

# Comparison of Coordination Communication and Expertise Communication in Software Development: Their Motives, Characteristics and Needs

Kumiyo Nakakoji<sup>1,2</sup>, Yunwen Ye<sup>3</sup>, Yasuhiro Yamamoto<sup>1</sup>

<sup>1</sup> RCAST, University of Tokyo, Japan

<sup>2</sup> SRA Key Technology Laboratory Inc., Japan

<sup>3</sup> Software Research Associates Inc., Japan

kumiyo@kid.rcast.u-tokyo.ac.jp, ye@sra.co.jp, yxy@kid.rcast.u-tokyo.ac.jp

**Abstract.** Nurturing communication in software development is not about increasing the amount of communication but about increasing the quality of communication experience in the context of software development. Existing studies have shown that different motives and needs are embedded when developers communicate with one another. Identifying *coordination communication* and *expertise communication* as two distinct types of communication, we characterize the difference between the two and discuss important factors to take into account in designing mechanisms to support each type of communication.

**Keywords:** nurturing communication in software development, knowledge collaboration, continuous coordination, unified interface for communication, coordination communication, expertise communication, design considerations

## 1 Introduction

Communication has been taken as an important element in software development. More and more studies argue that socio-technical aspects of software development need to be seriously taken into account in supporting software development. The underlying premise is that peer developers are important knowledge resources in the same way as other artifacts, such as source code, comments, design documents, release notes and bug reports, and that obtaining knowledge and information from their peers is quintessential in software development. Communication should not be regarded as something to get rid of, but instead as something to be nurtured [Nakakoji et al. 2010].

The media currently used in such communication demonstrate a variety of means, including face-to-face, telephone, personal email, mailing-list, Wiki, Internet Relay Chat (IRC), video conferencing, or digital and physical artifacts (e.g., comments inserted in source code or post-it notes pasted on a printed document). Awareness

mechanisms may also be regarded as a form of communication media in the sense that one can obtain information about what other members of the projects are doing. As the communication media vary, styles of communications in software development ranges from indirect to direct, from asynchronous to synchronous, and from intentional to unintentional. It might be one to one, one to designated some, or one to unknown numbers of many.

Such peer-to-peer communication in general aims at sharing information and knowledge about a developer's task at hand. By looking into motives of communicative activities of software developers, we have identified two distinctive types of needs in such communications: *coordination communication* and *expertise communication* [Nakakoji et al. 2010].

In coordination communication, a developer tries to coordinate his or her task with the dependent peers in order to avoid and/or to solve emerging or potential conflicts. In expertise communication, a developer seeks for information to solve his or her task at hand and asks his or her peers for help. Note that by expertise communication, we do not mean that a certain group of developers have general expertise therefore they are to transfer their knowledge to novice developers through communication. In contrast, our view is that expertise is always defined in terms of some context, for instance, in terms of a particular method, of a particular class, of a particular release, or of a particular bug report at a particular point in time; and that expertise is not something definable without context. With this view, each developer has his or her own expertise in some aspects of the system and the project. Expertise communication, therefore, may take place among all members of the peer developers in every direction [Ye et al. 2008].

Developers currently do not distinguish the two types of communication, which are driven by their “information needs” and are carried out through common communication channels. Coworkers were the most frequent source of information by software developers and that two most frequently sought information by software developers that depended on their coworkers were “what have my coworkers been doing?” and “in what situations does this failure occur?” [Ko et al. 2007]. The former information is sought primarily for the purpose of coordinating the work and the latter is for the purpose of getting some knowledge about the source code. Data on three well-known open source projects have shown that text-based communication (mailing lists and chat systems) are the developers’ primary source of acquiring both general knowledge about other developers (to know who has necessary expertise) and specific awareness of who is working on their relevant parts of the system (to coordinate their tasks) [Gutwin et al. 2004].

Developers often mix the two types of communication within a single discourse session without paying an attention to distinguish the two. For instance, a developer John first asks his colleague Mary over the cubicle wall if she knows why the class C calls a method X instead of Y, then Mary answers that it is because Y is planned to be thrown away, and that by the way Mary has just been working on X and checked-in the changes therefore he had better check the latest version of X if he is working on C.

Thus, while the initial question posed by John is expertise communication (i.e., he wanted to ask Mary to give him the answer as to why C calls X instead of Y), the subsequent conversation provided by Mary turns out to be coordination communication (i.e., C that John is working on depends on X that Mary is working on).

Why does it matter then to distinguish the two types of communication if developers do not distinguish the two? It matters because when it comes to design computational mechanisms for supporting communication in software development, each type of communication demands different types of concerns.

This position paper first describes what fundamental differences exist between the two types of communication in software development. We then explain how different aspects need to be considered in designing computational support mechanisms. We conclude with a list of research issues in developing such support.

## **2 Expertise communication and Coordination Communication**

A few distinctive features are involved in each of expertise communication and coordination communication.

Let us first illustrate coordination communication. Suppose a developer X initiates communication with another developer Y, which turns out to be coordination communication. The purpose of the coordination communication is to coordinate tasks to resolve emerging conflicts or to avoid possible future conflicts among the tasks X and Y are engaged in. X and Y are called “socially dependent developers” [de Souza et al. 2007] in the sense that they have to coordinate their tasks through social interactions when the perceived conflicts become necessary to be resolved. X and Y together form an “impact network” [de Souza et al. 2008]. 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 et al. 2008]. X may need to further involve those developers who are part of the impact network.

In contrast, suppose a developer A initiates communication with another developer B, which turns out to be expertise communication. The purpose of the expertise communication is for A to get some information about A’s task at hand. A is asking B to help A by providing some information for A’s particular task. As noted above, we use expertise communication to refer to the kind of activities by seeking information that is essential to accomplish A’s software development activities, not for the purpose of learning, but for the purpose of performing A’s job. If A does not get satisfying information from B, A might need to ask other peers for the same question.

Thus, while the relation between X and Y in the coordination communication is reciprocal, that of A and B in the expertise communication is not. In coordinating

communication, there is a symmetric or reciprocal relation between those who initiate communication and those who are asked to communicate with roughly equal interests and benefits. 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. The benefit would primarily for the communication initiator and the cost (i.e., additional efforts) is primarily paid by those who are asked to participate in the communication; that is, the cost of paying attention to the information request, of stopping his or her own ongoing development task, of composing an answer for the information-seeking developer while collecting relevant information when necessary, and of going back to the original task [Ye et al. 2007].

The role and value of resulting communicative actions would also differ between the two types of communication. When developers communicate with one another, their conversations as well as produced artifacts (mail message contents or white board drawings, for instance) can be stored (if appropriate media is used).

Such recorded communication can be useful if generated through expertise communication. Email exchange about a particular design of a class, for example, would serve as a valuable auxiliary document for the class, and another developer might find it useful to read when using the class at a later time.

Archived communication generated through coordination communication might be useful to inform other developers within the same impact network for the time being. However, the impact network constantly changes over time and such information communicated over a particular class would soon become obsolete. Moreover, coordination communication without its temporal context could be quite harmful when misused. A collection of the coordination communication about a particular object over a long period of time may serve as the object's development log but it would not be more than the existing developmental records captured within current development environments.

Table 1 summarizes the differences of coordination communication and expertise communication.

Table 1: Comparing Coordination Communication and Expertise Communication

	<b>Coordination Communication</b>	<b>Expertise Communication</b>
<i>purpose</i>	to coordinate work	to get information
<i>needs</i>	conflict avoidance, conflict resolution	problem solving
<i>cost &amp; benefit</i>	reciprocal between a communication initiator and the other communication participants	asymmetric between a communication initiator and the other communication participants
<i>expanding participants</i>	when others are part of the impact network	when the initiator could not get satisfying information
<i>recorded communication</i>	useful for the time-being until the impact network changes	become valuable document for later use

The next section compares the different aspects of concerns in designing mechanisms for supporting each type of the communication.

### 3. Different Needs for Supporting the Two Types of Communications

*“A thing is available at the bidding of the user--or could be--whereas a person formally becomes a skill resource only when he consents to do so, and he can also restrict time, place, and method as he chooses” [Illich 1971].*

In talking about depending on teachers as knowledge resources, Illich argued that their willingness to participate is essential in regarding people as information resources. Using peers as potentially relevant information resources is likely to increase cognitive load for both of those who initiate communication, and those who are asked to participate in the communication. Unlike the Help Desk where the jobs of those who are asked is to answer [Ackerman et al. 1990], peer developers are not there to communicate but to perform their own development tasks in a time-critical fashion. They might be willing to communication if they had more time and less stressful situations; they might not otherwise unless they see immediate needs for themselves to communicate.

Therefore, asymmetric nature of the beneficiary and benefactors in expertise communication demands a critical attention in designing communication support

Table 2: Different Present Research Emphases on the Two Types of Communication

	<b>Coordination Communication</b>	<b>Expertise Communication</b>
<i>key concepts</i>	continuous coordination [Redmiles et al. 07] impact management [de Souza et al. 08]	developer as knowledge resources [Nakakoji 06] communication channel [Ye et al. 07]
<i>primary functionality</i>	awareness visualization	finding expertise choosing experts socially-aware communication channel
<i>tools</i>	Palantir [Sarma et al. 03] Ariadne [de Souza et al. 07]	Expert Finder [Vivacqua et al. 2000] Expertise Browser [Mockus et al. 2002] STeP_IN_Java [Ye et al. 2007]
<i>socio-technical aspects</i>	social interaction needs are inferred from the technical (structural) dependencies of the tasks [Herbsleb et al. 1999]	communication participants are selected based on their technical experiences on sought information and previous social relations with an information seeker [Ye et al. 2007]

mechanisms. For an information-seeking developer, involving more participants in the communication means having more potential information resources, implying a better chance of obtaining necessary information at the cost of information overload; thus high quality ranking and triaging mechanisms would become essential. For those who are asked to participate in the communication and provide information, however, responding to the request becomes yet another task [Ye et al. 2007].

On one hand, when the relation between the communication initiator and the rest of the communication participants is symmetrical and reciprocal, those who are asked to participate in the communication would feel the equal importance of engaging in the communication. On the other hand, when the relation is asymmetrical where the initiator would be a beneficiary and the other participants would be benefactors, mechanisms to persuade people to participate in the communication are necessary.

Although there had been no explicit distinctions of the two types of communications in software development, existing research currently demonstrates different emphases on supporting each aspect of the communication with regard to key concepts, tools, and the primary functionality. Both approaches stress the importance of taking socio-technical aspects into account, but in different contexts.

Table 2 illustrates the two distinctive approaches.

Supporting coordination communication has been primarily studied in such research areas as coordinating programmers and programming tasks. Supporting expertise communication has been primarily studied in such research areas as knowledge sharing and experts finding.

Although they do not explicitly use the term coordination communication, Redmiles et al. [2007] present the continuous coordination paradigm for supporting coordination activities in software development. 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 paradigm stresses the importance of integrating the coordination activities within the programming environment, and of making developers aware of the need of communication and simultaneously minimizing the distraction of software developers by using formal configuration management mechanisms and informal visual notification and awareness techniques. They focus on socio-technical factors in the sense that peer-to-peer coordination communication needs are inferred by analyzing structural (technical) dependencies of the system components they are working on because they have to coordinate their tasks through social interactions when the perceived conflicts become necessary to be resolved [Wagstrom et al. 2006; de Souza et al. 2008].

Nakakoji et al. [2010] present nine design guidelines for expert communication support mechanisms. The guidelines state that expert communication support mechanisms should: be integrated with other development activities; be personalized and contextualized for the information-seeking developer; be minimized when other types of information artifacts are available; take into account the balance between the cost and benefit of an information seeking developer and the group productivity; consider social and organizational relationships when selecting developers for communication; minimize the interruption when approaching those who are selected to be communicated with; provide ways to make it easier for developers to ask for help; provide ways to make it easier for developers to answer or not to answer for the information requested; and be socially aware.

The guidelines stress the importance of finding communication participants who not only have necessary information, but are also willing to provide the sought information in an appropriate way in a timely manner. The guidelines also pay an attention to the cost of those who are asked to engage in expertise communication, and argue for the use of socially-aware communication channels. They focus on socio-technical aspect in a sense that finding potential communication participants

takes into account not only technical skills of developers but also their social relationship with the information-seeking developer.

Such differences of the two types of communication necessitate fundamental differences in designing communication support mechanisms in:

- how to select who to participate in communication,
- in what timing to start communication,
- how to invite people to participate in the communication,
- through which communication channel, and
- how to use the resulting communicative session (i.e., communication archives).

Table 3 lists factors that are common and distinctive to the two types of communications in software development.

Table 3: Comparison of Design Factors

	<b>Coordination Communication</b>	<b>Expertise Communication</b>
<i>in relation to the development environment</i>	integrate with the development environment	
<i>disturbance</i>	minimize	
<i>communication needs are identified when:</i>	conflicts are detected or possible conflicts are detected	a developer is in need of information about the task at hand
<i>trade-off of not communicating</i>	potential risks of conflicts that might arise by not coordinating	potential risks of problems when appropriate information is not provided to the information-seeking developer
<i>alternative means to minimize communication</i>	to visualize the status of the potential conflicts so that a developer may not need to engage in explicit communication by glancing at the visualized information	to guide the information seeking developer to relevant artifacts such as source code and documents so that a developer may not need to engage in explicit communication
<i>the use of the object a developer works on</i>	by looking at what objects a developer presently works on in order to infer the impact network	by looking at what objects a developer previously worked on in order to infer the technical expertise of the developer
<i>the use of who is initiating the communication</i>	by using the communication initiator's impact network in selecting communication participants	by using the communication initiator's social relations in selecting communication participants
<i>helping one in initiating communication</i>	mechanisms to switch to an explicit communication mode with the peers in the impact network when urgent communication needs are detected	mechanisms to ask without worrying about bothering peers
<i>helping those who are asked to participate in the communication</i>	mechanisms to judge how urgent and important the conflict is	mechanisms to minimize feeling guilty not responding to the request
<i>communication channel needs to be:</i>	impact-aware so that developers can easily judge and communicate how much impact the emerging conflict might have and how to avoid and solve the conflict.	socially-aware so that developers use the right channel instead of the channel that is easier to use (whom to ask, through which media)

Figure 1 illustrates how communication support mechanisms should be built in support of software developers. On one hand, there should be a unified interactive framework with communication for a software developer that is integrated within a

development environment. They should not be needed to explicitly choose which communication type they would like to be engaging in. Communication with peer developers should be supported as another type of information usage during software development, and needs to be integrated with a program- and document-authoring and browsing environment. On the other hand, how the communication is designed and structured needs to be tuned to each of the two types of communication. What is needed is to take the above differences seriously into account and design the communication support mechanisms accordingly.

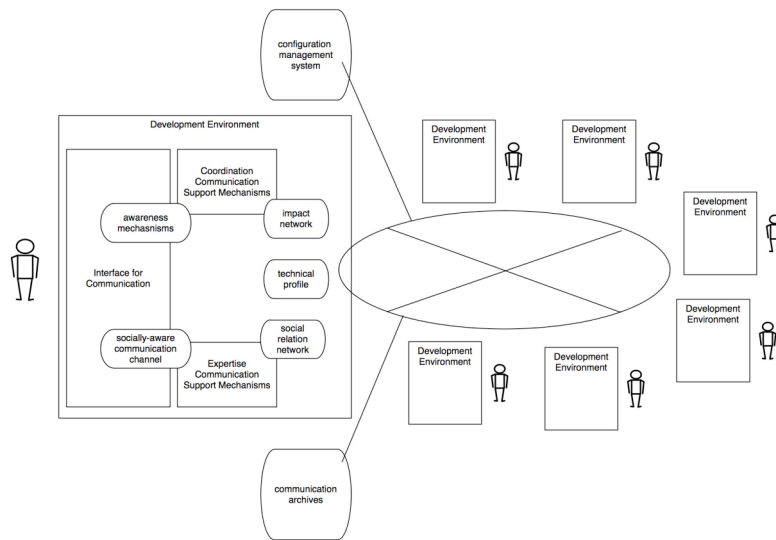


Figure 1: An Architecture of Communication Support Mechanisms that Takes Into Account Two Types of Communication

#### 4. Concluding Remarks

Nurturing communication in software development is not about increasing the amount of communication but about increasing the quality of communication experience in the context of software development. Although having been recognized merely as communicative acts, different motives and needs are embedded when developers communicate with one another. Different computational mechanisms are necessary to realize successful communication. This paper presents our initial attempt to list different aspects necessary to take into account in designing mechanisms to support each of the coordination communication and expertise communication. There are no general communication needs but either coordination communication needs or expertise communication needs. A real challenge would be to design a developer-centered unified interactive framework that seamlessly integrates the two.

## References

- [Ackerman et al. 1990] Ackerman, M. S., T. W. Malone, Answer Garden: A Tool for Growing Organizational Memory. Proceedings of the ACM Conference on Office Information Systems. Cambridge MA: 31-39, 1990.
- [de Souza et al. 2007] de Souza CRB, Quirk S, Trainer E, Redmiles D: Supporting collaborative software development through the visualization of socio-technical dependencies. In: Proc. of GROUP'07, pp 147–156, 2007.
- [de Souza et al. 2008] de Souza CRB, Redmiles D: An empirical study of software developers management of dependencies and changes. In: Proc. of ICSE'08, pp 241–250, 2008.
- [Gutwin et al. 2004] Gutwin C, Penner R, Schneider, K., Group awareness in distributed software development, Proceedings of the 2004 ACM conference on Computer supported cooperative work, 72-81, 2004.
- [Herbsleb et al. 1999] Herbsleb, J., Grinter, R. E. Splitting the Organization and Integrating the Code: Conway's Law Revisited. Proceedings of International Conference on Software Engineering (ICSE99): 85-95, 1999.
- [Illich 1971] Illich, I., Deschooling Society. New York, Harper and Row, 1971.
- [Ko et al. 2007] Ko AJ, DeLine R, Venolia G: Information needs in collocated software development teams. In: Proc. Of ICSE'08, pp 344–353, 2007.
- [Mockus et al. 2002] Mockus A, Herbsleb J: Expertise Browser: A quantitative approach to identifying expertise. In: Proc. of ICSE'02, pp 503–512, 2002.
- [Nakakoji 2006] Nakakoji K: Supporting software development as collective creative knowledge work. In: Proc. of KCSD2006, Tokyo, pp 1–8, 2006.
- [Nakakoji et al. 2010] Nakakoji, K., Ye, Y., and Yamamoto, Y.: Supporting expertise communication in developer-centered collaborative software development environments. In: Finkelstein, A., van der Hoek, A., Mistrik, I., Whitehead, J. (eds) Collaborative Software Engineering, Springer-Verlag, 2010 (forthcoming).
- [Redmiles et al. 2007] Redmiles D, van der Hoek A, Al-Ani B, Hildenbrand T, Quirk S, Sarma A, Filho RSS, de Souza C, Trainer E: Continuous coordination: a new paradigm to support globally distributed software development projects. *Wirtschaftsinformatik J*, 49: S28–S38, 2007.
- [Sarma et al. 2003] Sarma, A, Noroozi Z, van der Hoek, A.: Palantir: raising awareness among configuration management workspaces. In: Proc. of ICSE'03, pp 444–454, 2003
- [Vivacqua et al. 2000] Vivacqua A, Lieberman H: Agents to assist in finding help. In: Proc. of CHI'00, pp 65–72, 2000
- [Wagstrom, et al. 2006] Wagstrom, P. and J. Herbsleb, Dependency Forecasting, *CACM* 49(10): 55-56, 2006.
- [Ye et al. 2007] Ye Y, Yamamoto Y, Nakakoji K: A socio-technical framework for supporting programmers. In: Proc. of ESEC/FSE'07, pp 351–360, 2007.
- [Ye et al. 2008] Y. Ye, Y. Yamamoto, K. Nakakoji, 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, pp.41-58, 2008.