

Barriers Faced by Newcomers to Software-Crowdsourcing Projects

Alexandre Lazaretti Zanatta, Pontifical Catholic University of Rio Grande do Sul

Igor Steinmacher, Federal University of Technology—Paraná and Northern Arizona University

Leticia Santos Machado, Pontifical Catholic University of Rio Grande do Sul

Cleudson R.B. de Souza, Instituto Tecnológico Vale and Federal University of Pará

Rafael Prikladnicki, Pontifical Catholic University of Rio Grande do Sul

// Newcomers to software crowdsourcing can be hindered by lack of documentation, poor task management, problems understanding code structure or architecture, information overload, poor platform usability, and the language barrier. Fortunately, ways exist to minimize these barriers. //



SOFTWARE CROWDSOURCING

uses the cloud to outsource part or all of a software project to an unknown workforce of developers.¹ The “rise of vibrant online communities”² is fundamental to software crowdsourcing’s growth. So, it’s important to motivate, engage, and retain newcomers to promote a sustainable community.

However, bringing newcomers onboard isn’t easy. Newcomers to software crowdsourcing face barriers similar to newcomers participating in open source software (OSS) development,³ such as technical, documentation, and cultural problems. (For a comparison of crowdsourced software development with OSS, see the sidebar.)

We conducted a study to identify barriers to newcomers contributing to software-crowdsourcing projects for paid work⁴ in a competitive model.⁵ The study used two types of data:

- data extracted from Topcoder, a well-known crowdsourcing platform, and
- feedback from newcomers after they contributed to a software-crowdsourcing project for the first time.

Our analysis revealed the characteristics of contributors to a software-crowdsourcing platform, six barriers to newcomers, and guidelines for dealing with these barriers.

Crowdsourcing

Crowdsourcing is “a type of participative online activity in which an individual, an institution, a non-profit organization, or company proposes to a group of individuals of varying knowledge, heterogeneity, and number, via a flexible open call, the voluntary undertaking of a

SOFTWARE CROWDSOURCING AND OPEN SOURCE SOFTWARE DEVELOPMENT

Both software crowdsourcing and open source software (OSS) development rely on a crowd of developers in diverse locations, working on different yet related tasks in a common environment. In this sense, newcomers to software crowdsourcing are subject to barriers similar to those that newcomers to OSS development face.¹ These barriers include technical problems, poor or no documentation, community receptivity issues, unclearly described tasks, and cultural differences.

However, software crowdsourcing differs from OSS development in significant ways. For example, crowdsourcing developers are motivated mainly by financial incentives, the intellectual property is transferred, and developers are treated as contestants rather than collaborators.² In contrast, OSS developers are usually motivated by ideology, self-development, self-need, and altruism³ and maintain authorship of their work.⁴

In addition, software crowdsourcing introduces its own unique barriers. For example, Thomas LaToza and André van der Hoek reported that “in communities that engage in com-

petitions, established experts sometimes win most of the competitions, discouraging novices and effectively shutting them out of such competitions.”²

References

1. I. Steinmacher et al., “Social Barriers Faced by Newcomers Placing Their First Contribution in Open Source Software Projects,” *Proc. 18th ACM Conf. Computer Supported Cooperative Work & Social Computing*, 2015, pp. 1379–1392.
2. T.D. LaToza and A. van der Hoek, “Crowdsourcing in Software Engineering: Models, Motivations, and Challenges,” *IEEE Software*, vol. 33, no. 1, 2016, pp. 74–80.
3. S. Oreg and O. Nov, “Exploring Motivations for Contributing to Open Source Initiatives: The Roles of Contribution Context and Personal Values,” *Computers in Human Behavior*, vol. 24, no. 5, 2008, pp. 2055–2073.
4. E. Schenk and C. Guittard, “Towards a Characterization of Crowdsourcing Practices,” *J. Innovation Economics & Management*, no. 7, 2011, pp. 93–107.

task.”⁶ Crowdsourcing can reduce costs, provide on-demand access to specialized resources, and enable rapid mobilization of large numbers of people to accomplish tasks globally. In addition, it can encourage creativity and problem solving, which are applicable to software development.^{5,7}

Any crowdsourcing model has these basic components:

- The requester is the client or sponsor that requests tasks.
- The crowd is the participating community dispersed globally.
- The platform is the technological framework that intermediates communication and collaboration between the requester and the crowd.

In addition, Mahmood Hosseini

and his colleagues mention a fourth element: the task.⁸ The task might be a simple microtask or human-intelligence task, or it might be complex. A microtask is repetitive, self-contained, and can be completed in a few minutes;⁵ examples include audio or video transcription and image classification. Complex tasks often require cognitive effort and specialized skills; examples include software development tasks and innovation projects.

Thomas LaToza and André van der Hoek described three models for software-crowdsourcing scenarios.⁹ In *peer production*, control is decentralized, and participants contribute without receiving a monetary reward. In *competitions*, clients request tasks and pay for their completion. *Micro-tasking* usually occurs during testing.

Some crowdsourcing platforms

specifically support software crowdsourcing. Topcoder, CrowdPlat, GetACoder, and Upwork support multiple development phases. Other platforms target specific phases—for example, CrowdREquire (requirements), Bountify (coding), 99Tests (testing), and DesignCrowd (design).

Stack Overflow, Amazon Mechanical Turk, and Freelancer platforms aren’t specifically for software crowdsourcing but can support it.

Topcoder

Topcoder aims to gather experts in design, development, data science, and competitive programming to work on challenging problems. We investigated Topcoder because it is one of the largest and most successful software-crowdsourcing platforms,^{10,11} has one of the largest communities,¹¹ and has been used

to create software for many organizations, including AOL, Best Buy, and Facebook.^{10,12}

Applications developed on Topcoder follow a process involving phases such as requirements exploration (application specification), component architecture, component design, component development, deployment, and testing. Each phase is posted as a contest. Topcoder members can submit solutions to any contest. The winning solution for one phase becomes the input to the next phase.

During requirements exploration, enterprises' requirements are gathered and specified. During the component architecture phase, each application is split into components. Each component goes through component design and component development. Review and testing activities are leveraged in each component to identify any final fixes required for the solution. Then, the components are combined to create the application that's delivered to the customers. After these stages, and on the basis of quality scores, the first-place participant (and sometimes the second-place participant) gets paid.

Before newcomers compete, their ratings are provisional (entrance ratings). After a competition, Topcoder rates each competitor on the basis of his or her efficiency compared to the other competitors.

The Quantitative Study

Our quantitative analysis was exploratory; that is, it aimed to generate new insights and give us a better understanding of the problem. Moreover, it helped us understand part of the qualitative results, which we discuss later.

We employed Topcoder's search functionality to collect information

on its members from 23 May to 12 June 2016. We used these criteria:

- Members who participated in algorithm competitions. In these competitions, the participants must solve the same problems under the same time constraints. The algorithm competition is the most important Topcoder contest because the platform uses it to engage and retain members.¹²
- Members who had a maximum rating of 1,199 (that is, they were relative newcomers). Each member belongs to a tier indicating their efficiency: gray (0 to 899), green (900 to 1,199), blue (1,200 to 1,499), yellow (1,500 to 2,199), or red (2,200 or higher).
- Members whose most recent competition was fewer than 180 days (six months) ago.

We identified 3,403 members; we implemented a crawler (through Topcoder's public API) to get their nickname, country, number of wins, and entry month and year. We discarded members for whom information was missing; the final sample was 3,100. To obtain a sample size with a margin of error of ± 5 percent and confidence level of 95 percent, we selected 342 members for our analysis.

The data involved 54 countries. Two hundred five members were from Asia (primarily India, Japan, China, Bangladesh, Korea, and Mongolia), 69 members were from North America, and 68 members were from other continents. The top four countries were India, Japan, the US, and China—the world's largest economies.

Ninety-seven percent of these members had never won a competition. Those who had won competitions

weren't newcomers; they had at least nine years' experience with Topcoder and were in the gray or green tier. (Two members from India had won 66 times, and one from Bosnia and Herzegovina had won 134 times.) One hundred thirteen members joined Topcoder in 2015 or 2016 and had never won a competition but had participated for at least 180 days.

The Qualitative Study

We invited 20 newcomers to participate in the qualitative study. They were 10 students attending a Passo Fundo University software engineering course and 10 developers from industry with at least two years' experience. We interviewed eight respondents—three students and five professional developers—for a response rate of 40 percent.

The participants first registered in Topcoder and learned its structure and basic functions. Then, they could participate in any development challenge they regarded as not too complex—that is, one whose technical requirements were in accordance with their skills and abilities. They needed to complete the competition over a 12-day period.

After the competition, the participants answered the following question, inspired by Igor Steinmacher and his colleagues' research:³ "On the basis of this experience, what main barriers do newcomers face when they want to start contributing to Topcoder? (Please consider technical and nontechnical issues.)" We transcribed the interviews and analyzed the data using open-coding procedures.¹³ We chose a qualitative approach because this human behavior occurs in a social environment whose context is important. Furthermore, participants provided answers

that were free-form and thus adequate for qualitative analysis.

Our analysis revealed the following barriers (ordered from most to least frequent). The first barrier was a lack of documentation or diagrams. Most participants told of inadequate procedures and problems related to task documentation. One participant said, “The link to GitHub doesn’t help either, and it’s not clear to me how this works.”

this task can be difficult and time-consuming. The code must have a logical structure and be well-written, so that newcomers and maintainers can determine which routines (or modules) perform what function. One interviewer mentioned that “it’s necessary to structure programming rules to keep them understandable—each function (or module) must have comments, variable names should be meaningful, and so on.”

However, another participant commented that “the platform is nice,” and another said that “the platform is intuitive, and navigation speed is very good.” Most of them thought that Topcoder was easy to use. According to one participant, “The layout strategy used for menus and filters helped us look for competitions.” Overall, the participants’ perception of platform usability was positive, except for the platform’s effectiveness, which is a key aspect of its usability.

Most participants told of inadequate procedures and problems related to task documentation.

Another interviewee mentioned, “I can’t find proper documentation to do these tasks.”

The second barrier was poor task management. Some participants had trouble finding a task suited to their skills. Some also had trouble understanding how much effort and time it would take to develop a solution and finish the task. Many of the difficulties were due to poor time management. Such difficulties couldn’t be resolved in a simple manner because, in this kind of software-crowdsourcing project, time is money. When discussing the topic “time to accomplish the task,” the participants usually referred to matters of personal time management. One participant commented, “After this, in the Competitions menu, I’ll look for tasks according to my skills and my available time for the challenge.”

The third barrier was problems understanding code structure or architecture. Usually, developers navigate source code to understand its structure; because source code is typically large and complex,

The fourth barrier was information overload; that is, the amount of information exceeded participants’ capacity to manage it. One participant said, “I’m feeling a little stressed about all the information that’s necessary to make a decision.” Another participant commented, “I have a lot of information [available] on the Topcoder platform, and I don’t know what to do.”

The fifth barrier was poor platform usability. The ISO 9241-11 standard defines usability as “the extent to which a product can be used by specified users to achieve specified goals with effectiveness, efficiency, and satisfaction in a specified context of use.”¹⁴ Generally, the participants commented that completing a task was difficult and required much effort, and the platform needed to better motivate users to meet task deadlines. They also mentioned that they would have liked to discuss tasks with other users—for example, through a forum. One participant commented, “It’s very difficult to find the information about tasks.”

The sixth barrier was the language barrier. Topcoder is just in English and doesn’t support automatic translation to other languages. One participant commented, “I’d like to see a multilanguage system in Topcoder because I can’t speak and read English very well.” However, participants realized the importance of being able to communicate in English.

We observed an overlap between some of these barriers and traditional problems in distributed or collocated software development. For instance, lack of documentation about requirements or source code is a constant problem in software development. Problems with understanding code are also common in any development model.

However, we believe that when these problems occur in software crowdsourcing, they’re amplified. Poor or no documentation can’t be alleviated through direct formal or informal communication with the customers because such communication usually isn’t possible in crowdsourcing. Understanding the source code is also problematic because the developers can’t access the whole project and aren’t aware of the architecture or decisions that can influence their tasks. In Topcoder, one of

its employees mediates communication with customers.

Overcoming the Barriers

On the basis of our two studies, we offer the following guidelines to help Topcoder newcomers overcome the barriers we just described.

First, software-crowdsourcing platforms should provide clear, complete, and consistent documentation about tasks and their associated projects. When this documentation isn't available, platforms should provide mechanisms developers can use to ask questions. Both synchronous approaches (such as chat and phone conferences) and asynchronous approaches (such as email and forums) are relevant in these scenarios. Requesters and platforms must pay attention to the time spent on documentation practices and additional artifacts (such as UML diagrams and tutorial videos) during task posting. Too much attention to such details will likely decrease the time available to respond to newcomers' queries.

Second, a better match between developers' expertise and the task will increase the platform's usability. This can help motivate developers and allow them to better estimate the time required to finish tasks. So, platforms should include ways for developers, especially newcomers, to identify tasks' complexity. One strategy, borrowed from OSS projects, is to tag the tasks that are appropriate for newcomers.

Third, newcomers should concentrate on the tasks from which they'll derive the most benefit. LaToza and van der Hoek said that "to learn in today's crowdsourcing systems, software developers must first join, and acculturate themselves to, software projects."⁹ Task information is the key to software

crowdsourcing, but information overload can hinder newcomers' ability to manage that information.

Fourth, similarly to what Steinhilber and his colleagues proposed for OSS projects,³ platforms could provide newcomer-specific pages that give newcomers every resource they need, but only the resources they need. It's important to not flood newcomers with every possible resource, which could lead to information overload.

Fifth, platforms should provide mechanisms that help newcomers understand the source code structure. Depending on the project's size, simple approaches such as class diagrams and other UML diagrams generated from the source code might be appropriate, or more complex source code visualization tools might be necessary. This might well involve intellectual-property issues.

Finally, platforms should implement computer-assisted translation.

For our study participants who didn't speak English, participating in Topcoder projects was very difficult. Computer-assisted translation would greatly help such users.

Limitations and Further Research


As with any other empirical study, ours had limitations, and some topics require additional research. For example, our study covered only one crowdsourcing platform. Although that platform is the most popular, we can't say that the behavior we identified will occur on other platforms. Moreover, we didn't consider other Topcoder competitions, such as those for specification, architecture, design, development, assembly, or test suites.

In addition, we interviewed a small number of participants whom

we had invited to participate. Self-selection—agreeing to participate in the interviews—might have provided a bias leading to underrepresentation or overrepresentation of groups in the sample.

So, we plan to apply other research methods, including analyzing Topcoder's log analysis and experiments to help us better understand the barriers newcomers face. We also intend to work with different types of challenges and platforms and a larger number of newcomers to continue to identify patterns, behaviors, and ways to improve how newcomers contribute to software-crowdsourcing projects.

Although we didn't evaluate the recommended approaches to verify how well they mitigate the barriers, we think they could significantly benefit crowdsourcing platforms and customers.

Newcomers are key actors in crowdsourcing and can provide competitive advantages for the requesters. However, they need help overcoming the barriers to their contribution. Crowdsourcing platforms play an important role in developing strategies and mechanisms to support newcomers. 

Acknowledgments

Alexandre Zanatta was supported by a University of Passo Fundo grant. Cleidson de Souza is grateful for funding from CNPq (process nos. 440880/2013-0, 310468/2014-0, and 420801/2016-2).

References

1. W.-T. Tsai, W. Wu, and M.N. Huhns, "Cloud-Based Software Crowdsourcing," *IEEE Internet Computing*, vol. 18, no. 3, 2014, pp. 78–83.



ALEXANDRE LAZARETTI ZANATTA is a PhD candidate in the area of crowdsourcing for software development, at the Pontifical Catholic University of Rio Grande do Sul. He also teaches software engineering at the University of Passo Fundo. His research focuses on software development with alternative workforces and on agile development. Zanatta received an MSc in computer science from the Federal University of Santa Catarina. Contact him at alexandre.zanatta@acad.pucrs.br.



CLEIDSON R.B. DE SOUZA is a researcher at Instituto Tecnológico Vale and an associate professor at the Federal University of Pará. His research interests are at the intersection of software engineering and computer-supported cooperative work—how teams of software engineers work together to develop software systems. De Souza received a PhD in information and computer sciences from the University of California, Irvine. Contact him at cleidson.desouza@acm.org.



IGOR STEINMACHER is an associate professor in the Department of Computing at the Federal University of Technology—Paraná and a postdoctoral scholar in the School of Informatics, Computing, and Cyber Systems at Northern Arizona University, where he researches human aspects of software engineering and related topics. Steinmacher received a PhD in computer science from the University of São Paulo. Contact him at igorfs@utfpr.edu.br.



RAFAEL PRIKLADNICKI is an associate professor at the Computer Science School and the director of the Science and Technology Park (Tecnopuc) at the Pontifical Catholic University of Rio Grande do Sul (PUCRS), where he also leads the MuNDDoS research group. His research focuses on distributed and agile software development. Prikladnicki received a PhD in computer science from PUCRS. He's on *IEEE Software's* editorial board and chairs the magazine's advisory board. Contact him at rafaelp@pucrs.br.



LETICIA SANTOS MACHADO is a PhD candidate at the Computer Science School of the Pontifical Catholic University of Rio Grande do Sul (PUCRS). She also teaches information systems at Universidade do Vale do Rio dos Sinos. Her research focuses on understanding the roles of collaboration in software development. Machado received an MSc in computer science from PUCRS. Contact her at leticia.machado.001@acad.pucrs.br.



2. J. Howe, *Crowdsourcing: Why the Power of the Crowd Is Driving the Future of Business*, Crown Business, 2008.
3. I. Steinmacher et al., "Social Barriers Faced by Newcomers Placing Their First Contribution in Open Source Software Projects," *Proc. 18th ACM Conf. Computer Supported Cooperative Work & Social Computing*, 2015, pp. 1379–1392.
4. T.R.G. Green, S.J. Payne, and G.C. Van Der Veer, *Psychology of Computer Use*, Academic Press, 1983.
5. K.-J. Stol and B. Fitzgerald, "Two's Company, Three's a Crowd: A Case Study of Crowdsourcing Software Development," *Proc. 36th Int'l Conf. Software Eng. (ICSE 14)*, 2014, pp. 187–198.
6. E. Estellés-Arolas and F. González-Ladrón-De-Guevara, "Towards an Integrated Crowdsourcing Definition," *J. Information Science*, vol. 38, no. 2, 2012, pp. 189–200.
7. A. Kittur, "The Future of Crowd Work," *Proc. 2013 Conf. Computer Supported Cooperative Work (CSCW 13)*, 2013, pp. 1301–1318.
8. M. Hosseini et al., "The Four Pillars of Crowdsourcing: A Reference Model," *Proc. IEEE 8th Int'l Conf. Research Challenges in Information Science (RCIS 14)*, 2014, pp. 1–12.
9. T.D. LaToza and A. van der Hoek, "Crowdsourcing in Software Engineering: Models, Motivations, and Challenges," *IEEE Software*, vol. 33, no. 1, 2016, pp. 74–80.
10. K. Li et al., "Analysis of the Key Factors for Software Quality in Crowdsourcing Development: An Empirical Study on TopCoder.com," *Proc. IEEE 37th Ann. Computer Software and Applications Conf.*, 2013, pp. 812–817.
11. P.J. Ågerfalk, B. Fitzgerald, and K.-J. Stol, *Software Sourcing in the Age of Open: Leveraging the Unknown Workforce*, Springer, 2015.
12. D. Lakhani, D. Garvin, and E. Lonsstein, *TopCoder (A): Developing Software through Crowdsourcing*, case 9-610-032, Harvard Business School, 2010.
13. A. Strauss and J. Corbin, *Basics of Qualitative Research: Techniques and Procedures for Developing Grounded Theory*, 2nd ed., Sage Publications, 1998.
14. *Ergonomic Requirements for Office Work with Visual Display Terminals (VDTs)—Part 11: Guidance on Usability*, ISO 9241-11:1998, Int'l Org. for Standardization, 1998.

ACM - IEEE CS ECKERT-MAUCHLY AWARD

Call for Award Nominations • Deadline: 30 March 2017 • www.computer.org/awards • awards.acm.org

ACM and the IEEE Computer Society co-sponsor the **Eckert-Mauchly Award**, which was initiated in 1979. The award is known as the **computer architecture community's most prestigious award**.

The award recognizes outstanding contributions to computer and digital systems architecture. **It comes with a certificate and a \$5,000 prize.**

The award was named for John Presper Eckert and John William Mauchly, who collaborated on the design and construction of the Electronic Numerical Integrator and Computer (ENIAC), the first large-scale electronic computing machine, which was completed in 1947.

TO BE PRESENTED AT



ISCA 2017

The 44rd ACM/IEEE International Symposium on Computer Architecture
isca17.ece.utoronto.ca/doku.php

Nomination Guidelines:

- Open to all. Anyone may nominate. • Self-nominations are not accepted. • This award requires 3 endorsements.

Questions? Write to IEEE Computer Society Awards Administrator at awards@computer.org or the ACM Awards Committee Liaison at acm-awards@acm.org.

