Towards Patterns to Enhance the Communication in Distributed Software Development Environments

Ernst Oberortner, Boston University
Irwin Kwan, Oregon State University
Daniela Damian, University of Victoria

Distributed Software Development (DSD) is an emerging research area in software engineering. Several conducted research studies identified similar communication problems among DSD teams and tried to solve them. In this paper we present patterns that we have identified while surveying state of the art research studies. The patterns can help to organize DSD teams better in order to enhance their communication. We also highlight some potential future research challenges.

Categories and Subject Descriptors: K.4.3 [**Organizational Impacts**] Computer-supported collaborative work

Additional Key Words and Phrases: Collaborative Software Development, Pattern

ACM Reference Format:

Oberortner, E. and Kwan, I. and Damian, D. 2011. 18th Conference on Pattern Languages of Programs (PLoP), Portland, Oregon, USA (October 2011), 8 pages.

1. INTRODUCTION

Nowadays, several development teams of large software houses are distributed globally [Hildenbrand et al. 2008; Lanubile 2009]. The collaboration among distributed software development (DSD) teams is based upon the team members' communication. The better the communication, the better the collaboration, and the better the success of the resultant software. DSD teams communicate about various aspects of the intended software. For example, the software's requirements must be defined, updated, and clarified. Technical utilities, such as mailing lists, online forums, software repositories, or bug tracking systems, facilitate the communication during the whole software development process.

The literature consists of many studies and tools to enhance the communication of DSD teams, such as [Damian et al. 2010; Hinds and McGrath 2006; de Souza and Redmiles 2011]. Many studies use data mining techniques [Han and Kamber 2005] and utilize social network analysis methods [Wasserman and Faust 1994] to discover the communication structures of DSD teams. Mining the data of communication and software repositories can help, for example, to leverage invisible relations between globally distributed team members. Another example is to support newcomers, helping them to collaborate with task-related experienced team members in order to become productive more quickly.

Author's address: E. Oberortner, Boston University; email: ernstl@bu.edu; I. Kwan, Oregon State University; email: irwink@uvic.ca; D. Damian, University of Victoria; email: danielad@uvic.ca;

We present patterns to improve the communication within DSD teams. The patterns describe (1) how to organize co-located teams to enhance the communication with remote teams and (2) how to support the DSD team members to find other DSD team members with some wanted professional skills.

Agile software development is a software development process that is mostly utilized in DSD teams [Eckstein 2010]. In this paper, we do not focus on the software development process that DSD teams utilize. However, the communication during the whole software development process is an important aspect. The patterns presented in this paper can improve the communication independent of the utilized development process.

This paper is structured as follows: In Section 2 we justify the importance of the communication within DSD teams with an example. Section 3 explains the pattern's common terminology, the utilized pattern format, and the paper's main contribution, patterns to enhance the communication of DSD teams. In Section 4 we discuss the relationship between the patterns. In Section 5 we highlight some potential future research challenges for DSD environments. The paper concludes with Section 6.

2. MOTIVATING EXAMPLE

In this section we justify the importance of the presented patterns to enhance the communication within DSD teams. Imagine a DSD development team, as exemplified in 1, that is globally organized. The illustrated DSD team consists of one software designer and three software developers.

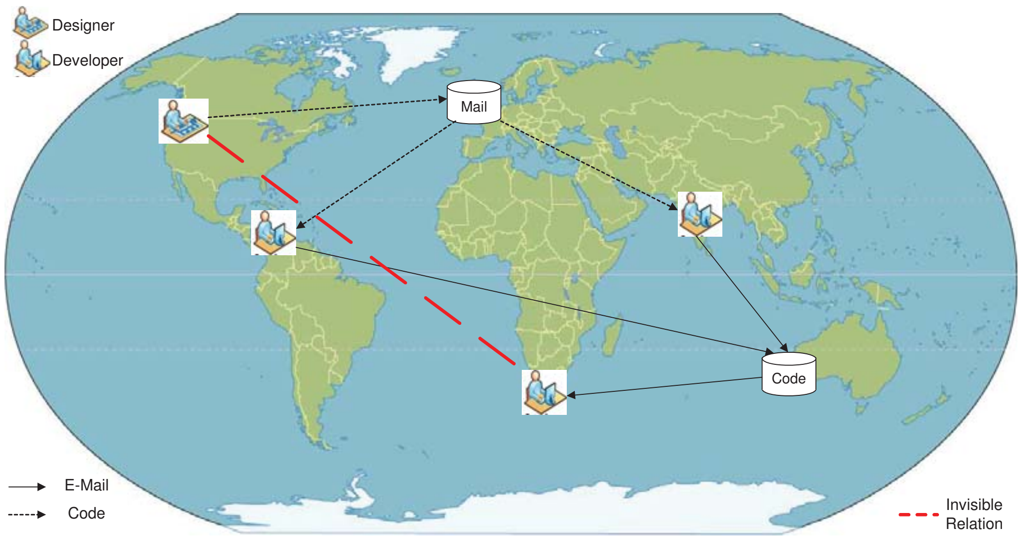

Fig. 1. A motivating example

For example, a software designer, located in North America, propagates design updates to, in his opinion, related software developers via e-mail. The developers implement the required software updates and submit the new code into the source code repository. All developers receive a notification about the updated code, enabled by a special feature of the source code repository.

As illustrated, the software designer in North America does not directly communicate with the one developer located in South Africa. The developer in South Africa recognizes software updates through the recently submitted code into the software repository. It can happen that the new code is inconsistent with the code of the developer in South Africa. Such situations occur often in DSD teams because there exist invisible dependencies between the DSD team members.

In the illustrated scenario, mining only the mailing list or just the source code repository is not sufficient because the invisible dependency cannot be uncovered from just mining the mail repository. Only a combination of mining the mail and the source code repository's data identifies the invisible dependency. Detecting invisible dependencies can enhance future communications between the globally distributed software designer and the developers.

3. PATTERNS TO ENHANCE TO COMMUNICATION OF DSD TEAMS

This section covers the main contributions of this proceeding. First, we define the patterns terminology and present the utilized pattern format. Then, we present two patterns that cover various aspects to enhance a DSD team's communication.

3.1 The Patterns' Terminology

The survey's studies use various terms that have a different naming but common meaning among all studies. In this section, we explain the survey's common terms in order to improve the understandability and to avoid misunderstandings.

The term **distributed software development** (DSD) elates to a software development strategy where the team members are distributed on various geographical locations. The team members are located at various **sites**. As illustrated in the motivating example (see Section 2), the designer is located in North America, whereas the developers are located in Asia, South Africa, and South America. In this case, North America, Asia, South Africa, and South America are the sites of the DSD team. Usually, each side consists of a group of team members. We term team members that reside on the same side as **co-located** team members, whereas team members that reside on different sites are **remote** team members.

All team members must **collaborate** with each other during the whole software development process in order to gather the requirements correctly, to design, develop, and test the software. The DSD team members use existing **technical communication facilities** to collaborate with each other, such as mailing lists, software repositories, bug tracking systems, or instant messengers. Such communication facilities can be used, for example, to coordinate task-related activities between the DSD team members and to propagate information to appropriate DSD team members. Similar to Damian et al. [Damian et al. 2007], we base the definition of **awareness** on Dourish and Bellotti [Dourish and Bellotti 1992]: Awareness is *an understanding of the activities of others, which provides a context for one's own activities*.

3.2 The Utilized Patterns Format

To describe the patterns, we are using a pattern format that is a subset of the pattern format utilized in the GoF book [Gamma et al. 1995].

Problem:	We utilize a driving question in order to explain the pattern's problem.
Forces:	Regarding the Language of Shepherding [Harrison 1999], forces drive the problem. In our pattern format, forces explain and motivate the problem in more detail. We confirm the problem with papers discovered during a literature review.
Solution:	In the solution section we explain the pattern's solution to the problem. We also try to visualize to pattern's solution graphically.
Consequences and Resulting Questions:	A pattern's consequences are the side effects and trade-offs of applying the pattern. Consequences describe arising questions in case of utilizing the pattern.
Related Patterns:	In this section we list similar patterns that we have discovered in the literature.
Known Uses:	In the Known Uses section we list several studies that utilize the pattern or that have discovered the importance of the pattern.

Following Alexander [Alexander et al. 1977], we annotate each pattern with one or two asterisks. One asterisk denotes that the pattern is subject to further investigations. Based on our observations and findings during the literature survey, patterns annotated with two asterisks are valid.

3.3 Pattern: COMMUNICATION BROKER**

How to enhance the communication between remote teams?

In DSD teams, challenges like miscommunication, misunderstandings, and how to share information are likely more severe than in co-located teams because of communication problems [HM06]. Also, in DSD environments the communication between co-located team members is more efficient than between remote team members. DSD teams are volatile and hence, the team members change frequently [DKM10].

Identify within each co-located team one COMMUNICATION BROKER that is responsible for the communication with the remote teams.

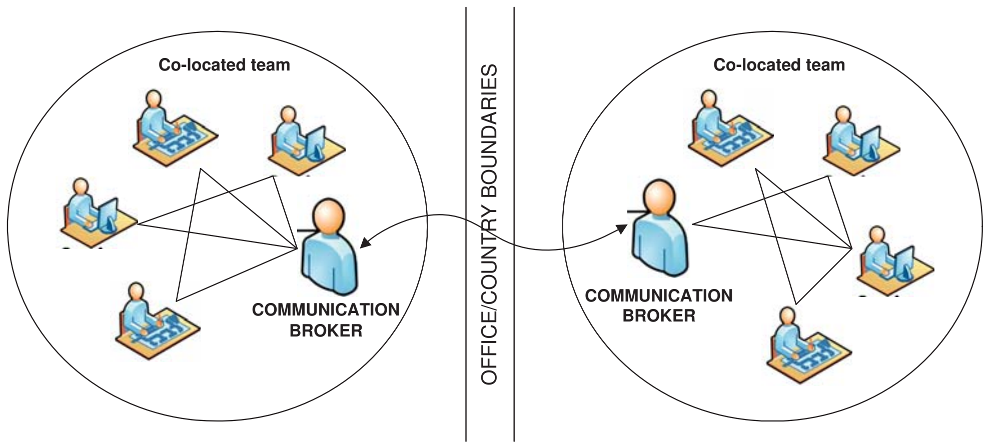

Fig. 2. COMMUNICATION-BROKER

In Figure 2 we visualize the COMMUNICATION BROKER pattern graphically. Within each co-located DSD team, one team member must be identified that is responsible for the communication with the remote teams and their team members. The COMMUNICATION BROKER observes the communication between its' co-located team members and propagates important issues, such as updates, changes, or extensions of the software, to the remote teams and their COMMUNICATION BROKER. The COMMUNICATION BROKER receives the important issues and propagates them to its' co-located team members.

One benefit of identifying COMMUNICATION BROKERS is that within each co-located team there is one central switching point that has knowledge about the communication within its' co-located team. The COMMUNICATION BROKER is mostly personally known within each co-located team, resulting in a trusted relationship between the COMMUNICATION BROKER and its' co-located team members. Because the COMMUNICATION BROKER communicates with remotely located COMMUNICATION BROKERS, a personal relationship evolves that can result in a trusted communication between the COMMUNICATION BROKERS. Because of the personal and trusted relationships between the COMMUNICATION BROKERS the communication can get enhanced.

COMMUNICATION BROKERS can have an information overflow of the communicated issues within and across co-located team members. Furthermore, the question about how to find appropriate COMMUNICATION BROKERS must be answered. This question relates to the DISCOVERING THE EXPERTS patterns (see Section 5.2). A further relevant question concentrates on the dynamic of DSD teams [DKM10]. What happens if the COMMUNICATION BROKER leaves the DSD team or the organization?

Related Patterns:

—Scott et al. introduce the FACADE pattern that is similar to the COMMUNICATION BROKER pattern [Scott et al. 2005].

—Each small team of should have a SCRUM MASTER [Rising and Janoff 2000] of an agile software development process is comparable to the COMMUNICATION-BROKER pattern. A SCRUM MASTER is responsible for coordinating the team and the communication within the team, but, a SCRUM MASTER is not a team leader.

Known Uses:

—Wolf et al. follow a three step filtering process to build a task-based social network [Wolf et al. 2009]. The authors use the filtering process to identify COMMUNICATION BROKER in order to discover which team members contributed to a build failure.

—During a conducted web-based survey within R&D DSD teams, Hinds and McGrath [Hinds and McGrath 2006] identified that a COMMUNICATION BROKER is necessary.

3.4 Pattern: AWARENESS NETWORK**

How to find DSD team members with a required expertise and professional skills?

At least 70% of development time is spent on communication [DeMarco and Lister 1999]. Finding appropriate experts in a globally DSD team to communicate with to gather additional experience, is a problem [Mockus and Herbsleb 2002]. Team members who recently joined the DSD team are not aware of the other team members' professional background, skills, activities and tasks [Ehrlich and Chang 2006]. The network of assigned tasks differs from the social interactions within the DSD team [Damian et al. 2007]. To become productive more quickly, newcomers must be aware of whom to contact in case they have some task-specific questions [de Souza and Redmiles 2011].

Construct an AWARENESS NETWORK, i.e., a social network mined from several project repositories, which reflects the DSD team members' communication content.

In Figure 3 we sketch how to construct an AWARENESS NETWORK by mining multiple repositories, such as mailing lists, source code repositories, or bug tracking repositories. It is particularly important to gather the topic of the DSD team members' interactions in order to discover the team members' professional skills and in which interactions they participated. For example, if a developer that submits frequently to the source code repository new code for the database access and participates frequently in database topics in the mailing lists, then it is highly possible that this developer is an expert in database development. The resultant AWARENESS NETWORK should be accessible to all DSD team members, making it possible to search for team members with a desired expertise.

An AWARENESS NETWORK can speed up the communication within the DSD team. Newcomers can increase their productivity because they immediately know whom to contact regarding which questions. Invisible relationships

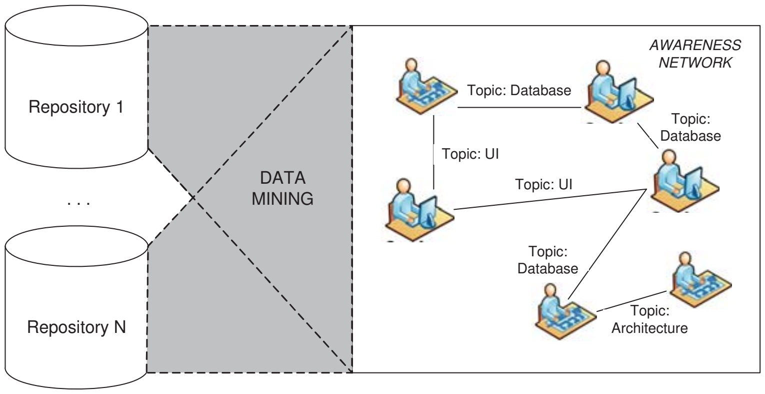

Fig. 3. AWARENESS-NETWORK

can be detected because the AWARENESS NETWORK differs from the network of assigned tasks to the DSD team members.

Mining the data of multiple repositories is necessary because mining just one repository can lead to an inaccurate AWARENESS NETWORK [Hossain et al. 2006]. Mining historical data can limit the AWARENESS NETWORK's accuracy. Therefore, an accurate and current status of work must be provided [Damian et al. 2007]. A permanent update of the AWARENESS NETWORK is needed, but keeping the AWARENESS NETWORK up-to-date is challenging because DSD teams are volatile and the team members change their roles or leave the team frequently [de Souza and Redmiles 2011]. Redmiles and deSouza [de Souza and Redmiles 2011] have discovered three factors that influence the awareness network: (1) the organizational reuse program, (2) the software developers' experience, and (3) the software architecture.

Known Uses:

—Hossain et al. [Hossain et al. 2006] mined the e-mail repository and performed a text-based keyword finding to construct the AWARENESS NETWORK. A limitation is that mining one repository is not sufficient. Also the authors state that a context-specific search would result in a more accurate AWARENESS NETWORK.

—The Expertise Browser [Mockus and Herbsleb 2002] is a web-based tool to support DSD team members finding experts to talk to. The tool mines a change management system and visualizes project-related expertise information.

—The Hipikat Tool [Cubranic and Murphy 2003] mines some project's source and mail repositories to support newcomers to find relevant solutions for their tasks.

4. DISCUSSION

In DSD teams, many challenges and problems exist that usually do not arise in co-located software development teams. In this section, we discuss the patterns and explain their relationships.

There exist a relationship between the COMMUNICATION BROKER and the AWARENESS NETWORK pattern. The observations by Hossain et al. [Hossain et al. 2006] are in accord with Conway's Law [Conway 1968]. In hierarchically organized companies it means that the higher a team member is in the organization's hierarchy, the higher the possibility to select this team member as the COMMUNICATION BROKER. To avoid an information overload of the COMMUNICATION BROKER, it is important to think about whom to identify as the COMMUNICATION BROKER. Furthermore, it can be possible to identify multiple COMMUNICATION BROKERS within the co-located teams that are responsible for the communication that covers the COMMUNICATION BROKER's working activities, tasks, or professional skills. Hence, an AWARENESS NETWORK of a co-located team can offer valuable clues to identify COMMUNICATION BROKERS.

DSD teams utilize various repositories, such as communication, source code, documentation, and bug tracking. Communication repositories, such as mailing lists or online forums, support DSD teams in order to define, clarify and modify the software's requirements. Source code repositories, such as concurrent version control (CVS) or Subversion (SVN), are utilized to version the software's source code. Documentation repositories keep track about a project's relevant documents, such as architecture documents, written deliverables, or change requests. Bug tracking repositories, such as Bugzilla, give information about identified and detected software bugs. According [Hassan 2008], mining just one repository is not sufficient to enhance the communication of DSD teams.

5. FUTURE RESEARCH CHALLENGES

Nowadays, model-driven development (MDD) is a popular paradigm in software development. Models can be used to define the software's requirements and technical artifacts in a platform-independent way, making it possible to generate recurring and schematic parts of the software automatically [Schmidt 2006]. Model repositories, such as EMFStore [Koegel and Helming 2010] or MORSE [Holmes et al. 2009], store and version the software's models. To the best of our knowledge, there exist no approaches yet that mine model repositories to support the DSD team communication. In our opinion, potential future research challenges exist to (1) mine model repositories and (2) to combine the mining results with the mining results of, for example, communication, source code, documentation, or bug tracking repositories.

A further identified research challenge deals with the broad variety of the repositories' data schemes. For example, the data schema of communication repositories differs from the data scheme of a bug repository. Many existing data mining approaches utilize the repositories' meta-data to construct, for example, social networks. But, just mining the meta-data is not sufficient to support the DSD team communication. Furthermore, the implementations of the various repository tools differ. For example, the CVS source code repository stores the data differently than the SVN source code repository. We ask if it is possible to define or standardize one universal meta-data and data scheme to ease the mining to improve the DSD teams' communication?

During physical DSD team meetings, the attendees formulate meeting minutes. Mostly, these do not cover all discussions during the meetings and are difficult to propagate to the team members who did not attend the meeting. Furthermore, personal communication among co-located team members, such as during coffee breaks, at the hallway, or just in the office, are difficult to propagate to the other co-located team members and especially to the remote team members. An important question is how to catch such conversations and how to propagate the information to the appropriate co-located and remote team members?

6. CONCLUSION

Communication in distributed software development (DSD) teams is challenging because several questions arise that usually do not arise in co-located software development teams. An efficient communication within DSD teams implies bigger success of the developed software. In this paper, we present two patterns to enhance the DSD team communication. We mined the patterns from surveying state of the art approaches published at several conferences and books that focus on DSD.

The COMMUNICATION BROKER pattern can improve the communication between co-located and remote DSD teams. The AWARENESS NETWORK pattern helps to find appropriate DSD team members with some desired professional skills. Both patterns can help to organize DSD teams better in order to enhance collaborative software development.

Acknowledgements

We want to thank Lise Hvatum. Lise gave us great suggestions to define and to improve the patterns and the paper.

REFERENCES

ALEXANDER, C., ISHIKAWA, S., SILVERSTEIN, M., JACOBSON, M., FIKSDAHL-KING, I., AND ANGEL, S. 1977. *A Pattern Language: Towns, Buildings, Construction*. Oxford University Press.

CONWAY, M. 1968. How do committees invent. *Datamation 14,* 4, 28–31.

CUBRANIC, D. AND MURPHY, G. C. 2003. Hipikat: Recommending pertinent software development artifacts. *International Conference on Software Engineering 0*, 408.

DAMIAN, D., IZQUIERDO, L., SINGER, J., AND KWAN, I. 2007. Awareness in the wild: Why communication breakdowns occur. In *Second IEEE International Conference on Global Software Engineering (ICGSE)*. 81 –90.

DAMIAN, D., KWAN, I., AND MARCZAK, S. 2010. Requirements-driven collaboration: Leveraging the invisible relationships between requirements and people. In *Collaborative Software Engineering*. Springer Berlin Heidelberg, 57–76. 10.1007/978-3-642-10294-3_3.

DE SOUZA, C. R. AND REDMILES, D. F. 2011. The awareness network, to whom should i display my actions? and, whose actions should i monitor? *IEEE Transactions on Software Engineering 37*, 325–340.

DEMARCO, T. AND LISTER, T. 1999. *Peopleware: Productive Projects and Teams (Second Edition)*. Dorset House Publishing Company, Incorporated.

DOURISH, P. AND BELLOTTI, V. 1992. Awareness and coordination in shared workspaces. In *Proceedings of the 1992 ACM conference on Computer-supported cooperative work*. CSCW '92. ACM, New York, NY, USA, 107–114.

ECKSTEIN, J. 2010. *Agile Software Development with Distributed Teams: Staying Agile in a Global World*. Dorset House Publishing.

EHRLICH, K. AND CHANG, K. 2006. Leveraging expertise in global software teams: Going outside boundaries. In *Proceedings of the IEEE international conference on Global Software Engineering*. ICGSE '06. IEEE Computer Society, Washington, DC, USA, 149–158.

GAMMA, E., HELM, R., JOHNSON, R., AND VLISSIDES, J. 1995. *Design patterns: elements of reusable object-oriented software.* Addison-Wesley Professional.

HAN, J. AND KAMBER, M. 2005. *Data Mining: Concepts and Techniques.* Morgan Kaufmann Publishers Inc., San Francisco, CA, USA.

HARRISON, N. 1999. The language of shepherding. `http://hillside.net/index.php/the-language-of-shepherding`.

HASSAN, A. E. 2008. The road ahead for Mining Software Repositories. In *Proc. FoSM 2008. Frontiers of Software Maintenance*. 48–57.

HILDENBRAND, T., ROTHLAUF, F., GEISSER, M., HEINZL, A., AND KUDE, T. 2008. *Approaches to Collaborative Software Development*. IEEE, 523–528.

HINDS, P. AND MCGRATH, C. 2006. Structures that work: social structure, work structure and coordination ease in geographically distributed teams. In *Proceedings of the 2006 20th anniversary conference on Computer supported cooperative work*. CSCW '06. ACM, New York, NY, USA, 343–352.

HOLMES, T., ZDUN, U., AND DUSTDAR, S. 2009. Morse: A model-aware service environment. In *Services Computing Conference, 2009. APSCC 2009. IEEE Asia-Pacific*. 470 –477.

HOSSAIN, L., WU, A., AND CHUNG, K. K. S. 2006. Actor centrality correlates to project based coordination. In *Proceedings of the 2006 20th anniversary conference on Computer supported cooperative work*. CSCW '06. ACM, New York, NY, USA, 363–372.

KOEGEL, M. AND HELMING, J. 2010. Emfstore: a model repository for emf models. In *Software Engineering, 2010 ACM/IEEE 32nd International Conference on*. Vol. 2. 307 –308.

LANUBILE, F. 2009. Collaboration in distributed software development. *Lecture Notes in Computer Science including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics 5413*, 174–193.

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.* ICSE '02. ACM, New York, NY, USA, 503–512.

RISING, L. AND JANOFF, N. S. 2000. The scrum software development process for small teams. *IEEE Softw. 17,* 4, 26–32.

SCHMIDT, D. C. 2006. Guest editor's introduction: Model-driven engineering. *Computer 39*, 25–31.

SCOTT, A., IZQUIERDO, L., GUPTA, S., ELVES, R., AND DAMIAN, D. 2005. Leveraging design patterns in global software development. In *International Workshop on Distributed Software Development*.

WASSERMAN, S. AND FAUST, K. 1994. *Social Network Analysis: Methods and Applications*. Vol. 24. Cambridge University Press.

WOLF, T., SCHRÖTER, A., DAMIAN, D., PANJER, L. D., AND NGUYEN, T. H. D. 2009. Mining task-based social networks to explore collaboration in software teams. *IEEE Softw. 26,* 1, 58–66.