Recommending Mentors to Software Project Newcomers

Igor Steinmacher, Igor Scaliante Wiese
Computer Sciences Coordination
Federal University of Technology – Paraná (UTFPR)
Campo Mourão, Brazil
{igorfs, igor}@utfpr.edu.br

Marco Aurélio Gerosa
Computer Science Department
University of São Paulo(USP)
São Paulo, Brazil
gerosa@ime.usp.br

***Abstract*—Open Source Software projects success depends on the continuous influx of newcomers and their contributions. Newcomers play an important role as they are the potential future developers, but they face difficulties and obstacles when initiating their interaction with a project, resulting in a high amount of withdrawals. This paper presents a recommendation system aiming to support newcomers finding the most appropriate project member to mentor them in a technical task. The proposed system uses temporal and social aspects of developer's behavior, in addition to recent contextual information to recommend the most suitable mentor at the moment.**

***Keywords*-recommendation system; open source software; newcomers; mentor recommendation**

I. INTRODUCTION

Many open source software (OSS) projects are self-organized and dynamic with volunteers from all over the world contributing and collaborating to a software product. A continuous influx of newcomers and their active engagement with development activities are crucial for the success of Open Source Software (OSS) projects [1].

However, newcomers face difficulties and obstacles when initiating their interaction within a project. Degenais et al. [2] compare OSS newcomers to explorers who must orient themselves in an unfamiliar landscape. On one hand, they are expected to learn about technical and social aspects of the project on their own, exploring the information available in mail lists, wikis, source code repositories, and issue tracking systems [3]. On the other hand, it is not easy to access the information because of its sheer volume, the lack of tools to effectively navigate the repositories, and the difficulty of making connections between logically related items in disparate repositories [4]. Additionally, there is no guarantee that the information available are up-to-date or complete enough to support a newcomer, what can result in misunderstandings and possible rework.

To avoid this kind of situation, newcomers often start their contribution by interacting with other members [1]. They use the mail lists or developers' contact information listed on the project website to help them choose a task, finding the right resources, report interest, etc. [5]. However, receiving an improper answer (or no answer) when sending an email can result in newcomers withdrawal. Von Krogh et al. [5] and Jensen et al. [6] analyzed the history of mail lists of OSS projects and demonstrated that receiving a (timely) reply is essential to make newcomers continue contributing. In Section II, we present two real cases in which improper communication, outdated information and lack of information discouraged newcomers.

In traditional software development teams, existing members are assigned as mentors to guide newcomers [7]. According to Degenais et al. [2], human guides make a key difference to how easy it is for newcomers to find their way and settle in. Newcomers can have their mentor as a safe harbor, who can warn them about possible problems to be faced and show them what is important to know when executing a given task. Mentors can emphasize hard-to-find information that is typically difficult for the newcomers to acquire on their own [4].

In order to help addressing the issues and obstacles faced by newcomers, in this paper, we propose a recommendation system to help them finding the most appropriate project member to mentor a specific technical task (e.g., a bug), guiding their initial steps in an OSS project. To proceed with the recommendation, we aim to use historical information available on source code repositories, mail lists and issue trackers, and users' interaction with the IDE. To check the most suitable person to mentor the newcomer, the system will take into account specific user behavior regarding temporal and social aspects.

The rest of the paper is organized as follows: in Section II we present some cases that motivated our research; in Section III we present the proposed recommendation system; Section IV brings some related works; and Section V presents some concluding remarks and future works.

II. (DE) MOTIVATION CASES

In this section, we will present two real cases that occurred in an Open Source course attended by a group of PhD candidates, including two authors of this paper. During the course, the students were separated in groups and requested to join and contribute to different OSS projects. These cases illustrate some obstacles that newcomers face when starting their contribution to an OSS project.

A. Case I

The first case occurred with a group joining an 8 years old project with 30+ developers and more than 5000 weekly downloads. They started lurking on documentation, mail lists and to set up their local workspaces. They sent an email to the developers that appear as project owners requesting some guidance on which bugs could be good to start with or what kind of technical work was needed at that moment. The email was not replied after one week. They insisted sending

another email and the day after they received a reply, that was not so helpful, apologizing for the delay.

The group started (by themselves) identifying some opened bugs and features that were apparently 'easy' to be addressed and analyzing the code to find out the classes and artifacts that needed to be changed. They selected an 'apparently small' feature request and implemented. But, when they started testing the feature, they found a shortcut key with the desired feature already in place. Although already addressed, the feature request information was outdated, with the status open, without any comment posted. After 2 months contributing, the group tried to address 12 features/bugs, however they found that 5 of them (42%) were already addressed, but the issues were not updated.

Another issue occurred when the group decided to start translating the software into their native language. They announced the translation in the open discussion forum (where translations to other languages were announced before). After 20 days working one member noted that the translation files had appeared on his workspace after pushing the changes from the central repository. Another contributor had already proceeded the translation. The forum thread announcing the translation did not receive any reply or comment. We contacted the committer and he said he does not even look at that forum and that the other translator contacted him in private and started translating.

We can see many demotivating facts that occurred in this case: emails not answered after a week could make the group withdraw; outdated information on the issue tracker made the developers waste time on an already existent feature and on checking each issue they pick to address; a message posted in a forum to announce a new translation was not read and resulted in concurrent work and wasted time.

B. Case II

The second case was reported by a group that started contributing to a large, successful and well known OSS project. They have also faced some issues when initiating their work. When asking the owners and core developers about what kind of technical work could be done by newcomers they were directed to the project page and to a specific page presenting a step by step on how to start contributing. The guide, according to them, is very well structured and present valuable information

The newcomers followed the guide and decided to start fixing some bugs. At this point they found the same problem as the aforementioned group faced in Case I: outdated information when picking bugs to solve. They found many bugs tagged as good for newcomers, however some were already fixed, but still with 'open' status without any information regarding how it was addressed. They also wasted time on an already existent feature and on checking each issue they pick to address before thinking about the solution. Once again, newcomers became demotivated due to outdated information.

III. RECOMMENDATION SYSTEM TO SUPPORT NEWCOMERS

Newcomers have difficulty on guiding themselves in the middle of a huge amount of information, which they do not know if is up-to-date or not, and with no clues on who can provide them a timely answer when they face a problem. To address this problem we propose a system that recommends a project member who can act as a mentor for a newcomer in a given technical task.

The proposed system focus on newcomers that want to start contributing directly on the source code, implementing a bug fix or addressing an issue. We are not interested in recommending the most appropriated person to answer a question sent to a forum or mail list, but in recommending the most adequate person to provide support to a newcomer in a given task. It is worth to notice that we also do not intend to support or guide newcomers selecting an issue, but recommending a mentor for the issue they select.

When newcomers want to start their technical contribution to a project, they can do it choosing a bug or an issue they find appropriate (some OSS projects – e.g. Mozilla – already tag the issues as 'easy' during the triage). According to the issue selected, a mentor is recommended to the newcomer, enabling him to start an interaction with other project member. The mentor is a person who can advise the newcomer and provide some help for that specific bug or issue. The support provided can include indicating the artifacts to look at or to change, the documents that can offer support, indicating the most appropriate forum to deliver a question, and how to get the code into the repository.

To understand how the recommendation system works, we present the proposed high level architecture in Fig. 1. The recommendation process will be triggered every time a new issue is reported. All recommendations will be recalculated periodically for all the issues opened, to update the mentor according to the most recent interactions and events. The system will recommend the most appropriate mentor according to developers' technical, social and current interest scores. The system inputs are information coming from different sources. These inputs are analyzed to calculate the scores for each developer and recommend the mentors. In Section III.A we present details of the inputs and in Section III.B we detail the modules that handle these inputs and calculate the technical, social and current interest score.

A. Recommendation System Inputs

In order to recommend the most appropriate mentor to an issue we will use as inputs the historical information and workspace information. Our goal is to recommend the most suitable mentor at that moment. By "most suitable", we mean that the system will recommend someone with skills that match the selected issue. However, our approach will also consider temporality of the interactions, and developers' social skills. We will use this information to check the recent activity of the developers to verify if they are still active and to verify their recent interests.

The historical information will be extracted from source code history, mail lists threads and issue tracker comments, like in other works found in the literature [8] [9] [4] [10]. The workspace context will be extracted from the stored developers' interaction events with the IDE, like in [11] [12].

The historical inputs enable to infer how developers and source code are related. From the **source code repository** it

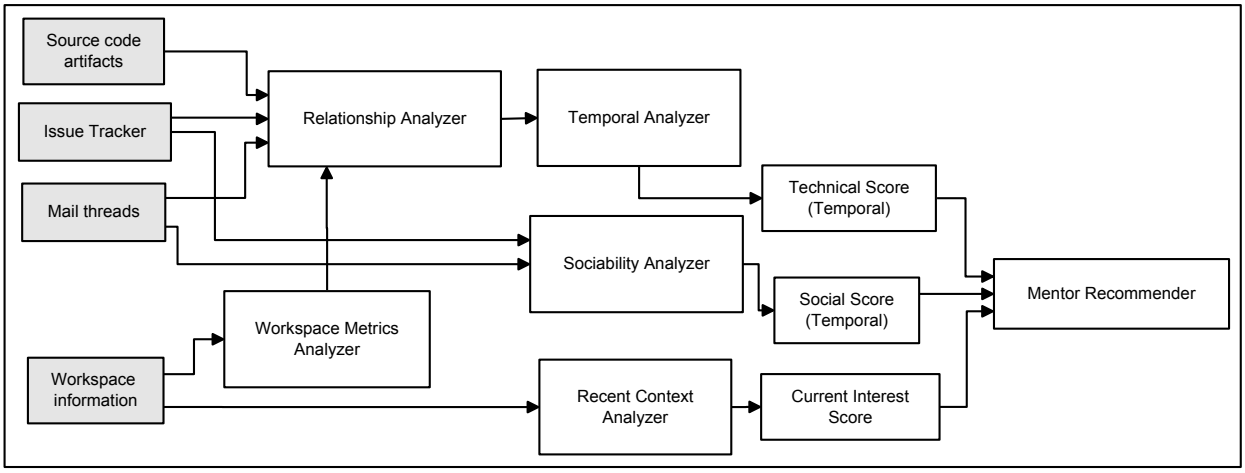

Figure 1. System high level architecture

is straightforward to extract the relationship among source code artifact and developer – '*who changed what'* – simply checking the history. It is also possible to infer logical dependencies among source code artifacts by checking the co-changes [13] – '*which artifacts were changed and committed together*'. Also using the source code repository it is possible to gather implicit relationships among developers using the method proposed by Cataldo et al. [14] – '*which developers cooperated by changing related artifacts*'. The information from source code repository will be used to calculate the technical score. From the issue tracker we can extract the developers that collaborated – '*who collaborated with a task*' – and information from the comments and attachments, that will compose the technical score.

From **mail lists** and **issue trackers** comments it is possible to check the social relationship among developers – *'who talked to who'* – checking the threads and discussions. This information will compose the social score.

The **workspace information** enables us to be aware of the micro interactions among developer and artifacts – 'which artifacts had been locally handled by who'. In addition, it provides richer information, enabling one to measure how much effort a developer put in editing a given artifact (and in a project), what is the frequency that a developer work on a project, the "degree of interest" [15] in a file, what are the most accessed artifacts (even if not edited). This information will also be used to compose the developers technical score.

The workspace information can also be used to gather developers' recent activities, enabling to retrieve their current interest and the amount of effort spent with the project recently, even before they commit to a repository or report something in mail lists or forums. This will be the input used to calculate developers current interest score.

B. Recommendation System Architecture

The inputs presented will be further analyzed by the recommendation system modules to calculate the technical, social and current interest score. In this subsection, we will briefly explain how the modules handle the inputs to recommend the mentors.

Relationship Analyzer: This module is responsible for linking historical information from different sources. The links will be created by using text similarity and a set of heuristics. The heuristics to be used include textual analysis and bug report attachments to verify explicit references that links source code, messages and bugs. For example, it is possible to link specific commits with bugs verifying the existence of a bug identifier in a commit message. Also, in some cases a bug report can present a stack trace, enabling to check where the root of the problem reported is. The output of the relationship analyzer is a data structure that links items from different sources, as show in Fig. 2, enabling the system to infer the relationship issue x developer.

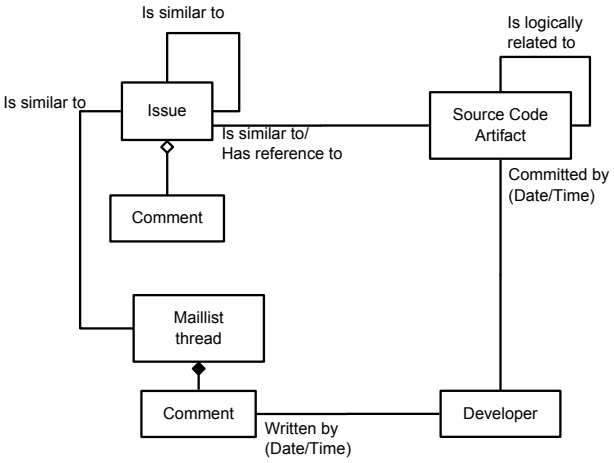

Figure 2. Links among items from different sources of information

Temporal Analyzer: The linked items will be then analyzed in order to make the recent relations more relevant than old ones. This will be made depreciating the older interactions and emphasizing recurrent interactions. For example, commits made in the current month will be considered fresh and their score will be fully considered.

Last month commits will have their score depreciated to 90%, commits made the month before last month will be depreciated to 80% and so on. If a developer commits a software artifact recurrently, this score of this interaction will be amplified. By analyzing the information temporally we aim to focus on recent and recurrent activities, enabling to find recent interests and developers that are currently active. It is worth noticing that the temporal analysis will be conducted only over the relations that involve developers.

Sociability Analyzer: This module is responsible for checking the social activity and skills of a developer. The goal is to verify developers who also contribute answering mail lists and posting comments on issue trackers. The output is a map linking developers and a sociability score that consider the amount and the temporality of comments.

Workspace Metrics Analyzer: This module will handle the data captured from developers' interaction with their integrated development environments (IDE). The events captured include editing, selecting and opening source code artifacts. These information will be used to calculate Micro Interaction Metrics [11], quantifying the complexity and intensity of developers' interaction activities such as browsing or editing of files. The metrics extracted mainly link developer and files regarding the effort and degree of interest [15]. The output will also be linked to the issues in the relationship analyzer module, since it can provide important information regarding the interaction between developers and source code artifacts.

Recent Context Analyzer: This module will be responsible for analyzing the information collected from developers' IDEs recently. The goal is to be aware of the developers' current activity level and their current interest (focus). This information can be used to fine tune our recommendation system. For example: recommending a currently active mentor can increase the chance of a timely response; recommending a mentor based on the current focus, can enhance the chance to choose a developer currently interested on a related issue or feature; dismissing developers with low recent activity level can avoid newcomers to have a non-replied email.

Mentor recommender: This module is responsible for combining the information generated by Temporal Analyzer, Sociability Analyzer and Recent Context Analyzer, classifying developers according to their scores and recommending the most suitable mentor for an issue at that moment. The most suitable mentor will be the developer that presents the best combination of current interest, social and technical skills.

IV. Related Works

There are many works in the literature that deal with the person recommendation in software engineering. Most part focus on recommending experts in a given subject or artifact.

Expertise Browser (ExB) [16] is a tool that uses the concept of Experience Atoms (EAs) to represent the expertise unities gathered from source code repositories. EAs are used to generate a sociotechnical network involving the relationships among artifacts, people, and tasks. It is used to identify experts and trace their relationships.

SmallBlue [17] uses social networks extracted from email messages and instant messenger to rank the experts. The analysis is performed by associating names and topics extracted from the messages. The experts search is made via web, where a user can provide a subject, and the system generates an ordered list with the experts on that subject.

STeP_IN [8] recommends developers with expertise in a specific Java class or method analyzing the source code and mail contacts. When a question about a piece of code is sent to the mail list, the system forward the message to a set of developers that are expected to answer the question. To choose the experts the tool check the developers who had already being in touch with that artifact and the social network of the developer who asked the question.

Conscius [9] also aims to facilitate access to experts on a given software project. The ranking of the experts is made by mining the SCM change history and archived mail threads. The tool analyzes the content of the communication to improve the usual recommendations based on the source code history and the relationship between the source code.

Codebook [18] is a tool that generates a social network from source code repositories, documentation and messages. Links are established between activities (work items), their artifacts, and developers involved. The resulting network can be consumed by tools, such as the Hoozizat [18], a web search portal to search for specific experts on features, APIs, products, or systems.

Emergent Expertise Locator (EEL) [10] also aims to recommend experts on a given subject. It uses the change history of source code to rank the experts of a given artifact. To sort the experts, the tool makes use of the coordination matrix proposed by Cataldo et al. [14].

None of the approaches proposed to recommend experts focus on the retention of newcomers, focusing on finding out experts in a given artifact or subject, based on historical perspective. Other than this, they do not look at messages temporality to verify the recent interests and activity to check who the 'most suitable person at the moment' is. Additionally, in our proposal we aim to use developer's workspace contextual information, to make recommendations more specific [19].

We also found in the literature a recommendation tool called Hipikat [4], that aims to support newcomers by building a group memory using source code, email discussions and bug trackers. The user proactively request recommendation based on existent artifacts. Hipikat returns a list of source code, mails messages and bug reports that present are related to the queried artifact. This is the closest approach we encountered, but it differs from ours because in our approach we believe that a human mentor can be more helpful than providing a set of artifacts (that not necessarily are up-to-date) to support user actions.

Our work is also related to researches that studies OSS projects joining process, and the importance of newcomers to OSS projects. Von Krogh et al. [5] conducted a qualitative study over Freenet project and proposed the concept of a "joining script" for new developers joining a community. They emphasize the importance of newcomers, noting the

high turnover rate among developers, and that recruiting is a concern among the developers. Ye and Kishida [20] describes a conceptual framework to analyze the motivational issues in OSS. They state that newcomers are vital to the success of OSS projects. Park and Jensen [1] studies the information needs of newcomers and the potential benefits of visualization tools to support newcomers learning about an OSS project and help them finding information more efficiently and effectively. Jensen et al. [6] study mailing lists of four OSS projects to understand how politeness, helpfulness and timeliness of the replies to newcomers questions influences their future interactions. They found that prompt feedback is essential to continued participation, and indifferent replies can have a chilling effect on lurkers, who may decide to give up.

V. Concluding Remarks

In this paper, we present the proposal of a recommendation system that aims to identify who is the most suitable person to mentor a newcomer in an OSS project task to reduce the problem of initial interactions. By using temporal and social analysis, we aim to recommend the mentors that are active and have social skills. The use of recent contextual information improves the quality of the recommendation, aiming to point the newcomers to mentors that are active and focusing on related issues.

Next steps include implementing the approach presented and conduct experiments based on historical information mined from bug trackers, and also case studies. Our final goal is to answer the following research questions:

- Do newcomers remain for longer periods in OSS projects if the right mentor is recommended to them?
- Is a mentor recommendation system a good approach to avoid core developers' overload?
- Can we improve the recall and precision of mentor recommendations if we analyze the temporality and sociability aspects of the relationships?

Acknowledgement

This work was partially funded by Fundação Araucária. The authors would also like to thank Ana Paula Chaves and Marcela Batista for proofreading preliminary versions of this document. Marco Aurélio Gerosa receives individual grant from Brazilian National Research Council (CNPq).

References

[1] Y. Park and C. Jensen, "Beyond pretty pictures: Examining the benefits of code visualization for Open Source newcomers," in 5th IEEE International Workshop on Visualizing Software for Understanding and Analysis (VISSOFT), 2009, pp. 3-10.

[2] B. Dagenais, H. Ossher, R. K. E. Bellamy, M. P. Robillard, and J. P. de Vries, "Moving into a new software project landscape," in Software Engineering, 2010 ACM/IEEE 32nd International Conference on, May 2010, pp. 275-284.

[3] W. Scacchi, "Understanding the requirements for developing open source software systems," Software, IEE Proceedings -, v. 149, no. 1, pp. 24-39, Feb. 2002.

[4] D. Cubranic, G. C. Murphy, J. Singer, and K. S. Booth, "Hipikat: a project memory for software development," IEEE Trans. Softw. Eng., v. 31, no. 6, pp. 446-465, Jun. 2005.

[5] G. von Krogh, S. Spaeth, and K. R. Lakhani, "Community, joining, and specialization in open source software innovation: a case study," Res. Policy, v. 32, no. 7, pp. 1217-1241, Jul. 2003.

[6] C. Jensen, S. King, and V. Kuechler, "Joining Free/Open Source Software Communities: An Analysis of Newbies' First Interactions on Project Mailing Lists," in Proceedings of the 2011 44th Hawaii International Conference on System Sciences, Washington, DC, USA, 2011, pp. 1-10.

[7] M. S. Elliott and W. Scacchi, "Free software developers as an occupational community: resolving conflicts and fostering collaboration," in Proceedings of the 2003 international ACM SIGGROUP conference on Supporting group work, New York, NY, USA, 2003, pp. 21-30.

[8] Y. Ye, K. Nakakoji, and Y. Yamamoto, "Reducing the Cost of Communication and Coordination in Distributed Software Development," in Software Engineering Approaches for Offshore and Outsourced Development, Springer Berlin / Heidelberg, 2007, v. 4716, pp. 152-169.

[9] A. Moraes, E. Silva, C. da Trindade, Y. Barbosa, and S. Meira, "Recommending experts using communication history," in Proceedings of the 2nd International Workshop on Recommendation Systems for Software Engineering, New York, USA, 2010, pp. 41-45.

[10] S. Minto and G. C. Murphy, "Recommending Emergent Teams," in Proceedings of the Fourth International Workshop on Mining Software Repositories, Washington, DC, USA, 2007.

[11] T. Lee, J. Nam, D. Han, S. Kim, and H. P. In, "Micro interaction metrics for defect prediction," in Proceedings of the 19th ACM SIGSOFT symposium and 13th European conference on Foundations of software engineering, New York, USA, 2011, pp. 311-321

[12] I. Omoronyia, J. Ferguson, M. Roper, and M. Wood, "Using Developer Activity Data to Enhance Awareness during Collaborative Software Development," Computer Supported Cooperative Work (CSCW), vol. 18, pp. 509-558, 2009.

[13] G. A. Oliva, F. W. S. Santana, M. A. Gerosa, and C. R. B. de Souza, "Towards a classification of logical dependencies origins: a case study," in Proceedings of the 12th International Workshop on Principles of Software Evolution and the 7th ERCIM Workshop on Software Evolution, New York, NY, USA, 2011, pp. 31-40.

[14] M. Cataldo, P. A. Wagstrom, J. D. Herbsleb, and K. M. Carley, "Identification of coordination requirements: implications for the Design of collaboration and awareness tools," in Proceedings of the 2006 conference on Computer supported cooperative work, New York, NY, USA, 2006, pp. 353-362.

[15] M. Kersten and G. C. Murphy, "Using task context to improve programmer productivity," in Proceedings of the 14th ACM SIGSOFT international symposium on Foundations of software engineering, New York, NY, USA, 2006, pp. 1-11.

[16] A. Mockus and J. D. Herbsleb, "Expertise Browser: a quantitative approach to identifying expertise," in. Proceedings of the 24rd International Conference on Software Engineering, 2002, May 2002, pp. 503-512.

[17] N. S. Shami, K. Ehrlich, and D. R. Millen, "Pick me!: link selection in expertise search results," in Proceedings of the 2008 Conference on Human Factors in Computing Systems, New York, 2008, pp. 1089-1092.

[18] A. Begel, Y. P. Khoo, and T. Zimmermann, "Codebook: discovering and exploiting relationships in software repositories," in Proceedings of the 32nd ACM/IEEE International Conference on Software Engineering, vol. 1, Cape Town, South Africa,, 2010, pp. 125-134.

[19] H. Happel and W. Maalej, "Potentials and challenges of recommendation systems for software development," in Proceedings of the 2008 international workshop on Recommendation systems for software engineering, New York, NY, USA, 2008, pp. 11-15.

[20] Y. Ye and K. Kishida, "Toward an understanding of the motivation Open Source Software developers," in Proceedings of the 25th International Conference on Software Engineering, Washington, DC, USA, 2003, pp. 419-429.