

Design Principles for Developer-Centered Collaborative Software Development Environments

Kumiyo Nakakoji(1,2), Yunwen Ye(3), Yasuhiro Yamamoto(1)

(1) Research Center for Advanced Science and Technology, University of Tokyo

(2) SRA Key Technology Laboratory, Inc., Japan

(3) SRA Inc., Japan

1. Introduction

Software development is in essence information-intensive collaborative knowledge activity. It is about using information; it is about generating information; and it is about making information artifacts. The wide acceptance of agile processes and success of many open source projects provide strong evidence that human aspects do matter in software development; cognitive and social processes play an essential role in successful software projects where individuals' creative knowledge engagements of using and generating information are nurtured. We argue that software engineering environments must be designed to foster such individuals' creative knowledge processes, and that collaboration must be supported in the context of individuals' development activities. Collaborative software development environments (CSDEs) should be designed to facilitate and nurture individuals' creative knowledge processes. We call the approach *developer-centered CSDEs*.

Collaboration takes place with or without explicit communication. On the one hand, software developers regularly engage in collaboration through artifacts without explicit communication (i.e., by writing comments in code to be read by others). On the other hand, communication be-

comes necessary when developers must ask their peers for the information that is otherwise not possible to obtain. Existing studies have provided ample evidences that both collocated and distributed software development teams frequently engages in communication to acquire necessary information from peer developers [Ko et al. 2007; LaToza et al. 2006; Herbsleb, Mockus 2003].

There are two distinctive types of communications in software development: one is communication for coordinating development activities (we call *coordination communication*), and the other is communication for acquiring knowledge and understanding of a particular aspect of the software artifact under investigation (we call *expertise communication*). Tools for the former have been primarily studied in such research areas as coordinating programmers and programming tasks, making developers aware of what other developers are doing, for instance, Ariadne [de Souza et al. 2007], Palantir [Sarma et al. 2003], or FastDASH [Biehl et al. 2007]. Tools for the latter have been primarily studied in such research areas as knowledge sharing and experts finding, helping developers ask questions for other developers, for instance, Expertise Recommender [McDonald and Ackerman 2000], Expert Browser [Mockus and Herbsleb 2002], and STeP_IN [Ye et al. 2007a]. The rather clear separation of the two research areas reflects the fact that the two types of communication have quintessential differences in nature; in their goals, challenges and concerns.

Studies on communication for a developer-centered CSDE, therefore, must be mindful of which type of communication they want to observe, analyze and support. In contrast to Redmiles et al.'s approach [Redmiles et al. 2007] that proposes the continuous coordination paradigm and suggests four design disciplines to support *coordination communication*, this chapter discusses design principles for supporting *expertise communication* in software development. We have derived the principles based on the theories of social capital [Fischer et al. 2004] and models of supporting collective creativity [Nakakoji et al. 2000; Nakakoji 2006], as well as existing tools in the research fields of intelligent support, groupware, knowledge management, and organizational memory.

In what follows, we first briefly describe the historical context for the developer-centered CSDE approach in software engineering research and discuss why communication must be supported as a first-class object in CSDEs. We then elaborate the differences between the two types of communications: *coordination communication* and *expertise communication*. Section 4 describes nine design principles for supporting expertise communication in a developer-centered CSDE. Each of the principles is out-

lined with theoretical grounds and is illustrated with technical instruments introduced by the existing tools and environments.

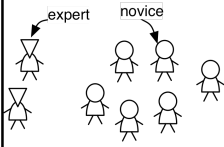
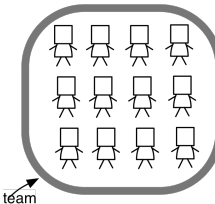
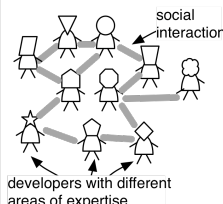
2. Historical Context: Three Schools of Research toward Developer-Centered CSDEs

Human factors and collaborative aspects in software engineering research are not new. In fact, software engineering research has looked at humans and their collaborations from the very beginning. Along the way, however, its emphases have been shifted several times during the last few decades. We identify three distinctive schools of research during the period. Table 1 illustrates the differences among the three schools.

The first school of research, which we call the *psychology-centered* approach, has investigated the inner cognitive process of programming by focusing on the differences between expert programmers and non-expert programmers through a number of psychological studies. That was the time right after Frederick Brook's *The Mythical Man-Months* was first published in 1975, which basically says that man-months is not an appropriate measure to software development project performance. It was realized that there is a huge performance difference between good programmers and not-so-good programmers. This had motivated a large number of studies to explore what psychological/cognitive factors in programming distinguishes experts from novices. The psychology of programming is a research area that primarily looked at the differences of programming productivity and efficiency between experts and novices, while studying the benefits as well as difficulties of mastering programming features (e.g. *if* statement design), methods (e.g. object orientation), and usage (e.g. mnemonic variable names) [Shneiderman 1980][Soloway and Ehrlich 1984].

The second school of research, which we call the *process-centered* approach, has its focus on the collaborative and managerial aspects of a software development project. It views software development as a group activity, or teamwork, and studies how to improve the capability of a software development organization, such as process traceability and repeatability [Humphrey 1989]. Interestingly, this second school of research is less concerned with programming skills of individual developers. Instead, it is more concerned about the skills of organization. This school advocates that a software development process is programmable, and software development should be treated as assembly lines where developers produce software by following predefined process instructions [Osterweil 1987]. Developers take specification documents and test specifications as input and

Table 1: Three Schools of Human-Oriented Software Engineering Research

	School 1	School 2	School 3
centered-around:	psychology-centered	process-centered	developer-centered
focuses on:	human programming skills	artifacts evolving processes	cognitive and social processes
software development is viewed:	as an individual skillful task	as an information processing task	as a creative knowledge task
primary research question:	how to develop programming skills	how to manage a project	how to nurture creative collaborative development processes
developers interact with:	programming language	documents and artifacts through step-by-step actions	tools and environments
looks at:	expert-novice differences	team-performance differences	individual developer difference in terms of areas of expertise
related disciplines:	cognitive psychology AI (knowledge-based systems) CHI	organizational science operation research measurements	tool building knowledge community CSCW
key phrases:	psychology of programming	process programming	creativity, motivation, collaboration, community
research materials:	- performance and productivity	- products (things) and processes (humans) - project management	- peer programmers as knowledge resources - making information relevant to the task-at-hand - social dependencies among developers - globally distributed collaborative development
beliefs:	There are two types of programmers: experts and non-experts. Their interactions are of little relevance to the project performance. 	All programmers perform task more or less equally as information processors (like assembly lines) but the team as a whole performs differently depending on the organizational maturity 	There are more than novice-expert differences among individuals. Each developer has his/her own area of expertise. They are loosely tied together through social interactions within a project. 

produce source code and test cases as output. Researchers in this second school have primarily focused on how to help project management in orchestrating and coordinating a number of work pieces that have been produced by a large number of developers.

The third school of research, which we call the *developer-centered* approach, is the focus of our research. It looks at both cognitive and social aspects of software development as well as their mutual interactions. The

focus has returned to an individual developer, who is now viewed as having his or her own area of expertise in terms of a specific context; for example, the expertise on a piece of source code, the expertise on a certain feature of the program, the expertise on a certain aspect of the application domain, or the expertise on a certain programming language. Thereby, there is symmetry of ignorance, or asymmetry of knowledge, among project members [Fischer 2000]. They have to collaborate with peer developers in order to accomplish their own programming tasks on which the success of the whole team depends. Individual developers are being loosely bonded together with peer developers through social interactions.

Researchers in the third school explore how to support developers in such a way that they collectively develop information artifacts. Project managers are expected to be concerned with how to ensure the creativity and productivity of individual developers by providing physically-, organizationally-, culturally-, and computationally-*right* environments, rather than worrying only about how to quantify project performances and how to keep an eye on the project milestones in regard with the produced artifacts.

There are two major factors that fueled the third school of research: open-source communities and agile development methods. Both of them demonstrate the great importance of an individual developer's motivation, engagement, and communication in software development.

Since a large number of open-source software development projects have emerged and made not only their source code but also related documents, development history data as well as mailing list archives openly available, a number of field studies have examined how software artifacts evolve through intensive communicative activities. As Augustin et al., who operated SourceForge, have noted, such data have revealed that successful open source community projects “employed a number of practices that were not well characterized by traditional software engineering methodologies” [Augustin et al. 2002]. The paper lists mobility of resources, culture of sharing, and peer review and peer glory as examples of such practices, and labels the practices as “collaborative software development or CSD.”

Many of the twelve practices of XP [Beck 1999], a representative agile method, are concerned with human and social aspects. By embracing individuals and interactions over processes and tools in their manifesto, agile software development methods aim to achieve successful software development by nurturing developer's collective creative processes [Tomayko and Hazzan 2004].

Communication has long been regarded as an important activity in software development. A software engineering textbook published in 1985 by

Fairley, for instance, shows that 37% of developers' time is spent in job communication and email [Fairley 1985]. However, communication was then regarded as an overhead rather than a part of the fundamental activities in software development. It was the trends of open source and agile methods that have made communication a first-class activity to support. The third school of research now views communication as something to be nurtured, not to be avoided.

It is very important to note that communication costs in software development remain very expensive even in the eyes of the third school of research. We argue that although supporting communication is important, encouraging more communication in general is not the research goal. Communication problems are not only caused by the lack of communicative acts; they are sometimes caused by too many communicative acts. For example, one case study reported that overwhelming incoming mail messages resulted in a significant coordination problem [Damian et al. 2007]. Studies have shown that programmers in general prefer to work in a solitary environment with long periods of uninterrupted time during which they can concentrate [DeMarco and Lister 1999; Perlow 1999]. As engaging in creative knowledge work, developers embrace flow experience, which is a situation "in which attention can be freely invested to achieve a person's goals, because there is no disorder to straighten out, no threat for the self to defended against" [p.40, Csikszentmihalyi 1990].

A developer-centered CSDE should first ensure that a developer can focus on his or her own task itself, and then facilitate easy communication with peer developers only when it becomes necessary. An important and often overlooked aspect is that when a developer wants to have communication, the one who is wanted to be communicated with is also a developer. Supporting communication must carefully balance one developer's needs for communication and the other developer's needs of concentrated flow experience.

3. Communications in Software Development

Many studies have observed how and what developers communicate with one another during software development. For instance, through a study on three well known open source projects, Gutwin et al. have found that text-based communication (mailing lists and chat systems) is the developers' primary source of acquiring both general awareness of the entire team, and more detailed information about people's expertise and activities [Gutwin et al. 2004]. An ethnographic study on an industrial project by Ko et al.

analyzes what information needs developers face during software development. Their findings include that coworkers were the most frequent source of information by software developers and that two types of most frequently sought for information that depended on their coworkers were “what have my coworkers been doing?” and “in what situations does this failure occur?” [Ko et al. 2007].

Such studies demonstrate that two distinctive types of communication are involved in software development. One is primarily for the purpose of coordinating tasks. A developer sends an email message, starts a chat, or walks to a coworker’s cubicle to invite them to start a discussion (and negotiation) to resolve emerging conflicts or to avoid possible future conflicts among the software components that they are working on. The structural dependency of software components may reflect “social dependency” among the developers who work on the components in the sense that they have to coordinate their tasks through social interactions when the perceived conflicts become necessary to be resolved [Wagstrom and Herbsleb 2006; de Souza and Redmiles 2008].

The other type of communication is primarily for obtaining knowledge and understanding about the design and/or behavior of a particular part of the system under development. A developer posts a question to a mailing list, sends a personal email message, or walks to a coworker’s cubicle and asks for information that is essential for performing his or her own task at hand [LaToza et al. 2006; Ye et al. 2008a].

A developer engages in communication with peer developers in both situations but the nature of the communication in each is quite different. We call the former *coordination communication* and the latter *expertise communication* to distinguish the two. In a developer-centered CSDE, communication support needs to be tuned to each type of communication based on their differences in nature. For instance, the relation between the developer who starts to communicate and those who are communicated with is different in *coordination communication* and *expertise communication*. In *coordination communication*, there is a symmetric or reciprocal relation between those who initiate communication and those who are asked to communicate with, with roughly equal amount of interests and expected benefits. Coordination communication is a part of impact management, which is “the work performed by software developers to minimize the impact of one’s effort on others and at the same time, the impact of others into one’s own effort” [de Souza and Redmiles 2008]. In *expertise communication*, in contrast, there is an asymmetric and unidirectional relation between the one who asks a question and the one who is asked to help [Ye et al. 2007a]. The benefit is primarily for the asker while the costs

are primarily paid by the helping developer. Such costs include the cost of paying attention to the information request, that of stopping his or her own ongoing development task, that of composing an answer for the information-seeking developer while collecting relevant information when necessary, and that of going back to the original task.

Different types of communication demand different sets of design principles in supporting communication in developer-centered CSDEs. Redmiles et al. present the continuous coordination paradigm for supporting *coordination communication* in software development [Redmiles et al. 2007]. The paradigm contains with the following four principles: first, to have multiple perspectives on activities and information; second, to have non-obtrusive integration through synchronous messages or through the representation of links between different sites and artifacts; third, to combine socio-technical factors by considering relations between artifacts and authorship so that distributed developers can infer important context information; and fourth, to integrate formal configuration management and informal change notification via the use of visualizations embedded in integrated software development environments [Redmiles et al. 2007].

This chapter presents design principles for supporting *expertise communication* in software development. By expertise communication, we do not mean knowledge exchange or knowledge transfer in a general sense. We use the phrase *expertise communication* to refer to the kind of activities by a software developer, who seeks, from his or her peer software developers, information that is essential yet not readily available in existing artifacts to accomplish his or her task, right in the middle of software development. The developer communicates with coworkers and asks for information not for the sake of increasing general knowledge in abstract but for the sake of performing his or her own immediate task.

4. Design Principles for Supporting Expertise Communication

This section presents nine design principles for supporting expertise communication. Before describing the principles, we first outline an overview of existing approaches that support expertise communication in software development.

4.1 An Overview of Existing Approaches for Expertise Communication

Existing research supports expertise communication primarily in three phases: how to locate expertise that an information-seeking developer is seeking for, how to select experts with whom to communicate, and how to approach experts with an appropriate communication channel.

Expertise sharing mechanisms first identify those peer developers who have expertise about the information sought by a developer. Previous research has shown that such knowledge as finding where the sought expertise is located and who potentially has the expertise takes extensive time to develop [McDonald and Ackerman 1998; Orlikowski 2002].

Answer Garden [Ackerman and Malone 1990] is an early attempt that routes questions about Unix to experts based on predefined expertise profiles, and accumulates answers from experts. The Expertise Recommender system [McDonald and Ackerman 2000] mines configuration management logs to identify experts and recommends experts based on organizational relations to support software maintainers. The approach of identifying experts from project history was further improved and validated in the Expertise Browser system [Mockus and Herbsleb 2002]. Other researchers have tried to identify expertise by analyzing programs that developers have written [Vivacqua and Lieberman 2000].

After finding who have the sought for expertise, expertise communication mechanisms need to choose a set of experts who would be willing to provide appropriate answers in a timely manner for the information-seeking developer [McDonald and Ackerman 1998]. Finding those with necessary expertise does not necessarily lead to the acquisition of the information being sought. Humans do not serve as knowledge resources unless they consent to engage in knowledge collaboration activities [Illich 1971]. As Reder and Schwab point out, automated attempts to “pin people down” may not bring about better communication or enhanced productivity because successful expertise sharing requires intensive engagement of those who answer [Reder and Schwab 1988].

The willingness of a developer to engage in helping other developers to solve their tasks depends on a set of perceived psychological and social factors. Studies have looked into social and psychological aspects, such as who have previously communicated with the asker [Ye et al. 2007a], who are motivated to answer at this very moment [Ribak et al. 2002], who have less workload (thereby are likely to have more time to answer) [McDonald and Ackerman 1998], and who have matching experience with the asker [Vivacqua and Lieberman 2000].

Finally, the expertise communication mechanisms need to consider how to approach the selected developers. Depending on which communication-channels are used people feel more or less comfortable joining or not joining the communication. A study by Herbsleb and Grinter, for instance, observed that collocated developers feel socially comfortable to initiate contact easily 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 [Herbsleb and Grinter 1999].

Forcing them into communication is neither good nor desirable. Expertise communication mechanisms should take into account the balance between the cost and benefit of an information seeking developer and the group productivity. Studies have found that the quality of answers would depend on how one asks a question. The system needs to support both who asks a question and those who are asked to answer.

4.2 Nine Design Principles for Expertise Communication

Principle #1: Expertise communication must be integrated with other development activities.

A need for expertise communication emerges during the development activity when a software developer finds his or her task in need of information that is only available through other developers. The developer must be able to acquire the necessary information in a timely fashion so that he or she can carry out the current task more effectively and productively in a fluid manner [Ye 2001]. Communication with peer developers for expertise seeking should be supported as a continuum of information search tasks from an information-seeking software developer's point of view. It needs to be integrated with a program- and document-authoring and browsing environment to minimize the cognitive cost of conscientiously switching to a different application, and then recalling and restructuring the context of the particular problem [Card et al. 1991].

Not many existing tools supporting expertise communication observe this principle. One of few tools that follow this principle is STeP_IN_Java [Ye et al. 2007b]. STeP_IN_Java has the "Ask Expert" feature embedded within the Java document-browsing interface. Each Java method is accompanied with the "Ask Expert" button (see Figure 1), and pressing the button will bring up a message-composing interface where the user can write a question about the Java method, which is then delivered to those developers who have expertise about the method. The system thus makes expertise communication a natural extension of browsing Java documents.

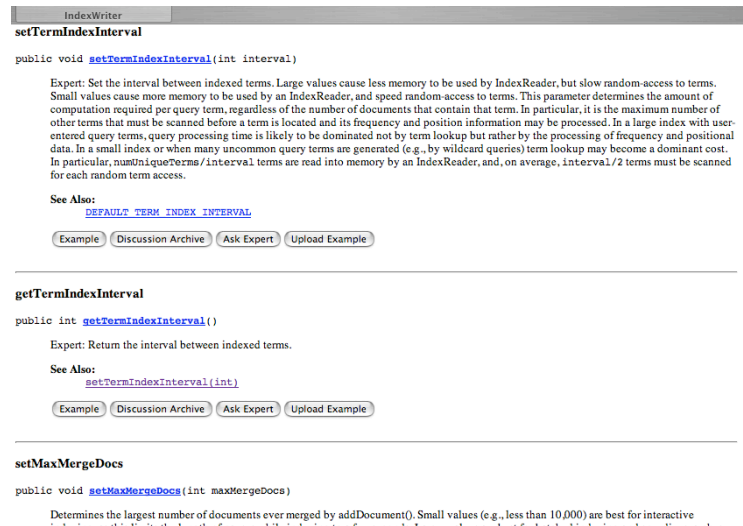


Figure 1: Expertise Communication Interface Integrated within Java Doc Browser in STeP_IN_Java [Ye et al. 2007b]

Principle #2: Expertise communication mechanisms should be personalized and contextualized for the information-seeking developer.

Information seeking in software development is an in situ and highly individualized action. A developer's needs for acquiring information from his or her coworkers arise when he or she is dealing with a specific task in a development environment. Integration with the development environment provides the context of the problem with which a developer is dealing. Such context should be utilized by an expertise communication mechanism to customize its support to the context and the background knowledge of the developer [Ye and Fischer 2002; Davor et al. 2003].

Expertise is not an absolute attribute but a relative attribute of a developer and changes over time. The developmental histories of developers (such as activities recorded in CVS repositories, mailing archives and written programs) should be used to identify who has needed expertise about the particular problem at the particular moment [Mockus and Herbsleb 2002]. The current socio-technical context surrounding the project and its team is determined by its historical development [Maturana and Varela 1992]. Having temporal information of the socio-technical context allows us to understand whether a developer has, and how he or she has gained,

the expertise being sought. Such information is not only useful for identifying expertise being sought, but also valuable for understanding the information seeker's background so that the system would be able to locate those who have similar mental models with the information seeking developer [Vivacqua and Lieberman 2000].

Principle #3: Expertise communication should be minimized when other types of information artifacts are available.

Resorting to peers as information resources involves not only the information-seeking developer but also those developers who are asked to provide information [Illich 1971]. Expertise communication, therefore, is a rather expensive means to get a developer's work done. It should not be promoted as the first choice, rather it should be avoided when code, documents, development history records, archived previous communications and/or other artifacts that satisfy the information needs are available.

Two mechanisms have been explored to conform to this principle in existing research: leading users initially to artifacts before providing the means of expertise communication, and archiving communication results to avoid unnecessarily repeated communications.

One example is Answer Garden [Ackerman and Malone 1990], which first lets a user to browse a database of commonly asked questions, and if the sought answer is not present, the system "automatically sends the question to the appropriate expert, and the answer is returned to the user as well as inserted into the branching network, thus evolving the organizational memory." The same design philosophy has been followed in the design of subsequent systems, such as Answer Garden 2 [Ackerman and McDonald 1996].

STeP_IN_Java [Ye et al. 2007b] takes a similar approach by first guiding a developer to attend to the search and browsing interface of Java source code, documents and communication archives. Only from the browsing interface does the system allow the developer to compose a question and ask other developers for information about the browsed artifact. The communication is again archived and associated with the artifact.

Other mechanisms, such as TagSEA, which is a shared waypoints mechanism to mark specific locations in Java source code elements or documents using social tagging [Storey et al. 2006], is also a useful instrument in guiding developers to access previously communicated information.

Principle #4: Expertise communication mechanisms should take into account the balance between the cost and benefit of an information seeking developer and the group productivity.

From the project team's perspective, expertise communication is a two-edged sword to solve collaboration problems in software development. On the one hand, broadcasting a question allows a developer to find the right people by letting other developers decide for themselves whether to respond or not [Gutwin et al. 2004]. On the other hand, if developers are frequently interrupted for offering help, their productivity is significantly reduced, resulting in lower group productivity [Ye et al. 2008b].

Goldhaber and others have eloquently argued that we are entering a world where our lives are guided more by the laws of the economics of attention because attention is quickly becoming the scarcest resource in our society [Goldhaber 1997]. Attention economy is concerned with the use or the patterns of allocation of attention for the best possible benefits. Following this thread of thought, the concept of collective attention economy has been proposed and used as an instrument to analyze the effective use of the sum of attentions of members in a group [Ye et al. 2008b].

We have made a rough estimate of how much attention (in terms of time) is collectively spent in expertise communication in the mailing list of the Open Source project Lucene [Ye et al. 2008c]. A discussion thread in the mailing list has 681 words in average. Each thread consumed a collective total of 647 minutes if 50% of the estimated 2,282 members skim (with the speed of 500 words/minute) initial request, and 10% of the members skim the entire thread. About 95 threads were posted every month in average, therefore, 61,394 minutes, which is over 1,000 hours, were collectively spent every month. In an organizational setting, this collective cost might even outweigh the benefits of knowledge collaboration and decreases the overall productivity of the whole project [Reder and Schwab 1988].

Some studies have looked into this problem. Both the Answer Garden approach [Ackerman 1998] and the STeP_IN approach [Ye et al. 2007a] try to reduce the cost incurred on expertise providers by limiting the recipients of the question only to those who are both able to (through the expert identification process) and very likely to willing to (through the expert selection process) to answer the question.

Principle #5: Expertise communication support mechanisms should consider social and organizational relationships when selecting developers for communication.

Favorable inter-personal relationship helps communicate expertise due to preexisting trust and mutual understanding [Ackerman and McDonald 1996]. An arduous relationship between an expertise seeker and an expertise provider often leads to the failure of expertise sharing [Cross and Borgatti 2004]. A recipient who does not like to work with the asker is not likely to provide help. People have very nuanced preferences concerning how and with whom they like to share expertise and like to maintain control of their social interaction [Halverson et al. 2004].

The theory of social capital provides an analytic framework to understand this decision-making process [Fischer et al. 2004]. 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” [Nahapiet and Ghoshal 1998]. Social capital 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 costs [Fukuyama 1999]. Social capital manifests itself in forms of obligations, expectations, trust, norms of generalized reciprocity, and reputations.

The feeling of expectation and obligation plays an important role during the helper’s process of deciding whether and when to help. Researchers see obligations and expectations as complementary features [Coleman 1988], incurred during prior interactions, creating value for the community in the future [Resnick 2002]. In other words, when B helps A, B would have a reasonable expectation that A will do something for him or her down the road, and A would feel obliged to help B [Coleman 1988].

Answer Garden 2 [Ackerman and McDonald 1996] uses organizational and physical proximities in the selection process. STeP_IN [Ye et al. 2007a] uses social relationships and nuanced perception of individual relationships. Table 2 illustrates the different strategies used in the selection steps.

Similar to STeP_IN, some tools give high priority to the individual preferences for expertise communication. For instance, ReachOut [Ribak et al. 2002] takes into consideration factors such as the helper’s motivation to answer questions on the topic, or to participate at this very moment, as well as the helper’s history of participation. The availability of choices and options helps the development of favorable attitudes toward expertise

communication [Salancik and Pfeffer 1978], and this favorable attitude is critical for expertise sharing to become sustainable in an organization.

Table 2: Selection Strategies Reported in Answer Garden 2 [McDonald and Ackerman 1998], in STeP_IN [Ye et al. 2007a], and Others Possible Strategies

Answer Garden 2 strategy	STeP_IN strategy	Other possible strategies
1. organizational criteria 1-1 keeping it local 1-2 cross department 1-3 last resort 2. load on the sources 2-1 selection based on regular workload 2-2 selection based on workload over time 3. performance 3-1 problem comprehension 3-2 providing a suitable explanation 3-3 attitude	1. inter-personal preferences of an individual 1-1 exclude 1-2 include 2. obligation 2-1. inter-personal obligation (has been helped by the information seeking developer) 2-2. social obligation (has been helped by others in the group) 3. external communication history (has previously communicated via email) 4. random selection	- communication recency - organizational hierarchy (relative significance and impact of the information seeking developer to potential helpers) - institutional secrecy - eager helper (who are very motivated to help others) [van den Hoff et al. 2004]

Principle #6: Expertise communication support mechanisms should minimize the interruption when approaching those who are selected to be communicated with.

When being approached to provide information for the benefit of another developer, a developer is likely to feel interrupted. Answering, or providing help, consumes the time and attention of the helping developers and distracts them from their own tasks.

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” [Szoestek and Markopoulos 2006]. The cost of interruption includes not only the attention spent on the interrupting event but the disruption of flow and continuity of the ongoing work [Jackson et al. 2001], and the accompanied work resumption efforts [Iqbal and Bailey 2006].

Expertise communication support tools, therefore, need to feature mechanisms that would minimize interruption when approaching potential helping developers. ReachOut [Ribak et al. 2002], for instance, a chat-based tool for peer support, collaboration, and community building, invites potential helpers to join a conference chat by pushing the question to a non-intrusive client on their screen. Incoming questions fade in and out until the user decides to answer.

The field of human-computer interaction has long been studying how to model interruption between humans and computer agents [Horvitz and Apacible 2003][Czerwinski et al. 2004]. Some parts of their models and findings should be taken into account to achieve more effective, less disruptive communication channels in support of expertise communication in software development.

In an attempt to minimize interruption for other developers by reducing the number of those who are asked to help, one may not be able to get the needed information. To address this issue, Answer Garden 2 has proposed the idea of escalation of support [Ackerman and McDonald 1996]. When no answers are provided from the selected group for a predefined period of time, the system automatically expands the recipients of the question to involve more people, larger groups, and wider range of areas.

Principle #7: Expertise communication support mechanisms should provide ways to make it easier for developers to ask for help.

Developers feel different levels of difficulty and easiness, depending on to whom they ask and through what communication channels. It is easy for developers to ask peers for information through face-to-face communication because many non-linguistic social signals are used to encourage or discourage the further occupation of attention in a socially acceptable way, without causing unwanted damages to social relationships.

As Gerstberger and Allen report, “engineers, in selecting among information channels, act in a manner which is intended not to maximize gain, but rather to minimize loss. The loss to be minimized is the cost in terms of effort” [Gerstberger and Allen 1968]. Thus, developers tend to choose face-to-face communication because it would be less likely to be turned down, and ask for help from coworkers that they feel easy to access rather than from the most appropriate person. This might end up in the wasteful use of a small set of “nice” people who keep helping others even if they do not have appropriate expertise.

Developers may immediately get necessary information or may never get any useful information, depending on how they ask. Rhetorical strategies, linguistic complexity and word choice of the question all influence the likelihood of others responding [Kraut et al. 1998] and receiving replies to a question [Cross and Borgatti 2004; Arguello et al. 2006].

Studies show that askers demonstrate different asking behaviors, depending on whether they are in public, in private, communicating with a stranger, or communicating with a friend due to the different levels of perceived psychological safety in admitting a lack of knowledge [Cross and Borgatti 2004]. If every question asked would always go to all members of the mailing list, the asker risks giving those colleagues the impression that he or she is rather ignorant and incompetent [Flammer 1981].

Perceived social burden on the potential helper side may affect how easy it is to ask a question. A field study of Answer Garden reports that because the asker's identity was not revealed in Answer Garden, the asker felt less pressure in asking questions and bother experts [Ackerman 1998]. It might also become easier for an information seeking developer to ask when the developer knows that the recipients of his or her request have options and freedom to ignore the request.

Reder and Shwab noted that tactical skill in selecting communication channels "often determines an individual's ability to influence and sometimes control the course and direction of group tasks and impact the success of particular projects" [Reder and Schwab 1988]. Expertise communication support mechanisms, therefore, need to consider social factors that affect expertise-seeking behaviors and help software developers in their expertise communication if they do not have enough tactical skill of selecting the right communication channel.

Principle #8: Expertise communication support mechanisms should provide ways to make it easier for developers to answer or not to answer for the information requested.

Developers who receive the request for help in expertise communication need to decide whether to answer or not. They may feel different levels of social pressure depending on by whom they are asked to help and through which communication channel. For instance, in direct emails, the receiver loses control and bears the interruption cost of reply or the social burden of no action [Tyler and Tang 2003].

The success of expertise communication should not come at the price of developers' reluctance of further participation in future collaboration.

Some developers might get bored by answering repeatedly asked questions that they deem too simple to worth their time and expertise; and some might want to guard their certain expertise to retain their “market value” in the organization [Reichling and Veith 2005]. The goodwill and limited attention of developers should be economically utilized to achieve sustainable and long-term success. They should not be forced into helping just for the fear of causing unnecessary disruptions to the social cohesion and norms of the project team, which is unlikely to be sustainable.

Unwillingness also leads to lower quality of communication. When workers are forced into sharing expertise without much of their willingness, they often use “verbal and intellectual skills as a defense to keep a person with a problem from consuming too much of their time,” and their answers are often “impressive-sounding” but not helpful [Cross and Borgatti 2004], resulting in the waste of time of both parties.

Developers may respond to a question not because they want to answer it, but because they do not want to ignore it. Even though helping is costly, taking no action also incurs a social cost. Saying “no” untactfully to an asker deteriorates the expert’s relation with the asker and negatively affects the expert’s social reputation among other peers because such behavior deviates from social norms [Pentland 2005].

The STeP_IN framework provides a communication mechanism called a *dynamic mailing list*; a temporal mailing list is created every time when an information-seeking developer posts a question, with the recipients who are decided dynamically [Ye et al. 2007a]. While the sender’s identity is shown to the recipients, the recipients’ identities are not revealed unless replying to the request. If some of the recipients do not answer, for whatever reasons, nobody will know it; therefore refusing to help becomes socially acceptable, similar to “hiding out to get some work done” [DeMarco and Lister 1999]. If one of the recipient answers the question, his or her identity is revealed to all members of the dynamic mailing list. This asymmetrical information disclosure is meant to reinforce positive social behaviors without forcing others into collaboration.

Principle #9: Expertise communication channels must be socially aware.

Socially aware communication [Pentland 2005] refers to the transmission of information or signals that does not violate social norms. Existing communication channels include face-to-face, direct email, mailing lists, Wikis, bulletin boards, IRC, telephone, or video conferences.

Different communication channels give various degrees of social controls of communication to either the information seeking developer (i.e.,

asker) or those who are asked to provide information (i.e., answerer). Decisions need to be made, depending on the goals and social context, about who should gain the social control of communication.

One prime example of such control is the disclosure of identities of askers and answerers. Table 3 summarizes the pros and cons of revealing identities of the askers and the answerers.

Table 3: Social Control of Communication through the Disclosure of Identities

	pros	cons
<i>Revealing identities of askers</i>	answerers know who is asking	revealing that the asker lacks knowledge
<i>Revealing identities of answerers</i>	motivate the answerers	unduly force the answerers into answering

Different tools take different approaches in designing such disclosure of identities. Answer Garden had an asker’s identity hidden and an answerer’s identity revealed. The field study [Ackerman 1998] has found that the askers felt easier to ask and that the expert answerers felt more “obliged” and tended to “show off” their expertise. STeP_IN [Ye et al. 2007a] in contrast made an asker’s identity revealed to those who receive the question while the receivers’ identities remain hidden unless they answer in a dynamically formulated temporal mailing list. The design decision is based on the viewpoint that the answerer should be granted more control because the asker is the main beneficiary and the answerer is benefactor.

Designing how to disclose what information has been studied in the field of access control. Stevens and Wulf have identified three types of access controls: ex-ante control if the permissions have been defined before the access, uno-tempore control if the permissions are defined at the moment of the access, and ex-post control if the permissions are checked after the access took place [Stevens and Wulf 2002]. Designing socially-aware communication channels may need to consider these three options.

Cohen et al. investigate through field studies of a legal firm the phenomena of adversarial collaboration, where peers who are adversaries having opposing goals, but who nonetheless have to collaborate to get their tasks done [Cohen et al. 2000]. They argue that adversarial collaborations are “the sine qua non of situations that call for the selective dissemination of information.” Although software developers in a project are by no means adversaries having no opposing goals, they may have different in-

terests and motivations in their own specific contexts, especially when a project is inter-organizational or involves subcontracted members. Mechanisms for supporting asymmetric disclosure of information may need to be designed within expertise communication channels.

5. Concluding Remarks

This chapter argues for a developer-centered CSDE where communication is considered as a first-class activity in software development. We identified two distinctive types of communication, *coordination communication* and *expertise communication*, in software development, and elaborated their differences.

Communication support mechanisms have features that imply suitable communication genres [Reder and Schwab 1988]. Such features include whether the communication is one-to-one or one-to-many, whether the communication happens synchronously or asynchronously, whether a sender and recipients are anonymous or identified, whether all the relevant information is disclosed symmetrically or asymmetrically among a sender, recipients, and others, whether the social control of communication is granted to the sender or to the recipient, whether it make it easier for the asker or the recipient, and what media to be used, such as text, voice, video or other types of multimedia, each of which demonstrates different degrees of archivability and searchability.

Taking into total consideration of above features and the distinctive nature of expertise communication in software development, this chapter presents a list of nine design principles for supporting expertise communication in software development. The principles listed above are interdependent on one another, and following one principle may also lead to follow a few other principles, or following one principle may conflict with following another principle. Each principle is important in a particular context. In designing expertise communication support mechanisms, one needs to understand what corporate and organizational culture exists and what types of collaboration their software projects want to nurture.

References

1. Ackerman, M. S., T. W. Malone (1990). Answer Garden: A Tool for Growing Organizational Memory. Proceedings of the ACM Conference on Office Information Systems. Cambridge MA: 31-39.

2. Ackerman, M. S., D. W. McDonald (1996). Answer Garden 2: Merging Organizational Memory with Collaborative Help. *Proceedings of the Conference on Computer Supported Collaborative Work 1996*. New York, ACM Press: 97-105.
3. Ackerman MS (1998) Augmenting organizational memory: a field study of answer garden. *ACM Transaction on Information Systems*, 16:3, 203-224.
4. Arguello, J., Butler, B. S., Joyce, E., Kraut, R., Ling, K. S., Rose, C., Wang, X. (2006) Talk to me: foundations for successful individual-group interactions in online communities. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems (Montreal, Canada, April 22 - 27, 2006)*. R. Grinter, T. Rodden, P. Aoki, E. Cutrell, R. Jeffries, and G. Olson, Eds. CHI '06. ACM, New York, NY, 959-968
5. Augustin L, Bressler D, Smith, G. (2002) Accelerating software development through collaboration. In *Proceedings of the 24th international Conference on Software Engineering*, ACM, New York, NY, 559-563.
6. Beck, K., (1999) *Extreme Programming Explained: Embrace Change*, Addison-Wesley.
7. Biehl, JT, Czerwinski, M, Smith, G, Robertson, GG, (2007) FASTDash: a visual dashboard for fostering awareness in software teams, *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems (CHI '07)* ACM, New York, NY, 1313-1322.
8. Brooks, F. P. J. (1995). *The Mythical Man-Month: Essays on Software Engineering*, 20th Anniversary edition. Reading, MA, Addison-Wesley.
9. Card, S., G. Robertson, Mackinlay, J. D. (1991). The Information Visualizer: An Information Workspace. *Proceedings of Conference on Human Factors in Computing Systems*, ACM Press: 181-188.
10. Cohen, A. L., Cash, D., and Muller, M. J. (2000) Designing to support adversarial collaboration. In *Proceedings of the 2000 ACM Conference on Computer Supported Cooperative Work (CSCW '00)* ACM, New York, NY, 31-39.
11. Coleman, J. C. (1988). Social Capital in the Creation of Human Capital, *American Journal of Sociology* 94: S95-S120.
12. Cross, R., Borgatti, S. P. (2004) The Ties That Share: Relational Characteristics That Facilitate Information Seeking. *Social Capital and Information Technology*. M. Huysman and V. Wulf. Cambridge, MA, The MIT Press: 137-161.
13. Csikszentmihalyi, M. (1990) *Flow: The Psychology of Optimal Experience*. HarperCollins Publishers, New York.
14. Czerwinski, M., Horvitz, E., Wilhite, S. (2004) A diary study of task switching and interruptions, *Proc. CHI 04*, ACM Press, 175-182.
15. Damian, D., Izquierdo, L., Singer, J., and Kwan, I. (2007) Awareness in the Wild: Why Communication Breakdowns Occur. *Proceedings of the international Conference on Global Software Engineering (ICGSE)* IEEE Computer Society, Washington, DC, 81-90.

16. Davor C. Cubranic and G. C. Murphy (2003). Hipikat: Recommending Pertinent Software Development Artifacts. Proceedings of the 25th International Conference on Software Engineering (ICSE03). Portland, OR: 408-418.
17. DeMarco, T., Lister, T. (1999). Peopleware: Productive Projects and Teams. New York, Dorset Housing Publishing.
18. de Souza, C.R.B., Quirk, S., Trainer, E., Redmiles, D. (2007a) Supporting Collaborative Software Development through the Visualization of Socio-Technical Dependencies. GROUP'07, Sanibel Island, FL, 147-156
19. de Souza, C. R. B., Redmiles, D. (2007b) The Awareness Network: Should I display my actions to whom? And, whose actions should I monitor?. Proceedings of the Tenth European Conference on Computer-Supported Cooperative Work (ECSCW2007), 99-117.
20. de Souza, C. R. B., Redmiles, D. (2008) An Empirical Study of Software Developers Management of Dependencies and Changes, Proceedings of the International Conference on Software Engineering (ICSE2008), 241-250.
21. Fairley, R., (1985) Software Engineering Concepts, McGraw-Hill College.
22. Fischer, G., (2000) Symmetry of Ignorance, Social Creativity, and Meta-Design, Knowledge-Based Systems, Vol.13, No.7-8, December, 527-537.
23. Fischer, G., E. Scharff, Ye, Y., (2004). Fostering Social Creativity by Increasing Social Capital. Social Capital. M. Huysman and V. Wulf: 355-399.
24. Flammer, A., (1981) Towards a Theory of Question Asking. Psychological Research, 43: 407-420.
25. Fukuyama, F. (1999). Social Capital and Civil Society. IMF Conference on Second Generation Reforms, Washington, DC.
26. Gerstberger, P.G., Allen, T.J., (1968) Criteria used by research and development engineers in the selection of an information source, Journal of Applied Psychology, Vol.52, No.4, 272-279.
27. Goldhaber, M. H. (1997). The Attention Economy. First Monday 2(4).
28. Gutwin C, Penner R, Schneider, K., (2004) Group awareness in distributed software development, Proceedings of the 2004 ACM conference on Computer supported cooperative work, 72-81
29. Halverson, C. A., Erickson, T., and Ackerman, M. S. (2004) Behind the help desk: evolution of a knowledge management system in a large organization, Proceedings of the 2004 ACM Conference on Computer Supported Cooperative Work (CSCW '04) ACM, New York, NY, 304-313.
30. Herbsleb, J., Grinter, R. E. (1999). Splitting the Organization and Integrating the Code: Conway's Law Revisited. Proceedings of International Conference on Software Engineering (ICSE99): 85-95.
31. Herbsleb, J. Mockus, A. (2003). An Empirical Study of Speed and Communication in Globally-Distributed Software Development, IEEE Transactions on Software Engineering 29(3): 1-14.
32. Horvitz, E., Apacible, J. (2003) Learning and reasoning about interruption. Proc. ICMI 03. ACM Press, 20-27.
33. Humphrey, W., (1989). Managing the Software Process. Massachusetts: Addison-Wesley Professional.
34. Illich, I. (1971). Deschooling Society. New York, Harper and Row.

-
35. Iqbal, S. T., Bailey, B. P., (2006) Leveraging characteristics of task structure to predict the cost of interruption, CHI 06, ACM Press, 741-750.
 36. Jackson T, Dawson R, Wilson, D, (2001) The Cost of Email Interruption, Journal of Systems and Information Technology, vol.5., 81-92.
 37. Ko, A. J. DeLine, R., Venolia, G. (2007). Information Needs in Collocated Software Development Teams. International Conference on Software Engineering (ICSE), May 20-26, 344-353.
 38. Kraut, R., Kiesler, S., Mukhopadhy, T., Scherlis, W., Patterson, M. (1998) Social impact of the Internet: what does it mean?. Commun. ACM 41, 12 (Dec. 1998), 21-22.
 39. LaToza, T. D., G. Venolia, et al. (2006). Maintaining Mental Models: A Study of Developer Work Habits. Proceedings of International Conference on Software Engineering. Shanghai: 492-501.
 40. Maturana, H.R., Varela, F.J., (1992) The Tree of Knowledge: The Biological Roots of Human Understanding, Shambhala Publications, Inc., Boston, MA.
 41. McDonald, D. W. Ackerman, M.S. (1998). Just Talk to Me: A Field Study of Expertise Location. Proceedings of Conference on Computer Supported Cooperative Work (CSCW'98). Seattle, WA: 315-324.
 42. McDonald, D. W. Ackerman, M.S. (2000). Expertise Recommender: A Flexible Recommendation System Architecture. Proceedings of CSCW 2000: 101-120.
 43. Mockus, A., Herbsleb, J. (2002). Expertise Browser: A Quantitative Approach to Identifying Expertise. Proceedings of 2002 International Conference on Software Engineering. Orlando, FL: 503-512.
 44. Nahapiet, J., Ghoshal, S., (1998) Social Capital, Intellectual Capital, and the Organizational Advantage. Academy of Management Review, 23, pp.242-266.
 45. Nakakoji, K., (2006) Supporting Software Development as Collective Creative Knowledge Work, Proceedings of the Second International Workshop on Supporting Knowledge Collaboration in Software Development (KCSE2006), Tokyo, 1-8.
 46. Nakakoji, K., Ohira, M., Yamamoto, Y., (2000) Computational Support for Collective Creativity, Knowledge-Based Systems Journal, Elsevier Science, Vol.13, No.7-8, pp.451-458, December, 2000.
 47. Orlikowski, W. J. (2002). "Knowing in Practice: Enacting a Collective Capability in Distributed Organizing." Organization Science 13(3): 249-273.
 48. Osterweil, L, (1987) Software processes are software too, Proceedings of the 9th international conference on Software Engineering table of contents, pp.2-13, 1987
 49. Pentland, A. (2005) Socially Aware Computation and Communication, Computer 38(3): 33-40.
 50. Perlow, L. A. (1999) The Time Famine: Toward a Sociology of Work Time, Administrative Science Quarterly 44(1): 57-81.
 51. Reder, S. Schwab, R.G. (1988). The Communication Economy of the Workgroup: Multi-Channel Genres of Communication. Proceedings of CSCW1988. New York, ACM Press: 354-368.

52. Redmiles, D., van der Hoek, A., Al-Ani, B., Hildenbrand, T., Quirk, S., Sarma, A., Filho, R. S. S., de Souza, C, Trainer, E., (2007) Continuous Coordination: A New Paradigm to Support Globally Distributed Software Development Projects, *Wirtschaftsinformatik Journal*, Vol.49, S28-S38.
53. Reichling, T. and Veith, M. (2005) Expertise sharing in a heterogeneous organizational environment. *Proceedings of the Ninth Conference on European Conference on Computer Supported Cooperative Work (ECSCW2005)* Springer-Verlag New York, New York, NY, 325-345.
54. Resnick, P. (2002) Beyond bowling together: sociotechnical capital. Carroll, J. M. (Ed.) *HCI in the New Millennium*, 247-272.
55. Ribak, A., Jacovi, M., and Soroka, V. (2002) Ask before you search: peer support and community building with reachout. *Proceedings of the 2002 ACM Conference on Computer Supported Cooperative Work (CSCW '02)* ACM, New York, NY, 126-135.
56. Salancik, G.R., Pfeffer, J. (1978) A Social Information Processing Approach to Job Attitudes and Task Design, *Administrative Science Quarterly* 23, 224-253.
57. Sarma, A, Noroozi Z., van der Hoek, A., (2003) Palantir: Raising Awareness among Configuration Management Workspaces, *Proceedings of Twenty-fifth International Conference on Software Engineering (ICSE 2003)*, 444-454.
58. Shneiderman, Ben (1980) *Software Psychology: Human Factors in Computer and Information Systems*. Cambridge, MA, Winthrop.
59. Soloway, E., Ehrlich, K. (1984) Empirical Studies of Programming Knowledge. *IEEE Transaction on Software Engineering* 10(5): 595-609.
60. Stevens, G. and Wulf, V. (2002) A new dimension in access control: studying maintenance engineering across organizational boundaries. In *Proceedings of the 2002 ACM Conference on Computer Supported Cooperative Work (New Orleans, Louisiana, USA, November 16 - 20, 2002)*. *CSCW '02*. ACM, New York, NY, 196-205.
61. Storey, M., Cheng, L., Bull, I., and Rigby, P. (2006) Shared waypoints and social tagging to support collaboration in software development. *Proceedings of the 2006 20th Anniversary Conference on Computer Supported Cooperative Work (CSCW '06)* ACM, New York, NY, 195-198.
62. Szoestek, A. M. and P. Markopoulos (2006). Factors Defining Face-To-Face Interruptions in the Office Environment. *Proceedings of Conference on Human Factors in Computer Systems*: 1379-1384.
63. Tomayko, J.E., Hazzan, O., (2004) *Human Aspects of Software Engineering (Electrical and Computer Engineering Series)*, Charles River Media, Inc., Rockland, MA.
64. Tyler, J. R. and J. C. Tang (2003) When Can I Expect an Email Response? A Study of Rhythms in Email Usage. *Proceedings of the Eighth European Conference on Computer Supported Cooperative Work (ECSCW2003)*. Helsinki: 239-258.
65. van den Hooff, B., De Ridder, J. A., Aukema, E. J. (2004) Exploring the Eagerness to Share Knowledge: The Role of Social Capital and ICT in Knowl-

-
- edge Sharing. Social Capital and Information Technology. M. Huysman and V. Wulf. Cambridge, MA, The MIT Press: 163-186.
66. Vivacqua, A., Lieberman, H. (2000) Agents to assist in finding help. In Proceedings of the SIGCHI Conference on Human Factors in Computing Systems (CHI '00), ACM, New York, NY, 65-72.
 67. Wagstrom, P. and J. Herbsleb (2006) Dependency Forecasting, CACM 49(10): 55-56.
 68. Ye, Y. (2001). Information-Enriched Workspaces. Proceedings of IFIP TC.13 International Conference on Human-Computer Interaction (INTERACT'01). M. Hirose. Tokyo, Japan, IOS Press: 206-213.
 69. Ye, Y. and G. Fischer (2002). Supporting Reuse by Delivering Task-Relevant and Personalized Information. Proceedings of 2002 International Conference on Software Engineering (ICSE'02). Orlando, FL.: 513-523.
 70. Ye, Y. Yamamoto, Y., Nakakoji, K. (2007a), A Socio-Technical Framework for Supporting Programmers, Proceedings of the 6th Joint Meeting of the European Software Engineering Conference and the Symposium on the Foundations of Software Engineering (ESEC/FSE 2007), ACM Press, Dubrovnik, Croatia, 351-360.
 71. Ye, Y., Yamamoto, Y., Nakakoji, K., Nishinaka, Y., Asada, M., (2007b) Searching the Library and Asking the Peers: Learning to Use Java APIs on Demand. Proceedings of 2007 International Conference on Principles and Practices of Programming in Java. V. Amaral, L. Veiga, L. Marcelino and H. C. Cunningham. Lisbon, Portugal, ACM Press: 41-50.
 72. Ye, Y. Yamamoto, Y., Nakakoji, K. (2008a) Expanding the Knowing Capability of Software Developers through Knowledge Collaboration, IJTPM (International Journal of Technology, Policy and Management), Special Issue on Human Aspects of Information Technology Development, Inderscience Publishers, Vol.8, No.1, 41-58.
 73. Ye, Y. Yamamoto, Y., Nakakoji, K. (2008b) Understanding and Improving Collective Attention Economy for Expertise Sharing, Proceedings of the 20th International Conference on Advanced Information Systems Engineering (CAiSE 2008), Montpellier, France, Lecture Notes in Computer Science, Vol.5074, Springer Berlin Heidelberg, June, 167-181
 74. Ye, Y., Nakakoji, K., Yamamoto, Y. (2008c) The economy of collective attention for situated knowledge collaboration in software development. Proceedings of the 2008 international Workshop on Cooperative and Human Aspects of Software Engineering (Leipzig, Germany, May 13 - 13, 2008). CHASE 08 Workshop, ACM, New York, NY, 109-112