Who Cares About My Feature Request?

Lukas Heppler, Remo Eckert, Matthias Stuermer

Who cares about my feature request?

Lukas Heppler[1], Remo Eckert[2], Matthias Stuermer[3]

[1] University of Bern, Switzerland
lukas.heppler@students.unibe.ch
[2] University of Bern, Switzerland
remo.eckert@iwi.unibe.ch
[3] University of Bern, Switzerland
matthias.stuermer@iwi.unibe.ch

Abstract. Previous studies on issue tracking systems for open source software (OSS) focused mainly on requests for bug fixes. However, requests to add a new feature or an improvement to an OSS project are often also made in an issue tracking system. These inquiries are particularly important because they determine the further development of the software. This study examines if there is any difference between requests of the IBM developer community and other sources in terms of the likelihood of successful implementation. Our study consists of a case study of the issue tracking system BugZilla in the Eclipse integrated development environment (IDE). Our hypothesis, which was that feature requests from outsiders have less chances of being implemented, than feature requests from IBM developers, was confirmed.

Keywords. Open Source Software; Issue Tracking System; Feature Request; Eclipse; Bugzilla

1 Introduction

Collaboration of core developers with the outside community is an important key to the longevity of an Open Source Software (OSS) project [1]. For users, it is particularly important to have their requirements integrated into future versions of the software product. Issue tracking systems are the main instrument used to integrate the needs of external participants. In an OSS development project, the community can report a bug or feature using issue tracking systems such as Bugzilla [2]. Issue tracking systems have received considerable attention in the OSS literature [3–5]. Due to the open nature of these systems and the ease of data collection, they are an ideal subject for examination when investigating the OSS development process. Issue tracking systems also have various advantages for the community: more problems with the software can be identified, because they are easy to report, and more bugs can be fixed, because there are more developers contributing to solutions. This not only helps to improve the product, but also to tailor a software to the users' needs [6, 7]. Moreover, issue tracking systems are a way to integrate more externals into the OSS community. In the context of the Eclipse IDE, where IBM revealed the source code of its software under an OSS license, their goal was to increase its

popularity as a development platform on a larger market, while retaining control over the future path of the software development. The present study examines the area of conflict between the contributors from IBM and the outsiders - people not paid by IBM. Do feature requests from outsiders have less chance of being implemented than those which originate from an IBM employee?

To answer this question, this paper is structured as follows: Chapter 2 describes theories on community integration as well as an analysis of previous studies of issue tracking systems. Moreover, the Eclipse IDE is presented in detail. Chapter 3 describes the method applied to analyse our hypothesis, Chapter 4 presents the results, which are then discussed in Chapter 5.

2 Theory section

Since Netscape released their source code for the Internet browser Mozilla, an increasing number of business companies have revealed their prior proprietary source code under an OSS license. For the most part, the intention behind such a move is not altruistic, but is based instead on the hope that the community will help to improve and maintain the future code base, in order to reduce internal development efforts [1]. The involvement of a community in an OSS project is a vital factor in the success of the project because the community promotes the project and its development [8, 9]. Moreover, as stated by Grammel et al [7], integrating the community in the OSS development plays a key role for the success of an OSS project. Ways to involve people in an OSS project include the marketing of the OSS project to attract potential contributors and integrating their efforts into the project. As the project grows, governance structures become necessary [1, 10]. On the technical side, increasing modularity in the source code is one incentive to attract new developers. A modular software makes the software more attractive to outsiders, since the effort required to get to know the code is lower [11]. Another way to integrate a community in an OSS project is to use an issue tracking system; this will be discussed in more detail in the next subsection.

2.1 Issue tracking systems

Although there is no strict hierarchical structure in an OSS community, the structure is not completely flat. According to [12], roles and their associated influence can be earned through contributions to the community. The resulting community structure, called the “onion-model” can be shown in a layer where the roles closer to the center (e.g. the project leader and core members) have a greater influence than the roles in the external area (e.g. readers and passive users). Contributions to the community can be made through an issue tracking system, thereby influencing the community structure and the impact each individual has on the OSS project. OSS projects typically have an open issue tracker where developers and users of the software can report bugs and feature requests [3]. Previous studies of issue tracking systems covered topics such as the automatic assignment of bug reports to developers [3, 13, 14], the automatic assignment of priority and severity labels to bug reports [15], the identification

of duplicate reports [5, 16], the automatic summarization of reports [17, 18] and the prediction of bug fixing times [19, 20].

Bug fixing times, respectively the speed at which bug reports are processed, are influenced by several factors. Bugs which are critical for the proper functioning of the software (i.e. have a high severity) receive more attention and resources and are therefore fixed faster than more trivial bugs [21]. However, the bug fixing time is also influenced by the characteristics of the person filing the bug report. The popularity of the reporting person within the community reduces the bug fixing time. A bug filed by a reporter whose bugs are usually fixed quickly, has a high chance of any future bugs also being fixed quickly [21].
A study of the issue tracking system BugZilla of the Mozilla Firefox project revealed that bug reports from 'outsiders' tend to be ignored by the developer community. These were processed far more slowly than bugs reported by core developers. Furthermore, the study showed that reports from 'outsiders' tend to be ignored only in the more recent versions of Mozilla Firefox, and not in the earlier stages of development [22].

2.2 Feature requests in issue tracking system

Issue tracking systems are not used only to report bugs, but also serve to request new features or enhancements to the software [3, 23, 24]. Previous research excluded reports containing feature requests or enhancements, despite the fact that these are very interesting for research on OSS communities since they determine the further development of the product, under the influence not only of developers, but also based on the opinions of its users. Other than the famous first lesson by Eric S. Raymond "Every good work of software starts by scratching a developer's personal itch." [25]. Feature requests also allow non-programmers to express their demand for further development of the OSS product. Using issue tracking systems to highlight new functional needs thus allows their ideas and requirements to be integrated.

In this paper we identify measures that influence whether or not a feature request is successful. A key variable indicates whether a feature request was reported by a core developer or by an outsider. Based on the findings of Dalle, den Besten, and Masmoudi [22], we expect that requests from outsiders are less likely to be implemented than feature requests from core developers (IBM developers). Other independent variables included in our analysis will be introduced later in this study.

2.3 Eclipse IDE

The Eclipse IDE project is a longstanding and well-established OSS project, with a wide installation base in both the OSS and in the commercial development field: It presents a large and mature OSS project [20]. While the focus has been on projects such as Linux, Apache or Gnome, the Eclipse IDE was not founded as a "grassroots" community of user-developers [1]. The project was initially owned by IBM and was released as OSS in 2001. By releasing the source code under an OSS license, IBM made a source code available with a value estimated at $40 million [26]. While this seemed a somewhat surprising decision at the time, this step increased its popularity as a

development platform across a larger market, attracting more attention to IBM's complementary products. From 2001 until 2004, the control over the development strategy remained in the hands of IBM [27]. In 2004, IBM ceded control over the project and the Eclipse Foundation was established, which now owns the intellectual property rights (IPR). With this decision, IBM allowed other firms to become equal members in the project. Today, the Eclipse Foundation serves as a "steward" of the Eclipse community. In general, the Eclipse Foundation provides four services to their community: IT-Infrastructure management, IPR-management, development processes and the ecosystem development.

Eclipse was the subject of many studies in the OSS literature [3, 6, 19, 20, 14]. Source code contribution and mailing list activity have already been investigated in terms of participation by IBM developers vs. outsiders. The results indicated that IBM developers initially dominated mailing list and source code contributions, but the participation of outsiders increased over time [27].

3 Method

We obtained a database dump of Eclipse's issue tracking system BugZilla from the website of the MSR Mining Challenge 2011. The dataset included 316'911 reports from 30'230 different reporters for the period from 2001 to 2010. We then used the command line tool Bicho to crawl additional data for the period from 2010 to December 2015 from the Eclipse issue tracking system.

The reports were filtered according to their value for the attribute "severity". Only reports with the severity attribute set to "enhancement" were included in the analysis. "Enhancement" is the label used to tag reports that contain feature requests or enhancements in BugZilla. Our subsample consisted of 24'856 reports.

3.1 Logistic Regression Model

To investigate which factors led to a successful feature request, we included only resolved feature requests and excluded all open feature requests. Furthermore, we investigated only requests marked as "FIXED" or "WONTFIX", because it was only with respect to these issues that a decision was made on whether or not to implement the requested feature. "FIXED" feature requests were considered to be successful and "WONTFIX" were considered unsuccessful. We excluded feature requests with any other resolution such as duplicate reports, invalid requests or requests not related to Eclipse. Therefore, the categories "INVALID", "WORKSFORME", "DUPLICATE" and "NOT ECLIPSE" were omitted from our analysis.

To distinguish IBM developers from outsiders, we used a similar approach as Spaeth, Stuermer, and von Krogh [27]. Reports from users whose e-mail addresses contained @ibm or @oti were classified as IBM developers; all other users were classified as outsiders (early members communicated with @oti e-mail addresses because the initial version of Eclipse was developed by OTI). The original data did not include any e-mail addresses, which made it necessary to crawl the corresponding e-mail address for every user ID from BugZilla via

their API interface and merge them into the dataset. Eleven user accounts did not contain any e-mail address, leading to the exclusion of 134 reports. The remaining 11'479 feature requests were included in the analysis.

The following were also included as additional independent variables: the date of the request and the assigned priority (P1 up to P5) of the request; the number of times the request was reassigned to another developer; and the number of times the request was reopened to control for their effect. To measure the attention the feature request received, we counted the number of votes for the feature request and the number of people who had the feature request on their "watch list". Both attributes were visible for anyone working on the request. As a measure of the volume of discussion the feature request generated, we included the number of comments written and the number of separate authors writing comments. To investigate whether an extensively written description had any impact on the success of the feature request, we measured the length of the description in 100s of characters.

In order to include the type of software, we included variables for the three major "products" within the Eclipse SDK: the Java Development Toolkit (JDT), the Plugin Development Environment (PDE) and the Eclipse development platform (Platform) itself. We also included another set of variables for the four most frequent subcomponents: the Core of the product (Core), the User Interface (UI), the Debugging component (Debug) and the Text Editor component (Text).

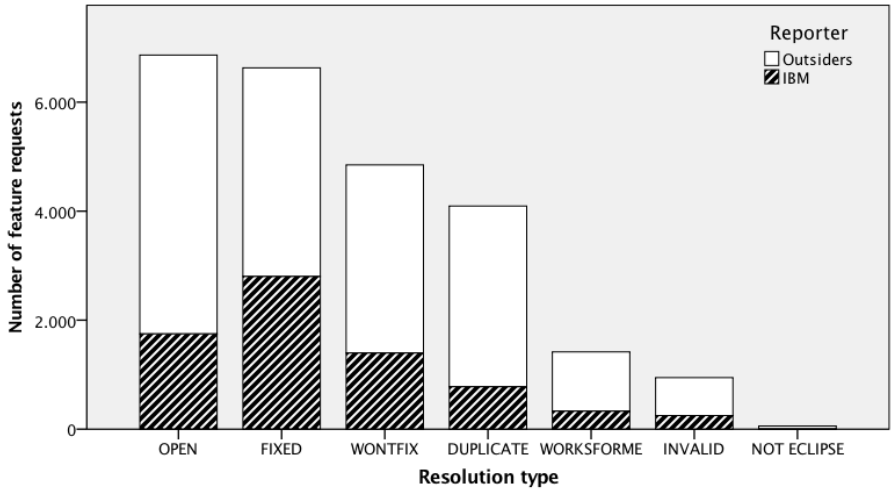

Fig. 1. Number of feature requests from IBM developers and outsiders by resolution type

4 Results

Our dataset consisted of 24'856 feature requests from the period between October 2001 and December 2015. A total of *n*=7'303 (29.4%) feature requests were submitted by IBM developers, *n*=17'553 (70.6%) were submitted by outsiders.

Table 1. Mean and SD (in parenthesis) for variables included in the Logistic Regression Model

	FIXED		WONTFIX	
	Outsiders	IBM	Outsiders	IBM
Priority	2.98 (.47)	3.02 (.50)	2.81 (.49)	2.77 (.55)
# Comments	9.83 (13.11)	7.57 (9.03)	4.33 (4.41)	4.29 (3.67)
# Authors	2.88 (3.30)	2.35 (2.12)	2.29 (1.45)	2.18 (1.34)
# Watching	.60 (2.42)	.48 (1.72)	.17 (.85)	.13 (.83)
Votes	.59 (3.60)	.19 (1.94)	.20 (.95)	.06 (.33)
# Reassignments	1.46 (1.24)	1.27 (1.21)	1.00 (1.19)	1.12 (1.35)
# Reopened	.23 (.51)	.21 (.52)	.15 (.42)	.24 (.48)
Characters in description (in 100 chars)	5.62 (12.70)	4.69 (7.63)	5.76 (11.84)	5.73 (14.09)

As Figure 1 shows, the relative share of feature requests submitted by outsiders varies strongly depending on the resolution type. Outsiders are responsible for over 70% of all invalid feature requests (INVALID, NOT ECLIPSE, WORKSFORME) and for over 80% of all duplicate feature requests. The proportion of all feature requests which are later implemented (FIXED) is more balanced. Around 42% of all implemented feature requests stem from IBM developers and 58% from outsiders.

Figure 2 shows that the participation of outsiders grew rapidly after the first year of development. Prior to the release of version 2.0 of Eclipse in 2002, the majority of all feature requests (n=1’374, 56.1%) were submitted by IBM developers. In the post 2.0 release phase, outsiders were responsible for the majority of 73% (n=16’477) of all feature requests. Figure 2 also shows that the total number of feature requests filed every year decreased in the later years of development. The number of feature requests from both IBM developers and outsiders decreased after the release of version 3.0 in 2004.

4.1 Logistic Regression

A logistic regression was performed to ascertain the effects on the likelihood that the feature request was successfully implemented of: being an IBM developer vs. outsider; the year the request was submitted; assigned priority; number of characters in the description; number of comments; number of separate authors of comments; number of people watching the request; number of reassignments and times the request was reopened; and the product and component to which the feature request referred. Table 1 shows the mean and standard deviation of variables included in the regression model. The logistic regression model was statistically significant, $\chi^2(17) = 3148.69$, $p < .001$. The model explained 32.2% (Nagelkerke R^2) of the variance in success rate and correctly classified 72.5% of all cases (76.6% of successful and 67% of unsuccessful requests).

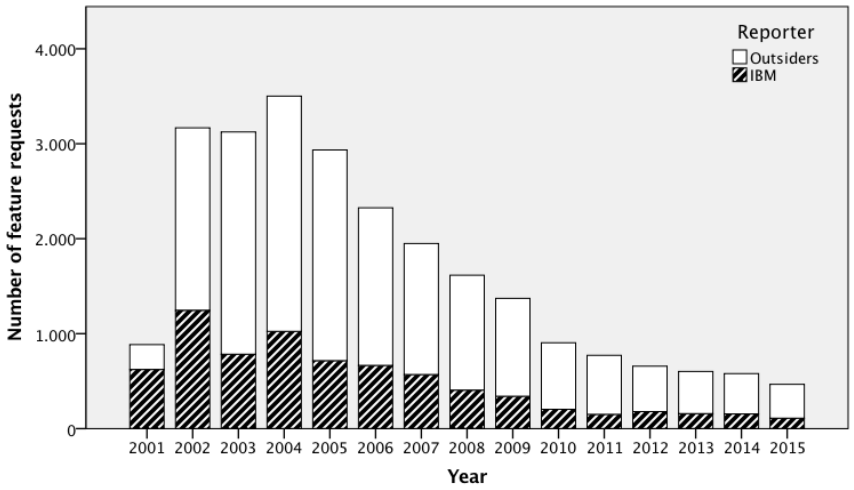

Fig. 2. Number of feature requests from IBM developers and outsiders from 2001 to 2015

Table 2. Summary for Logistic Regression Analysis for variables predicting success of a feature request (*n*=11'479)

Variable	B[a]	S.E.	e^{β}
IBM	.733***	.046	2.08
Year	.108***	.009	1.11
Priority	.742***	.049	2.10
# Comments	.210***	.008	1.23
# Authors	-.427***	.023	0.65
# Watchers	.054*	.023	1.06
Votes	.095***	.026	1.10
# Reassignments	.317***	.021	1.37
# Reopened	-.298***	.055	0.74
Characters in description	-.012***	.003	0.99
Products			
Plug-in Dev. Environment	-.133	.374	0.88
Java Development Tools	-1.614***	.363	0.20
Platform	-1.616***	.361	0.20
Components			
Core	-.235*	.104	0.79
Debug	-.260**	.080	0.77
Text	.845***	.102	2.33
UI	-.044	.060	0.96
Constant	2.993	.397	19.94

a. *p < .05. **p < .01. ***p < .001.

Table 2 indicates feature requests from IBM developers were two times more likely to be successfully implemented than requests from outsiders, $e^{\beta} = 2.08$, $p < .001$. Feature requests in the later years of development were more likely to be implemented than feature requests in the earlier years of development, $e^{\beta} = 1.11$, $p < .001$. Increasing priority level by one unit (out of five) doubled the chances of success, $e^{\beta} = 2.10$, $p < .001$. An increasing number of comments on a feature request was associated with an increased likelihood of the request being successful ($e^{\beta} = 1.23$, $p < .001$), but an increasing number of separate authors submitting comments on a request was associated with a reduction in the likelihood of success, $e^{\beta} = .65$, $p < .001$. The number of people watching the feature request slightly increased the likelihood of success, $e^{\beta} = 1.06$, $p < .05$. While the number of times the feature request was reassigned had a positive effect ($e^{\beta} = 1.37$, $p < .001$), the number of times the request was reopened ($e^{\beta} = .74$, $p < .001$) and the length of the description (in 100's of characters) had a minimal negative effect on the likelihood of success, $e^{\beta} = .99$, $p < .001$. The number of votes slightly increased the likelihood of success, $e^{\beta} = 1.10$, $p < .001$.

Feature requests concerning the Java development tools (JDT) or Platform yielded noticeably lower likelihoods of success ($e^{\beta} = .20$, $p < .001$) where there was no significant effect on requests concerning the PDE, $e^{\beta} = .88$, $p = .723$. Requests concerning the Core of the product ($e^{\beta} = .79$, $p < .05$) and the Debugging component ($e^{\beta} = .77$, $p < .01$) were less likely to be successful, whereas feature requests concerning the Text Editor had a significantly higher likelihood of being successful, $e^{\beta} = 2.33$, $p < .001$. There was no significant effect on the likelihood of success for feature requests concerning the User Interface (UI), all other variables being equal, $e^{\beta} = .96$, $p = .458$.

5 Conclusions

In this last section, we discuss the results and implications of this research paper. We also explain its limitations and future research issues.

5.1 Discussion of results

Our analysis indicates that feature requests from non-IBM employees are less likely to be implemented than feature requests from IBM developers. Thus, we confirm the finding that core developers tend to ignore reports from outsiders as indicated previously in the Mozilla Firefox project [22]. While ignorance might be a possible cause of the identified effect, there might also be other reasons. Feature requests formulated by IBM employees might be more qualified based on their in-depth knowledge and experience with the source code and available software features. Feature requests by IBM developers likely target highly relevant areas of improvement within the Eclipse software and, thus, might be implemented more often than feature requests by outsiders. This is somewhat contradictory considering core developers are essentially not required to file feature requests for certain requirements. Through their commit access to the code repository they have the opportunity to implement new functionalities directly into the source code, without going through the feature request process. We assume core developers often file feature reports voluntarily in order to comply with community rules and norms.

The results show that besides IBM affiliation, several other characteristics of feature requests also influence the probability of implementation. Our results suggest the highest positive impact on resolution is induced by the priority level of a feature request. Raising the priority level of a feature request by one doubles the probability that the feature will be implemented. The reason for this might be signaling effects indicating high relevance and demand, thus motivating developers to actually prioritize implementation of a certain feature. Based on this insight, ambitious users might now be tempted to categorize all of their feature requests as high priority. However, issue reporting users in Bugzilla cannot define the priority of their issue. This function is limited to the person to whom the issue is assigned, who can change the priority level, thus inhibiting opportunistic behavior of feature reporters.

Besides reporter origin and feature priority, the number of reassignments also had a strong positive effect on its implementation. Reassignments indicate an issue has been directed to persons with the best skills required to implement

the work, following the principle of knowledge specificity [28]. Interestingly, in this context the number of authors involved in a feature request had a negative impact, thus indicating that the mere fact that a large number of people are involved discussing a feature request did not help to implement it. Only when responsibility to resolve the issue changed were the chances of success raised.

In addition the number of comments and the number of votes an issue had received, as well as the number of people watching a feature request positively influenced the probability of its implementation. It is possibly the activity level surrounding a feature request and, thus, the level of interest in the resolution of a certain enhancement that increased the chances of its successful completion.

While we started with a research question concerning the factors influencing implementation probability of a feature request, we coincidentally found the interesting observation of decreasing request numbers over the years. While Eclipse is the leading Java development platform [29] it apparently receives less issues as others researching bug tracking have also found [30]. On the one hand, this might indicate a decreasing level of interest in Eclipse in the long-term. On the other hand, a more mature software solution justifiably receives less feature requests because it already fulfills most user requirements. This unexpected finding raises new questions about software maturity related to community activity and necessitates further research in other OSS projects to test whether their feature request pattern behaves the same, or if it is different - and if so, why.

5.2 Limitations and future research

The distinction between IBM developers and outsiders using their e-mail address is one limitation of this empirical research paper. Holders of an IBM e-mail address might not necessarily be core developers, and not all core developers in Eclipse are employed by IBM. In the Mozilla Firefox study [22] bug reports were differentiated according to their initial state when they enter the issue tracking system. Experienced developers with the “CanConfirm” privilege were able to enter their reports as “NEW”, all other reports were initially in the state “UNCONFIRMED”. Therefore, reports which entered the issue tracking system in the state “NEW” were considered to come from core developers, whereas reports which entered the issue tracking system in the state “UNCONFIRMED” were considered to come from outsiders. This approach was not applicable to the Eclipse project, because all reports are initially flagged as “NEW”, irrespective of the privileges of the reporter. Other methods of identifying core developers in the Eclipse project, such as measuring activity levels in terms of source code contribution, issue tracker and newsgroup activity, could be applied to distinguish between core developers and outsiders.

Future research could analyze if and how differences in the likelihood of success of feature requests from outsiders evolved over time. The effect that reports from outsiders tend to be ignored by the core developers in the Mozilla Firefox project could be shown only for the more recent versions of Firefox and not in the earlier stages of development. Eclipse evolved from being under the strict control of IBM to an independently governed OSS project. The effects of this evolution have already been shown in terms of increased source code contribution and newsgroup activity from outsiders in the Eclipse project [27]. It

would be of particular interest to study the effect of governance on the treatment of feature requests from outsiders. Further, a comparison between requirements from IBM developers and outsiders could be made to understand the differences of our quantitative study. Perhaps internal requests are more clear and feasible than those from outsiders and are therefore preferred by the Eclipse IDE developers.

Future research could obtain a dataset of further Eclipse projects or other OSS communities to expand the scope of the analysis. It would allow the reliability of the results of this study to be tested, including the accidently discovered effect of the decreasing number of feature requests. As the issue tracking system keeps record of historical information, it is possible to analyze in-depth the process between the reporting and solving of bugs and feature requests. This represents a reliable and promising source for further studies.

References

1. West, J., O'Mahony, S.: Contrasting community building in sponsored and community founded open source projects. In: System Sciences, 2005. HICSS'05. Proceedings of the 38th Annual Hawaii International Conference on System Sciences (2005).
2. Scacchi, W.: Understanding the requirements for developing open source software systems. IEE Softw. Proc. 149, 24–39 (2002).
3. Anvik, J., Hiew, L., Murphy, G.C.: Who Should Fix This Bug? In: Proceedings of the 28th International Conference on Software Engineering. pp. 361–370. ACM, New York, NY, USA (2006).
4. Mockus, A., Fielding, R.T., Herbsleb, J.D.: Two case studies of open source software development: Apache and Mozilla. ACM Trans. Softw. Eng. Methodol. TOSEM. 11, 309–346 (2002).
5. Runeson, P., Alexandersson, M., Nyholm, O.: Detection of duplicate defect reports using natural language processing. In: Software Engineering, 2007. ICSE 2007. 29th International Conference on. pp. 499–510. IEEE (2007).
6. Anvik, J., Hiew, L., Murphy, G.C.: Coping with an Open Bug Repository. In: Proceedings of the 2005 OOPSLA Workshop on Eclipse Technology eXchange. pp. 35–39. ACM, New York, NY, USA (2005).
7. Grammel, L., Schackmann, H., Schröter, A., Treude, C., Storey, M.-A.: Attracting the Community's Many Eyes: An Exploration of User Involvement in Issue Tracking. In: Human Aspects of Software Engineering. pp. 3:1–3:6. ACM, New York, NY, USA (2010).
8. Bagozzi, R.P., Dholakia, U.M.: Open Source Software User Communities: A Study of Participation in Linux User Groups. Manag. Sci. 52, 1099–1115 (2006).
9. Iivari, N.: Empowering the users? A critical textual analysis of the role of users in open source software development. AI Soc. 23, 511–528 (2009).
10. Dahlander, L., Magnusson, M.G.: Relationships between open source software companies and communities: Observations from Nordic firms. Res. Policy. 34, 481–493 (2005).

11. MacCormack, A., Rusnak, J., Baldwin, C.Y.: Exploring the Structure of Complex Software Designs: An Empirical Study of Open Source and Proprietary Code. Manag. Sci. 52, 1015–1030 (2006).
12. Nakakoji, K., Yamamoto, Y., Nishinaka, Y., Kishida, K., Ye, Y.: Evolution patterns of open-source software systems and communities. In: Proceedings of the international workshop on Principles of software evolution. pp. 76–85. ACM (2002).
13. Kagdi, H., Gethers, M., Poshyvanyk, D., Hammad, M.: Assigning change requests to software developers. J. Softw. Evol. Process. 24, 3–33 (2012).
14. Rahman, M.M., Ruhe, G., Zimmermann, T.: Optimized assignment of developers for fixing bugs an initial evaluation for eclipse projects. In: Proceedings of the 2009 3rd International Symposium on Empirical Software Engineering and Measurement. pp. 439–442. IEEE Computer Society (2009).
15. Xuan, J., Jiang, H., Ren, Z., Zou, W.: Developer prioritization in bug repositories. In: 2012 34th International Conference on Software Engineering (ICSE). pp. 25–35 (2012).
16. Wang, X., Zhang, L., Xie, T., Anvik, J., Sun, J.: An approach to detecting duplicate bug reports using natural language and execution information. In: Proceedings of the 30th international conference on Software engineering. pp. 461–470. ACM (2008).
17. Mani, S., Catherine, R., Sinha, V.S., Dubey, A.: Ausum: approach for unsupervised bug report summarization. In: Proceedings of the ACM SIGSOFT 20th International Symposium on the Foundations of Software Engineering. p. 11. ACM (2012).
18. Rastkar, S., Murphy, G.C., Murray, G.: Summarizing Software Artifacts: A Case Study of Bug Reports. In: Proceedings of the 32Nd ACM/IEEE International Conference on Software Engineering - Volume 1. pp. 505–514. ACM, New York, NY, USA (2010).
19. Giger, E., Pinzger, M., Gall, H.: Predicting the Fix Time of Bugs. In: Proceedings of the 2Nd International Workshop on Recommendation Systems for Software Engineering. pp. 52–56. ACM, New York, NY, USA (2010).
20. Panjer, L.D.: Predicting Eclipse Bug Lifetimes. In: Fourth International Workshop on Mining Software Repositories, 2007. ICSE Workshops MSR '07. pp. 29–29 (2007).
21. Marks, L., Zou, Y., Hassan, A.E.: Studying the Fix-time for Bugs in Large Open Source Projects. In: Proceedings of the 7th International Conference on Predictive Models in Software Engineering. pp. 11:1–11:8. ACM, New York, NY, USA (2011).
22. Dalle, J.-M., Besten, M. den, Masmoudi, H.: Channeling Firefox Developers: Mom and Dad Aren't Happy Yet. In: Russo, B., Damiani, E., Hissam, S., Lundell, B., and Succi, G. (eds.) Open Source Development, Communities and Quality. pp. 265–271. Springer US (2008).
23. Bissyande, T.F., Lo, D., Jiang, L., Reveillere, L., Klein, J., Le Traon, Y.: Got issues? Who cares about it? A large scale investigation of issue trackers from GitHub. In: 2013 IEEE 24th International Symposium on Software Reliability Engineering (ISSRE). pp. 188–197 (2013).

24. Koru, A.G., Tian, J.: Defect handling in medium and large open source projects. IEEE Softw. 21, 54–61 (2004).
25. Raymond, E.S.: The Cathedral and the Bazaar: Musings on Linux and Open Source by an Accidental Revolutionary. O'Reilly Media (1999).
26. Fitzgerald, B.: The transformation of open source software. Mis Q. 587–598 (2006).
27. Spaeth, S., Stuermer, M., Von Krogh, G.: Enabling knowledge creation through outsiders: towards a push model of open innovation. Int. J. Technol. Manag. 52, 411–431 (2010).
28. Sampler, J.L.: Redefining industry structure for the information age. Strateg. Manag. J. 19, 343–355 (1998).
29. Rebellabs: Developer Productivity Report 2012: Java Tools, Tech, Devs & Data. Zero Turnaround (2012).
30. Banerjee, S., Helmick, J., Syed, Z., Cukic, B.: Eclipse vs. Mozilla: A Comparison of Two Large-Scale Open Source Problem Report Repositories. In: High Assurance Systems Engineering (HASE), 2015 IEEE 16th International Symposium on. pp. 263–270. IEEE (2015).