Recommending Experts Using Communication History

Alan Moraes, Eduardo Silva
Department of Informatics
Federal University of Paraíba
João Pessoa, PB, Brazil
alan@di.ufpb.br, eduvfsilva@di.ufpb.br

Cleyton da Trindade, Yuri Barbosa, Silvio Meira
Center of Informatics
Federal University of Pernambuco
Recife, PE, Brazil
cct@cin.ufpe.br, yamb@cin.ufpe.br, srlm@cin.ufpe.br

ABSTRACT
In distributed software development the communication is inefficient because of geographical and temporal distances, affecting the team's performance and awareness. The low level of awareness makes hard the task of finding the expert of a piece of source code, delaying the implementation whenever a developer needs help. To identify and to recommend the people with right knowledge to people in trouble during the implementation can improve the collaboration and awareness of the team because it can reduce the waiting time for an answer, since the expert can be contacted directly. In this paper we propose recommender system for expert location with the aim to reduce delays of finding the right person whenever somebody needs assistance during coding. Our approach uses the communication history of the project (the developer's mailing list) in addition to usual source code history. We also present results which show the practical potential of our approach.

Categories and Subject Descriptors
K.6.1 [**Management of Computing and Information Systems**]: Project and People Management – *staffing*.

General Terms
Algorithms, Experimentation, Human Factors.

Keywords
Expert Recommender System, Knowledge Management, Distributed Software Development, Global Software Engineering

1. INTRODUCTION
In today's world, software development is increasingly spread across national and geographic boundaries. The companies involved in distributed software development are seeking mainly to lower costs, to access skilled resources and the business advantages of proximity to the market [1].

But the advantages do not come without drawbacks. During the software development, team members typically share and exchange knowledge about their work interacting constantly with others developers [3][8]. Examining the communication process in distributed teams, we note the lack of opportunities for synchronous interactions that naturally occur in co-located development, making difficult to share the knowledge of the software under development itself [9].

The lack of deep and confident knowledge of the software itself could potentially affect negatively the team's productivity. This problem is clearly perceived whenever somebody needs help during the implementation of a piece of source code from someone yet unknown and remotely located, because the access to remote team members and the synchronous communication options are often restricted. These limitations delay the starting of the collaboration process among the team members and make hard to find the source code expert [2].

As pointed in the literature, the problem of expert location is a real challenge distributed teams face every day [2][3][4]. To identify and to recommend the people with right knowledge to people in trouble during the implementation can improve the collaboration and awareness of the team because it can reduce the waiting time for an answer, since the expert can be contacted directly. Thus people, their specialties and their responsibilities become clearer to each of the developers, in this way contributing to more effective communication and then possibly improving the team's productivity. Quality expert recommendation systems can supply an important part of the awareness lost due geographical and temporal distances.

Despite the fact of poor quality of the communication in distributed teams, there is still a lot of communication happening in various channels, such as mailing lists, version management systems, bug tracking systems and so on. To the best of our knowledge, none of the expert recommender system uses the mailing list content to recommend source code experts.

So our research questions are: (1) Can we recommend only using the project communication history? (2) Can we to improve the recommendation quality by using the project communication history in addition to source code history?

In this paper we propose recommender system for expert location with the aim to reduce delays of finding the right person whenever somebody needs assistance during coding. Our approach uses the communication history of the project (the developer's mailing list) in addition to usual source code history. In the sections that follow, we present the related works, our approach and its evaluation through an experiment, and the final remarks.

2. RELATED WORKS
The Expertise Browser [19] uses experience atoms (EA), basic units of experience, as the basis for recommending experts.

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. To copy otherwise, or republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee.
RSSE'10, May 4, 2010, Cape Town, South Africa.
Copyright © 2010 ACM 978-1-60558-974-9 /10/05…$10.00.

Experience atoms are created by mining the version control system for the author of each file revision and the changes made to the file. Rules of thumb were applied to identifying candidates which had expertise on a particular software project and, more specifically, a piece of source code. A simple counting of experience atoms in question is then used to determine the experience in that area.

Expert Finder [6] is a tool to locating an expert through the use of existing organizational information. In practice this information is the documentation generated during a project or similar activity. Expertise models are created using text modeling algorithms based on the vector space model to analyze the documents. There are filters that receive the results of analysis and select the people most related with a specific document.

Expertise Recommender [7] uses software artifacts to recommend experts to people asking for help. The tool uses both past interacting between the people in the project and the artifacts history. The recommendation is done after filtering by organizational, social and personal preference criteria.

STeP_IN [5] recommend a set of developers with expertise in a specific method of a Java source code. The set of experts is based on the analysis of the source code and the contacts in the mailbox of each developer. The tool recommends the developers that have some usage of the method, and also recommends archived discussions about a specific piece of source code but did not use the content of the archived discussion to recommend the experts.

None of the approaches indeed analyze the content of the communication among team members of the project which we find could be very valuable. Our approach analysis the content of communication to improve the usual recommendations based on the source code history and the relationship (technical dependency) between the source code.

3. CONSCIUS EXPERT RECOMMENDER SYSTEM

Conscius uses the source code, its history, the project documentation (javadoc) and the developer's mailing list archives to recommend source code experts. Differently of Ye and colleagues, that use the mail archives only to build a social network for each developer and recommend people inside it, our tool analyses the content of each email in the mailing list to identify its subject and related source code.

The mailing list archive is very valuable because the developers tend to write about things they are working on or have knowledge about. Using mining algorithms we can relate the emails to documentation or source code.

From the inputs we build the following relationships: (1) **developer–source code**, from the source code history; (2) **source code – source code**, from the technical dependency between them; (3) **developer – developer**, from the threads of the mailing list; and (4) **documentation – source code**, from the documentation.

The chosen inputs can indicate the knowledge owners from these elements since its inception. The relationship among the projects activities, its artifacts and the communication supply to Conscius the needed information to infer who is the expert in each part (package, class, method) of the software.

3.1 Approach

Instead of write to mailing list seeking for help, the user writes the message in the Conscius tool. Our tool looks for referenced classes in the message and finds classes depending on those.

The Classifier component identifies the keywords in a Javadoc or message ignoring the stopwords and invalid characters [10]. It computes the frequency of each keyword on the text.

The Knowledge component receives the list of keywords from the Classifier and analyzes and associates them with one top-level Javadoc package. This component does automatic classification using the technique of fuzzy similarity described by Galho [11] to determine the score of similarity among the list of keywords from one message and the list of keywords from one Javadoc package. We compute the equity score (*gi*) for each common keyword between the two lists (from the message and from the Javadoc packages) with the Formula 1. The parameters a and b are the frequency of the common keyword in each list. The resulting score must be in the interval [0,1]. Values upper than 1 are normalized to 1; values lower than 0 are normalized to 0.

$$gi(a,b) = \frac{1}{2}\left[(a \rightarrow b) \wedge (b \rightarrow a) + \left(\overline{a} \rightarrow \overline{b}\right) \wedge (\overline{b} \rightarrow \overline{a})\right]$$

Where:

$a \rightarrow b = \max\{gi \in [0,1] \mid a\ x\ c \leq b\}$

$\overline{a} = 1 - a$

$\wedge = min.$

Formula 1. Equity score [11]

After the end of the computation of equity score among the keywords, we compute the similarity score between the message and the top-level Javadoc package with the Formula 2.

$$gs(X,Y) = \sum_{h=1}^{k} gi_h(\mathrm{a},\mathrm{b})$$

Where:

gs is the similarity score between X and Y.

gi is the equity between the weights of *h* keyword (weight *a* for document *X* and *b* for document *Y);*

h is an index for the commons keywords;

k is the total number of common keywords.

Formula 2. Similarity score [11]

This computation is repeated for each top-level Javadoc package in the project. The package with the highest similarity score is chosen. For now on, we are going to refer to this package as the knowledge of the message.

The Recommender component identifies the people most qualified to reply the original message from the user. The component searches the users that sent emails with the requested knowledge (identified in the step above) and computes his communication score. The developer increases his score for each message containing the knowledge. We call this *C Heuristic*.

The next step in the Recommender component is to compute the development score of the developers in each class referenced in the message and each class that depends on them. The developer increases his score for each commit on any these classes in the source code version control system. This as an extension to "Line 10 Rule" of McDonald and Ackerman [16] and we call this heuristic *D Heuristic*.

The final score for each developer is the sum of the communication score and the development score. The developers with highest score are recommended as experts and receive the original message.

4. PRELIMINARY EVALUATION

The experiment planning followed the model proposed by Wohlin and colleagues [12] and adapted by Brito [13]. There are five activities: definition, planning, operation, analysis and interpretation, and presentation.

4.1 Definition

The goal of this experiment is to analyze *the Conscius tool* for the purpose of *evaluating it* with respect to *its recommendation efficiency* of the tool from the point of view of *developers looking for help* in the context of implementation phase of *distributed software development projects*.

4.2 Planning

The purpose of the experiment is to assess the viability to use the project communication to recommend experts for distributed software development. The Apache Lucene-Java was chosen because it's written in Java, is open source software and has its developers spread worldwide.

4.2.1 Subjects

All the developers that sent at least one email to the developer's mailing list of the project.

4.2.2 Hypotheses

Q1. Does the tool recommend efficiently using only the project communication history?

Q2. Does the tool improve the recommendation quality by using the project communication history in addition to source code history?

Three metrics where collected to assess the quality of the recommendations of the tool:

M1. Precision: the number of correct recommendations of the tool divided by the total number of recommendations of the tool [14];

M2. Recall: the number of correct recommendations of the tool divided by the number of people which actually replied the message in the mailing list [14].

M3. Accuracy: the number of the set of recommendations with at least one correct recommendation divided by the total number of the set of recommendations [15].

The **Q1** is answered comparing the values of precision, recall and accuracy from *C Heuristic* and the values of *D Heuristic*. In the same way **Q2** is answered comparing the values of precision, recall and accuracy from *C and D Heuristics combined* and the values of *C Heuristic.*

4.2.3 Independent variables

The independent variables are the tool, the mailing list archive, the source code and the source code history.

4.2.4 Dependent Variable

The dependent variable is the quality of the recommendation of the tool. The quality will be measured through the metrics precision, recall and accuracy.

4.2.5 Internal validity

The mailing list has the problem that not all emails sent are directly related to questions about the implementation nor all repliers are truly experts on the email subject, or the emails do not have enough text to allow the identification of the correct knowledge. This is minimized because we will be using a big quantity of emails (6 months of archives) and this quantity guarantees good internal validity.

4.2.6 External validity

In this experiment the results may be affected by the chosen project because of the nature of the open source software development, where participants are free to contribute whenever they are willing to do so, differently from the enterprise scenario. However the external validity of the study is considered sufficient to assess the quality of the recommendations of the tool. The experiment replication depends only from the necessary inputs.

4.2.7 Conclusion validity

The conclusions will be draw by the use of analysis of precision, recall and accuracy of the recommendations.

4.2.8 Instrumentation

The inputs of the experiment were the developer's mailing list archives, the source code and its history in the versioning control system (VCS), and the Javadoc. There were 6.327 emails sent from 292 different participants in the year 2008. Among the participants 17 of them had at least one commit in the VCS. The VCS had 10.764 records from September 2001 to July 2009.

4.3 Operation

We selected all the threads in the mailing list from January 2009 to June 2009 with at least one reply. The selected threads (375) where grouped by months. All the threads accounts for 1187 emails including questions (the first email in thread) and replies.

In order to also experiment the evolution of the software implementation together with communication made about it, six versions of source code, representing each month above, where extracted from the VCS. In order to Conscius recommend the experts, the questions and the source code are from the same month.

We run three configurations of Conscius varying the enabled heuristics. The first configuration had the two heuristics enabled as described in the section 3.1. The second configuration had the *C Heuristic* disabled and *D Heuristic* enabled, so the communication score was always 0. The third configuration had the *C Heuristic* enabled and *D Heuristic* disabled, so the development score was always zero. In all the runs the tool always recommended three experts.

5. ANALYSIS AND INTERPRETATION

At the end of the experiment execution the recommendations of the tool where collected to compute the three metrics – precision, recall and accuracy.

5.1 Assessing the quality of recommendations for each heuristic

The purpose of this analysis is to assess whether the communication is more relevant than the VCS history for identifying and recommending experts. The table 1 shows the mean values of the metrics collected during the operation.

Table 1 – Comparing the C and D Heuristics

	C Heuristic	D Heuristic
Precision	27,38%	20,71%
Recall	46,06%	33,73%
Accuracy	68,53%	54,13%

All the values of the *C Heuristic* for precision, recall and accuracy are higher than the values of the *D Heuristic*, thus **Q1** is true. A possible explanation is that the *D Heuristic* could be too simple but it demonstrated to be quite competitive in comparison with some others algorithms [17].

5.2 Improving the quality of recommendation

The purpose of this analysis is to assess the strength of combining both heuristics to recommend the experts as described in section 3.1. The table 2 shows the mean values of the metrics collected.

Table 2 – Comparing the combined heuristics against the D Heuristics

	C + D Heuristics	D Heuristic
Precision	27,29%	20,71%
Recall	46,49%	33,73%
Accuracy	69,07%	54,13%

The results for the combined heuristics are also higher than the *D Heuristics* alone, thus **Q2** is also true. Surprisingly the results of the combined heuristics did not improve the results of the *C Heuristic* alone in the previous section as we intuitively expected. We believe this occurs because of the way we combine the scores of *C and D Heuristics*. At least the results did not go worse.

6. CONCLUSION AND FUTURE WORKS

The proposed tool Conscius can find the expert in the source code level from the descriptions of faced difficulties during the implementation of the software, the source code and its history, and mainly from the communication history and its content. We believe that distributed teams can improve their communication and awareness with this tool because they can easily locate the right person to talk.

Our experiment showed evidence of the potential value of communication content as a source to expert location. The results even overcome results relying only on the source code history. We are planning to contact the Apache Lucene-Java's developers to check the results with them, so we can assess their opinions on the quality and utility of the recommendations.

This evidence enables us to generate new hypotheses about the value of the content already available in other textual sources such as bug reports and wikis. Future works also include more exploration on the potential of email communication as a source for expert location, such as using metrics of social network analysis or more elaborated sociotechnical analysis – the social call graph – to improve our heuristics [18].

The current design of the tool only allows the message be classified by one knowledge. We did not have enough time to analyze all 7500 email messages in the experiment but we presume a part of unsuccessful recommendation were because our design limitation. We are also working to improve this area to allow multiple classifications.

7. REFERENCES

[1] Carmel, E. (1999). Global Software Teams: Collaborating Across Borders and Time Zones. Prentice Hall.

[2] Espinosa, A. J. and Carmel, E. 2004. "The Effect of Time Separation on Coordination Costs in Global Software Teams: A Dyad Model". In Proceedings of the Proceedings of the 37th Annual Hawaii International Conference on System Sciences, IEEE Computer Society, Washington, DC, USA.

[3] Espinosa, J., Slaughter, S., Kraut, R., and Herbsleb, J. 2007. Team Knowledge and Coordination in Geographically Distributed Software Development. J. Manage. Inf. Syst. 24, 1 (Jul. 2007), 135-169.

[4] Herbsleb, J. D. (2007). Global software engineering: The future of socio-technical coordination. FOSE '07: 2007 Future of Software Engineering, pages 188–198.

[5] Ye, Y.; Nakakoji, K. and Yamamoto, Y. (2007). "Reducing the cost of communication and coordination in distributed software development", Lecture Notes in Computer Science, 2007, LNCS 4716, 152 – 169.

[6] Sim, Y. and Crowder, R. (2004). "Evaluation of an Approach to Expertise Finding", In PAKM, Seiten 141–152.

[7] McDonald, D. W. and Ackerman, M. S. (2000) "Expertise recommender: a flexible recommendation system and architecture", In CSCW '00: Proceedings of the 2000 ACM conference on Computer supported cooperative work, Seiten 231–240, New York, NY, USA. ACM.

[8] Hogan, B. (2006) "Lessons Learned from an eXtremely Distributed Project", In Proceedings of the conference on AGILE 2006, publisher IEEE Computer Society.

[9] Trindade, C. C., Moraes, A. K. O., and Meira, S. R. L. (2008). Communication in distributed software teams: a systematic review (in Portuguese). In ESELAW '08: Proceedings of the 5th Experimental Software Engineering Latin American Workshop.

[10] Wives, L. K. (1999) Um Estudo sobre Agrupamento de Documentos Textuais em Processamento de Informações não Estruturadas Usando Técnicas de Clustering. Dissertação de Mestrado, PPGC/UFRGS, Porto Alegre (RS).

[11] Galho, T. S.; Moraes, S. M. W. (2004) Categorização Automática de Documentos de Texto utilizando Lógica Difusa. Logos (Rio de Janeiro), Canoas, v. 15, n. 1, p. 91-104, 2004.

[12] Wohlin, C.; Runeson, P.; Host, M.; Ohlsson, C.; Regnell, B. & Wesslén (2000), "A. Experimentation in Software Engineering: an Introduction Kluver Academic Publishers".

[13] Brito, K. S (2007). "LIFT: A Legacy InFormation retrieval Tool Universidade Federal de Pernambuco".

[14] Minto, S. & Murphy, G. C. (2007) "Recommending Emergent Teams", In: ICSEW '07: Proceedings of the 29th International Conference on Software Engineering Workshops, IEEE Computer Society, 2007, 5.

[15] Kagdi, H. H.; Hammad, M. and Maletic, J. I. "Who can help me with this source code change?" ICSM, IEEE, 2008, 157-166.

[16] McDonald, D. W. and Ackerman, M. S. (2000) "Expertise recommender: a flexible recommendation system and architecture", In CSCW ’00: Proceedings of the 2000 ACM conference on Computer supported cooperative work, Seiten 231–240, New York, NY, USA. ACM.

[17] Da Tindade, C. C. 2009. PRESLEY: uma ferramenta de recomendação de especialistas para apoio à colaboração em desenvolvimento distribuído de software. Dissertação de Mestrado, Centro de Informática, Universidade Federal de Pernambuco, Recife, Brasil.

[18] de Souza, C. R., Quirk, S., Trainer, E., and Redmiles, D. F. 2007. Supporting collaborative software development through the visualization of socio-technical dependencies. In Proceedings of the 2007 international ACM Conference on Supporting Group Work (Sanibel Island, Florida, USA, November 04 - 07, 2007). GROUP '07. ACM, New York, NY, 147-156.

[19] Mockus, A. and Herbsleb, J. D. 2002. Expertise browser: a quantitative approach to identifying expertise. In Proceedings of the 24th international Conference on Software Engineering (Orlando, Florida, May 19 - 25, 2002). ICSE '02. ACM, New York, NY, 503-512.