Open Source Software Onboarding as a University Course: An Experience Report

Hao He*, Minghui Zhou*§, Qingye Wang*, Jingyue Li†
*School of Computer Science, Peking University, China
*Key Laboratory of High Confidence Software Technologies, Ministry of Education, China
†Department of Computer Science, Norwegian University of Science and Technology, Norway
{heh, zhmh, wqye}@pku.edu.cn, jingyue.li@ntnu.no

***Abstract*—Without newcomers, open source software (OSS) projects are hardly sustainable. Yet, newcomers face a steep learning curve during OSS onboarding in which they must overcome a multitude of technical, social, and knowledge barriers. To ease the onboarding process, OSS communities are utilizing mentoring, task recommendation (e.g., "good first issues"), and engagement programs (e.g., Google Summer of Code). However, newcomers must first cultivate their motivation for OSS contribution and learn the necessary preliminaries before they can take advantage of these mechanisms. We believe this gap can be filled by a dedicated, practice-oriented OSS onboarding course. In this paper, we present our experience of teaching an OSS onboarding course at Peking University. The course contains a series of lectures, labs, and invited talks to prepare students with the required skills and motivate them to contribute to OSS. In addition, students are required to complete a semester-long course project in which they plan and make actual contributions to OSS projects. They can either contribute to some recommended OSS projects with dedicated mentors, or contribute to any OSS project they prefer. Finally, 16 out of the 19 enrolled students have successfully contributed to OSS projects, and five have retained. However, the onboarding trajectories, final contributions, and retention outcomes differ vastly between the two groups of students with different course project choices, yielding lessons for software engineering education.**

***Index Terms*—open source software, open source onboarding, software engineering education**

I. Introduction

Open Source Software (OSS) has achieved tremendous success in the past decades with thriving projects and communities in almost all domains of science, engineering, and society. Meanwhile, people are raising concerns about whether an OSS project's success can be sustained over the next decades. For example, OpenSSL, despite being a critical component of the whole Internet, is only maintained by two overworked and unpaid developers (aged 46 and 59 at that time) before the disclosure of the notorious HeartBleed vulnerability [1]. Even for the Linux kernel, most of the maintainers are already in their fifties or sixties and *it's really hard to find people who are maintainers* [2], [3].

To secure sustainability, OSS projects need to maintain a healthy influx of newcomers [4], [5], but newcomer onboarding can be very difficult, especially for mature OSS projects, with technical, social, documentation, and knowledge barriers [6]. To alleviate this issue, researchers and practitioners have proposed strategies, guidelines, and tools for *projects* to support onboarding, such as mentoring [7], task recommendation [5], [8], [9], communication [10], and engagement programs [11]. However, the flip side of the coin is much less considered: what can we (i.e., researchers/educators) do to help *newcomers* of diverse backgrounds onboard OSS? Intuitively, newcomers need to prepare themselves with the necessary motivation, skills, knowledge, and behavior traits, before they can overcome the onboarding barriers and make successful contributions to OSS. Such preparations may also encourage their long-term commitment to OSS projects (where newcomer retention rate is often very low ($<1\%$) [12], [13]).

University students are an important source of OSS contributors, but their knowledge and skills are often insufficient for them to onboard and contribute effectively. Despite the prevalence of OSS in computer science (CS) and software engineering (SE) education [14]–[19], most of the effort focuses on the utilization of OSS for other pedagogical purposes (e.g., teaching SE [19]). The experience of teaching OSS onboarding in CS/SE courses is relatively scarce in the literature [20]. Previous work reports a variety of onboarding difficulties and mistakes when students attempt to contribute [18], [19], [21]. We believe more evidence is needed to uncover lectures, assignments, and interventions that are helpful for cultivating CS students into the next generation of OSS contributors.

We present our experience of teaching a course specifically designed for OSS onboarding to students majoring in CS or SE at Peking University. The main goal of this course is to prepare students with the necessary preliminaries (e.g., software engineering skills, social abilities, and motivation) so that they may face fewer barriers in the future when they attempt to contribute to OSS. To meet the teaching goals, we organize a series of lectures (for introducing the basics of OSS development), labs (for learning OSS-related techniques, tools, and best practices), and invited talks from OSS veterans (for cultivating students' motivation and awareness) throughout the course. At the same time, the course includes a semester-long course project in which students make contributions to OSS projects of their choice while being assisted (and supervised) by the professor and TAs. Grading includes the quality of lab submissions, in-class presentations, and the significance of their final OSS contributions.

In the 2021 Fall semester, 19 students enrolled in the course.

[§] Minghui Zhou is the corresponding author.

In the course project, they were offered two options: one was to contribute to some recommended OSS projects with the assistance of a dedicated mentor from each project; the other was to contribute to any OSS project they prefer while the professor and TAs offered (often high-level) guidance and feedback during their onboarding trajectories. Eventually, 16 students managed to make their first contribution to some OSS projects (the remaining three withdrew in mid-term). Their amount of contribution varied, from bug fixes of only several lines of changes to full-fledged new features with thousands of lines of code. They faced the typical barriers of project selection, task selection, communication, and technical hurdles during the entire process. Seven out of 16 (43.75%) students had to try out several OSS projects before making their first contribution. Despite the difficulties, students reported that they felt amazed by how OSS communities work, obtained a great sense of achievement when they successfully contributed, and learned the necessary technical and communication skills for overcoming the onboarding barriers.

In addition, we observe that the outcome of OSS onboarding differs significantly between students who selected to onboard a given OSS project (i.e., *the mentored group*) and students who searched and contributed to arbitrary OSS projects (i.e., *the free group*). More specifically, while students from the mentored group managed to make more significant contributions with fewer onboarding barriers, more students from the free group retained in their selected projects after the course. Through our anecdotal observations, we hypothesize that such differences may be related to their thinking patterns according to the theory of fast and slow thinking [22]. Such differences may indicate that although extrinsic factors (e.g., mentoring) accelerate onboarding, it is intrinsic motivations (e.g., personal interest) that help retention in OSS.

To summarize, this paper makes the following contributions:

- A detailed description of an OSS onboarding course design, which can be referred to by CS/SE educators interested in teaching OSS onboarding.
- Experience from the onboarding trajectories of students and our lessons for SE education and OSS onboarding.

II. Background

A. Newcomer Onboarding

Newcomers are driven by a variety of motivations to contribute to OSS projects (see Von Krogh et al. [23] for a literature review), including intrinsic motivation (i.e., ideology, altruism, kinship, and fun), extrinsic motivation (i.e., career and pay), and internalized extrinsic motivation (i.e., reputation, reciprocity, own-use, and learning). For example, they can be motivated by their desire to learn and accumulate software development experience, their willingness to help others, or a personal obligation of giving back to the OSS community as a user [23]–[25]. Recently, Gerosa et al. [26] investigate the shift of OSS contribution motivation. They find that motivations related to social aspects become more frequent. Although intrinsic and internalized extrinsic motivations are still prevalent, the motivation of “scratching one’s own itch” becomes rarer. They also discover that contributing to OSS often transforms extrinsic motivations into intrinsic ones.

When newcomers attempt to contribute to OSS projects, they can face a multitude of barriers. In a systematic literature review, Steinmacher et al. [6] identify six main categories of newcomer onboarding barriers: newcomers’ characteristics, newcomers’ orientation, reception issues, documentation problems, technical hurdles, and cultural differences. Mendez et al. [27] further discover that tool and infrastructure issues are prevalent in newcomer onboarding barriers in which social and culture factors (e.g., gender biases) are embedded .

To facilitate newcomer onboarding, researchers have proposed guidelines for newcomers and project mentors [7], [10], [28]. Even with the guidelines, it is still hard for newcomers to locate suitable development tasks on their own. Therefore, a series of studies investigate the good first issue mechanism and reveal its limitations [4], [29], [30]. Approaches are also proposed to help newcomers choose tasks by, e.g., tagging issues’ difficulty [8], [31], or automatically recommending “good first issues” to newcomers [5], [9], [32], [33].

B. Education for Open Source

1) Mentoring: Mentoring is a common and effective way to help newcomers onboard and grow their competence in an OSS project [34]. It is common for seasoned OSS developers to serve as mentors at any point of the OSS contribution process in which they recommend tasks, explain the code architecture, or assist various software development activities for newcomers. Through mentoring, mentors can help newcomers improve communication, collaboration, and information collection skills [34], [35]. OSS communities also offer specialized engagement programs (e.g., Google Summer of Code [11]) to mentor and educate students in the program. To improve mentoring practices, researchers have investigated various aspects of mentoring in OSS communities, including the characterization of its impact [36], [37], barriers of mentors [7], strategies during mentoring process [7], [11], and automated mentor recommendation [35], [38], [39].

2) Courses: The idea of incorporating OSS projects into CS/SE education is not new: educators have documented in the CS/SE education literature about their experience of using OSS to teach software engineering [17]–[19], [21], [40], database systems [14], software design [41], software evolution [42], object-oriented programming [43], etc. It is generally believed that OSS projects are ideal for teaching SE-related topics as they are easily accessible and offer opportunities for real-world development experience. Two systematic mapping studies [16], [44] summarize how OSS projects had been used to facilitate students’ learning of software engineering. However, students may encounter the typical OSS onboarding barriers which hinder the realization of pedagogical goals [18], [19], [21]. To alleviate the barriers, many courses opt to provide a predefined list of OSS projects in which the teaching staff has contact with [14], [17], [18], [40], [42], with only some exceptions [19], [20]. Morgan et al. [20] describe

TABLE I: The Weekly Course Schedule

Week	Lecture	Assignment
1	An Overview of OSS and Its Development	
2	Platforms, Tools, and Technologies for OSS Development	Lab 1: Familiarizing with Git and GitHub
3	OSS Contribution Guidelines	Lab 2: Knowing an OSS Project
4	*Invited Talks from OSS Veterans*: OSS Onboarding and Maintenance	
5	Task Selection and the Good First Issue Mechanism	Lab 3: OSS Task Selection
6	Course Project Opening Presentations	
7	*Lab Introduction*: Continuous Integration (CI) & Continuous Delivery (CD)	Lab 4: Configuring CI/CD Pipelines
8	Differences between Open Source and Proprietary Software Development	
9	*Lab Introduction*: Packaging Ecosystems & Dependency Management	Lab 5: Managing and Releasing Python Packages
10	*Invited Talks from OSS Veterans*: OSS Culture and Community	
11	Course Project Mid-Term Presentations	
12	Communication and Social Skills in OSS Development	Lab 6: Reflection on OSS Communication
13	OSS Governance and Community Operation	
14	Ongoing Challenges in OSS Sustainability: Ecosystems and Supply Chains	
15	Course Project Final Presentations	Project Report

two OSS development courses (in USA) which share some similarities with our course. As a comparison, our course is taught in a different cultural setting and our experiences significantly differ from theirs. Tan et al. [19] propose GitHub-OSS Fixit, an SE course project in which students select Java OSS projects they prefer and contribute bug fixes following eXtreme Programming (XP) practices. However, a course project alone may be insufficient to prepare students for OSS, as their students report many typical difficulties during OSS onboarding. This is one of the reasons that motivated us to design and teach a course specifically for OSS onboarding and overcoming the OSS contribution barriers.

III. Course Design

Compared with traditional software development, OSS development has a new set of models and techniques. Despite the growing importance of OSS in the industry and society, there was a lack of relevant courses in Peking University to help students understand the nature of OSS, master the hands-on techniques and tools for OSS development, and learn how to overcome the OSS onboarding barriers. These are essential for students regardless of their career path choices (academic or industry), as they are very likely to use, contribute to, or launch their own OSS projects in the future. As a bonus, students can also accumulate experiences on how to apply their CS knowledge into real-world SE tasks through OSS development.

To teach students OSS development methods, process, and techniques, we started a new elective course, *Open Source Software Development*, mainly for senior undergraduate students with a CS or SE major. As preliminary requirements, we advise students to have fluency in at least one programming language and possess some reasonable knowledge of SE. The preliminaries are not mandatory: we believe that students of any background can contribute to OSS in some way as long as they are highly motivated and master the necessary skills and knowledge that will be covered in this course.

A. The Course Schedule

The course consists of four main parts: lectures, invited talks, lab assignments, and a course project. Table I provides an overview of the course schedule. Note that the labs are assigned mapping the progress of course project.

The first five weeks are designed to familiarize students with the basics of OSS development, with a special focus on the strategies to overcome the OSS onboarding barriers. To this end, we cover topics including the fundamentals of software engineering, the notion and history of OSS, common platforms, tools, and techniques for OSS development, OSS contribution guidelines, and OSS task selection mechanisms. Moreover, we invite four OSS developers (all are currently core members of different OSS projects) to give presentations about how to onboard their OSS project and their experiences in OSS participation and maintenance. We expect these invited talks to provide vivid examples of real OSS development, which can help students select OSS projects, reduce the fear of unknown and motivate them to contribute to OSS projects. Meantime, we design three labs for students to practice with the knowledge learned. In Lab 1, students are asked to practice with git and GitHub by initializing a git repository, pushing the repository to GitHub, opening issues, forking the repository, and contributing back to the parent repository using a PR. In Lab 2, students are asked to choose an OSS project of their interest, locate and read contribution documentation, and setup the required development environment for that project. In Lab 3, students are asked to explore possible ways to contribute to the OSS project they choose in Lab 2. In Lab 2 and Lab 3, they are also required to produce reports to justify their selections, describe their experiences, and report any encountered challenges.

The remaining ten weeks are mixed with course project arrangements, lectures about advanced topics, and two labs covering CI/CD and dependency/release management, respectively, and one lab reflecting on communication practices. The course project is intentionally interleaved with other lectures and labs, as we expect OSS onboarding to be a time-consuming trial-and-error process hardly achievable in a short duration. Details about the course project design are described in Section III-B. There are also lectures and invited talks designed to broaden students' view of OSS and cover topics that are generally beneficial to have an understanding of, such as OSS community, culture, license, governance, sustainability challenges, etc. Lab 4 and Lab 5 cover some increasingly important SE topics in both open source and proprietary software development (yet not covered by Peking University's SE course). In Lab 4, we introduce the topic of continuous integration (CI) and continuous delivery (CD), which is leveraged by almost every OSS project for automated quality assurance. In the lab assignment, we provide a small Python graph library for which students are required to set up pre-commit hooks, code linters, unit tests, and a CI/CD pipeline using GitHub Action. In Lab 5, we introduce the topic of OSS packaging ecosystems and dependency management, as we believe it is increasing important for software developers to understand how to select, manage, and release OSS packages, and possibly students will practice in their course project. In the lab assignment, students further implement new features in the Python graph library with external dependencies and configure an automated CD pipeline to release the library to TestPyPI [45], a platform for experimenting with Python packaging. Finally, Lab 6 is designed to give students an opportunity to write down their own reflections on communication, for them to build a deeper understanding of communication best practices in OSS communities. Communication over Internet differs from physical communication in many ways, which can be both encouraging and challenging. Although the best practices are already covered in the first five weeks, we expect that such an in-depth understanding can only be built after students have had some real experiences.

B. The Course Project

Obviously, the pedagogical goals of our OSS onboarding course cannot be fulfilled without *actual* OSS onboarding and contribution experiences. However, given the known difficulties of OSS onboarding, it will be probably inappropriate to arrange the course project in a very short duration or to ask students to contribute to arbitrary OSS without any predefined guidelines, processes, and interventions. On the other hand, if we make the predefined processes too strict (e.g., contributing to a specific OSS project), the course project would deviate from real OSS onboarding and it would be hard to find an OSS project that matches the background and interest of all enrolled students. Therefore, the course project is designed to balance structure and flexibility, in which students have freedom in OSS project and task selection while we define (loosely) the processes and tasks they must complete.

At the end of the first lecture, we introduce to students the requirements and arrangements of the course project. They are offered two options. The first option is to contribute to a recommended list of OSS projects in which they can be helped in person by a mentor. In the Fall 2021 semester, we provided four OSS projects within our reach: OpenEuler [46], Kata Containers [47], PaddlePaddle [48], and TiDB [49]; all of their mentors also gave an invited talk in Week 4. The second option is to contribute to any OSS project of their choice. In this case, they must choose OSS projects and tasks according to the following criteria (to minimize the onboarding barriers and student pitfalls shown in prior work [6], [19], [20]):

- The project must be sufficiently popular (as a rule of thumb, $\geq$100 GitHub stars) with a non-trivial user base (e.g., a high number of downloads each week).
- The project must be actively maintained with rapid issue response time, sustained commit activity in the past six months, and more than ten previous contributors;
- The project should have a roadmap or an issue tracker that provides opportunities for external contributions;
- The student must be able to set up a valid development environment for their selected OSS projects;
- The student should try their best to align projects and tasks with their skills, background, and personal interest;
- The student should always evaluate the workload, difficulty, and risks of each task they attempt to take.

All the criteria are designed to ensure students to be able to learn and contribute within a limited timeframe ($\leq$10 weeks). Although they are not enforced, students are required to justify any deviations in the in-class presentations and we may intervene if necessary. In Lab 2 and Lab 3, students will have the opportunity to practice and familiarize themselves with these project and task selection criteria. Students are allowed to form teams but will be graded independently based on the presentations, the final report, and the significance of OSS contributions (more details in Section III-C).

For students choosing the first option, a WeChat group (similar to a Slack channel) is arranged with the presence of a mentor from their interested project. In this way, students can directly ask questions, request good first issues, and seek help if they encounter any technical hurdles. For students choosing the second option, they need to find projects and tasks, learn the necessary preliminaries, and figure out all the low-level details on their own, all of which make the second option inherently more challenging.

We arrange three in-class presentations (opening, mid-term, and final) throughout the course. The presentations serve two purposes: the first is for us to get informed of the students' progress and intervene if necessary; the second is for students to inspire and motivate each other through learning from peers [20]. In the opening presentation, students need to describe the OSS project they selected, their selection rationales, their understanding of the project (e.g., toolchains, tech stacks, architecture, and contribution guidelines), and their contribution plans. In the mid-term presentation, students must present their communication and development activities, the issues and barriers encountered, and their further contribution plans. In the final presentations, students need to summarize their onboarding trajectories, the final successful (and failed) contributions, their reflections, and their lessons learned.

After the final presentation (Week 15), students need to produce a project report as a formal document of their onboarding processes, successful (or failed) contributions, and individual reflections. We expect that a formal written report can push students to carefully reflect on their own experiences and lessons learned so that they can avoid similar mistakes in

TABLE II: An Overview of Students and Their Contributions to OSS

	ID	Final Project	Interactions (in GitHub)	Contributions	Line Changes		Retention
Mentored Group	S04	PaddlePaddle	1 PR	fix (3)	430++	176--	None
	S08	PaddlePaddle	1 Issue, 3 PRs, 2 Comments	feat (1)	3130++	0--	None
	S09	PaddlePaddle	2 PRs, 6 Comments	feat (1) fix (1)	150++	41--	None
	S11[‡]	PaddlePaddle & Other(s)	1 Issue, 2 PRs, 5 Comments	feat (7) fix (3) doc (1)	10782++	4442--	Issues (in Other(s))
	S12	PaddlePaddle	1 Issue, 5 PRs, 16 Comments	feat (1) fix(1)	3088++	1--	None
Free Group	S01	Other(s)	4 PRs, 17 Comments	fix (1) doc (1)	715++	82--	None
	S02*	Other(s)	2 Issues, 2 PRs, 12 Comments	feat (1) fix (1)	390++	7--	None
	S03*	Other(s)	2 Issues, 4 PRs, 39 Comments	feat (1) fix (1) doc (1)	68++	22--	None
	S05*	Other(s)	2 PRs, 2 Comments	fix (1)	8++	8--	Issues & Commits & PRs
	S06[†]	Other(s)	3 Issues, 3 PRs, 7 Comments	fix (4)	318++	298--	Code Reviews
	S07	Other(s)	4 Issues, 11 PRs, 40 Comments	fix (1) doc (2)	430++	24--	Commits & PRs
	S10*[†]	Other(s)	2 Issues, 2 PRs, 1 Comment	feat (1) doc (1)	278++	403--	Commits & PRs
	S13*	Other(s)	2 Issues, 2 PRs, 4 Comments	feat (1) doc (1)	137++	7--	None
	S14	Other(s)	4 Issues, 4 PRs, 4 Comments	fix (2)	6++	10--	None
	S15*	Other(s)	2 PRs, 4 Comments	fix (4)	238++	77--	None
	S16*	Other(s)	5 Issues, 4 PRs, 8 Comments	feat (4)	1033++	913--	None

* These students have switched projects during the course.
[†] These students already have OSS contributions prior to taking our course.
[‡] We put S11 into the mentored group because he mainly contributes to PaddlePaddle.

the future. We will also use these reports as an invaluable reference for grading and improving course design.

C. Grading

Grading is based on students' in-class participation (10%), lab assignments (50%), and the course project (40%). For each lab, the lab assignments are given clear grading criteria. We expect that the 60% of in-class participation and lab assignments can be easily seized by students with a reasonable amount of devotion and hardworking, to relieve their anxiety with grades. The classes and labs are designed for learning, not for evaluating performance after all.

The main part where the students' grades can greatly differ is the 40% of course project, which is composed of 15% of in-class presentations, 15% of OSS contribution significance, and 10% of the final project report. The presentations and reports are evaluated w.r.t. their adherence to our requirements and their depth of thought; the OSS contributions are evaluated with respect to the amount of work, the appropriateness of interaction with OSS communities, and the overall difficulty and significance levels. The acceptance of several non-trivial contributions is needed for a full grade, but failed attempts can still count for some grade points.

IV. Student Onboarding Trajectories

In the Fall 2021 semester, 19 students enrolled in our class, and we will use S01 - S19 to refer to them in this Section. During the semester, three withdrew in mid-term by following our university's procedures (the withdrawal rate was high that year due to the ongoing COVID-19 pandemic; another possibility is that some were afraid of the difficulty in accomplishing actual OSS contributions). We provide an overview of the remaining 16 students and their contributions in Table II. We refer to students who choose to contribute to our given OSS projects with the help of the mentor as *the mentored group* and students who locate OSS projects on their own as *the free group* (Section III-B).

In this Section, we will describe: 1) students' projects & tasks selection, 2) their interactions with OSS communities, 3) their contribution outcomes, 4) their own reflections, and 5) their retention in OSS. The results come in part from a thematic analysis [50], [51] of their presentations, lab submissions, and project reports, and in part from our own observations. Where appropriate, we also include (translated) quotations to support our observations.

A. Project and Task Selection

We provided the following interventions before students chose their OSS projects and tasks. In the lectures of Week 3 and Week 5, we explained in detail the rationales and the common pitfalls behind our provided project and task selection criteria. In Week 4, the talks covered processes, guidelines, and contribution opportunities for each invited project. Students were given opportunities to practice with the typical project and task selection processes in Lab 2 and Lab 3, respectively. To understand the effectiveness of the interventions, in the opening presentations and the project reports, students were asked to document their choices and the considerations behind their choices.

1) Project Selection: Five students opted for option one (i.e., the mentored group) and 14 students opted for option two (i.e., the free group, among which three withdrew from the

TABLE III: Students' Considerations during Project Selection

	# of Students		
Theme	Mentored	Free	All
Alignment	5 (100%)	14 (100%)	19 (100%)
Skill	1 (20%)	10 (71%)	11 (58%)
Personal Interest	4 (80%)	4 (29%)	8 (42%)
Experience as User	0 (0%)	7 (50%)	7 (37%)
Perceived Ability	2 (40%)	2 (14%)	4 (21%)
Community	4 (80%)	10 (71%)	14 (74%)
Active	2 (40%)	9 (64%)	11 (58%)
Newcomer Friendly	4 (80%)	6 (43%)	10 (53%)
Welcome External Contribution	2 (40%)	2 (14%)	4 (21%)
Project	3 (60%)	8 (57%)	11 (58%)
Low Contribution Barrier	2 (40%)	5 (36%)	7 (37%)
High Quality	1 (20%)	3 (21%)	4 (21%)
Intrinsic Motivation	0 (0%)	1 (7%)	1 (5%)

course). All students in the mentored group chose PaddlePaddle. In contrast, students from the free group selected OSS projects of high diversity (in terms of programming language, application domain, size, complexity, and maturity).

We provide a summary of their justifications in Table III. As expected, all students (19, 100%) mention the alignment of OSS projects with their specific skills (11, 57.9%), interest (8, 42.1%), user experience (7, 36.8%), and perceived ability (4, 21.1%). In terms of skills, students generally wish to onboard OSS projects that align well with their familiar programming languages and application domains. For example, S05 mentions that: *Because I am doing research on computer vision and deep learning, and I am mainly programming in Python and PyTorch for object detection, I would like to participate in deep learning projects and projects mainly written in Python.* Besides, their personal interest also plays an important role and many wish to contribute to the OSS that they had some experiences as a user. As stated by S06: *My main interests are data science, quantitative finance, and blockchain, so I would like to contribute to software libraries in this domain that I like and I use a lot.* Finally, some care whether the project is "not too difficult" for their current ability.

The majority (14, 73.7%) of students also care about whether the corresponding OSS community has a benign atmosphere and is capable to support their onboarding. The most frequently mentioned community attributes are activity (11, 57.9%) and newcomer friendliness (10, 52.6%). For example, S03 mentions: *If a community can respond to my questions quickly, the chance of my contribution getting accepted is much higher.* S17 notes: *My selection criteria for OSS projects is that they should have a healthy community willing to accept contributions and offer help to newcomers. Ideally, they should signal that through the labeling of "good first issues."* Some students mention whether the project is willing to accept external contributions (4, 21.1%) as an additional factor because some projects only "seem to be open-source" but actually harder to contribute as an outsider.

Certain characteristics of the software projects themselves are also frequently considered (11, 57.9%). In general, some students (7, 36.8%) care about whether the project seems to have low contribution barriers and some students care about whether the project is of high quality (4, 21.1%). For the former case, students expect 1) they can easily set up a development environment for the projects they onboard; 2) the projects are in rapid development and need contribution; 3) the projects they onboard are of small size, well documented, and have high modularity. For the latter case, they wish the projects they onboard are among the "prestigious" ones that are mature, impactful, sustainable, and well-maintained.

Only S06 has mentioned intrinsic motivation of contributing, including the eagerness to improve and seeking challenges, as noted: *I am eager to improve the project ...Using QML to compose graphic interfaces is not only outstandingly effective but also a good challenge.*

The students' reports reflect that they have learned well about the rationales for OSS project selection. In terms of the justifications, differences between the mentored and free groups are small. The free group focuses more on the alignment of skills (71%) compared with the mentored group (20%); the mentored group focuses more on newcomer friendliness (80%) compared with the free group (43%).

It is surprising for us to find that all students in the mentored group choose PaddlePaddle while the other three projects are not chosen at all. Through our inquiries, it turns out that these students are generally more interested in and familiar with deep learning, and they think that projects related to operating systems, cloud computing, and database are too "hardcore" for their current programming ability and expertise. This can be an example of the mere exposure effect [52], [53] under the availability heuristic [54], that they immediately prefer PaddlePaddle because they are familiar with AI and consider PaddlePaddle as "the easier choice" for them (Table III). For some students in the mentored group, we conjecture that they choose to be mentored mainly because they are less intrinsically motivated compared to students in the free group (i.e., they only want to earn credits with ease instead of facing the unknown and overcoming difficulties).

2) Task Selection: Similar to project selection, we summarize student common considerations for task selection in

TABLE IV: Students' Considerations during Task Selection

	# of Students		
Theme	Mentored	Free	All
Preferred Task Type	5 (100%)	13 (93%)	18 (95%)
Bug Fix	2 (40%)	7 (50%)	9 (47%)
Good First Issue	0 (0%)	6 (43%)	6 (32%)
Feature	5 (100%)	1 (7%)	6 (32%)
Documentation	0 (0%)	3 (21%)	3 (16%)
Test	0 (0%)	3 (21%)	3 (16%)
Other	1 (20%)	2 (14%)	3 (16%)
Task Selection Strategy	0 (0%)	10 (71%)	10 (53%)
Scratch One's Own Itch	0 (0%)	4 (29%)	4 (21%)
Start from Easy to Difficult	0 (0%)	4 (29%)	4 (21%)
Limited Change Scope	0 (0%)	2 (14%)	2 (11%)
Reproducible Bug	0 (0%)	1 (7%)	1 (5%)
Has Reference Implementation	0 (0%)	1 (7%)	1 (5%)
Recent Task	0 (0%)	1 (7%)	1 (5%)

Table IV. Almost all students (18, 94.74%) have a specific task type in mind when they locate their initial tasks and over a half (10, 52.63%) explicitly mention their task selection strategies.

We notice a huge difference between the two groups. The mentored group preferred more on implementing new features, while the free group preferred to begin with easier tasks (e.g., bug fixes, good first issues, writing documentation, testing, etc). Only the free group mentioned task selection strategies. The differences come from the fact that the mentored group can often quickly propose or get assigned a task by their mentor (e.g., implementation of a state-of-the-art deep learning model). On the other hand, the free group is faced with much more challenges in finding a starting task that aligns well with their past experiences. For this purpose, they need to apply various task selection strategies, such as 1) search for bug fixes that scratch their own itch (i.e., they have encountered the same bug during their use of the OSS); 2) begin with easy tasks to familiarize themselves with the development process and increase their ability for more difficult tasks, 3) solve reproducible, recent issues with a limited change scope, etc.

3) The Switch of Projects and Tasks: However, even with those considerations, project and task selection is still prone to fail for the free group. In the mid-term presentation, seven out of 11 (63.6%) students reported that they have abandoned the initial projects and seek to find contributions to other projects. Among them, six mentioned their reasons: *hard to find tasks* (S02 and S15), *lack response* (S10 and S16), *lack ability* S13), and *failure to setup a working development environment* (S03).

S02 and S15 found that locating suitable tasks to work on can be difficult and the "low-hanging fruits" may be easily preempted by others. For example, S02 wanted to contribute to `pandas` [55] but eventually concluded that it is already highly mature and he could not find anything to work with.

S10 and S16 switched because they did not receive timely feedback from project maintainers due to their inactivity or reluctance to help. S10 notes: *Some projects have a low maintenance frequency and your issue may stay unresponded for several months*; S16 mentions that: *I did all things requested by the contribution guidelines and encountered issues, but nobody replies to me and helps me*.

S13 found his selected projects and tasks were beyond his current ability. This was common among students because they were still learning and yet to be software professionals. As noted: *Selecting open-source projects is very hard because I do not have the ability to contribute to those I use and it is difficult to contribute to those that I do not use.*

For S03, his initial selection was blocked by development environment setup, as mentioned: *most of the bug reports in [an IoT Application] need non-trivial hardware configuration to replicate*.

B. Communication and Interactions

In Fall 2021, we included communication related topics in two lectures (Week 3 and Week 12). In Week 3, we provided a brief coverage of common means for communication (issues, PRs, mailing lists, etc.) and best practices on how to work with OSS communities. In Week 12, we revisited the topic with in-depth discussions, taking the Linux Kernel and several studies as examples, to show what should be expected and what should be avoided when interacting with OSS communities. Through our teaching, we expect students to be equipped with some knowledge about communication in OSS communities. To evaluate these teaching activities, in the three presentations, students were requested to report their communication activities and we provided feedback where necessary. We report their communication activities in this Section.

Students used various means for communication, among which the most frequently used is GitHub issues (mentioned by 13 students). The less frequent ones include mailing lists (six students), Slack channels (three students), user forums (three students), Twitter (two students), Stack Overflow (two students), and WeChat Groups (two students).

We summarize the effort of communication spent by the two groups of students, respectively, as shown in Table V. We can observe that the mentored group has more opened issues and pull requests, indicating that they may be more focused on task-related communication. Most of their communication is straightforward because they have already been told by their mentor about what to do for their assigned tasks. On the other hand, the free group has more comments and significantly more words in their issue/PR bodies (9.4x) and comments (5.6x) than the mentioned group. This is probably because they have spent much more effort in locating tasks, communicating with other developers to figure out what they should do, and may iteratively refine their contribution according to the maintainers' requests.

Communicating with strangers in OSS community is mostly a new experience for our students. Therefore, we encouraged them to communicate bravely and actively, especially when they were struggling at certain aspects. This turned out to be beneficial for both task selection and getting contributions accepted. For example:

1) When having no idea how to contribute, students explicitly asked for good first issues. For example, S07 commented in an issue: *I am new to the community of [OSS Name] and I really want to contribute to the project of [OSS Name]. Are there any suitable issues to start with right now?* This comment was replied: *Thanks for the enthusiasm. We have a similar issue at the [OSS Name] repository. You may find the issues that fit your interests here [issue link].*

2) When interested in a certain issue or task, students expressed their interest to be assigned by project maintainers, boosting their motivation to contribute. For example, S11

TABLE V: Communication Effort in Different Groups (Mean).

Metric	Mentored	Free	All
# of Issues	2.00	1.55	1.68
# of PRs	4.40	2.81	3.31
# of Comments	10.09	11.20	10.43
Issue/PR Body Length	10.56	99.59	79.56
Comment Length	22.76	128.17	102.48

commented on an issue: *Is the feature being implemented? If not, I'm willing to give it a try*. He was answered with: *please go ahead with your plans. This will be greatly appreciated.*

3) When implementing complex features, students discussed with community before implementing. For example, S11 proposed an issue to discuss what he/she wants to contribute (a multi-label classification feature). He/she then further opened an implementation PR after the proposal is approved.

4) If there was no response from reviewers, students sent reminder messages asking for comments on their issues or code reviews for their PRs. For example, S16 commented "*can anyone take a look at this?*" after his PR was delayed for four days. The PR was merged the next day after his reminder.

5) If the contributions were questioned by reviewers, some students were persistent and eager to defend their contributions. The example of S01 especially impressed us: when he opened a PR for translating documentation, a reviewer replied: *...It failed to meet our criteria for documentation. We cannot accept unedited machine translations...* He insisted on improving the PR despite the discouraging reply (he was not machine translating, just not proficient enough in the target language). The PR eventually got merged after several rounds of improvements. The use of polite terms like *thanks for your help*, *thanks, my pleasure*, *sorry for the late response*, etc., can help in winning assistance from core maintainers.

However, we also observed some (unexpected) communication problems from some students, which brought a negative impact on their onboarding process. One especially prevalent problem was the inappropriate use of English (e.g., broken sentences, abbreviations of non-English terms), which in part caused unwelcoming receptions. There were also cases where students did not follow the project contribution process and issue templates, causing their issues to be closed. Some students gave up on their contributions entirely when challenged or requested changes by the project maintainers.

C. Contribution Outcomes

All students have successfully contributed to at least one OSS project and their contribution status is summarized in Table II. In total, they submit 51 contributions among which 46 have been accepted and merged into the main branch. Their contributions fall in three categories: implementing new features (feat), fixing bugs (fix), and improving documentation (doc). The size of their contribution significantly varies, ranging from merely changing one line of code to huge patches with thousands of lines of changes. For example, the first successful OSS contribution of S03 is to improve a warning message to make it less confusing to users, which only requires one line of change to the corresponding string literal. In the other extreme, several students (S09, S11, S12) contribute state-of-the-art models to PaddlePaddle which require thousands of lines of changes to implement. The five non-accepted contributions happen due to re-submissions of PRs (two cases), issue preemption (one case), project inactivity (one case), and student giving up to contribute (one case).

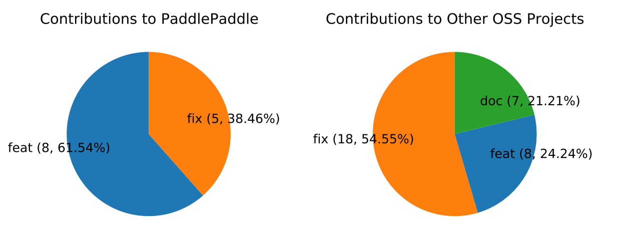

Fig. 1: Distribution of Task Type in the Two Groups

Both the quantity of change and the distribution of task type differ significantly between the two groups. As shown in Figure 1, the mentored group mainly contributes to PaddlePaddle with implementations of new features, with a much higher number of changed files and added lines (see Table II). Contributions by the free group are mainly bug fixes, with some new feature implementation and some documentation improvement. In general, the mentored group makes contributions of a much bigger size and a higher amount of work. And the free group presents a diversity in terms of the tasks and projects.

We conjecture that the differences between the mentored and free group in their OSS contribution outcomes are due to three main reasons. First, the presence of mentorship plays a strong factor. The mentor (one of PaddlePaddle's core members) communicated directly with students via instant message to answer any questions from students. He also offered guidance on how to propose a major feature (e.g., a new state-of-the-art model) and follow project conventions. This encouraged students to try bigger tasks (without the fear of mistakes or rejection). Second, authentic demonstrations of successful contributions (i.e., a role model) encouraged students to try bigger tasks. For the mentored group, S11 was such a role model: he had not only submitted many PRs but was also actively helping others when they are struggling to contribute. Finally, the free group had spent much more (and often) invisible effort on selecting projects & tasks, learning, and developing their own motivations for OSS contribution. Having made less significant contributions does not necessarily mean that students in the free group have spent less effort or learned less from their experiences. They may benefit more in the long run.

D. Students' Reflection

In the project reports, we ask students to reflect on their own experiences. We identify common themes in their reflections through a thematic analysis of their reflections, as shown in Table VI.

1) Students' Positive Feedback: We are impressed to find out that students feel highly positive about the unique experiences brought by our course.

Students mention that they feel a strong sense of achievement after they manage to make contributions to an OSS project. As noted by S04: *I really thank this course, which helps me experience the charm of participating in open source* *and I really like maintaining open source projects. When I successfully solve problems for someone else, I feel a great sense of achievement and happiness.*

Another frequently mentioned feedback is that they are impressed with how warm and welcoming OSS communities are compared with other online communities. For example, S05 notes: *Open source communities are very friendly...Everyone is smart, speaks nicely, and is very helpful. This is very touching in a world where the Internet is becoming increasingly polarized and even hostile.* S15 also notes: *People from the open source community are very passionate and polite.*

Some students once thought that GitHub is just a place where you can get software "for free"; they had no idea how open source works and why such a model can incubate great software. Now, they realize the complexity and charm behind. As noted by S05: *Hosting an open-source project is more than simply adding a license to your code and putting it online. You need to spend a lot of energy writing documentation, organizing files, communicating, and enhancing the compatibility and robustness of your code*; and S16: *I am impressed how a group of people who have even not met each other can start a heated discussion and contribute...From my perspective, open-source is more than a sort of license or development model. It is an open mindset where you respect everyone, trust newcomers, appreciate and value others' opinions, sharing a common goal, and a common vision.*

2) Challenges Encountered: Even under a structured course setting, they still encounter onboarding challenges and barriers. In their reports, they documented and reflected on the challenges they had encountered and how they overcome them.

The most frequently reported challenge is *Communication*. Students find it hard to properly seek help, ask questions, use English properly, etc. For example, project maintainers refuse to help S01 when S01 asks a question without trying to read project documentation or investigating by himself. S05 notes: *I feel my English is weak and need to think twice on how to communicate without causing confusions, especially for technical terms and expressions.* In their reflections, they conclude many lessons on effective communication. For example, S02 concludes: *You need to have a sufficient understanding of the project before asking questions and communicating with others. They will not treat you as a student but an independent professional. You need to learn a lot from project documentation by yourself...You should not underestimate yourself and be brave to communicate.* S03 notes: *You should be active and show a good attitude. Try your best to help developers understand what you are doing, and sometimes it will help to directly tell them you are new to their project.*

TABLE VI: Students' Reflections after the Course

	# of Students		
Theme	Mentored	Free	All
Positive Feedback	2 (40%)	3 (27%)	5 (26%)
OSS Communities are Welcoming	1 (20%)	2 (18%)	3 (19%)
Sense of Achievement	1 (20%)	1 (9%)	2 (13%)
OSS is Non-Trivial	0 (0%)	2 (18%)	2 (13%)
Challenges Encountered	2 (40%)	8 (73%)	10 (63%)
Communication	1 (20%)	5 (45%)	6 (38%)
Project & Task Selection	0 (0%)	4 (36%)	4 (25%)
Environment/Process/Tool Usage	2 (40%)	2 (18%)	4 (25%)
Life Lessons	1 (20%)	2 (18%)	3 (19%)

Many students also report *Project & Task Selection* as their main onboarding challenge. For example, S07 notes: *[project name] is highly mature and if you want to contribute to such a community, you need to find bugs and improvement directions from your own specific applications.* S10 concludes: *Selecting an open source project is not easy: your initial explorations for "intentionally seeking contributions" cannot progress, while some unexpected problems can trigger interactions with the open-source community and get positive feedback.* In retrospection, they realized the necessity to have a strong personal interest in an OSS project and focus more on the process instead of deliberately seeking outcomes, *doing good deeds without asking for reward.* Even if many initially failed to contribute to some projects, they acknowledge that they have accumulated experiences throughout their trials-and-errors.

Environment, Process, and Tool Usage also pose barriers to students. Even with the courses covering common tools and processes, each OSS project may use a different toolchains and process. The perfectionism of renowned OSS projects also pose challenges to many students. Consequently, they still need to learn a large amount of new knowledge and get into unexpected failures, before figuring out the right path to do everything. Although the process is painful, in the aftermath, they begin to understand software development in the real world. For example, in terms of development environment, S05 realizes: *Software needs to run on diverse OS, environment, devices, and in different people, which makes the problem really complex. You will never imagine how a program can come with all kinds of problems in other people's device.* S04 also writes: *I learn to always check twice before submitting a PR and follow community conventions closely to ensure there will be no unexpected problems.*

Through their experiences of OSS participation, students realize the importance of having a strong interest and passion for what they do, being brave and resilient, and focusing more on the experience instead of the outcome. For example, S02 writes: *Do not mistake desire for passion and do what you really want to do*, and S13 notes: *Through the process of "gold washing," I have learned many things and tried a lot. Even if I may not really do it, at last, everything I learn in this process is invaluable, from the first build to the last contribution.*

E. Retention in OSS

Six months later, we observe further OSS contributions from five students (Table II). Their contributions include issue reporting, bug fixes, and code reviews. However, their quantity of post-course contribution is still minor. Whether they will remain as long-term contributors is still yet to be seen.

Two students have started their own OSS projects and both of them have achieved a noticeable impact. S05 created a utility app for students at Peking University to automatically submit COVID-19 health monitoring information. At the time of writing, it has 173 commits, 179 stars, five external contributors, and a healthy stream of issue reports. S10 created and maintained a custom build for a not actively maintained software distribution to include the latest version of compilers in it. This custom build attracted a number of users and earned 15 stars before the official build begins to update recently.

Interestingly, although the mentored group has larger contributions, none of them remain in PaddlePaddle after the course. This is quite alarming, indicating that although close mentoring helps them make more significant contributions, it may not be sufficient for retaining newcomers. They may be overly focused on code implementation and have learned less about OSS participation and intrinsic motivations.

V. Lessons Learned

In terms of the OSS contribution outcomes, students' reflections, and their retention, we consider our course as mostly a success. Students report to have increased their ability in several dimensions: communication, social behaviors, technical skills, problem solving, ways of thinking, etc.

The main complaint about our course is grading: students report that the opaqueness of grading criteria increased their anxiety about grades in the course project. In general, students are highly concerned about their GPA, and many set GPA-oriented goals in their courses, but the outcome of OSS onboarding is inherently unpredictable. Moreover, we refrained from providing detailed grading criteria beforehand in Fall 2021 to prevent students from overfitting the criteria. This, in hindsight, turned out to cause anxiety and reluctance to choose challenging projects and tasks among some students. To balance between fair grading and students' learning achievement, we decide to disclose the detailed grading criteria at the beginning of the course in Fall 2022 with more emphasis on the effort and process than on the contribution outcome. We also advise students to set learning-oriented goals to maximize their gain. The impact of these improvements is yet to be seen.

Even in a structured and guided setting, students still report many technical and social barriers during their onboarding. For some of the barriers with project & task selection, tools, and communication, we believe this can be further addressed by better-designed hands-on lab assignments. However, as also pointed out by Morgan et al. [20], a significant amount of self-directed learning is inevitable during OSS onboarding (e.g., on project-specific tools) and it is impossible to cover everything in a course. Instead, it is vital for an OSS onboarding course to cover "the philosophy of learning" (i.e., learning how to self-learn) in the context of OSS development.

For reducing the social barriers, apart from teaching best practices, we could have done better by telling students to expect criticism from the OSS community and handle them correctly. We could also provide vivid examples of past successes and failures (ideally, from past students) to help them set proper expectations and be more mentally ready for possible challenges. That is exactly what we did in the Fall 2022 semester.

We are especially concerned with the fact that students from the mentored group are less likely to remain in OSS compared with students in the free group. Through our observations, we hypothesize that the choice of being mentored or not is related to their thinking patterns: the mentored group is more tempted to behave with "fast thinking" while the free group more often applies "slow thinking" deliberately. According to psychology research [22], the human mind has two thinking systems: "fast thinking" and "slow thinking". The former is fast, instinctive, and emotional while the latter is deliberative and logical. All of us possess both systems and we may be affected by either one in decision-making and problem-solving, in which "fast thinking" works autonomously and deliberate attention and effort are needed to activate the "slow thinking" system.

For some students in the mentored group, they may be affected by intuitive heuristics from their prior experience [54] (which is a typical pattern of "fast thinking"), causing them to replace the challenging problem of *"what should I do to contribute to an OSS project?"* with an easier one: *"can I quickly write some code within my expertise so that I can earn good grades with minimum effort?"* The latter option becomes possible with the presence of an extremely friendly mentor from PaddlePaddle, who basically removed all typical onboarding barriers and enable students to focus entirely on implementation (using their knowledge in deep learning). It is questionable whether they really learned the sufficient key aspects of OSS development from our course.

On the contrary, we observe some typical patterns of "slow thinking" from the free group: students often deliberately, rationally, and logically think about the potential solutions to challenging problems, overcoming the heuristics and biases induced by "fast thinking." When the reception from project maintainers does not go well or they are faced with a daunting task, they know how to identify possible solutions and self-learn (by carefully reading project guidelines, documentation, code, etc). Despite their difficulties in project & task selection and their less significant contributions, more of them report having benefited from our course, continue contributing to OSS projects, and some even launch their own OSS projects.

Such contrast yields several additional lessons. First, the OSS onboarding barriers can be beneficial in fostering "slow thinking" and deliberately lowering them may harm the fulfillment of teaching goals. Even if mentors are involved in the course, they should not behave like a "babysitter" but more like a "receptionist" guiding students to the learning resources, social events, and contribution opportunities. This will create a more challenging environment in the course project, pushing students to learn about the essence of OSS development.

Second, the development of intrinsic motivation for software development should also be an important teaching goal. Without such motivation, students are more prone to setting GPA-oriented goals in the course and apply more "fast thinking." According to student reflections (Section IV-D) and previous studies [23], [26], newcomers are more intrinsically motivated if they have experienced the rewarding part of OSS development. Students can feel especially rewarded by the process of overcoming onboarding barriers and their contributions to OSS projects of their true interest. This highlights the importance of retaining onboarding challenges in course project design, giving students high flexibility in project & task selection, and advising them to follow their true interests. According to the Self-Determination Theory [56], [57], intrinsic motivation can be also fostered by building relatedness to others in the course. Similar to Morgan et al. [20], it may be beneficial to organize more in-class discussions about the progress of course projects, apart from the three more formal presentations.

VI. Limitations

It is important to note that our report is based on a relatively small number of students and it is hard to use systematic methods to support our findings or to exclude possible confounding factors in our interpretation. To mitigate biases, we try our best to use our own judgments and our observation of students (both in class and in personal Q&As) to calibrate the reported results. In total, the professor and two TAs are involved and this paper is presented as a consensus of all authors.

Although the results are based on running the course in one specific university, i.e., Peking University, we believe that our experiences and the lessons learned should be of sufficient practical value for SE educators interested in teaching OSS in a similar cultural and academic setting. As future work, we plan to replicate this course in different universities to investigate the impact of culture, pedagogical methods, students' learning habits, and personalities on the course design and outcome.

In addition, our hypotheses of the students' thinking patterns are based on our own observations and inferences, not on a solid empirical foundation. A large part of the evidence that helped us reach these hypotheses come from our casual, off-class talks with the enrolled students, but we did not record or took notes, nor did we apply systematic scientific methods to analyze them. Future empirical research is necessary to establish the precise relationship between newcomer personality and newcomer OSS onboarding outcomes.

VII. Conclusion

In this experience report, we have presented our experience of teaching an OSS onboarding course, with a detailed course design and the onboarding outcomes of students in Fall 2021. In general, we feel that the course is a success and will benefit the enrolled students in the long term (one invited speaker was even jealous of this course and hoped he could have enrolled when he was an undergraduate). The course will continue to enroll students each Fall semester and we will iteratively improve it; we have already implemented some of our learned lessons in the Fall 2022 course. We hope this paper can provide inspiration and practical values for others willing to integrate OSS onboarding in SE education courses.

VIII. Data Availability

The course materials (in Chinese) are currently maintained as an open source project accessible at:

https://github.com/osslab-pku/OSSDevelopment

To avoid leakage of sensitive data and ensure privacy, we choose to anonymize all student information provided in this paper and not to disclose any raw data related to students.

Acknowledgment

This work is supported by the National Natural Science Foundation of China Grant 61825201 and 62142201. We thank Pengcheng Li for his contribution as a course TA in Fall 2021 and all the brilliant students who took our course.

References

[1] C. Stokel-Walker. (2014, April) The Internet is being protected by two guys named Steve. [Online]. Available: https://www.buzzfeed.com/chrisstokelwalker/the-internet-is-being-protected-by-two-guys-named-st

[2] T. Anderson. (2020, June) 'it's really hard to find maintainers...' Linus Torvalds ponders the future of Linux. [Online]. Available: https://www.theregister.com/2020/06/30/hard_to_find_linux_maintainers_says_torvalds/

[3] M. Zhou, Q. Chen, A. Mockus, and F. Wu, "On the scalability of linux kernel maintainers' work," in *Proceedings of the 2017 11th Joint Meeting on Foundations of Software Engineering*, ser. ESEC/FSE 2017. New York, NY, USA: Association for Computing Machinery, 2017, p. 27–37. [Online]. Available: https://doi.org/10.1145/3106237.3106287

[4] X. Tan, M. Zhou, and Z. Sun, "A first look at good first issues on GitHub," in *ESEC/FSE '20: 28th ACM Joint European Software Engineering Conference and Symposium on the Foundations of Software Engineering, Virtual Event, USA, November 8-13, 2020*, P. Devanbu, M. B. Cohen, and T. Zimmermann, Eds. ACM, 2020, pp. 398–409. [Online]. Available: https://doi.org/10.1145/3368089.3409746

[5] W. Xiao, H. He, W. Xu, X. Tan, J. Dong, and M. Zhou, "Recommending good first issues in github OSS projects," in *44th IEEE/ACM 44th International Conference on Software Engineering, ICSE 2022, Pittsburgh, PA, USA, May 25-27, 2022*. ACM, 2022, pp. 1830–1842. [Online]. Available: https://doi.org/10.1145/3510003.3510196

[6] I. Steinmacher, M. A. G. Silva, M. A. Gerosa, and D. F. Redmiles, "A systematic literature review on the barriers faced by newcomers to open source software projects," *Inf. Softw. Technol.*, vol. 59, pp. 67–85, 2015. [Online]. Available: https://doi.org/10.1016/j.infsof.2014.11.001

[7] S. Balali, I. Steinmacher, U. Annamalai, A. Sarma, and M. A. Gerosa, "Newcomers' barriers. . . is that all? An analysis of mentors' and newcomers' barriers in OSS projects," *Comput. Support. Cooperative Work.*, vol. 27, no. 3-6, pp. 679–714, 2018. [Online]. Available: https://doi.org/10.1007/s10606-018-9310-8

[8] S. Balali, U. Annamalai, H. S. Padala, B. Trinkenreich, M. A. Gerosa, I. Steinmacher, and A. Sarma, "Recommending tasks to newcomers in OSS projects: How do mentors handle it?" in *OpenSym 2020: 16th International Symposium on Open Collaboration, Virtual Conference, Spain, August 26-27, 2020*, G. Robles, K. Stol, and X. Wang, Eds. ACM, 2020, pp. 7:1–7:14. [Online]. Available: https://doi.org/10.1145/3412569.3412571

[9] H. He, H. Su, W. Xiao, R. He, and M. Zhou, "GFI-Bot: Automated good first issue recommendation on GitHub," in *Proceedings of the 30th ACM Joint European Software Engineering Conference and Symposium on the Foundations of Software Engineering, ESEC/FSE 2022, Singapore, Singapore, November 14-18, 2022*, A. Roychoudhury, C. Cadar, and M. Kim, Eds. ACM, 2022, pp. 1751–1755. [Online]. Available: https://doi.org/10.1145/3540250.3558922

[10] X. Tan and M. Zhou, "How to communicate when submitting patches: An empirical study of the Linux kernel," *Proc. ACM Hum. Comput. Interact.*, vol. 3, no. CSCW, pp. 108:1–108:26, 2019. [Online]. Available: https://doi.org/10.1145/3359210

[11] J. D. O. Silva, I. Wiese, D. M. Germán, C. Treude, M. A. Gerosa, and I. Steinmacher, "A theory of the engagement in open source projects via summer of code programs," in *ESEC/FSE '20: 28th ACM Joint European Software Engineering Conference and Symposium on the Foundations of Software Engineering, Virtual Event, USA, November 8-13, 2020*, P. Devanbu, M. B. Cohen, and T. Zimmermann, Eds. ACM, 2020, pp. 421–431. [Online]. Available: https://doi.org/10.1145/3368089.3409724

[12] M. Zhou and A. Mockus, “What make long term contributors: Willingness and opportunity in OSS community,” in *34th International Conference on Software Engineering, ICSE 2012, June 2-9, 2012, Zurich, Switzerland*, M. Glinz, G. C. Murphy, and M. Pezzè, Eds. IEEE Computer Society, 2012, pp. 518–528. [Online]. Available: https://doi.org/10.1109/ICSE.2012.6227164

[13] ——, “Who will stay in the FLOSS community? Modeling participant’s initial behavior,” *IEEE Trans. Software Eng.*, vol. 41, no. 1, pp. 82–99, 2015. [Online]. Available: https://doi.org/10.1109/TSE.2014.2349496

[14] R. K. Raj and F. Kazemian, “Using open source software in computer science courses,” in *Proceedings. Frontiers in Education. 36th Annual Conference*. IEEE, 2006, pp. 21–26.

[15] J. Buchta, M. Petrenko, D. Poshyvanyk, and V. Rajlich, “Teaching evolution of open-source projects in software engineering courses,” in *22nd IEEE International Conference on Software Maintenance (ICSM 2006), 24-27 September 2006, Philadelphia, Pennsylvania, USA*. IEEE Computer Society, 2006, pp. 136–144. [Online]. Available: https://doi.org/10.1109/ICSM.2006.66

[16] D. M. C. Nascimento, K. Cox, T. Almeida, W. Sampaio, R. A. Bittencourt, R. R. G. Souza, and C. von Flach G. Chavez, “Using open source projects in software engineering education: A systematic mapping study,” in *IEEE Frontiers in Education Conference, FIE 2013, Oklahoma City, Oklahoma, USA, October 23-26, 2013*, R. L. Shehab, J. J. Sluss, and D. A. Trytten, Eds. IEEE Computer Society, 2013, pp. 1837–1843. [Online]. Available: https://doi.org/10.1109/FIE.2013.6685155

[17] T. M. Smith, R. McCartney, S. S. Gokhale, and L. C. Kaczmarczyk, “Selecting open source software projects to teach software engineering,” in *The 45th ACM Technical Symposium on Computer Science Education, SIGCSE 2014, Atlanta, GA, USA, March 5-8, 2014*, J. D. Dougherty, K. Nagel, A. Decker, and K. Eiselt, Eds. ACM, 2014, pp. 397–402. [Online]. Available: https://doi.org/10.1145/2538862.2538932

[18] Z. Hu, Y. Song, and E. F. Gehringer, “Open-source software in class: students’ common mistakes,” in *Proceedings of the 40th International Conference on Software Engineering: Software Engineering Education and Training, ICSE (SEET) 2018, Gothenburg, Sweden, May 27 - June 03, 2018*, P. Lago and M. Young, Eds. ACM, 2018, pp. 40–48. [Online]. Available: https://doi.org/10.1145/3183377.3183394

[19] S. H. Tan, C. Hu, Z. Li, X. Zhang, and Y. Zhou, “GitHub-OSS Fixit: Fixing bugs at scale in a software engineering course,” in *43rd IEEE/ACM International Conference on Software Engineering: Software Engineering Education and Training, ICSE (SEET) 2021, Madrid, Spain, May 25-28, 2021*. IEEE, 2021, pp. 1–10. [Online]. Available: https://doi.org/10.1109/ICSE-SEET52601.2021.00009

[20] B. Morgan and C. Jensen, “Lessons learned from teaching open source software development,” in *Open Source Software: Mobile Open Source Technologies - 10th IFIP WG 2.13 International Conference on Open Source Systems, OSS 2014, San José, Costa Rica, May 6-9, 2014. Proceedings*, ser. IFIP Advances in Information and Communication Technology, L. Corral, A. Sillitti, G. Succi, J. Vlasenko, and A. I. Wasserman, Eds., vol. 427. Springer, 2014, pp. 133–142. [Online]. Available: https://doi.org/10.1007/978-3-642-55128-4_18

[21] G. Pinto, C. Ferreira, C. Souza, I. Steinmacher, and P. Meirelles, “Training software engineers using open-source software: the students’ perspective,” in *Proceedings of the 41st International Conference on Software Engineering: Software Engineering Education and Training, ICSE (SEET) 2019, Montreal, QC, Canada, May 25-31, 2019*, S. Beecham and D. E. Damian, Eds. IEEE / ACM, 2019, pp. 147–157. [Online]. Available: https://doi.org/10.1109/ICSE-SEET.2019.00024

[22] D. Kahneman, *Thinking, Fast and Slow*. Macmillan, 2011.

[23] G. von Krogh, S. Haefliger, S. Spaeth, and M. W. Wallin, “Carrots and rainbows: Motivation and social practice in open source software development,” *MIS Q.*, vol. 36, no. 2, pp. 649–676, 2012.

[24] Y. Ye and K. Kishida, “Toward an understanding of the motivation of open source software developers,” in *Proceedings of the 25th International Conference on Software Engineering, May 3-10, 2003, Portland, Oregon, USA*, 2003, pp. 419–429. [Online]. Available: https://doi.org/10.1109/ICSE.2003.1201220

[25] A. Hars and S. Ou, “Working for free? motivations of participating in open source software projects,” *HICSS’04*, pp. 25–31, 2004.

[26] M. A. Gerosa, I. Wiese, B. Trinkenreich, G. Link, G. Robles, C. Treude, I. Steinmacher, and A. Sarma, “The shifting sands of motivation: Revisiting what drives contributors in open source,” in *43rd IEEE/ACM International Conference on Software Engineering, ICSE 2021, Madrid, Spain, 22-30 May 2021*. IEEE, 2021, pp. 1046–1058. [Online]. Available: https://doi.org/10.1109/ICSE43902.2021.00098

[27] C. J. Mendez, H. S. Padala, Z. Steine-Hanson, C. Hilderbrand, A. Horvath, C. Hill, L. Simpson, N. Patil, A. Sarma, and M. M. Burnett, “Open source barriers to entry, revisited: A sociotechnical perspective,” in *Proceedings of the 40th International Conference on Software Engineering, ICSE 2018, Gothenburg, Sweden, May 27 - June 03, 2018*, M. Chaudron, I. Crnkovic, M. Chechik, and M. Harman, Eds. ACM, 2018, pp. 1004–1015. [Online]. Available: https://doi.org/10.1145/3180155.3180241

[28] I. Steinmacher, M. A. Gerosa, T. U. Conte, and D. F. Redmiles, “Overcoming social barriers when contributing to open source software projects,” *Comput. Support. Cooperative Work.*, vol. 28, no. 1-2, pp. 247–290, 2019. [Online]. Available: https://doi.org/10.1007/s10606-018-9335-z

[29] J. W. D. Alderliesten and A. Zaidman, “An initial exploration of the ”good first issue” label for newcomer developers,” in *14th IEEE/ACM International Workshop on Cooperative and Human Aspects of Software Engineering, CHASE@ICSE 2021, Madrid, Spain, May 20-21, 2021*. IEEE, 2021, pp. 117–118. [Online]. Available: https://doi.org/10.1109/CHASE52884.2021.00023

[30] H. Horiguchi, I. Omori, and M. Ohira, “Onboarding to open source projects with good first issues: A preliminary analysis,” in *28th IEEE International Conference on Software Analysis, Evolution and Reengineering, SANER 2021, Honolulu, HI, USA, March 9-12, 2021*. IEEE, 2021, pp. 501–505. [Online]. Available: https://doi.org/10.1109/SANER50967.2021.00054

[31] I. Steinmacher, T. U. Conte, and M. A. Gerosa, “Understanding and supporting the choice of an appropriate task to start with in open source software communities,” in *48th Hawaii International Conference on System Sciences, HICSS 2015, Kauai, Hawaii, USA, January 5-8, 2015*, T. X. Bui and R. H. S. Jr., Eds. IEEE Computer Society, 2015, pp. 5299–5308. [Online]. Available: https://doi.org/10.1109/HICSS.2015.624

[32] C. Stanik, L. Montgomery, D. Martens, D. Fucci, and W. Maalej, “A simple NLP-based approach to support onboarding and retention in open source communities,” in *2018 IEEE International Conference on Software Maintenance and Evolution, ICSME 2018, Madrid, Spain, September 23-29, 2018*. IEEE Computer Society, 2018, pp. 172–182. [Online]. Available: https://doi.org/10.1109/ICSME.2018.00027

[33] Y. Huang, J. Wang, S. Wang, Z. Liu, D. Wang, and Q. Wang, “Characterizing and predicting good first issues,” in *ESEM ’21: ACM / IEEE International Symposium on Empirical Software Engineering and Measurement, Bari, Italy, October 11-15, 2021*, F. Lanubile, M. Kalinowski, and M. T. Baldassarre, Eds. ACM, 2021, pp. 13:1–13:12. [Online]. Available: https://doi.org/10.1145/3475716.3475789

[34] F. Fagerholm, A. S. Guinea, J. Münch, and J. Borenstein, “The role of mentoring and project characteristics for onboarding in open source software projects,” in *2014 ACM-IEEE International Symposium on Empirical Software Engineering and Measurement, ESEM ’14, Torino, Italy, September 18-19, 2014*, M. Morisio, T. Dybå, and M. Torchiano, Eds. ACM, 2014, pp. 55:1–55:10. [Online]. Available: https://doi.org/10.1145/2652524.2652540

[35] S. Panichella, “Supporting newcomers in software development projects,” in *2015 IEEE International Conference on Software Maintenance and Evolution, ICSME 2015, Bremen, Germany, September 29 - October 1, 2015*, R. Koschke, J. Krinke, and M. P. Robillard, Eds. IEEE Computer Society, 2015, pp. 586–589. [Online]. Available: https://doi.org/10.1109/ICSM.2015.7332519

[36] J. D. O. Silva, I. S. Wiese, I. Steinmacher, and M. A. Gerosa, “Students’ engagement in open source projects: An analysis of google summer of code,” in *Proceedings of the 31st Brazilian Symposium on Software Engineering, SBES 2017, Fortaleza, CE, Brazil, September 20-22, 2017*, J. C. Maldonado, F. C. Ferrari, U. Kulesza, and T. U. Conte, Eds. ACM, 2017, pp. 224–233. [Online]. Available: https://doi.org/10.1145/3131151.3131156

[37] J. D. O. Silva, I. S. Wiese, D. M. Germán, I. F. Steinmacher, and M. A. Gerosa, “How long and how much: What to expect from summer of code participants?” in *2017 IEEE International Conference on Software Maintenance and Evolution, ICSME 2017, Shanghai, China, September 17-22, 2017*. IEEE Computer Society, 2017, pp. 69–79. [Online]. Available: https://doi.org/10.1109/ICSME.2017.81

[38] G. Canfora, M. D. Penta, R. Oliveto, and S. Panichella, “Who is going to mentor newcomers in open source projects?” in *20th ACM*

SIGSOFT Symposium on the Foundations of Software Engineering (FSE-20), SIGSOFT/FSE'12, Cary, NC, USA - November 11 - 16, 2012, W. Tracz, M. P. Robillard, and T. Bultan, Eds. ACM, 2012, p. 44. [Online]. Available: https://doi.org/10.1145/2393596.2393647

[39] I. Steinmacher, I. S. Wiese, and M. A. Gerosa, "Recommending mentors to software project newcomers," in *Proceedings of the Third International Workshop on Recommendation Systems for Software Engineering, RSSE 2012, Zurich, Switzerland, June 4, 2012*, W. Maalej, M. P. Robillard, R. J. Walker, and T. Zimmermann, Eds. IEEE, 2012, pp. 63–67. [Online]. Available: https://doi.org/10.1109/RSSE.2012.6233413

[40] H. J. C. Ellis, R. A. Morelli, T. R. de Lanerolle, and G. W. Hislop, "Holistic software engineering education based on a humanitarian open source project," in *20th Conference on Software Engineering Education and Training (CSEE&T 2007), 3-5 July 2007, Dublin, Ireland*. IEEE Computer Society, 2007, pp. 327–335. [Online]. Available: https://doi.org/10.1109/CSEET.2007.26

[41] D. Carrington and S.-K. Kim, "Teaching software design with open source software," in *33rd Annual Frontiers in Education, 2003. FIE 2003.*, vol. 3. IEEE, 2003, pp. S1C–9.

[42] E. E. Allen, R. Cartwright, and C. Reis, "Production programming in the classroom," in *Proceedings of the 34th SIGCSE Technical Symposium on Computer Science Education, SIGCSE 2003, Reno, Nevada, USA, February 19-23, 2003*, S. Grissom, D. Knox, D. T. Joyce, and W. P. Dann, Eds. ACM, 2003, pp. 89–93. [Online]. Available: https://doi.org/10.1145/611892.611940

[43] E. F. Gehringer, "From the manager's perspective: Classroom contributions to open-source projects," in *2011 Frontiers in Education Conference, FIE 2011, Rapid City, SD, USA, October 12-15, 2011*. IEEE Computer Society, 2011, p. 1. [Online]. Available: https://doi.org/10.1109/FIE.2011.6143028

[44] D. M. C. Nascimento, R. A. Bittencourt, and C. Chavez, "Open source projects in software engineering education: A mapping study," *Comput. Sci. Educ.*, vol. 25, no. 1, pp. 67–114, 2015. [Online]. Available: https://doi.org/10.1080/08993408.2015.1033159

[45] (2022) Test PyPI. [Online]. Available: https://test.pypi.org/

[46] M. Zhou, X. Hu, and W. Xiong, "openEuler: Advancing a hardware and software application ecosystem," *IEEE Software*, vol. 39, no. 2, pp. 101–105, 2022.

[47] (2022) Kata Containers. [Online]. Available: https://katacontainers.io/

[48] (2022) PaddlePaddle. [Online]. Available: https://www.paddlepaddle.org.cn/

[49] (2022) TiDB. [Online]. Available: https://github.com/pingcap/tidb

[50] V. Braun and V. Clarke, "Thematic analysis." 2012.

[51] D. S. Cruzes and T. Dybå, "Recommended steps for thematic synthesis in software engineering," in *Proceedings of the 5th International Symposium on Empirical Software Engineering and Measurement, ESEM 2011, Banff, AB, Canada, September 22-23, 2011*. IEEE Computer Society, 2011, pp. 275–284. [Online]. Available: https://doi.org/10.1109/ESEM.2011.36

[52] R. B. Zajonc, "Attitudinal effects of mere exposure." *Journal of Personality and Social Psychology*, vol. 9, no. 2p2, p. 1, 1968.

[53] R. M. Montoya, R. S. Horton, J. L. Vevea, M. Citkowicz, and E. A. Lauber, "A re-examination of the mere exposure effect: The influence of repeated exposure on recognition, familiarity, and liking." *Psychological Bulletin*, vol. 143, no. 5, p. 459, 2017.

[54] T. Gilovich, D. Griffin, D. Kahneman *et al.*, *Heuristics and Biases: The Psychology of Intuitive Judgment*. Cambridge university press, 2002.

[55] (2022) pandas. [Online]. Available: https://pandas.pydata.org/

[56] E. L. Deci and R. M. Ryan, "The" what" and" why" of goal pursuits: Human needs and the self-determination of behavior," *Psychological Tnquiry*, vol. 11, no. 4, pp. 227–268, 2000.

[57] ——, "Self-determination theory: A macrotheory of human motivation, development, and health." *Canadian Psychology/Psychologie Canadienne*, vol. 49, no. 3, p. 182, 2008.