To appear in New Frontiers in Artificial Intelligence: JSAI-isAI 2010 Workshops, LNAI Series, Springer, fall 2010.

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

Kumiyo Nakakoji[1], Yunwen Ye[1], Yasuhiro Yamamoto[2]

[1]Key Technology Laboratory, SRA Inc., Japan
[2] Precision and Intelligence Laboratories, Tokyo Institute of Technology, Japan

kumiyo@sra.co.jp, ye@sra.co.jp, yxy@acm.org

Abstract. The research question we pursue is how to go beyond existing communication media to nurture communication in software development. Nurturing communication in software development is not about increasing the amount of communication but about increasing the quality of the communication experience in the context of software development. Existing studies have shown that different motives and needs are inherent when developers communicate with one another. Identifying *coordination communication* (*c-comm* for short) and *expertise communication* (*e-comm*) 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, c-comm, expertise communication, e-comm, design considerations

1 Introduction

Communication has been regarded as an important element in software development. Increasingly 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 peers is quintessential in software development. Communication should not be regarded as something to eschew, but instead as something to be nurtured [10].

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 communication media vary, styles of communication in software development range from indirect to direct, from asynchronous to synchronous, and from intentional to unintentional. It might be one to one, one to a designated some, or one to unknown numbers of many.

Most of communication media that software developers currently use have been built for general purposes (with few exceptions such as Wiki). The goal of our research is to design innovative communication media to nurture communication for software developers. Nurturing communication in software development is not about increasing the amount of communication but about increasing the quality of the communication experience in the context of software development. The primary task of a software developer is to develop software, and not to communicate. Communicative activities should be seamlessly integrated within the context of software development activities. Communication is a means, not a goal.

In order to address the research question of how to go beyond existing communication media to nurture communication in software development, we need to better understand *why* software developer communicate with each other. By looking into the motives of communicative activities of software developers, we have identified two distinctive types of needs in such communications: *coordination communication* and *expertise communicatio*n [10].

In coordination communication, or *c-comm* for short, a developer tries to coordinate his or her task with dependent peers in order to avoid and/or to solve emerging or potential conflicts. In expertise communication, or *e-comm* for short, a developer seeks information to solve his or her task at hand and asks peers for help. Note that by expertise communication, we do *not* mean that a certain group of developers who have general expertise thereby 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, a particular class, a particular release, or a particular bug report at a particular point in time; and that expertise is not something definable without context. In 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 of the peer developers in every direction [16].

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 for software developers, and the two types of information most frequently sought by software developers from their coworkers were “What have my coworkers been doing?” and “In what situations does this failure occur?” [7]. 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) is the developers' primary source of acquiring both general knowledge about other developers (who has the necessary expertise) and specific awareness (who is working on their relevant parts of the system—to coordinate their tasks) [4].

It seems that developers often mix the two types of communication within a single discourse session without paying any attention to distinguishing the two. For instance, developer John first asks his colleague Mary over the cubicle wall whether she knows why class C calls a method X instead of Y; then Mary answers that it is because Y is designed to be thrown away, and that, by the way she has just been working on X and checked-in the changes, so he had better check the latest version of X if he is working on C. Thus, while the initial question posed by John is e-comm (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 c-comm (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 them? It matters because when it comes to design computational mechanisms for supporting communication in software development, each type of communication demands different types of design concerns.

In this paper, we 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 to be considered in developing such support.

2 Expertise Communication and Coordination Communication

A few features distinguish e-comm (expertise communication) from c-comm (coordination communication).

We first illustrate c-comm. Suppose developer X initiates communication with developer Y, which turns out to be c-comm. The purpose of the c-comm is to coordinate tasks to resolve emerging conflicts or to avoid possible future conflicts among the tasks in which X and Y are engaged. Developers X and Y are called "socially dependent" [2] in the sense that they have to coordinate their tasks through social interactions when it becomes necessary to resolve the perceived conflicts. X and Y together form an "impact network" [3]. 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" [3]. X may need to further involve those developers who are part of the impact network.

In contrast, suppose developer A initiates communication with developer B, which turns out to be e-comm. The purpose of this e-comm 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 earlier, e-comm refers to the activities to seek 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 the same question.

Thus, while the relation between X and Y in the c-comm is reciprocal, that of A and B in the e-comm is not. In c-comm, 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 e-comm, 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., the additional effort) 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 their own ongoing development task; of composing an answer for the information-seeking developer, including collecting relevant information when necessary; and of going back to the original task [15].

The role and value of the 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 e-comm. Email exchange about a particular design of a class, for example, would serve as a valuable auxiliary document for the class because another developer might find it useful to read when using the class at a later time.

Archived communication generated through c-comm 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 may soon become obsolete. Moreover, c-comm 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 between c-comm and e-comm.

Table 1: Comparing Coordination Communication (c-comm) and Expertise Communication (e-comm)

	Coordination Communication (c-comm)	**Expertise Communication (e-comm)**
purpose	to coordinate work	to get information
needs	conflict avoidance, conflict resolution	problem solving
cost & benefit	reciprocal between a communication initiator and the other communication participants	asymmetric between a communication initiator and the other communication participants
expanding participants	when others are part of the impact network	when the initiator could not get satisfactory information
recorded communication	useful for the time being until the impact network changes	becomes valuable for later use

The next section compares the different aspects of concern in designing mechanisms for supporting each of the two types of 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 [6].

In talking about depending on other people, such as teachers, as knowledge resources, Illich argued that their willingness to participate is essential in regarding them as information resources. Using peers as potentially relevant information resources is likely to increase the cognitive load for both those who initiate communication and those who are asked to participate in the communication. Unlike a Help Desk, where it is the job of those who are asked to answer [1], peer developers are there not to communicate but to perform their own development tasks in a time-critical fashion. They might be willing to communicate if they had more time and less stressful situations; otherwise, they might not be willing unless they see an immediate need to communicate.

Therefore, the asymmetric nature of the beneficiary and benefactors in e-comm demands critical attention in designing communication support mechanisms. For an

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

	Coordination Communication (c-comm)	**Expertise Communication (e-comm)**
key concepts	continuous coordination [11] impact management [3]	developer as knowledge resources [9] communication channel [15]
primary functionality	awareness visualization	finding expertise choosing experts socially-aware communication channel
tools	Palantir [12] Ariadne [2]	Expert Finder [13] Expertise Browser [8] STeP_IN_Java [15]
socio-technical aspects	social interaction needs are inferred from the technical (structural) dependencies of the tasks [5]	communication participants are selected based on their technical experiences on sought information and previous social relations with an information seeker [15]

information-seeking developer, involving more participants in the communication means having more potential information resources, implying a better chance of obtaining the necessary information but 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 [15].

On the 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 an equal importance of engaging in the communication and would therefore participate. 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 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 c-comm has been studied primarily in such research areas as coordinating programmers and programming tasks. Supporting e-comm has been studied primarily in such research areas as knowledge sharing and expert finding.

Although they do not explicitly use the term "coordination communication," Redmiles et al. [11] present the continuous coordination paradigm for supporting coordination activities in software development. The paradigm contains the following four principles: (1) to have multiple perspectives on activities and information; (2) to have nonobtrusive integration through synchronous messages or through the representation of links between different sites and artifacts; (3) to combine socio-technical factors by considering relations between artifacts and authorship so that distributed developers can infer important context information; and (4) to integrate formal configuration management and informal change notification via the use of visualizations embedded in integrated software development environments [11].

This paradigm stresses the importance of integrating coordination activities within the programming environment, and of making developers aware of the need for communication and simultaneously minimizing the distraction of software developers by using formal configuration management mechanisms and informal visual notification and awareness techniques. Redmiles et al. (2007) 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 on which developers are working because they have to coordinate their tasks through social interactions when the resolution of perceived conflicts becomes necessary [3], [14].

Nakakoji et al. [10] present nine design guidelines for expertise communication support mechanisms. The guidelines state that expertise 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 group productivity, consider social and organizational relationships when selecting developers for communication, minimize the interruption when approaching those who are selected for communication, 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 the information request, and be socially aware.

The guidelines presented by Nakakoji et al. [10] 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 attention to the cost to those who are asked to engage in expertise communication, and argue for the use of socially aware communication channels. They focus on the socio-technical aspect in the 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.

Each approachs take socio-technical aspects into account differently. Research on c-comm focuses on *socio-technical congruence*, where the structural similarities between an organizational structure and software structure are primarily studied. Research on e-comm focuses on a *socio-technical space*, where social relations among developers are considered in finding communication partners who would be willing to engage in the communication.

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

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

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

Table 3: Comparison of Design Factors

	Coordination Communication	**Expertise Communication**
in relation to the development environment	integrate with the development environment	
disturbance	minimize	
when communication needs are identified	conflicts are detected or possible conflicts are detected	a developer is in need of information about the task at hand
trade-off of not communicating	potential risks of rework caused by conflicts that might arise by not coordinating	potential risks of slowing work when appropriate information is not provided to the information-seeking developer
alternative means to reduce communication	to visualize the status of the potential conflicts so that by glancing at the visualized information a developer may not need to engage in explicit communication	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
the use of the object on which a developer is working	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
the use of who is initiating the communication	by using the communication initiator's impact network in selecting communication participants	by using the communication initiator's social relations in selecting communication participants
helping one in initiating communication	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
helping those who are asked to participate in the communication	mechanisms to judge how urgent and important the conflict is	mechanisms to minimize feeling guilty for not responding to the request
awareness of communication channel	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 the one hand, there should be a unified interactive framework with communication for the software developer that is integrated within a development environment. Developers should not need to explicitly choose which communication type in which they would like to be engaging. 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.

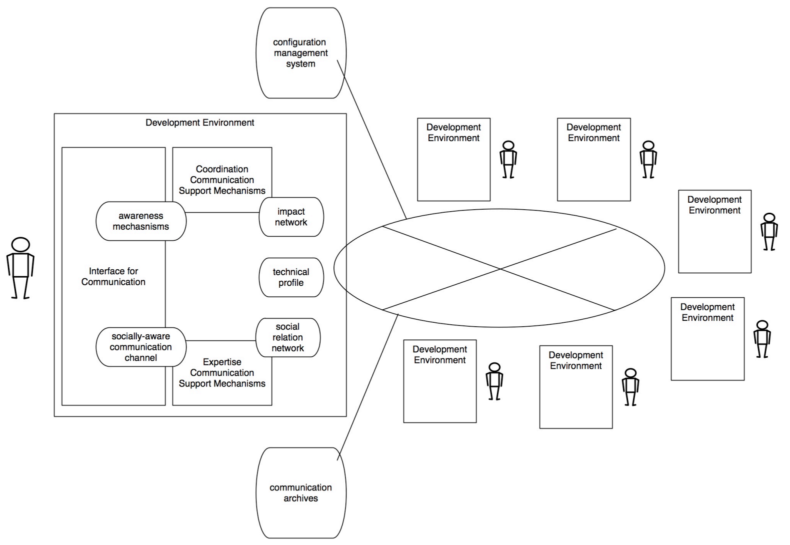

Figure 1: An Architecture of Communication Support Mechanisms
that Takes into Account Two Types of Communication

4. Concluding Remarks

Nurturing communication in software development is not about increasing the amount of communication but about increasing the quality of the communication experience in the context of software development. Although having been recognized merely as communicative acts, different motives and needs are inherent 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 coordination communication and expertise communication. As opposed to general communication needs, there are 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

1. Ackerman M S, Malone T W: Answer Garden: a tool for growing organizational memory. Proceedings of the ACM Conference on Office Information Systems. Cambridge, MA, pp 31--39 (1990)
2. [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. Proceedings of GROUP'07, pp 147–156 (2007)
3. [de Souza & Redmiles 2008] de Souza CRB, Redmiles D: An empirical study of software developers management of dependencies and changes. Proceedings of International Conference on Software Engineering (ICSE'08), pp 241--250 (2008)
4. [Gutwin et al. 2004] Gutwin C, Penner R, Schneider, K.: Group awareness in distributed software development, Proceedings of the ACM Conference on Computer Supported Cooperative Work (CSCW2004), pp 72--81 (2004)
5. [Herbsleb & Grinter 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), pp 85--95 (1999)
6. [Illich 1971] Illich, I., Deschooling Society. Harper and Row (1971)
7. [Ko et al. 2007] Ko AJ, DeLine R, Venolia G: Information needs in collocated software development teams. Proceedings of International Conference on Software Engineering (ICSE’07), pp 344--353 (2007)
8. [Mockus & Herbsleb 2002] Mockus A, Herbsleb J: Expertise Browser: a quantitative approach to identifying expertise. Proceedings of International Conference on Software Engineering (ICSE'02), pp 503--512 (2002)
9. [Nakakoji 2006] Nakakoji K: Supporting software development as collective creative knowledge work. Proceedings of International Workshop on Knowledge Collaboration in Software Development (KCSD2006), Tokyo, pp 1--8 (2006)
10. [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, Chapter 11, Springer-Verlag, January (2010)
11. [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)
12. [Sarma et al. 2003] Sarma A, Noroozi Z, van der Hoek A: Palantir: raising awareness among configuration management workspaces. Proceedings of International Conference on Software Engineering (ICSE'03), pp 444--454 (2003)
13. [Vivacqua & Lieberman 2000] Vivacqua A, Lieberman H: Agents to assist in finding help. Proceedings of Human Factors in Computing Systems (CHI'00), pp 65--72 (2000)
14. [Wagstrom & Herbsleb 2006] Wagstrom P, Herbsleb J: Dependency forecasting, Communications of ACM 49(10): 55--56 (2006)
15. [Ye et al. 2007] Ye Y, Yamamoto Y, Nakakoji K: A socio-technical framework for supporting programmers. Proceedings of ESEC/FSE'07, pp 351--360 (2007)
16. [Ye et al. 2008] Ye Y, Yamamoto Y, Nakakoji K: Expanding the knowing capability of software developers through knowledge collaboration, International Journal of Technology, Policy and Management (IJTPM), Special Issue on Human Aspects of Information Technology Development, Interscience Publishers, 8(1): 41-58 (2008)