A linguistic analysis on how contributors solve software problems in a distributed context

Héla Masmoudi, Imed Boughzala

A Linguistic Analysis on how Contributors solve Software Problems in a Distributed Context

Héla Masmoudi[1], Imed Boughzala[2]
1 Paris Dauphine at Paris, Place du Maréchal de Lattre de Tassigny 75116, Paris, France,
hela.masmoudi.09@campus.dauphine.fr

2 Telecom EM Research Center, Institut Mines-Télécom,
9 rue Charles Fourrier, 91011 Evry, France
Imed.boughzala@it-sudparis.eu

Abstract. There is a little understanding of distributed solving activities in Open Source communities. This study aimed to provide some insights in this way. It was applied to the context of Bugzilla, the bug tracking system of Mozilla community. This study investigated the organizational aspects of this meditated, complex and highly distributed context through a linguistic analysis method. The main finding of this research shows that the organization of distributed problem-solving activities in Bugzilla isn't based only on the hierarchical distribution of the work between core and periphery participants but on their implication in the interactions. This implication varies according to the status of each one participant in the community. That is why we distinguish their roles, as well as, the established modes to manage such activity.

1 Introduction

Distributed problem-solving in Open Source context is a complex phenomenon and a fundamental issue at the organizational level. Research in this field was conducted from different theoretical and methodological perspectives. Realized on several projects Open Source such as Apache, Mozilla, Linux, researchers stated that organization of the work in this project is not totally democratic and observed that the coding is reserved to core developers (a limited number of developers which have code source access), when repairing defects and reported problems are periphery tasks (a large number of users/developers members). This distinction between Open Source community members tasks by previous results are interesting but need more empirical investigation and validation concerning the characterization how distributed problem-solving is organized in Open Source context by core and periphery contribution categorization and using linguistic techniques.

We have considered the case of Free/Libre Open Source Software (FLOSS) development as an open and distributed process. Mostly, we were interested in software problems found in Bugzilla' bug reports. This study investigates particularly the interaction between contributors in the case of the community associated with Mozilla's Firefox Internet browser. It analyzes the activities related to the participants in Bugzilla, the Mozilla's bug tracking system.

The rest of this paper is structured as follows: Section 2 presents the related research works. Section 3 describes our research method that has been used for conducting the linguistic analysis. Results are discussed and summarized in section 4. It describes linguistic specificities used in bug reports to identify the division of labor and contributors' roles in the organization of problem-solving. Section 5 includes concluding remarks and sketches the limits of this work as well as its future perspectives.

2 Related research

A lot of research has been done on coordination in Open Source Software community. If Raymond [21, p.4] describes the software debugging task in Open Source development style as "*self-correcting systems of agents*", an open model that he qualifies as a "*bazaar*" and argues, from a Linux experience, that "*Given enough eyeballs, all bugs are shallow*". In contrast, some research suggests that the openness in Open Source development does not imply necessarily democratic processes.

In a study on Linux Kernel, Cox [2] reveals that the access to exchanges is not granted to all the participants but reserved only for a category of developers (core developers). The resultant structure of exchange is quite hierarchical. Mockus and colleagues [15] have examined two Open Source projects and studied a division of labor in Mozilla and Apache. They have observed that the coding is reserved to a limited number of developers. Only 15 developers have contributed for the greater part to the design, the coding (80 %), and the validation of the solutions. They suggest that "*a group larger by an order of magnitude than the core will repair defects, and a yet larger group (by another order of magnitude) will report problems*" [18, p.9]. Crowston and Howison [4] have focused on social organization of Open Source projects and proposed a general description of this organization by providing the "Onion Model" of software development. According to this model, a small group of core developers is surrounded by several layers of peripheral helpers ranging from occasional problem solvers (close to the core) to mainstream users whose contribution is limited to the occasional submission of crash reports.

In a study on the governance of the Open Source Software (OSS) projects, Markus [16] proposes a definition according to which the mechanisms of coordination are perceived as answers to the problems of control and more generally as solutions to manage the development work: "*In the operational coordination literature, OSS governance is understood as a solution to* [*... the problem of*] *loss of operational control and the solution is techniques for managing the process of OSS development work*" [16, p.156]. Grinter and colleagues [8] said that "*the coordination of a distributed activity, bases on the communication and the interaction between developers"* [8, p.308]. Malone and Crowston [15] underline that coordination is an activity that is not directly observable. It is often studied through the communication in particular contexts where artifacts, e-mails, forums or lines of code, shape the structure of interaction and favor the teamwork. In a study of the Mozilla project, Gasser and collegues [7] shows that interactions participate to realize the coordination process and made through negotiation between contributors.

Until now according to our knowledge, little focused research has been done to understand how problems are being solved and described in OSS communities by using linguistic analysis. Indeed, apart from the interesting studies of Ripoche and Sansonnet [22], Ko and colleagues [10] and recently Maalej and colleagues [11,17], linguistic techniques have not really been used to describe the organization of distributed problem-solving related to Open Source software by interaction categorization.

3 Method

In this study, the bug report is the primary unit of analysis. We suppose that the organization of the community is reflected through the used tools that enable coordination of its activities. The traces of the structure should be visible in the community's bug tracking system. Therefore, a sample of 4109 bug reports was extracted from Bugzilla bug report repository. These bugs are selected in 2008 and specified the problems mentioned in Mozilla's CVS code archive before 2007 with a high complexity level (i.e. number of patches upper to 4) which were declared as solved. The analysis of our sample allows identifying the roles of participants in this activity according to their hierarchical statutes in the community. We specifically studied linguistic aspects of Bugzilla' traces since each bug report generates a discussion and all the exchanged messages between participants in the bug solving were recorded. Inspired by register linguistics analysis [1], our method is based on language categorization. A register is a variety associated with a particular situation of use and described for her linguistic features (lexical and grammatical characteristics). Registers can be identified and described based on analysis of texts or a collection of text. Especially, we look statistically at words that people use in Bugzilla to discern differences in the discourse and representation between participants (core and periphery).

4 Results through the linguistic analysis: Characterizing core and periphery contribution:

The emerging empirical literature on OSS communities indicates that a majority of code writing and communication activity is concentrated on a few individuals, the "core". Yet, these communities allow and encourage wide scale participation by anybody in their community, the periphery. Actually a large number of organizations and projects contribute to Bugzilla. The aim of this section is to explain by defining the roles of contributors, the division of labor amongst the core and periphery in a distributed problem-solving community and in essence to determine the value of the periphery to the core.

4.1 Contribution Frequency

Table 1 shows that 8072 individuals participated in our selection of 4019 bugs whose numbers are identified in comments to revisions to code belonging to the Firefox branch, or Firebird or Phoenix branch (as Firefox was formally known) in a version of Mozilla's CVS code archive dating from 2007. One hundred and twelve of these participants had CVS commit access, that is, the decision right to make changes to the community's software repository, were considered core community members.

Participation was defined by contributing code to the OSS community or more exactly by technical contribution such as patches submission [3, 16]. In our sample, a majority (91%) of patches are provided by core contributors (approximately 28471 patches). Similarly Table 1 shows, according to the bug status, that core dominate discussion. Considering all emails posted by contributors in our selection of bug reports, approximately 89% (182369 messages) of all messages provides from core.

We notice on the basis of this first result, that the contribution of the core members is globally more significant than the periphery. We observe that the technical work (submission of patches) is made by the core participants and the contribution of the periphery is not significant at this level. Consequently, if problem-solving activity is mostly technical and do not impose normally frequent exchanges, the core developers multiply nevertheless the exchanges with others contributors because only 16% of posted messages from the core contain patches. These observations approve the need of the core participants to communicate with the periphery, and to strengthen the hypothesis according to which the contribution of the periphery is also important as that of the core, in the problem-solving activity [11]. To characterize better these contributions, we tried to study in the second part of this section, the exchanges through an analysis of e-mail discussions.

Table 1. Core and periphery contribution

Community status	Number of participants	Number of messages	Number of patches
Core	112	182369(89%)	28481(91%)
Periphery	7960	22538(11%)	2796 (9%)
Total	8072	204908(100%)	31277(100%)

4.2 Contribution categorization

In this section, we characterize core and periphery with language categorization. In order to do so, we first define a subset of organizational proprieties identified in linguistic analysis and encode these proprieties as description, directives, activity and interaction. We then code bug reports into proprieties strings made up of this categorization. As expressed, we consider four proprieties:

1) Diagnosis and evaluation by description (**DESCR**): supply a maximum of information to the community to orient the problem-solving. The participants contribute to the activity by proposing information which allows:

- To describe the problem context: states provoked by the problem and the environment in which it is located (e.g. "`frequently the ad server that is used by one of the forums I read is down, and when the request for the ad fails; Firebird shows a modal dialog telling me the connection was refused or something like that`").
- To display and reproduce the problem (e.g. "`go to www.mlb.com and click on a 'gameday' icon`").
- To validate the state of normal functioning of the module further to integrate solution (e.g. "`I downloaded the latest release today (V 1.0.7) installed it and it works exactly the way it did previously with regard to this bug!`").

2) Coordination by the directives which (**DIREC**):

- To conduct the execution of some tasks (e.g. “`make it listen to command rather than click, add Alt+Down / Up and F4 as equivalents to the dropdown button`”).

- To attribute some tasks to a particular contributor by explicit requests (e.g. “`Tim could you review this and land it on branch and trunk`”).

- To verify that the instructions are correctly led (“`The patch can bereviewed I tested it today and didn't run into any problems`”).

3) Activity explicitness (**ACTIV**): to perform an action directly or report on the performance of an action by:

- The creation of a patch, i.e. a suggested change in the code base (e.g. “`Created an attachment (id=135235) [details]` ”).

- The update of a designed solution: to indicate the link which allows to download the update of the solution (e.g. “`From update of attachment 135235 [details]` ”).

- The verification of the conceived solutions: when one or several solutions are proposed, several stages of reviewing and tests are implemented (e.g. “`review=me,superreview=brendan@mozilla.org for trunk checkin`).

- The fixing of bugs: after checking, the solution is validated to fix the bug (e.g. “`verified, fixed` ”).

4) Activity articulation by direct interaction (**INTER**): Communication between participants, is transformed into dialogue between two or three contributors by the use of expressive linguistic forms such as:

- Agreement or disagreement (e.g. “`Actually, I'm not sure I agree with Gerv here. I understand that RMS agrees Mitchell`”).

- Interrogative forms (e.g. “`Benjamin, could you explain why?` ”).

- Forms which express the emotion or gratitude (e.g. “`Thanks for writing this, Daniel! I'd say it's a pretty good starting point; Nice one :-) That's what I couldn't work out. Go to it. I'm away for the next nine days; thank you so much for clearing up after me :-) Gerv`”).

Having thus transformed the sample, we focus the frequency of the occurrence of organizational proprieties according contributors status. Figure 1 reveals that when considering organizational categorization, as a percentage, the used language by core members is essentially descriptive (42,45%). If we consider the language from the point of view of action, we notice that the core contributors often use the linguistic units, to clarify an action or to anticipate the actions to be realized (26,30%) with equivalent proportions of directives (28,19%). Finally, we note a small proportion (3%) of linguistic units indicating the presence of direct interactions between core members, and other contributors. Figure 1 also reveals that proportions of actions, descriptions, and directives are even more important after the conception of the first patch. These observations confirm our first observations , indicating that the core members intervenes not only in the critical phases of software design, but also in the phases of reviewing, validation and integration of solutions, realized in the last phases of the problem-solving. These reports can be understandable by the fact that the Mozilla project is strongly structured, and that the integration of contributors is not completely free and requires the downstream of administrators of the project.

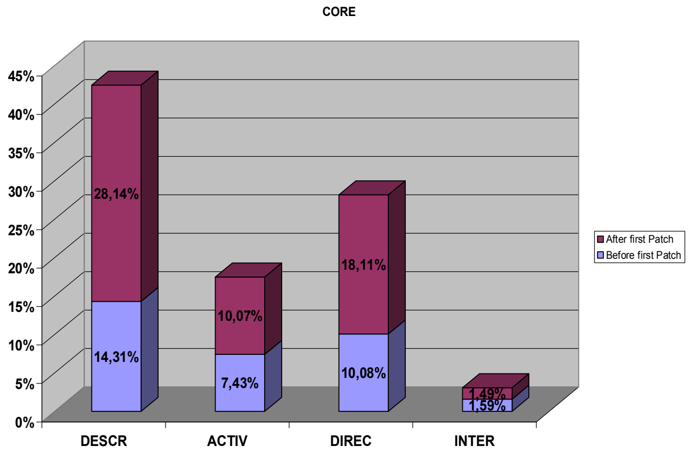

Fig.1. Core contribution to manage problem-solving before and after the first patch

Figure 2 shows that peripheral members use an important rate of descriptive words (49,39%) in comparison with the other categories. Besides, we observe less important rates of interactive words (18,47%) and linguistic forms reporting action performed (15,68%). We note finally of small proportions (8,91%) of directives in comparison with the core contribution.

These results support works supporting that the contribution of the periphery in Open Source communities consists mainly in declaring problems, asking for instructions, or for instructions concerning it, without intervening in a significant way in its solving (patch submission) [22,4]. This thus explains the high frequencies of messages of the category description, followed by the category interactivity, and particularly before beginning to conceive a solution to the problems. The contribution of the periphery is thus more significant and important in the first phases of the problem-solving, which are characterized by a very strong rate of description.

Our results suggest that most common forms employed by the core are used for assigning explicitly tasks to developers and asking questions about problems. In general, the core members use more professional languages to ask questions about program output and purpose technical solutions. However, it is important to have a detailed understanding of problems by identifying problem context. Our results suggest the peripheral role to perform this categorization by use descriptive words as an indicator. For example, to reproduce contextual elements by using words such as “when”, “during”, and “after”, in order to indicate the situation in which a problem is occurred.

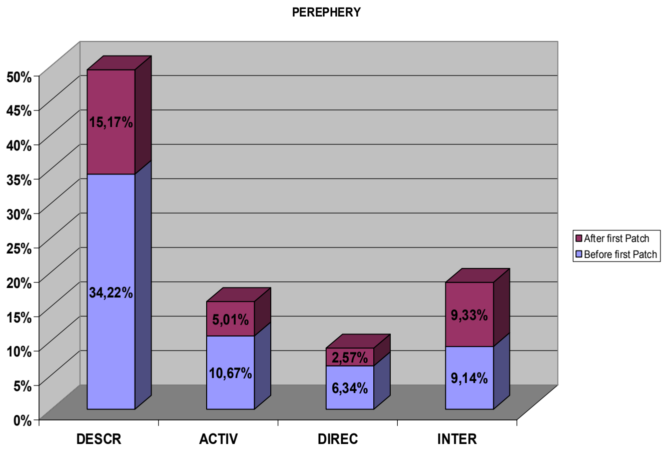

Fig.2. Periphery contribution to manage problem solving before and after first patch

5 Discussion and Conclusion

This study serves as a starting point of a larger effort to better understanding of how the core and the periphery of an online community contribute to manage software problem-solving by focusing on Bugzilla community, and using several methodologies derived from computational linguistics. We have observed in this study same differences in the structure and content of language used by contributors in our dataset. This difference has a variety of implications to describe the organization of Bugzilla activities in the case of Firefox and distinguish between the core and periphery roles. Many research works found that most of the technical activities with respect to bug solving are carried out by a small minority of core members while periphery contributions are less significant.

We have initially suggested that collaboration and interaction between the core and periphery of the community is an important aspect of problem-solving in the Bugzilla, and then identified the peripheral role in managing this activity. The main difficulty in managing reports is how to determine the most qualified developer for each report. There have been attempts to automatically match developers with specific action or task, based on the correspondence between the task and the contributor’s skills.

This study provides an empirical validation of previous results concerning the role of core and periphery OSS community members using a different analysis technique: linguistic categorization. It shows that if the writing of the code and more globally the technical work are the principal role of the core members. Peripheral members play also an important role in the problem-solving activity. Indeed, we examined that the role of the periphery is not only in helping to formulate the exact context by describing problems, but also in proposing potential information on solutions. The results presented here are interesting, but still very preliminary. Further investigations are so needed, for example, to analyze higher level issues in problem-solving processes such as roles’ evolution according problem-solving phases.

References

1. Biber, D., and Conrad, S.: Registre, genre and style, C*ambridge and New York: Cambridge University Press*, 344p (2009)

2. Bird, C., Gourley, A., Devanbu, P., Swaminathan, A., and Hsu, G.: Open borders? immigration in open source projects. In MSR '07: Proceedings of the Fourth International Workshop on Mining Software Repositories. *IEEE Computer Society*, Washington, DC, USA (2007)

3. Cox, A.: Cathedrals, bazaars and the town council. Available at http://www.linux.org.uk/Papers_CathPaper.cs (1998)

4. Crowston, K. and Howison, J.: Hierarchy and centralization in free and open source software team communications. Knowledge, *Technology & Policy*, Vol.18, N°4, pp. 65–85 (2006)

5. Ghosh R.: Managing rights, in free/libre/open source software, http://www.infonomics.nl /FLOSS/papers/20060423/GHOSH-licensing.pdf (2006)

6. Gasser, L., Gabriel, R.: Distributed Collective Practices and Free/Open-Source Software Problem, Management: Perspectives and Methods. *Conference on Cooperation, Innovation & Technologie* (CITE'03), 17p (2003)

7. Grinter, R. E., Herbsleb, J.D., Perry, D.W: The geography of coordination: Deling with distance in R&D work. In Proceedings of SIGGROUP Conference on Supporting Group Work, New York: ACM Press, pp.306-315 (1999)

8. Herbsleb, J. D., and Cataldo, M.: Factors leading to integration failures in global feature-oriented development: an empirical analysis. *In Proceedings, International Conference on Software Engineering, Honolulu*, HI, pp. 161-170 (2011)

9. Ko, A. J., Myers, B.A., Chau, D. H. : A Linguistic Analysis of How People Describe Software Problems in Bug Reports. Visual Languages and Human-Centric Computing, Brighton, United Kingdom, September 4-8, 127-134 (2006)

10. Lakhani, K.R.: The core and the periphery in distributed and self-organizing innovation systems. Thesis (Ph. D.) from Sloan School of Management, Cambridge, MA: Massachusetts Institute of Technology, pp. 331 (2006)

11. Lanzara, G.F. and Morner, M.: Artifacts rule! how organizing happens in open source software projects. In Czarniawska, B. and Hernes, T., editors, Actor-Network Theory and Organizing, Copenhagen Business School Press, pp. 67–90 (2005)

12. Maalej, W., and Happel, H-J: Can Development Work Describe Itself? In Proceedings of the 7th IEEE Conference on Mining Software Repositories (MSR 2012), IEEE CS, 191-200 (2012)

13. Malone, T.W.: Toward an interdisciplinary theory of coordination, Tech. Rept. CCS 120, *MIT Sloan School of Management*, Cambridge, MA (1991)

14. Markus, M.L.: The governance of free/open source software projects: monolithic, multidimensional, or configurational?. *Journal of Management and Governance*, vol. 11, N° 2, pp. 151-163 (2007)

15. Masmoudi, H. : La résolution distribuée dans les communautés Open Source : propriétés organisationnelles et modes de coordination. Thesis (Ph. D.) from Paris Dauphine University, Paris, France, pp. 265 (2011)

16. Mockus, A., Fielding, R.T., Herbsleb, J. D.: Two case studies of open source software development: Apache and mozilla. *ACM Transactions on Software Engineering and Methodology*, Vol. 11, N°3, pp. 309–346 (2002)

17. O'Mahony, S., Ferraro, F.: The emergence of governance in an open source community. *Academy of Management Journal*, Vol. 50, N° 5, pp. 1079-1106 (2007)

18. Pagano, D., and Maalej, G.: How do developers blog?: an exploratory study. In Proceedings of the 8th Working Conference on Mining Software Repositories (MSR '11). ACM, New York, NY, USA, 123-132 (2011)

19. Raymond, E. S.: The cathedral and the bazaar. *First Monday*, N° 3 (1998)

20. Ripoche, G. and Sansonnet, J.-P.: Experiences in automating the analysis of linguistic interactions for the study of distributed collectives. Computer Supported Cooperative Work, vol.15, PP.149–183 (2006).