Presented at the ACM CSCW Workshop on Supporting the Social Side of Large-Scale Software Development, Banff, Alberta, Canada, November, 2006.

Supporting Software Development as Knowledge Community Evolution

Kumiyo Nakakoji[1,2]
[1]RCAST, University of Tokyo
4-6-1 Komaba
Meguro, Tokyo, 153-8904, Japan
kumiyo@kid.rcast.u-tokyo.ac.jp

Yasuhiro Yamamoto[1]
[2]SRA-KTL Inc.
3-12 Yotsuya
Shinjyuku, Tokyo, 160-0004, Japan
yxy@kid.rcast.u-tokyo.ac.jp

Yunwen Ye[2,3]
[3]Dept. of Computer Science
University of Colorado, Boulder
CB430 Boulder, CO. 80303
yunwen@cs.colorado.edu

ABSTRACT
We view software project as a knowledge ecology consisting of three interrelated elements: (1) artifacts, (2) individual developers, and (3) a community of developers. How developers relate with each other in the community affects how they share knowledge during the development and therefore impacts the overall quality of the software system that have to be built through continuous knowledge collaboration. This paper analyzes this social relation and its impacts on software development, and presents an approach to help developers make use of peer expertise by asking and helping other developers. It then describes the STeP_IN (Socio-Technical Platform for In situ Networking) framework to illustrate the approach.

Keywords
Knowledge collaboration, knowledge community, software development, reuse, community, socially aware communication, socio-technical approach

1. SOFTWARE DEVELOPMENT AS KNOWLEDGE COLLABORATION
The development of large-scale software systems is a social activity, carried out through the collaboration by a group of software developers. The social aspects of software development have been studied mostly in the context of how developers and users work together in designing systems [22], in the organizational context of a software project [17], or in distributed software development teams [8]. This position paper in contrast focuses on the knowledge collaboration of software developers: how developers can make use of peer expertise in collectively creating the software system.

Software development is essentially a knowledge construction process that needs knowledge in a variety of fields, which is constantly changing. For example, application domains are subject to rapid change; component libraries are continually updated; new features and functionalities continue to be introduced in programming tools and environments. Software development can therefore be viewed as a learning process and software developers have to constantly acquire new knowledge.

It may come as a surprise that software developers also need to learn about the system that they are developing. One may argue that since the software developer participates in the creation of the system, he/she should know the system inside out. However, because large scale software systems are created collaboratively by many developers, not all developers, if any, would have complete knowledge about the whole system. At the same time, With the increasingly widely accepted view of software systems as evolving entities, the percentage of incremental, continuous development tasks in software development has risen quickly. Such software systems need to be continuously developed with iterative processes to adapt to the ever-changing user requirements and execution environments. Coupled with the high turnover rate in software industry, many software developers find themselves working to make incremental changes to systems that have been partially developed, or even are operating on a daily base (such as those web-based systems) .

For software developers, software code is the ultimate knowledge resource about the system. During the development process, they intensively engage in recovering "implicit knowledge" embedded within the code [11]. Due to the essential invisibility of software code, however, the needs of creating documents that provide high level descriptions of the code and the design rationale have been recognized.

Code and documents, however, are often still not enough. Documents often do not exist or are not in sync with the code. Moreover, a culture exists in software development that prevents developers from sharing knowledge over the entire source code. As LaToza et al. observed, "implicit knowledge retention is made possible by a strong, yet often implicit, sense of code ownership, the practice of a developer or a team being responsible for fixing bugs and writing new features in a well defined section of code" [11]. Much of the knowledge about the code and the design decisions remain in the head of developers. This "symmetry of ignorance" [4] within a development team is neither a problem nor an accident; it is a matter of fact in software development,

Supporting knowledge collaboration among software developers thus becomes an important research topic in supporting software development. This paper first conceptualizes software project as a knowledge ecology that has intertwined and dynamically changing relationships among software artifacts (code and documents), software developers, and developer community, followed by the analysis of social factors in supporting knowledge collaboration in software development based on this conceptualization. Finally, the paper describes the STeP_IN (Socio-Technical Platform for In situ Networking) framework that supports knowledge collaboration in software development by taking into full consideration those identified social factors.

2. THREE ELEMENTS IN SOFTWARE DEVELOPMENT
We view software project as a knowledge ecology that consists of three interrelated elements: (1) artifacts, (2) individual developers, and (3) a community of developers (Figure 1). A group of

developers engaging in software development can be viewed as a knowledge community, defined as a group of people who collaborate with one another for the construction of artifacts of lasting value [2]. In a knowledge community, people are bonded through the construction of common artifacts.

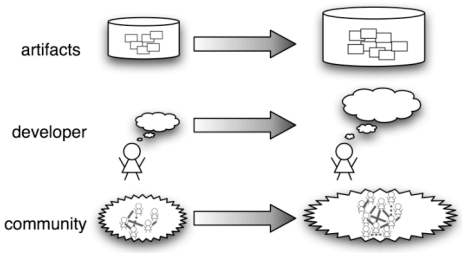

Figure 1: Software Project as a Knowledge Ecology Consisted of Three Interrelated Elements

The community element is essential when viewing software development as collective creative knowledge work. The roles of individual developers, both formally assigned ones and informally perceived ones, change over time during a project. The social relationships among the developers grow through their engagement in the project, affecting how they collaborate, communicate, and coordinate with one another, which results in different ways of sharing knowledge.

Because knowledge sharing is indispensable in software development, the quality of the resulting software depends not only on the skills and knowledge of individual developers, but also on the roles and social relationships among the developers. In other words, the quality of the software to be developed is determined not only by the sum of each developer's knowledge, but also on the social relationships of software developers that impacts the sharing of knowledge during the development process.

All three elements constantly evolve during the process of software development. Artifacts change over time throughout the development. Individual developers—or, more precisely, what individual developers know—grow by gaining experience through the engagement with artifacts and peer developers. The community of developers changes when new developers join, old developers leave, and both the assigned and perceived roles of members change.

Existing studies on understanding and supporting software evolution have primarily focused on the evolution of artifacts. More recent work has started to look at how individuals change through learning about the system. People learn by reading source code and documents, and they learn by asking peers questions. They also learn by solving new problems and experiencing unfamiliar situations. Their old knowledge is replaced with new knowledge and is restructured during the development process.

In contrast, not much has been studied on the aspects of the evolution of the developer community in the context of software development [15]. A community evolves through individual activities in software development that result in either the change of software artifacts or the individual growth of knowledge about the system. This paper views the evoluationary process of a community from the following three relationships (Figure 2).

(1) *The relationship of an individual with artifacts.* How one relates with artifacts is concerned with what knowledge, expertise, and experience the individual has on what artifacts. This information is useful in identifying a set of people who are likely to have expertise with a certain artifact.

(2) *The relationship of an individual with other developers.* How one relates with other individuals impacts social relationships among developers. This information helps a developer determine whom to ask for help about a certain artifact as well as decide whether and how to respond to a question being posed by an asker (Figure 3).

(3) *The relationship of an individual with the community as a whole.* How one relates to the community is concerned with that individual's role within the community: whether he/she is a peripheral member, a core member, or a member in between. This relationship helps a developer decide how much he/she should contribute to the community by gaining trust and social reputation within the community. One's role evolves within a community through legitimate peripheral participation [21]. By looking at how and what a developer's peers who are closer to the core of the community do within the community, the developer gradually acquires skills through learning, and develops his/her identity within the community.

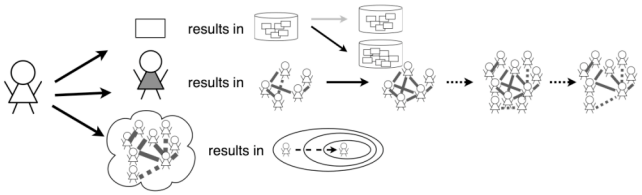

Figure 2: Three Aspects of the Community's Evolutionary Process

3. SOCIAL FACTORS IN SOFTWARE DEVELOPMENT

To support software project as a knowledge ecology that consists of the interrelationship among software artifacts, individual developers, and developer community, we have focused on the following aspect: how to help developers make use of peer expertise in development activities.

A number of researchers have already recognized the needs of using the expertise of other software developers. Berlin has found that expert developers are experts not only because they have more expertise but are able to use other experts more [1]. Several systems, notably Expertise Recommender [12] and Expertise Browser [13] that help software developers to find experts, have been proposed in the past years.

Finding experts, however, does not necessarily lead to the acquisition of their expertise [23]. As knowledge resources, experts are different from other resources that are things. "A thing is available at the bidding of the user—or could be—whereas a person formally becomes a skill resource only when he consents to do so, and he can also restrict time, place, and method as he chooses" [9].

Thus, when peers' expertise becomes critical resources for a programming task, simply knowing who has the expertise is not enough. The expertise seeker (i.e., asker) needs to establish a communication channel with the potential expertise providers (i.e., helpers) and asks the question. The expertise providers have to consent to engage in the communication with the asker to share their expertise. The communication channels used, the contents of the question and answer, the ways the questions is asked and the

answers provided, as well as the timings of questioning and asking depend on a set of perceived social variables.

Awareness. Because asking a question implies that the asker is missing some knowledge, the asker needs to take a risk of looking ignorant. Studies show that askers demonstrate different asking behaviors when they are in public or in private or communicating with a stranger or with a friend due to the different levels of feeling psychological safety of admitting the lack of knowledge [3]. Research has also shown that previous social interactions between an asker and a helper leads to easier quality judgment, and helps the interpretation of answers [10].

Access. Social factors in accessing expertise from peers include how and when an asker asks for help from a potential helper. A study has concluded that collocated developers feel socially comfortable to initiate contact because they know each other, know how to approach them, and have a good sense of how important their question is related to what the experts seem to be doing at the moment [8]. Such social cues are heavily used in face-to-face communication through informal interruptions among software development project team members [11]. Rhetorical strategies, linguistic complexity and word choice of the question all influence the likelihood of others responding to a question [10]. Making a personal appeal (e.g. "I need help") in the question results in better and faster responses than making non-personal appeals (e.g. "I have a problem that might be of interest to you") [3]. The expectation of how soon a help would come has been found to be shaped by the history of interactions with the other party [20].

Interruption. Answering, or providing help, consumes the time and attention of the helpers and interrupts their primary task. An interruption is regarded as an unexpected encounter initiated by another person, that disturbs "the flow and continuity of an individual's work and brings that work to a temporary halt to the one who is interrupted" [19].

Collective attention cost. In addition to the cost of the helpers, considerable collective cost could also be incurred. Mailing lists have been heavily used as a means for mediating peer-to-peer knowledge sharing in software development. All the people who have received the question through a mailing list would at least spend some attention about the question before they decide not to answer. When the number of people who receives the question becomes large, the collective attention consumed also becomes considerably large. Attention is quickly becoming the scarcest resource in our society [7].

Social capital. Upon receiving a question, the expert developers need to decide whether and how to engage in collaboration with the asker by expending their precious time and contributing their expertise. This decision is primarily based on their perceived social relationship both with the asker and with the social environment at large. The theory of social capital provides an analytic framework to understand this decision-making process [5]. Social capital is the "sum of the actual and potential resources embedded within, available through, and derived from the network of relationships possessed by an individual or social unit" [14]. It is regarded as important as financial capital and intellectual capital for an individual as well as a social organization because it would promote cooperation and reduce transaction cost [6]. While helping is costly, taking no action also incurs social cost. Saying "no" untactfully to an asker deteriorates the expert's relation with the asker, and affects negatively the expert's social reputation among other peers because it deviates from social norms [18].

4. AN APPOACH: STeP_IN

We have developed the STeP_IN (Socio-Technical Platform for In situ Networking) framework to help developers to use peer expertise based on the above considerations [23]. The goal of STeP_IN is twofold: (1) to increase the ease of accessing peer experts by asking questions, and at the same time (2) to reduce the total cost of experts being interrupted and that of providing help. We try to achieve this goal by creating an ephemeral knowledge network, called a Dynamic Community (DynC) to connect an expertise seeking developer with other developers who have not only technical expertise but also good social relations with the expertise seeker, and support their collaboration with socially aware communication mechanisms.

STeP_IN presupposes a knowledge workspace, which consists of a group of developers, artifacts (their code and related documents), and the three types of relations among them (Figure 3): artifact-artifact, developer-artifact (a developer's *technical profiles*), and developer-developer (a developer's *social profile*). The framework uses those relations to retrieve relevant artifacts for a developer's task at hand, and then to create a DynC for the developer first by identifying experts for the task, and then by selecting experts based on the developer's social profile.

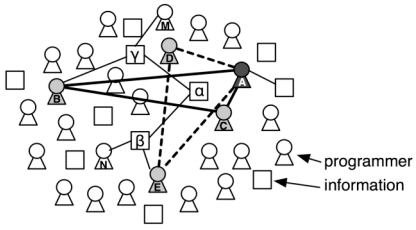

Figure 3: Knowledge Workspace and Relations in STeP_IN

The framework is instantiated in Step_IN_Java (SIJ) for supporting Java developers (see Figure 4) [23]. In SIJ, a Java developer can (1) search for methods, (2) read documents and examples, and (3) ask questions about a specific method to selected experts through the formation of a DynC. See [23] for more details.

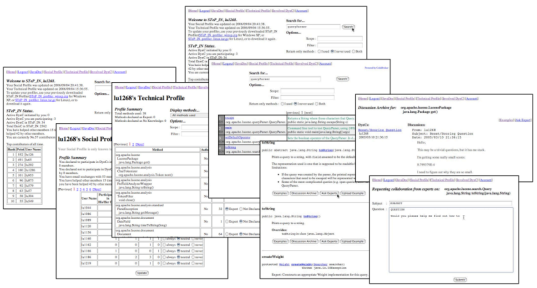

Figure 4: STeP_IN_Java

By using SIJ, developers do not need to have the awareness of who are the experts for the problem that he/she has in seeking for peer expertise. Potential shame of ignorance in asking a question is reduced because only experts with established good relationships are selected. The established social relationships also increase the likelihood for the asker to obtain timely responses because such social relationships are likely to motivate the experts to actively engage in communications with the asker.

A DynC in SIJ complies with the principle of asymmetrical disclosure of information. The membership is not revealed unless one explicitly posts a reply to the DynC. A member, therefore, may leave the DynC (a social equivalent of saying "no") at any moment without being publicly known. Due to this principle, no participation does not constitute the violation of social norms, which is punishable by the "iron hand of social pressure" of enforcing required individual behavior in a social unit [18]. On the other side, because replying to the DynC reveals the identity of the sender of the message, the DynC members' contribution is publicly acknowledged and can lead to the improvement of motivation [5].

This socially aware mechanism that allows unwilling peer developers exit socially safely has two implications. The remaining peers are the participants of willing, and hence the expertise sharing becomes more effective. From the perspective of the asker, knowing that other developers could easily exit, he/she feels less pressured to post a question because the availability is controlled by the experts.

Unlike a mailing list, because questions are only sent to DynC members, other developers who have neither interest nor expertise on the topic are not disturbed. The collective cost of attention and interruption is reduced by the reduction of the number of receivers.

5. Summary

This paper analyzed the social factors that affect the knowledge sharing practice during the software development from the perspective of viewing software project as evolving knowledge ecology. The STeP_IN framework was described to support the use of peer expertise with socially aware mechanisms. The framework was illustrated in the SIJ system that supports knowledge collaboration among Java developers.

6. REFERENCES

[1] L.M. Berlin, "Beyond Program Understanding: A Look at Programming Expertise in Industry," in Empirical Studies of Programmers: Fifth Workshop, C.R. Cook, J.C. Scholtz, and J.C. Spohrer, Eds. Palo Alto, CA: Ablex Publishing Corporation, 1993, pp. 6-25.

[2] Cosley, D., Frankowski, D., Terveen, L., Riedl, J., Using Intelligent Task Routing and Contribution Review to Help Communities Build Artifacts of Lasting Value, Proc. CHI06, ACM Press, pp. 1037-1046, 2006.

[3] R. Cross and S.P. Borgatti, "The Ties That Share: Relational Characteristics That Facilitate Information Seeking," in Social Capital and Information Technology, M. Huysman and V. Wulf, Eds. Cambridge, MA: The MIT Press, 2004, pp. 137-161.

[4] Fischer, G., "Symmetry of Ignorance, Social Creativity, and Meta-Design," Knowledge-Based Systems Journal, Elsevier Science B.V., Oxford, UK, Vol. 13, No. 7-8, pp. 527-537, 2000.

[5] G. Fischer, E. Scharff, and Y. Ye, "Fostering Social Creativity by Increasing Social Capital," in Social Capital, M. Huysman and V. Wulf, Eds., 2004, pp. 355-399.

[6] F. Fukuyama, "Social Capital and Civil Society," presented at IMF Conference on Second Generation Reforms, Washington, DC, 1999.

[7] M.H. Goldhaber, "The Attention Economy," First Monday, vol. 2, 1997.

[8] J. Herbsleb and A. Mockus, "An Empirical Study of Speed and Communication in Globally-Distributed Software Development," IEEE Transactions on Software Engineering, vol. 29, pp. 1-14, 2003.

[9] J.D. Herbsleb and R.E. Grinter, "Architectures, Coordination, and Distance: Conway's Law and Beyond," IEEE Software, vol. 1999, pp. 63-70, 1999.

[10] I. Illich, Deschooling Society. New York: Harper and Row, 1971.

[11] R.E. Kraut, W. Scherlis, M. Patterson, S. Kiesler, and T. Mukhopadhyay, "Social Impact of the Internet: What Does It Mean?" Communications of the ACM, vol. 41, pp. 21-22, 1998.

[12] T.D. LaToza, G. Venolia, and R. DeLine, "Maintaining Mental Models: A Study of Developer Work Habits," presented at Proceedings of International Conference on Software Engineering, Shanghai, 2006.

[13] FD.W. McDonald and M.S. Ackerman, "Expertise Recommender: A Flexible Recommendation System Architecture," Proceedings of CSCW 2000, 2000.

[14] A. Mockus and J. Herbsleb, "Expertise Browser: A Quantitative Approach to Identifying Expertise," in Proceedings of ICSE02. Orlando, FL, 2002, pp. 503-512.

[15] J. Nahapiet and S. Ghoshal, "Social Capital, Intellectual Capital, and the Organizational Advantage," Academy of Management Review, vol. 23, pp. 242-266, 1998.

[16] Nakakoji, K., Ohira, M., Yamamoto, Y., Computational Support for Collective Creativity, Knowledge-Based Systems Journal, Elsevier Science, Vol. 13, No. 7-8, pp. 451-458, December, 2000.

[17] Nakakoji, K., Yamamoto, Y., Nishinaka, Y., Kishida, K., Ye, Y., Evolution Patterns of Open-Source Software Systems and Communities, Proc. IWPSE2002, ACM Press, Orlando, FL, pp. 76-85, May, 2002.

[18] A. Pentland, "Socially Aware Computation and Cmmunication," Computer, vol. 38, pp. 33-40, 2005.

[19] M.P. Robillard, W. Coelho, and G.C. Murphy, "How Effective Developers Investigate Source Code: An Exploratory Study," IEEE Transactions on Software Engineering, vol. 30, pp. 889-903, 2004.

[20] A.M. Szoestek and P. Markopoulos, "Factors Defining Face-to-Face Interruptions in the Office Environment," in Proceedings of Conference on Human Factors in Computer Systems, 2006, pp. 1379-1384.

[21] J.R. Tyler and J.C. Tang, "When Can I Expect an Email Response? A Study of Rhythms in Email Usage," in Proceedings of the Eighth European Conference on Computer Supported Cooperative Work (Ecscw2003). Helsinki, 2003, pp. 239-258.

[22] Wenger, E., Communities of Practice − Learning, Meaning, and Identity. Cambridge, UK: Cambridge University Press, 1998.

[23] C. Westrup, "On Retrieving Skilled Practices: The Contribution of Ethnography to Software Development," in Social Thinking: Software Practice, Y. Dittrich, C. Floyd, and R. Klischewski, Eds. Cambridge, MA: MIT Press, 2002, pp. 95-110.

[24] Y. Ye, Y. Yamamoto, K. Nakakoji, Helping Programmers through In Situ Networking of Peer Expertise, ICSE 2007 (submitted).