Chapter 7
Onboarding and Retaining of Contributors in FLOSS Ecosystem

Minghui Zhou

Abstract There is a saying that the type of developers that an ecosystem *wants* do not have trouble getting involved. They are good at finding tasks and issuing pull requests. The type of developers that needs hand-holding—you do not want them joining your project/ecosystem due to their lack of skill. This might be true for a popular project like the Linux kernel which never worried attracting new developers. The (difficult) process of working around to get (a patch) in for a contributor is a process of getting the right people for the community. However, many other projects/ecosystem, e.g., GNOME, do not have many people who desperately want to work for them. And they have many to-do tasks. Projects even as popular as the Linux kernel are often in the need of resources. Moreover, the tasks in an ecosystem are quite different, what if the community just wants people who are able to review English documents? We may be able to train them well with a good design. In summary, there might be something we could do to help people with willingness (and no right skills yet) to get to the right track needed by ecosystems.

7.1 Onboarding

7.1.1 Background

The start of participation in a FLOSS ecosystem is fraught with difficulties [23, 31], as the new contributors may not be familiar with project's practices and norms and the existing participants have to rely on the scant information in a bug report or a comment made by the newcomer to judge the competence and reliability of the

M. Zhou (✉)
Key Laboratory of High Confidence Software Technologies, Ministry of Education, Peking University, Beijing 100871, China
e-mail: zhmh@pku.edu.cn

© Springer Nature Singapore Pte Ltd. 2019

B. Fitzgerald et al. (eds.), *Towards Engineering Free/Libre Open Source Software (FLOSS) Ecosystems for Impact and Sustainability*,
https://doi.org/10.1007/978-981-13-7099-1_7

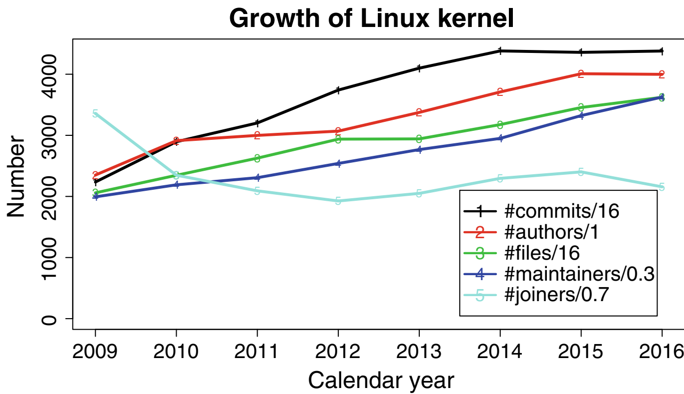

Fig. 7.1 Evolution of the Linux kernel over time

new contributor. Figure 7.1 shows the evolution of the Linux kernel over time.[1] In particular, the number of joiners decreases in recent years while the amount of work represented by files, commits, and authors which the community needs to take care of keeps growing. Getting a newbie on board in an ecosystem may be much more complicated than in a traditional project, because an ecosystem often has different projects and these projects often have interdependencies and require more learning. Moreover, the complexity of an ecosystem grows over time substantially, e.g., the Linux kernel has grown from 10.2 thousand lines of code in 1991 (version 0.01) to 22.3 million lines of code in 2016 (version 4.9), and from several authors to more than 2000 authors [27]. However, the nature of learning for individuals is the same, what differs may lie on the scale and content of learning.

The research questions which are critical to onboarding include the following:

1. How do newcomers learn? It involves what they need to learn and how to learn. For example, except the programming skills, they need to learn a methodology they did not invent and they need to learn how to communicate with the community. It also involves intermediaries (e.g., tools) that help to transfer knowledge and facilitate learning. How to learn? For example, learn by doing, or learn from experts or artifacts.
2. How do existing participants learn? The existing participants in the community are often busy with various tasks. Even if they want to spend effort on nurturing newcomers, they may not know what is needed for the newcomers due to the knowledge gap between them and newcomers—though they may naturally educate newcomers in the process of resolving problems (while newcomers learn by doing).

[1]The calculation is based on the data retrieved from the mainline repository of Linux kernel maintained by Linus Torvalds: http://www.git://git.kernel.org/pub/scm/linux/kernel/git/torvalds/linux.git.

3. What would be a good way to structure a community/ecosystem to get newcomers onboarding? For example, an often adopted policy is to have a division of modules and tasks. So newbies could focus their effort on easier modules or tasks—an often mentioned barrier faced by newcomers is that they do not know where to start.

These questions sketch an ecosystem-learning-focused agenda that needs to address the outlined challenges. Several important aspects that require extensive attention are discussed below.

7.1.2 Communication

The new developer population needs to learn the norms that enable them into the ecosystem. However, the culture of a long-lived ecosystem is difficult to understand. How to communicate with the community to acquire skills and knowledge that are needed to proceed in the ecosystem is a critical challenge in onboarding.

Newbies may acquire the skills and knowledge embodied in the community by directly interacting with master members (by reading their code and by asking them questions) [14]. The communication between newcomers and experts of a community is a two-way communication. On the one hand, newbies need to spend effort to learn basic norms and practices of the community by themselves (before asking questions); otherwise, the experts in the community may "get grumpy" because many simple mistakes are made over and over again, as described by one Linux kernel maintainer: "...you can answer a lot of questions like this for yourself very easily, simply by reading the source code."[2] As discovered by Steinmacher et al. [18], not needed pull requests are among the most common cause for code nonacceptance in FLOSS projects—because the newbies often submit "superseded/duplicated pull-requests".

On the other hand, experts may not be able to understand the confusion of newbies and may not communicate with them in an efficient way. There is a well-understood construct of the "zone of proximal development" [24], which describes the case where experts are not usually effective at training or teaching novices. The gap is too wide, the assumptions of what is known, too great. For example, Steinmacher et al. [17] found that one of the social barriers for people to contribute is "receiving answers with too advanced/complex contents." Therefore, people who are closer in experience level to newbies may be more effective at helping newbies learn practices. Due to the variation of new participants by nature, experts also need to learn how to communicate with different kinds of newbies.

In other words, for newcomers, it is important to understand how to do their homework and how to communicate with experts if necessary; and for experts, it is important to understand the (technical and social) needs of newcomers and make their nurturing effort worthwhile. Therefore, better communicating practices and mechanisms could be designed or adopted to help onboarding.

[2] https://github.com/gregkh/presentation-linuxmaintainer/blob/master/maintainer.pdf.

Moreover, experts inside the community may not have time to handle newbies. There is a need for intermediaries whose main purpose is to communicate with others. An idea of Social Mechanism of Interaction introduced by Schmidt et al., emphasizes the role of product itself in supporting the articulation of the distributed activities of multiple actors [15]. Not only codebase [6], but also bug report forms [3] are means by which the articulation work of the project, and therefore the communication can be carried out. Sometimes, the artifacts like MR/change repositories might be the only possible mechanism for developers to communicate [33], for example, in the offshoring or trans-generation scenarios (like a long-lived FLOSS system), since there may be no traditional opportunities to communicate in many situations, and a new generation of developers may be unable to communicate with original creators who have retired or died a long time ago.

To improve communication through these social mechanisms (and many others that are not discussed here), more investigations are needed, involving how and why these mechanisms work or do not work, particularly for newcomers, and what could be improved.

7.1.3 Division of Tasks and Modularization

Segregation of tasks at the architecture level is valuable to a community for a variety of reasons. The tasks that are core could be reserved for experts; it also enables a dispersion of less risky code tasks toward newer contributors which may help to facilitate the onboarding process. In particular, for ecosystems that have evolved for decades, the scale and complexity of the software system is way too complicated for a newbie to master, let alone to revise the code. However, if the tasks can be well divided, the newbies may be able to start from the easy tasks, e.g., ones that have no or few dependencies on the other parts of the system, and get on board quickly.

Though the tasks suitable for newbies have been rarely addressed explicitly, the division of labor and distribution of tasks is a common theme in the FLOSS literature. Researchers, e.g., Ducheneaut [7], characterized a community as a series of concentric circles; each circle is occupied by people playing a particular role in the development process. The core team accomplishes central tasks and oversees the community [1, 13, 27]. Peripheral roles, e.g., triagers, are found to be good at filtering invalid issues and as accurate as developers in filling in missing issue attributes [26]. These peripheral roles may suit newbies who are not familiar with code yet, as suggested by many FLOSS communities.

Modularization is adopted in software projects for the convenience of separating tasks. In particular, for a long-lived ecosystem, it is extremely difficult for any newcomer to join the development. A well-modularized architecture might help with that. For example, the key to the success of the Linux is its modularity according to its creator [20]. Inside the system, the combination of modules has a structured hierarchy of dependence relations, but modules entering at the same level of the system can be developed independently from each other [1]. Therefore, different modules

could evolve according to its own nature and some parts that require minimal interaction with other developers may fit newbies. In the Linux kernel, after more than two decades of evolution, the core modules like mm (memory management) appear to have become mature and very few newbies could participate in the development [27]. The peripheral modules like drivers keep growing to satisfy various needs of hardware manufacturers. "In order to support many independent devices and therefore many independent authors, it is important to make the subsystem extensible, so each hardware device driver is implemented as a separate (sub-)module that supports a common interface."[3] As a result, the tasks of driver development are often considered to represent lower entry barrier for newcomers.[4] However, modularization is often aspirational, and different projects and organizations are in different points along this continuum. This could be examined from existing software repositories and build processes.

In practice, a variety of FLOSS projects/ecosystems post the possible tasks they perceive that would be suitable for newbies to work on in the project page. For example, the Linux community has KernelNewbies which "is all about sharing knowledge and experience" for newbies.[5] Mozilla has a website called Bugs Ahoy that allows people to search through all of Mozilla's bug reports to find the ones that are most relevant to their areas of interest, for example, newbies could choose to display only "simple bugs".[6] Further investigation on how the roles are separated and how the tasks are distributed among the roles in large-scale FLOSS ecosystems are needed. It could certainly help onboarding in addition to many other benefits, for example, it helps to understand the governance of a community.

7.1.4 Learning of Experts

What is known about experts is important not because all learners are expected to become experts, but because the knowledge of expertise provides valuable insights into what the results of effective learning look like [2]. Understanding how experts learn and how they develop knowledge structure may provide ways to help newbies.

First, we need to understand the project/ecosystem practice trajectories that experts take. The issues include how a developer starts from a novice (a newcomer) and becomes an expert (a core team member), how she grows her expertise, and what kind of expertise she has to master (and in what order) to become central [29]. Some studies have been conducted about how the developers grow their strength in terms of task difficulty and task centrality [28], but much broader and deeper investigation is needed, for example, of what leads to that trajectory. Further, an ecosystem requires

[3] linux.org/threads/the-linux-kernel-the-source-code.4204/.

[4] https://www.linux.com/news/software/linux-kernel/804403-three-ways-for-beginners-to-contribute-to-the-linux-kernel/eudyptula-challenge.org.

[5] https://kernelnewbies.org/.

[6] https://www.joshmatthews.net/bugsahoy/?simple=1.

different kinds of participants who have different skills, it is important to separate their skills and trajectories (if it is possible) so people would know which trajectory to take based on their own preferences and skills.

Second, knowing how learners develop coherent structures of information has been particularly useful to understand the nature of organized knowledge that underlies effective comprehension and thinking. For example, the difference between seniors and novices, might lie in the ability to combine and apply what is learned to perform more complex activities creatively and in new situations [28]. Psychologists tried to aid software engineering through programmer selection testing since the 1950s. For example, McKeithan et al. [12] observed that experts are able to remember language commands based on their position in the structure of the language. Novices, not having an adequate mental representation of the language structure, often use mnemonic tricks to remember command names. Curtis [4] considered the performance of someone tackling a complicated programming task to be related to the richness of their knowledge about the problem area. However, the initial attempt had failed poorly, not because the principles and technologies of psychology were not up to the task, but because the psychologists failed to adequately model the mental and behavioral aspects of programming before selecting tests to measure it [4]. Learning theory can now account for how learners acquire skills to search a problem space and then use these general strategies in many problem-solving situations.

Overall, a better understanding of the programmer knowledge base, and why and how the programmer learn could help prepare newbies more efficiently. Different communities/organizations may have different cultures that suit how people get involved, empirical studies on existing ecosystems could benefit us in this regard.

7.2 Retaining

7.2.1 Background

It is critical for ecosystems to retain participants (who have become familiar with ecosystem practices and norms and have worked and established rapport with other participants), because people with multiyear participation in a project (or ecosystem at a higher level) tends to accomplish more and more important tasks, to provide greater value to the community than others, and are critical to the long-term viability of the community [16, 28, 30, 32]. While it is challenging to attract people, it is even more challenging to retain them. For example, Shah [16] found that a need for software-related improvements drives the initial participation, but only a small subset hobbyists remain involved. We found that only 3.6% of Gnome and 0.9% of Mozilla joiners would stay with the ecosystem for at least 3 years [32]. Figure 7.2 shows that the conversion of new joiners to long-term contributors who would stay with the

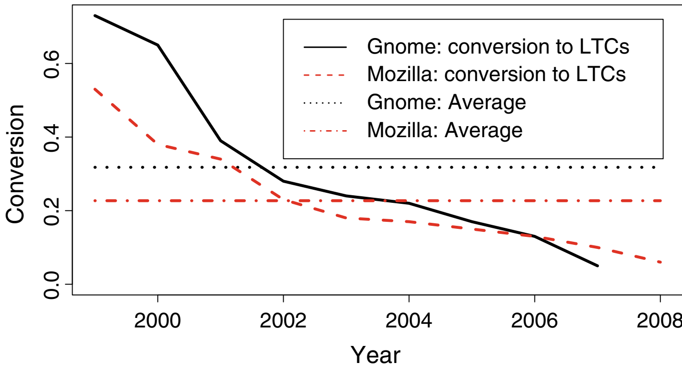

Fig. 7.2 Conversion of new joiners to long-term contributors (LTCs)

ecosystem for at least 3 years has been decreasing in Mozilla and GNOME.[7] It raises challenges that ecosystems must take seriously in order to survive and sustain.

The research questions which are critical to retaining include the following:

1. Why do people leave or stay?
2. What kind of people/expertise is needed (to stay long) by an ecosystem?
3. What could be designed in a community to retain contributors?

These questions motivate the following two important aspects that relate to retaining participants in an ecosystem.

7.2.2 Spectrum of Contributors

An ecosystem prospers with diverse contributions from diverse contributors. Ducheneaut [7] presented a pattern with core developers in the center, surrounded by the maintainers, often responsible for one or more subcomponents (modules) of a project. Around these are patchers (who fix bugs), bug reporters, documenters, and, finally, the users of the software.

In the spectrum of contributors, it is important for an ecosystem to know what kind of people it needs to retain, or what kind of people are needed to stay long for the survivability and sustainability of the ecosystem. This requires an understanding about the distribution of expertise needed in an ecosystem which is much more complicated than that of a single project due to the complicated dependencies between projects. For example, Wang et al. [25] proposed a novel view about the types of contributors needed in software development. They view software development as a combination of activities that require creation and activities that follow the routine

[7]The calculation is based on the data retrieved in [32].

manufacturing processes. Different activities call for different types of developers who need to be inspired and retained by different strategies.

Developer's expertise could be considered from various aspects. For example, the difficulty and centrality of tasks represent expertise or competence in a project/ecosystem, people who could accomplish central tasks are extremely valuable to sustain long-lived projects [28]. For example, Vasilescu et al. [21] showed that tenure diversity improves a team's productivity and turnover rate, which suggests that all levels of tenure are essential and what is critical might be how to keep a balance.

The spectrum of contributors required by an ecosystem could be explained through the media of people making contributions. The contributing media include mailing list, issue tracking system, version control system, question & answer websites, etc. These channels nurture different expertises required by an ecosystem in different ways. For example, the majority of the tasks of a senior QA in the issue tracking system of Mozilla is "going through the NEW/UNCONFIRMED pile of bugs contributed from outside sources (i.e., non-Netscape-paid employees)".[8] The responsibility of a maintainer in the Linux kernel is to "review patches from submitters (and then accept or reject it), handle questions from both developers and users about things related to the subsystem (usually bug reports)".[9] Both experienced QA and maintainers are critical to the sustainability of ecosystems but may require different skill sets, and therefore different methods to train and retain.

On the other hand, people come to join an ecosystem with different motivations and only a small fraction of them have the possibility to stay long. Some people would be simply one-time contributors, because they never attempt to stay no matter how attractive the ecosystem is. For example, some users run into problems when using Firefox, they may come to report the bugs (which are also important contributions for the software) and never come back. Some people may stay for long simply because that is their job. For example, in the Linux kernel some maintainers work for years maintaining drivers from companies such as Intel. Therefore, people who could be retained may occupy a small proportion of contributors. In order to understand how to retain them, this group of people needs to be located and carefully investigated. For example, the nature of the initial behavior of this group (e.g., the tasks they start may represent the motivation they have) and why they leave or stay.

7.2.3 Forces of Retaining

In order to sustain a community, it is important to understand what factors/mechanisms might be at play to achieve that goal. The most influential factor to affect participation might be the motivation of a developer. In particular, FLOSS developers are likely to be motivated and involved in the project for fundamentally

[8]http://weblogs.mozillazine.org/stephend/.

[9]http://www.kroah.com/log/linux/what_greg_does.html.

different reasons. For example, Lakhani et al. [11] suggested that enjoyment-based intrinsic motivation is the strongest and most pervasive driver, with user need, intellectual stimulation derived from writing code, and improving programming skills being the top motivators for project participation (which may or may not suit sustaining). Nakakoji et al. [14] found that the willingness to get involved determines the role played by a FLOSS member in the community. We found that joiners who are more willing to contribute more than double their odds of becoming a long-term contributor [31].

The relationship between individuals and their environment might affect retention and have been extensively studied in the organizational literature. For example, the extent to which an individual's values are consistent with those revealed in his or her organization/environment was found to yield significant effects on a variety of attitudinal outcomes like job satisfaction and organizational commitment, and behavioral outcomes like job performance and turnover [8, 10, 22]. Similarly, in FLOSS projects, identity-based and bond-based commitments are found important for contributor retention [9]. If developers shared the beliefs and norms of the community, they engaged more in the effort related to the community [5, 19]. An ecosystem is combined with different cultures, the Linux Foundation, for example, does not have "a way" that all projects are compelled to follow, which makes retention even more challenging.

The macro-environment of an ecosystem, such as relatively sociality [30], user base (of the product), commercial support [27], and the popularity of the technology, has a substantial impact on the sustaining of contributors (and even the sustainability of the ecosystem itself). It is important to understand to what extent these factors play their roles and what is left for the community to tailor to retain valuable contributors.

Overall, the retention (or sustainability) of FLOSS participants is determined by a variety of factors, ranging from individual motivation to interaction between individuals and their environment. Further investigation may lie in the deeper understanding and quantification of the impact of various factors in large-scale ecosystems, and therefore helping to build mechanisms that could help retain participants.

Acknowledgements This work is supported by the National key research and development program Grant 2018YFB10044200, and the National Natural Science Foundation of China Grants 61432001 and 61825201.

References

1. C.R. Andrea Bonaccorsi, Why open source software can succeed. Res. Policy **32**, 1243–1258 (2003)
2. J. Bransford, A. Brown, R. Cocking, *How People Learn: Brain, Mind, Experience and School* (National Academy Press, Washington, 2003)
3. P. Carstensen, The bug report form (1994), http://cscw.dk/schmidt/papers/comic_d3.2.pdf
4. B. Curtis, Fifteen years of psychology in software engineering: individual differences & cognitive science, in *ICSE'84* (1984), pp. 97–106

5. S. Daniel, L. Maruping, M. Cataldo, J. Herbsleb, When cultures clash: participation in open source communities and its implications for organizational commitment, in *ICIS 2011 Proceedings* (7 Dec 2011), page Paper 7
6. C. de Souza, J. Froehlich, P. Dourish, Seeking the source: software source code as a social and technical artifact, in *GROUP '05: Proceedings of the 2005 International ACM SIGGROUP Conference on Supporting Group Work* (ACM, New York, USA, 2005), pp. 197–206
7. N. Ducheneaut, Socialization in an open source software community: a socio-technical analysis. Comput. Support. Coop. Work (CSCW) **14**(4), 323–368 (2005)
8. B.J. Hoffman, D.J. Woehr, A quantitative review of the relationship between person-organization fit and behavioral outcomes. J. Vocat. Behav. **68**(3), 389–399 (2006)
9. R.E. Kraut, P. Resnick, *Building Successful Online Communities: Evidence-Based Social Design* (MIT Press, Cambridge, 2012)
10. A.L. KRISTOF-BROWN, R.D. ZIMMERMAN, E.C. JOHNSON, Consequences of individuals' fit at work: a meta-analysis of person-job, person-organization, person-group, and person-supervisor fit. Pers. Psychol. **58**(2), 281–342 (2005)
11. K. Lakhani, R. Wolf, *Why Hackers Do What They Do: Understanding Motivation and Effort in Free/Open Source Software Projects* (MIT Press, Cambridge, 2005)
12. K. McKeithen, J. Reitman, H. Rueter, S. Hirtle, Knowledge organization and skill differences in computer programmers. Cogn. Psychol. **13**, 307–325 (1981)
13. A. Mockus, R.F. Fielding, J. Herbsleb, A case study of open source development: the Apache server, in *22nd International Conference on Software Engineering* (Limerick, Ireland, 4–11 June 2000), pp. 263–272
14. K. Nakakoji, Y. Yamamoto, Y. Nishinaka, K. Kishida, Y. Ye, Evolution patterns of open-source software systems and communities, in *IWPSE '02: Proceedings of the International Workshop on Principles of Software Evolution* (Orlando, FL, 19–20 May 2002), pp. 76–85
15. K. Schmidt, C. Simone, Coordination mechanisms: towards a conceptual foundation of CSCW systems design. J. Collab. Comput. **5**, 155–200 (1996)
16. S.K. Shah, Motivation, governance, and the viability of hybrid forms in open source software development. Manag. Sci. **52**(7), 1000–1014 (2006). July
17. I. Steinmacher, T. Conte, M.A. Gerosa, D. Redmiles, Social barriers faced by newcomers placing their first contribution in open source software projects, in *Proceedings of the 18th ACM Conference on Computer Supported Cooperative Work & Social Computing, CSCW '15* (ACM, New York, USA, 2015), pp. 1379–1392
18. I. Steinmacher, G. Pinto, I.S. Wiese, M.A. Gerosa, Almost there: a study on quasi-contributors in open source software projects, in *Proceedings of the 40th International Conference on Software Engineering, ICSE '18* (ACM, New York, USA, 2018), pp. 256–266
19. K.J. Stewart, S. Gosain, The impact of ideology on effectiveness in open source software development teams. MIS Q. **30**(2), 291–314 (2006)
20. L. Torvalds, The linux edge. Commun. ACM **42**(4), 38–39 (1999). Apr
21. B. Vasilescu, D. Posnett, B. Ray, M.G. van den Brand, A. Serebrenik, P. Devanbu, V. Filkov, Gender and tenure diversity in github teams, in *Proceedings of the 33rd Annual ACM Conference on Human Factors in Computing Systems* (ACM, 2015), pp. 3789–3798
22. M.L. Verquer, T.A. Beehr, S.H. Wagner, A meta-analysis of relations between person-organization fit and work attitudes. J. Vocat. Behav. **63**(3), 473–489 (2003)
23. G. von Krogh, S. Spaeth, K.R. Lakhani, Community, joining, and specialization in open source software innovation: a case study. Res. Policy **32**(7), 1217–1241 (2003). July
24. L. Vygotsky, Interaction between learning and development. Read. Dev. Child. **23**(3), 34–41 (1978)
25. H. Wang, G. Yin, X. Li, X. Li, *TRUSTIE: A Software Development Platform for Crowdsourcing* (Springer, Berlin, 2015)
26. J. Xie, M. Zhou, A. Mockus, Impact of triage: a study of mozilla and gnome, in *ESEM 2013* (Baltimore, Maryland, USA, 10–11 Oct 2013), pp. 247–250
27. M. Zhou, Q. Chen, A. Mockus, F. Wu, On the scalability of linux kernel maintainers' work, in *Proceedings of the 2017 11th Joint Meeting on Foundations of Software Engineering, ESEC/FSE 2017* (ACM, New York, USA, 2017), pp. 27–37

28. M. Zhou, A. Mockus, Developer fluency: achieving true mastery in software projects, in *ACM SIGSOFT / FSE* (Santa Fe, New Mexico, 7–11 Nov 2010), pp. 137–146
29. M. Zhou, A. Mockus, Growth of newcomer competence: challenges of globalization, in *FSE/SDP Workshop on the Future of Software Engineering Research* (Santa Fe, New Mexico, 7–8 Nov 2010), pp. 442–447
30. M. Zhou, A. Mockus, Does the initial environment impact the future of developers?, in *ICSE 2011* (Honolulu, Hawaii, 21–28 May 2011), pp. 271–280
31. M. Zhou, A. Mockus, What make long term contributors: willingness and opportunity in OSS community, in *ICSE 2012* (Zürich, Switzerland, 2012), pp. 518–528
32. M. Zhou, A. Mockus, Who will stay in the floss community? modeling participant's initial behavior. IEEE Trans. Softw. Eng. **41**(1), 82–99 (2015). Jan
33. M. Zhou, A. Mockus, D. Weiss, Learning in offshored and legacy software projects: how product structure shapes organization, in *ICSE Workshop on Socio-Technical Congruence* (Vancouver, Canada, 19 May 2009)