

Supporting Software Development as Collective Creative Knowledge Work

Kumiyo Nakakoji^{1,2}

¹*RCAST, University of Tokyo* ²*SRA Key Technology Laboratory Inc.,
kumiyo@kid.rcast.u-tokyo.ac.jp*

Abstract

We view software development as a system of evolution consisting of the three elements: (1) artifacts, (2) individual developers, and (3) a community of developers. An individual's determining what artifacts to contribute and how, with whom to communicate by asking or answering, and which role to play within the community affects the quality of software to be developed; how the developers relate to each other does matter. Software development should be viewed as a system of evolution driven through metabolic processes of how artifacts, developers, and the community grow. This paper describes the framework of viewing software design as a collective creative knowledge work, and outlines possible research areas to pursue.

1. Introduction

Software development is knowledge intensive work, involving both planning and presentation activities [34]. Developers need to locate source code potentially relevant to the task at hand, understand how to modify the source code while identifying why the way it is, and/or write new code where necessary [20]. Although requirement specifications, design documents, comments, and design rationale are provided to help developers in this process, they are often not enough. Developers need to be familiar with the programming language for the code, component libraries used and potentially usable for implementing the code, design methods applied to develop the code, programming tools and environments available to develop the code, and application domains of the code.

While experience is certainly helpful, it does not necessarily work in such a way that a longer experience of engaging in a development project provides more knowledge about the entire project. Software development needs knowledge in a variety of fields, which require constant updates. There are no absolute *experts* in software development. Application domains are subject to rapid change. Component libraries are continually updated. New features and functionalities keep being introduced in programming tools and environments. Moreover, there is such

culture in software development that keeps 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” [20]. Thus, the “symmetry of ignorance” among a development team is neither a problem nor an accident; it is a matter of fact in software development [8].

This makes software development a fundamentally social activity [30]. The activity is carried out by a group of developers, forming a community, engaging in collective creative knowledge work [26]. It is a social activity mediated through artifacts, which are primarily source codes and documents. Even a single-person project has such a community aspect because the project is likely to use component libraries and existing modules, which have been developed by a number of other developers over a long period of time.

2. Social Aspects of Software Development

Social aspects of software development have been studied mostly in the context of how developers and end-users work together in designing a computer technology. Ethnographers and social scientists have explored ways to help them develop a shared understanding and shared context during the process [41]. Another social aspect that has been studied is the organizational context of a software development project [30].

This paper in contrast focuses on the peer-to-peer level of knowledge collaboration of software developers. How developers use other developers as knowledge resources and what social issues are involved during the process, such as the cost of interruption and the motivation for contribution.

Let me first illustrate what kinds of social issues I am referring to.

While sharing knowledge and information within a community of developers is indispensable, the primary means for developers to obtain knowledge is not

through communicating with their peers, but through artifacts.

In understanding source code, developers ask questions such as where to focus as an initial point, exploring the related parts, understand concepts involved the related parts, and understand the relationships among the concepts [35]. During the process, software developers “invest great effort recovering implicit knowledge by exploring code” [20].

However, this exploration process often does not succeed primarily because of the lack of detailed knowledge articulated in the source code. If this becomes the case, software developers would start depending on distributed knowledge resources; namely, the other developers in the community.

By conducting two surveys and eleven interviews with software developers at Microsoft Corporation, [20] have observed that “Developers go to great lengths to create and maintain rich mental models of code that are rarely permanently recorded, and when trying to understand a piece of code, developers turn first to the code itself but when that fails, to their social network.” This would work because source code is often *owned* by a certain developer or a team of the small number of developers, who has a detailed, almost complete knowledge of the source code.

This way of knowledge sharing and collaboration involves two types of social issues. First, asking the *owner* of the source code, either through face-to-face or via email would cost some additional work for the person who is being asked for help, and may interrupt his/her primary work [20]. An interruption is regarded as an unexpected encounter initiated by another person, which disturbs “the flow and continuity of an individual’s work and brings that work to a temporary halt to the one who is interrupted” [37]. Different interruption moments have different impacts on user emotional state and positive social attribution [1].

Second, even if the one understands the source code by being helped by his/her peer, this understanding is not likely to be articulated nor recorded because not only of the overhead of writing it down, but also of the feeling that the newly found information “is not authoritative enough to add permanently to the code” or that checking in the comment under his/her own name “would inappropriately make them experts” [20]. This thereby often results in “institutional memory loss” [20].

Supporting software developers would need to support their collaboration with their peers. Support for collaboration, then, would need more than simply finding the “right” person for completing the task. Social factors, such as motivation, trust, self-confidence and social recognition, need to be dealt with.

3. Three Elements of Software Development: Artifacts, Developers, and a Community

The goal of supporting software development as collective creative knowledge work is to support software developers to develop software. This is different from that of social matching systems, which is to introduce people to people [38].

This position paper views software development as a system of evolution consisting of the three 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 forming a knowledge community. A knowledge community is a group or people that collaborate with one another for the construction of artifacts of lasting value [4]. In a knowledge community, people are bonded through the construction of artifacts.

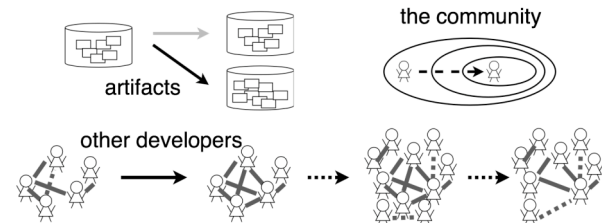


Figure 1: Software Development as a System of Evolution Consisting of the Three Elements

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 the engagement in the project. Such factors affect how the developers collaborate, communicate, and coordinate with one another, resulting in different ways of how they share knowledge.

Because sharing knowledge is indispensable in software development, the quality of resulting software thus depends not only on the skills and knowledge of individual developers but also on the roles of and the social relationships among the developers. The sum of the amount of each developer’s knowledge does not simply determine the quality of software to be developed; how the developers relate to each other also does.

None of the three elements are constant during the software development. Artifacts change over time throughout the development. Individual developers, or more precisely, what individual developers know, grow by gaining experiences of engaging in the development and learning about the artifacts. A

community of developers change by having new developers join, and some developers leave the development project. Their officially assigned roles and informally perceived roles change over time, and the social relationships among them also change.

Existing studies on supporting software development have primarily focused on the evolution of artifacts. More recent work has started to look at how individuals change over time through learning. In contrast, not much has been studied on the aspect of the evolutionary community in the context of software development processes [27].

The rest of the position paper focuses on how the community element evolves and how technologies ought to support such processes.

4. The Metabolic Process of a System of Evolution

A software system needs to evolve to improve its quality in terms of efficiency and robustness, or to cope with the external changes in the environment where the software is used. This type of evolution, recently referred as incremental change [32], should be viewed as not simply adding new objects or mending broken ones; rather it should be viewed as a metabolic process.

Artifacts go through such a metabolic process by adding, modifying, and refactoring the source codes. New parts are added and old parts are rewritten. Some parts may be replaced with other parts.

Individuals' knowledge evolves through learning [22]. They learn by reading source code and information sources such as documents. They learn by asking peers questions. They also learn by solving new problems and experiencing unfamiliar situations. Their old knowledge is replaced with new ones and restructured during the learning process.

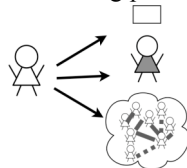


Figure 2: Three Aspects of the Community's Metabolic Process

A community grows through a metabolic process through individual activities. This paper views the metabolic process of a community from the following three aspects (Figure 2): (1) the relationship of an individual with artifacts; (2) the relationship of an individual with other developers; and (3) the relationship of an individual to the community as a whole.

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

An early social navigation system, Expert Browser [24], provides this type of information. Expert browser uses data from change management systems to locate people with desired expertise, by using a quantification of experience. The system then presents evidence to validate this quantification as a measure of expertise.

As a more recent tool, LifeSource [14] provides two visualizations of CVS code repositories. CodeConnections provides file-centric, temporary-animated visualizations, where color-coded authors (i.e., developers) are indicated in terms of the file-structures. CodeSaw provides the author-centric visualizations of weighted collection of email and code contributions of each developer, where the view can overlay multiple developers' contribution to make comparisons.

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

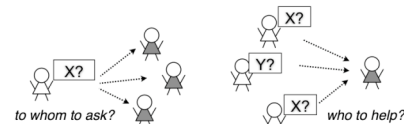


Figure 3: Asker-Helper Relationship

Answering to a question costs the helper (i.e., the answerer) additional work as well as interrupts the helper's current task. Resuming the original task after such an interruption has also been found quite costly [19]. How one would, then, help another if answering is such a costly task.

To help people to decide to whom to ask, social awareness tools [36] help community members become aware of what is going on within a community, and primarily helps askers to decide who and when to ask a question by looking at how intensively potential helpers are currently engaging in their own tasks.

Comparing to the number of approaches that aim at supporting askers, very few studies exist that focus on supporting helpers (Figure 3).

The feeling of *expectation* and *obligation* plays an important role during the helper's process of deciding whether and when to help. Having information about one's social relationships with the other individual developers helps him/her develop a feeling of *obligation* and *expectation* with each of them [28] as people tend to favor reciprocal acts. If Person X gives a service to Person Y, X feels an expectation for Y while Y feels obliged to return the service to X in the near future. Thus, one may feel obliged to answer to a question being asked by his/her peer developer who had kindly helped him/her the week before. *Obligations* "represent a commitment of duty to undertake some activity in the future" [25]. *Expectations* are what one has on others based on one's trust to them and vice versa. Researchers see obligations and expectations as complementary features [3], incurred during prior interactions, creating value for the community in the future [31].

A few systems have been developed to explore individual relationships to help one decide how to engage in the communication. For instance, SoyLent [9] provides temporal and social structures of an online activity by visualizing email messages and their traffics. The system provides a nice ego-centric view to identify with whom one has been communicating at what time, helping him/her to develop a feeling of obligation and expectation.

(3) *the relationship of an individual to the community as a whole*. How one relates with the community is concerned with his/her role within the community, belonging to either a peripheral part, a core part, or an intermediate layer. This aspect helps a developer to decide how much he/she contributes to the community by getting trust and social reputations within the community. One's role evolves within a community through legitimate peripheral participation [40]. By looking at how and what a developer's peers who are closer to the *core* of the community do within the community, he/she gradually acquires skills through learning, and develops his/her identity within the community.

One-to-one communication and collaboration also contributes to the development of social reputation. *Obligations* and *expectations* also play a role in this context. When other peers in the community look at X giving service to Y, X might gain not only expectation to Y but also social reward from the community in the form of good reputation and trust from the community. This might then lead to shifting the role of X within the community from the *periphery* closer to the *core*.

Tools have been developed to use Usenet Newsgroup communities to identify this type of relationships of an individual with the community.

Tools described in [10][12] provide a second-degree ego-centric network for each author together with out-degree histograms of each community, and identify types of users (e.g., answer-only group) and characterizes each community. Newsgroup Crowds and AuthorLines [39] identify authors and types of authors in terms of how they are engaged in the community. The tools visually represent for each user the number of posting per thread and that of active days over a month. They highlight recently posted messages and the encodes the number of posts to the entire set (Usenet Newsgroup as a whole) as the size, allowing people to understand the "role" of a user as a whole and for a particular newsgroup.

In order to support the evolutionary metabolic process of a community, we need technologies for individuals to become aware of the current state as well as its history from the three aspects; that is, to help them determine what artifacts to contribute and how, with whom to communicate by asking or answering, and which role to play within the community.

5. Technical Support for Metabolic Processes of the Community Evolution

Our approach is to use the interaction histories as a source for such decision making by allowing developers to deal with social factors, such as motivation, trust, self-confidence and social recognition.

A number of social navigation systems have been studied to support community activities in a variety of domains [17]. Many of them visualize the history of the community members' activities to analyze the community as a whole, and/or to help a user decide which community to join or to find people with whom the user should communicate. Many of them, however, suffer from not having a clear goal of who is to use the visualizations for what purposes. Having clear goals would determine what types of data to show in what ways, for instance, whether to use *my-own-data* or *collective-social-data* as a *collective snapshot* or as *temporal transitions* [11].

Our goal here is to use the interaction history data to help software developers determine how to engage in the community by interacting with which artifacts and with whom. How developers engage in the community then would shape the metabolic processes of the system of evolution from the community aspect. In considering this, this section argues for the following claims.

(1) *Such data should describe the state of the community, as well as the trends and temporal changes over a long period of time.*

The evolution of an organism depends not only on the type of perturbation, but also on the current structure of the organism. The current structure is determined through its historical development [22]. Having temporal views that allow us to understand how the community has evolved is quintessential.

For instance, even when two developers worked the equal amount of time on a certain module, if the one has worked over the period of two years and the other has been working during the last two months, the latter developer is likely to know better about the current version of the module. This kind of information is important to identify to whom ask about the module [24].

(2) Such data should support not only views for the summaries and overviews of the interaction history data but support ego-centric views, those based on individuals' perspectives.

Because it is situated within a social context, knowing the current state and its history of one's relationships with artifacts, the other developers, and the community, is not as straightforward as it seems. Such relationships are by no means objectively countable or measurable. One could only assume, or feel, what the relationships currently are and have been. One could also assume, thereby, how the relationships would look by another developer.

For instance, you think you have the X amount of expertise on a particular part of the source code. You may think that you are a little bit overestimated by one of your colleagues, Bob, and have a feeling that Bob thinks that you have the Y amount of expertise on the part. You think that Bob has the Z amount of expertise on the part, but again, Bob might think a little differently.

Thus, such technologies that support a community's metabolic process should help an individual to feel or assume the current state as well as its history of his/her relationships with artifacts, the other developers, and the community. They need to aim at providing data not only from an objective standpoint, but also from an individual, ego-centric viewpoint.

(3) Such data can be collected within the scope of a single community activity, as well as from that of external activities.

People's social relationships might be determined not only through activities within the community but also through those external to the community or within another community [43]. A software developer might be a member of another project, belonging to multiple communities.

A developer might be able to better understand the skill level of his/her peer by knowing the role of the peer member within another development community.

(4) Some parts of such data should only be partially disclosed to the community members, creating asymmetric information disclosure.

Software developers may not want to disclose all the historical information of his/her activities within the community. He/she should be able to explicitly specify some of the properties of his/her relationships with artifacts, developers and the community (e.g, the skill level with a certain module) because it is not always possible to adequately assume how such relationships are and have been.

The Saori system [15] provides users with awareness of and control over the information dissemination process within social networks. Saori allows users to specify types of information to be shared and a sharing policy at the level of mostly public and mostly private, not at the level of individuals. The STeP_IN (Socio-Technical Platform for in situ Networking) system [29] allows users to explicitly specify with whom developers want to communicate in what topics. This information is kept invisible to the other developers.

6. Social Factors

This section briefly examines social factors that affect software development driven by a knowledge community: motivation and interruption.

6.1 Motivation

Studies have recently been reported on how to motivate people to make contributions of higher quality to community-maintained artifacts of lasting value (CALVs). Ludford et al. [21] reports that telling people how they are special with respect to the group and its purpose increases member contributions and levels of satisfaction. Cosley et al. [4] argues that what they call "intelligent task routing," which is matching people with work, can be helpful to increase people's contribution, and that such intelligent task routing should consider not only the community's needs but also a person's knowledge and ability. Rashid et al. [33] has found that giving feedback about the value of a participant's contribution in terms of a small group the user has affinity with is most effective in motivating people to contribute.

Although the domain of these projects is movie recommendation and not software development, these findings seem to be equally applicable to software development as a collective creative knowledge community activity. On the other hand, this domain has fundamentally different nature from that of software development. In making a community repository of movie recommendations, the members of

the community has no clear purpose of *finishing it* having no explicit incentives for doing so. With many of software development projects, developers of each community share the clear goal of finishing a project, and they may be more motivated to help one another.

In either case, we need to conduct empirical studies to draw any significant conclusions on this matter, and further studies are necessary on how to motivate developers to contribute high quality artifacts and sustain the community as a system of evolution.

6.2 Interruption

Although interruptions between humans have mainly been studied in face-to-face communication settings, many findings seem to also be applicable to communications through email. In a face-to-face communication, an asker and a helper first need to go through a negotiate process making an agreement on when to interrupt the helper. People use a variety of social cues to decide when to start the negotiation process and making an agreement [42].

In using email for communication, it is much easier for a helper to ignore email message that asks for help. On the other hand, it is more difficult for an asker to get a timely help as the asker cannot tell when a helper would reply to the message. Studies by LaToza [20] found that this makes developers to go more and more face-to-face communications rather than using email, which causes serious problems of interruption especially employing agile development styles.

Wiberg and Whittaker [42] report that in their face-to-face interruption studies, users preferred to take interruptions as soon as possible. People preferred to take interruptions now, incurring the cost of disrupting their currently activity in order to avoid the future overhead of having to schedule and remember later commitments to talk. The authors also argue that users felt a social obligation to return calls and a need for being polite rather than delegating them even though it require more effort to do this.

These phenomena seem to also hold true for email communications. Although it is not as socially critical as in face-to-face communication, putting off replying to information-seeking messages often makes one to feel guilty. One may feel that he/she wants to reply to a message as soon as possible so that he/she would not need to worry about not forgetting replying.

To address this issue, the STeP_IN system [29][44] uses a mechanism to automatically set up anonymously addressed mailing list for an asker's request. The tool produces such a mailing list by taking into an account who is asking what question (i.e., the topic) and identifying a several set of developers in a community who have expertise in the topic and have *good* social relationships with the asker. The mechanism allows

receivers of the message to remain anonymous, letting them from not feeling bad by not replying to the message. When one of the recipients replies to the message, the identity of the helper is revealed to the asker and the regular ways of social interaction will follow, helping them to develop feelings of *expectation* and *obligation*. The approach is unique where the cost of interruption is treated as a collective manner. This aspect needs to be studied further in order to better support software development as collective creative knowledge work.

The field of human-computer interaction has long been studying how to model interruption between humans and computer agents [18][5]. Some parts of their models and findings should be taken into account to achieve more effective, less disturbing communication channels in support of software development within a social setting. For instance, one possible approach is to model the timing of when a potential helper should receive an email message by deliberately delaying the message delivery.

7. Related Work

The previous sections list existing tools and studies that address specific aspect of the approach. This section addresses three projects that have similar research goals with us in the domain of supporting software development as social activities.

The Augur system can be viewed as an example technique to look at software development as the system of evolution. The Augur system [6][13] simultaneously visualizes the structure of a software system (i.e., artifacts) and the structure of the development process carried out by developers (i.e. developers and the community). Augur visualizes the result of call graph analysis, and networks of contributors to a project, relating those who worked together on a single module. By looking at how developers worked together on what parts of a software system, a user of Augur could tell how relationships between artifacts (software system module structures) and developers change over time, including phenomena such as types of projects, how different roles different developers take, how such roles shifts between core and periphery, how authorship changes, and what patterns of stability and changes are observable. Augur currently supports ways to view the structural changes from an objective standpoint. Providing ego-centric individual viewpoints, for instance, from a particular developer's point of view, such as similar to the ones provided by Soyent [9].

Another example is Hybrid Networks [23], which integrates links from multiple development data sources. The tool uses the Probabilistic Latent

Semantic Indexing clustering technique to associate and cluster data from email discussions, authors, and CVS source code tree branches. The result is integrated and displayed in a single visualized view. The tool currently does not support temporal views or ego-centric views.

As mentioned above, Storey et al. [36] argues for the importance of supporting awareness in software development by visualizing artifact and activity data, and report the result of comparing then-existing 13 tools that support such awareness. They have developed a survey framework, which consists of intention of the visualization, information that are visualized, presentation used in the visualization, interaction provided for the visualization, and effectiveness of the visualizations. Some parts of the framework, such as whether tools address temporal and historical changes over time, and what types of artifacts tools support, are important for our purpose. However, the framework does not focus on the relationships among artifacts, developers and the community, and how they change over time.

8. Discussion

Human aspects of software development have long been not highly focused [30][7] except in few approaches, such as empirical software engineering [2] and considerations of cognitive aspects of software engineering [16]. Recent trends in software engineering cannot be taken into a full account without seriously taking the social aspect of knowledge-intensive software development as a central theme. Using open source software, adapting agile methods through incremental change, and engaging in global software development equally aware of the importance of the collective, creative aspect. This would demand us to develop inter-disciplinary research agenda to cope with the issue. We as researchers and practitioners in this field need to engage in socio-technical collaboration for ourselves.

Acknowledgements

This research is partially supported by the Ministry of Education, Science, Sports and Culture (MEXT) Grant-in-Aid for Exploratory Research, 17650038, 2005.

Reference

[1] Adamczyk, P.D., Bailey, B.P., If not now, when?: the effects of interruption at different moments within task execution, Proc. . CHI04, ACM Press, pp.271-278, 2004.

[2] Basili, V., The Role of Experiments in Software Engineering: Past, Current, and Future, Proc. ICSE'96, pp.442-449, ACM, 1996.

[3] Coleman, J.S., Social capital in the creation of human capital. *American Journal of Sociology*, 94: pp. S95-S120, 1998.

[4] 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.

[5] Czerwinski, M., Horvitz, E., Wilhite, S. 2004. A diary study of task switching and interruptions, Proc. CHI'04, ACM Press, pp.175-182, 2004.

[6] de Souza, C., Froehlich, J., Dourish, P., Seeking the source: software source code as a social and technical artifact, Proc. GROUP05, ACM Press, New York, NY, pp. 197-206, 2005.

[7] Dittrich, Y., Doing Empirical Research on Software Development: Finding a Path Between Understanding, Intervention, and Method Development, *Software Practice is Social Practice, Social Thinking - Social Practice*, Dittrich, Y., Floyd, C., Klischewski, R. (Eds.), pp.243-262, MIT Press, 2002.

[8] 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.

[9] Fisher, D., Dourish, P., Social and temporal structures in everyday collaboration, Proc. CHI04, p.551-558, Vienna, Austria, 2004.

[10] Fisher, D. Understanding Communication Using Social Networks. *IEEE Internet Computing*. September/October, 2005.

[11] Fisher, D., Ask Not for Whom the Visualization is Rendered; It is Rendered for Thee. Workshop paper, presented at the Social Visualization Workshop, CHI 2006.

[12] Fisher, D., Smith, M., Welser, H. You Are Who You Talk To, Proc. HICSS, January 2006.

[13] Froehlich, J., Dourish, P. 2004. Unifying Artifacts and Activities in a Visual Tool for Distributed Software Development Teams. ICSE'04. IEEE Computer Society, 387-396.

[14] Gilbert, E., Karahalios, K., LifeSource: two CVS visualizations. CHI '06 Extended Abstracts on Human Factors in Computing Systems ACM Press, 791-796, 2006.

[15] Goecks, J., Mynatt, E. D. Leveraging social networks for information sharing. Proc. CSCW '04. ACM Press, 328-331, 2004.

[16] Herbsleb, J. D., Beyond computer science. Proc. ICSE '05. ACM Press, 23-27, 2005.

[17] Hook, K., Benyon, D., Munro, A.J. (Eds.), *Designing Information Spaces: The Social Navigation Approach*, CompSpringer, 2003.

- [18] Horvitz, E., Apacible, J. 2003. Learning and reasoning about interruption. Proc. ICMI '03. ACM Press, pp.20-27, 2003.
- [19] Iqbal, S. T., Bailey, B. P., Leveraging characteristics of task structure to predict the cost of interruption, CHI'06, ACM Press, 741-750, 2006.
- [20] LaToza, T.D., Venolia, G., DeLine, R. Maintaining mental models: a study of developer work habits, Proceeding of the ICSE '06. ACM Press, 492-501, 2006.
- [21] Ludford, P.J., Cosley, D., Frankowski, D., Terveen, L., Think different: increasing online community participation using uniqueness and group dissimilarity, Proc. CHI'04, ACM Press, 631-638, 2004.
- [22] Maturana, H.R., Varela, F.J., The Tree of Knowledge: The Biological Roots of Human Understanding, Shambhala Publications, Inc., Boston, MA, 1998.
- [23] Medynskiy, Y., Ducheneaut, N., Farahat, A., Using hybrid networks for the analysis of online software development communities, Proc. CHI'06, ACM Press, 513-516, 2006.
- [24] Mockus, A., Herbsleb, J. D., Expertise browser: a quantitative approach to identifying expertise, Proceedings ICSE'02. ACM Press, 503-512, 2002.
- [25] Nahapiet, J., Ghoshal, S., Social Capital, Intellectual Capital, and the Organizational Advantage. Academy of Management Review, 23, pp.242-266, 1998.
- [26] 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.
- [27] 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.
- [28] Nakakoji, K., Humane Requirements for Enabling and Nurturing Collective Creativity, Proc. HCHI05, Las Vegas, NV, CD-ROM, Jul. 22-27, 2005.
- [29] Nishinaka, Y., Asada, M., Yamamoto, Y., Ye, Y., Please STeP_IN: A Socio-Technical Platform for in situ Networking, Proc. APSEC'05, Taipei, pp. 813-820, Dec. 2005.
- [30] Noerbjerg, J., Kraft, P., Software Practice is Social Practice, Social Thinking - Social Practice, Dittrich, Y., Floyd, C., Klischewski, R. (Eds.), pp.205-222, MIT Press, 2002.
- [31] Resnick, P. Beyond bowling together: sociotechnical capital. Carroll, J. M. (Ed.) HCI in the New Millennium, pp. 247-272, 2002.
- [32] Rajlich, V., Changing the Paradigm of Software Engineering, Communications of ACM, Vol.49, No.8, pp.67-70, August, 2006.
- [33] Rashid, A. M., Ling, K., Tassone, R. D., Resnick, P., Kraut, R., Riedl, J., Motivating participation by displaying the value of contribution, Proc. CHI'06, ACM Press, New York, NY, 955-958, 2006.
- [34] Robillard, P. N., The role of knowledge in software development. Comm. ACM 42, 1 87-92, 1999.
- [35] Sillito, J., Murphy, G., De Volder, K., Questions Programmers Ask During Software Evolution Tasks, Proc. Symposium on Foundations of Software Engineering, November 2006 (to appear).
- [36] Storey, M-A. D. Cubranic, D., German, D.M., On the use of Visualization to Support Awareness of Human Activities in Software Development: a Survey and a Framework, Proc. SoftVis'05, ACM Press, pp.193-202, 2005.
- [37] Szoestek, A.M., Markopoulos, P. Factors Defining Face-To-Face Interruptions in the Office Environment, CHI2006, Work-in-Progress, pp.1379-1384, 2006.
- [38] Terveen, L., McDonald, D. W. 2005. Social matching: A framework and research agenda, ACM Trans. of Comput.-Hum. Interact. 12, 3, 401-434, 2005.
- [39] Viegas, F., Smith, M., Newsgroup Crowds and Authorlines: Visualizing the Activity of Individuals in Conversational Cyberspaces, HICSS-37, Hawaii, January 2004.
- [40] Wenger, E., Communities of Practice - Learning, Meaning, and Identity. Cambridge, England: Cambridge University Press, 1998.
- [41] Westrup, C., On Retrieving Skilled Practices: The Contribution of Ethnography to Software Development, Social Thinking - Social Practice, Dittrich, Y., Floyd, C., Klischewski, R. (Eds.), pp.95-110, MIT Press, 2002.
- [42] Wiberg, M., Whittaker, S., Managing availability: Supporting lightweight negotiations to handle interruptions. ACM Trans. of Comput.-Hum. Interact.,12,4, 356-387, 2005.
- [43] Ye, Y., Yamamoto, Y., Dynamic Communities in Support of Situated Knowledge Collaboration, Proceedings HCHI05, Las Vegas, NV, CD-ROM, Jul. 22-27, 2005a.
- [44] Ye, Y., Dimensions and Forms of Knowledge Collaboration in Software Development, Proceedings APSEC, Taipei, pp. 805-812, Dec. 2005b.