Reducing the Cost of Communication and Coordination in Distributed Software Development

Yunwen Ye[1,3], Kumiyo Nakakoji[1,2], and Yasuhiro Yamamoto[2]

[1] SRA Key Technology Laboratory, Inc.
3-12 Yotsuya, Shinjuku, Tokyo 160-0004, Japan
[2] Research Center for Advanced Science and Technology, University of Tokyo
4-6-1 Komaba, Meguro, Tokyo, 153-8904, Japan
[3] Center for LifeLong Learning and Design, University of Colorado at Boulder
Boulder, CO80309-0430, USA
yunwen@colorado.edu; kumiyo@kid.rcast.u-tokyo.ac.jp;
yxy@kid.rcast.u-tokyo.ac.jp

Abstract. Decades of software engineering research have tried to reduce the interdependency of source code to make parallel development possible. However, code remains helplessly interlinked and software development requires frequent formal and informal communication and coordination among software developers. Communication and coordination cost still dominates the cost of software development. When the development team is separated by oceans, the cost of communication and coordination increases dramatically. To better understand the cost of communication and coordination in software development, this paper proposes to conceptualize software as a knowledge ecosystem that consists of three interlinked elements: *code*, *documents*, and *developers*. This conceptualization enables us to understand and pinpoint the social dependency of developers created by the code dependency. We show that a better understanding of the social dependency would increase the economic use of the collective attention of software developers with a proposed new communication mechanism that frees developers from the overload of communication that does not interest them, and thus reduces the overall cost of communication and coordination in software development.

Keywords: distributed software development; knowledge collaboration; cost of communication and coordination; attention cost.

1 Introduction

When a large software project is created by developers separated by oceans and time zones, communication and collaboration becomes the more dominant forces in determining the productivity and quality of software development [13]. Most of the current research in supporting offshore outsourcing software development has mainly focused on the brawny power brought by *many hands* through collaboration. The major concerns have been on the cooperation, communication, and coordination problems brought by the consequences of division of labor [12, 34].

This paper focuses on another aspect of collaboration in offshore outsourcing software development that has not been paid enough attention—the brainy power brought by *multiple heads* of software developers; that is, the knowledge collaboration that takes place during the process of software development.

Software development is a knowledge-intensive and creative activity [22]. It requires knowledge from several different domains, including both the computing domain and the application domain. As computer applications get larger and more complex, the amount and kinds of knowledge required grow [38]. Few developers, if any, have all the knowledge needed in their own heads. The knowledge necessary for software development is distributed between the developer and the external world, and the development of a software system requires learning from and integrating the knowledge from various external sources in the world. Knowledge in the world comes from cognitive tools that support programming and from peer developers. The development of software is therefore no longer confined to an individual developer but has to rely on distributed cognition by reaching into a complex networked world of information and computer mediated collaboration.

With the current trend of offshore outsourcing, software projects are increasingly become distributed along different times zones, locations, and cultures. The distribution of software projects has become necessary due to not only the needs of shifting labors to places that have lower costs, but also the pursuit of local talents that are otherwise unavailable. In other words, in addition to delegating the task of development to the most economically viable places, which is the current driving force of outsourcing, software development companies need also to ship the task to the most talented and suitable people regardless of location, time zone, and national boundary. This, we strongly believe, will soon be the upcoming driving force of offshore outsourcing. Knowledge-based collaboration is becoming as important as, if not more than, the current labor-based collaboration in outsourcing.

2 Knowledge Distribution and Collaboration

Software development involves the application of knowledge from a variety of sources, which are constantly changing. For example, application domains are subject to rapid change; a vast amount of third-party libraries are continually updated; new features and functionalities continue to be introduced in programming tools and environments. Software development therefore is a continual learning process during which developers have to constantly acquire new knowledge [39].

The knowledge required in software development is not only about the process knowledge and domain knowledge that are applied in the software system; it also includes knowledge about the software system itself that developers are currently creating. One may argue that since the software developer participates in the creation of the system, he/she should know the system inside out. However, because large systems are created collaboratively by many developers, not all developers, if any, would have complete knowledge about the whole system. Meanwhile, with the increasingly accepted view of software systems as evolving entities, the percentage of incremental, continuous development in software has risen quickly. Such software systems need to be continuously developed with iterative processes. Coupled with the high turnover rate in software industry, many software developers find themselves working to make incremental changes to systems that have been partially developed. This is especially true in offshore outsourcing software development: local developers do not have the global knowledge of the whole system.

Software development, therefore, should be viewed as a distributed cognitive activity [10, 16]. The overall capability of a project team, termed as *group knowing*, is determined not only by the sum of the capability of individual developers, but also by the socio-technical environment consisting of the developers, tools, and accessible communication channels that affect how they contribute their knowledge to the project and how they collaborate with each other.

In offshore outsourcing development where software developers are dispersed geographically, they lost the opportunities of spontaneous and informal fact-to-face communication that has been shown critical in sharing context awareness and knowledge in software development [18]. The lost of communication opportunities, however, is not unique to offshore development; it is similarly detrimental to large software projects where all developers cannot be collocated in closeness. Allen [1] reported that when engineers' office were about 30 meters or more apart, the frequency of informal communication dropped to nearly the same level as people with offices separated by many miles.

The key challenge of supporting offshore development, therefore, lies not in developing tools that make offshore development same as same-site development, but in seamless integration of individual knowledge to enhance the group knowing regardless of location. Software developers, especially in offshore development, do not have a uniformed knowledge structure; each of them has a unique set of skills and expertise. The key is how to integrate this diversity of expertise and synthesize it into the group knowing of a software project team through knowledge collaboration in which ideas and inspiration cross fertilize and feed on each other.

3 Software = Code + Documents + Developers

3.1 Knowledge Resources for Software Development

As a knowledge artifact, software code is the ultimate knowledge resource about the system. Due to the essential invisibility of software code [3], however, it is not easy to recover knowledge about the system by simply reading the code. It has long been recognized that documents that provide high level descriptions of the code and the design rationale are needed to coordinate the development.

Code and documents, however, are often insufficient for supporting knowledge collaboration. Documents do not always exist, or quickly fall out of sync with the code. Much of the knowledge about the code and the design decisions remain in the head of developers. Many empirically studies have shown that software developers routinely access their peer developers for knowledge during the development process through informal communications [19, 21]. Peer developers remain the most commonly used and valued sources of expertise in software projects [32].

3.2 Software Project as an Evolving Knowledge Ecosystem

We conceptualize a software project as a self-organizing and evolving knowledge ecosystem [4] that consists of three interlinked elements: *code*, *documents* and *developers*. In such a knowledge ecosystem, knowledge is embedded in both its constituting elements and its structure that regulates their inter-relationship, and flows along the hyperlinked relationship. As developers create artifacts (code and documents), their knowledge gets distilled into the artifacts. Knowledge gets shared, exchanged and combined through the dynamic interactions between software developers, mediated by code, document, and communications.

This conceptualization enables us to model a software project as a socio-technical information space that has triangulated relationships among code parts, documents and developers. The nodes that constitute the socio-technical information space associated with a software system include not only parts of code at different levels of granularities, but also the documents that have been generated during the development process, as well as the developers that hold knowledge about them. Code, documents, and developers are therefore equally important knowledge resources that should be utilized during software development.

In this knowledge ecosystem, relationships among code, documents and developers dynamically change as the development process proceeds. The interacting developers form a knowledge community, defined as a group of people who collaborate with one another for the construction of knowledge artifacts. In a knowledge community, people are bonded through the construction of common artifacts. This is especially true in the case of offshore outsourcing development because, unlike a collocated software project in a single organization, those developers often do not have a shared identity defined by their shared belongingness to the organization. In most cases, they have different organizational and cultural identities [8]; and when they come together for a software project, they are bonded by the needs of constructing a common artifact.

3.3 Evolution in Software Projects

The knowledge community aspect has important implications when viewing software development as collective creative knowledge work that depends on the learning of developers through knowledge collaboration. The roles of individual developers, both formally assigned ones and informally perceived ones, change over time during a project. The social relationships among the developers grow through their engagement in the project, affecting how they collaborate, communicate, and coordinate with each other, which results in different ways of sharing and integrating knowledge.

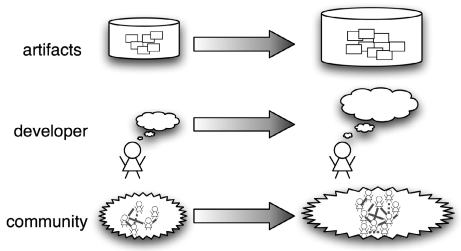


Fig. 1. Software Project as a Knowledge Ecosystem

All three elements constantly evolve during the process of software development (Fig. 1). Artifacts (code and elements) change over time throughout the development. Individual developers—or, more precisely, what individual developers know—grow by gaining experience through the engagement with artifacts and peer developers. The community of developers changes when new members join, old members leave, both the assigned or perceived roles of members change, and members' relationships change.

Existing studies on understanding and supporting software evolution have primarily focused on the evolution of artifacts. More recent work has started to look at how individuals change through learning about the system. People learn by reading source code and documents, and they learn by asking peers questions. They also learn by solving new problems and experiencing unfamiliar situations. Their old knowledge is replaced with new knowledge and is restructured during the development process. A community evolves through individual activities in software development that result in the change of software artifacts and/or in the individual growth of knowledge about the system. This paper views the evolutionary process of the developer community and software systems from the following three relationships (Fig. 2):

(1) *The relationship of an individual with artifacts.* How one relates with artifacts is concerned with what knowledge, expertise, and experience the individual has on what artifacts. This information is useful in identifying a set of people who are likely to have expertise with a certain artifact.
(2) *The relationship of an individual with other developers.* How one relates with other developers impacts knowledge collaboration among developers. This information helps a developer determine whom to ask for help about a certain artifact as well as decide whether and how to respond to a question posed by an asker.
(3) *The relationship of an individual with the community as a whole.* How one relates with the community is concerned with that individual's role within the community: whether he/she is a peripheral member, a core member, or a member in between. This relationship helps a developer decide how much he/she should contribute to the community by gaining trust and social reputation within the community.

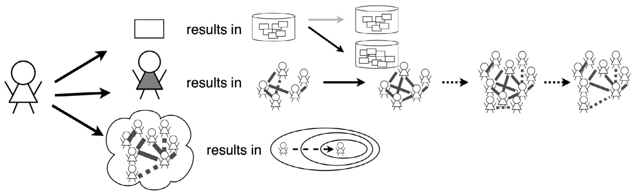


Fig. 2. Evolutionary Process in a Software Project

3.4 Socio-technical Costs in Knowledge Collaboration

When peer developers become critical resources for expertise, simply knowing who has the knowledge is not enough. The knowledge seeker needs to contact the knowledge providers and ask the question, and the knowledge providers then have to consent to engage in knowledge collaboration activities [17]. These steps become exceptionally costly in a globally distributed development project because developers in one site often do not "know" about those located in another site.

This knowledge collaboration act is affected by and affects the characteristics of the social relationship between developers and of their relationships with the community. The communication channels used, the contents of the question and answer, the ways the questions is asked and the answers provided, as well as the timing of asking and answering depends on a set of perceived social variables.

All the communication and coordination required for knowledge collaboration among developers come with a great cost that demands attention and time that can otherwise be used for development [19]. The technology used in supporting knowledge collaboration could affect positively or negatively of the perception of social variables, and the associated total cost of communication and coordination [33]. From the socio-technical perspective [24], we analyze those social factors that affect knowledge collaboration behaviors and cost (both social and attentional).

Awareness. For a developer to seek external expertise from peers, he/she has to know who might have the expertise. From a set of potential expertise providers, he/she needs to choose whom to ask, and then make the decision to ask. This conscientious decision making process is related to the following social and technical factors.

The asker needs to find where the needed expertise is located, and who potentially has the expertise. Previous research has shown that such transactive knowledge takes extensive time to develop [21, 30], and its utilization consumes intensive attentional cost [23]. The geographical distance in offshore development lowers significantly the knowledge of knowing who are the experts [18].

Asking a question shows that the developer is missing some knowledge, and he/she risks of appearing ignorant that impacts his/her overall relationship with the community. People demonstrate different asking behaviors when they are in public or in private; to a stranger or to a friend. Generally speaking, people are more willing to ask questions covertly to those that they are closely related because the close social relationship provides a psychological safety of admitting a lack of knowledge [6].

Asking question is also challenging because the expertise seeker needs to assess the reliability of and then understand the answer. Research has shown that a strong tie between the expertise seeker and the provider resulted from previous interactions leads to easier quality judgment and helps the interpretation of answers [29].

Asking. When a developer decides to ask a question, he/she needs to make contact with the experts. A study by Herbsleb and Grinter [14] 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. When collocation is replaced with remote communication tools in offshore development, initiating a contact became more difficult due to the loss of such social cues.

The way that the question is presented has a direct impact on the response it will receive. Rhetorical strategies, linguistic complexity and word choice of the question all influence the likelihood of others responding to a question [2]. The needs for a developer to seek expertise mostly arise from a problematic situation that needs to be resolved in a specific timeframe. The expectation of how soon a help would come is shaped by a history of interactions with the other party [36].

Engagement. Upon receiving a question, the experts need to decide whether to engage in collaboration with the expertise seeker based on social factors: their perceived social relationship with the expertise seeker and the community at large.

The theory of social capital provides an analytic framework to understand this decision-making process. 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 [25]” and is regarded as important as financial capital and intellectual capital for an individual as well as a social organization because it promotes cooperation and reduce transaction cost. Social capital manifests itself in forms of obligations, expectations, trust, norms of generalized reciprocity, and reputations.

Social capitals are derived from social interactions. If *A* helps *B*, *A* then holds a credit slip for *B* to reciprocate the favor in the future. In other words, *A* can have a reasonable expectation that *B* will do something for him or her down the road, and *B* will feel an obligation to help *A* [5]. Regularly reciprocated fulfillment of obligations leads to the development of trust among the interacting parties. When this direct interpersonal reciprocity becomes a norm, it promotes generalized reciprocity. Persons with a large amount of credit slips are easier to draw collaboration in the social unit. The norm would also apply social pressure for those who have a large amount of obligations to engage actively in collaboration with others.

Engagement consumes time and attention. No action, however, has social cost too. Saying no untactfully to a peer who seeks for your help deteriorate your relation with him or her, and affects negatively your social reputation among other peers because it deviates from social norms.

Collective Attention Cost. Asking and answering a question takes cost. In addition to the time and attention for the asker to formulate and compose the question, and the expertise provider to read, think and post the reply, considerable collective cost is also incurred.

All the people who have received the question would at least spend some attention about the question before they decide to answer or not. When the number of people who receive the question becomes large, the collective attention consumed also becomes considerably large. Given the fact that we are now 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 [11], it is imperative for system designers to take this factor into consideration because the project has a limited supply of collective attention and should be used economically.

A question means an interruption. The cost of interruption includes both the loss of working context and the destruction of flow [35]. When multiple project members receive the request for help, for example, if the request is sent through the project mailing list, this interruption cost is multiplied with the number of receivers. Collectively, this cost might outweigh the benefits of knowledge collaboration and decreases the overall productivity of the whole project [33]. Communication mechanisms used for knowledge collaboration have to be carefully designed and chosen by paying attention that the communication would not impact negatively the overall performance of the project team.

4 A Socio-technical Framework to Supporting Knowledge Collaboration

We have developed the Dynamic Community framework to help software developers engage in knowledge collaboration during the process of software development through sharing and exchanging expertise required for the project. The goals of the framework are twofold: to increase the ease of accessing external expertise either through a knowledge repository or from peer experts, and at the same time to reduce the total cost of experts being interrupted and that of providing help. The essential guidelines of the Dynamic Community framework are:

(1) Expertise is not an absolute attribute but a relative attribute of a developer. A group of experts can be identified only after the task is known.
(2) Knowledge collaboration is not the goal; it is only the means to support developers to solve their current task. The social and technical cost of knowledge collaboration has to be balanced with the primary goal: to improve the productivity of the team.
(3) Existing social relationships among developers play an important role and should be taken into consideration to facilitate knowledge collaboration.
(4) The success of one knowledge collaboration transaction should not come at the price of developers' reluctance of further participation in future knowledge collaboration. The goodwill and limited attention of experts should be economically utilized to achieve sustainable and long term success. Rather than focusing only on the success of one act of knowledge collaboration; we focus on the sustainability of knowledge collaboration because it has to recur repeatedly during the whole lifecycle of the project.
(5) Social support is costly and should only be used as a back up mechanism for technical approaches.

4.1 Modeling the Knowledge Ecosystem of a Software Project

The knowledge elements in a software project create a knowledge ecosystem with complicated interdependency. It consists of a group of developers, their code, related documents, and the relationships among them (Fig. 3). Three kinds of relationships exist: those between programmers, those between a programmer and information, and those between information. We use the term information to refer to both code and documents (such as design documents, configuration management logs, bug reports, and email archives that are associated with the development of the code). The relationship between programmers captures the social relationship between them, including who helped whom, and who sent emails to whom, as well as their social dependency derived from the

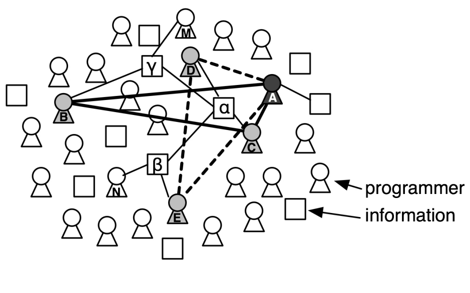


Fig. 3. An Actor-Network of a Software Project

technical dependency of the code and documents, such as which software developers depend on which other software developers for a given piece of code or a document through calling, using, or describing [7].

The relationship between information includes the syntactic and semantic dependency among code parts that are linked through data flow, control flow or linear order. Code nodes in Fig. 4 can have different levels of granularity: code segments, methods, files, classes and packages. Documents are related to code through multiple dimensions. For example, a code node implements a portion of a design document; the design rationale of the code is described in a series of email discussions; a bug report is fixed by modifying several nodes of code; or a document describes the functionality of reusable code components.

The relationship between a developer and information includes who writes or changes the code, who has commented on the code, and who has reused the code component in his/her own programs.

The knowledge network in Fig. 3 is an actor-network that consists of actants (both human and artifacts) [20]. The knowledge embedded in each node as well as the links constitutes the group knowing of a project. The network, as well as the group knowing, changes as new actants are brought into or removed from the network (e.g. new information is added or a developer leaves), and as new relationships are developed, strengthened, or weakened (e.g. another developer started working on a module, a link between documents were discovered). An individual's capability about the project progresses as he/she develops more relationships with other actants.

4.2 A Continuum of Technical Support and Social Support

Using external expertise can be viewed as a software developer's activating the links in the actor-network in Fig. 3, and engaging in collaboration with actants. To do so, a software developer are faced with the following challenges:

(1) He/she might not be aware of where the expertise is located: what is the relevant information, and who has the expertise on this particular problem?
(2) When the actants are peer developers, how should he/she approach them, without causing too much communication cost of interruption?
(3) Whether the human actants are willing to engage in providing help?

Accordingly, the Dynamic Community approach (Fig. 4) provides three kinds of support for in situ knowledge collaboration. Assume a developer (*A*) is dealing with a task (α) and needs external expertise.

First, it employs both information access and information delivery mechanisms [27] to help developers find task-relevant information in the repository that models the actor-network of the knowledge ecosystem of a project (Fig. 4). *Information access* includes browsing or searching, in which the developer articulates what he/she needs through either traversing the links between the information (browsing) or formulating a query (searching). Contrary to information access that has to be initiated by the developer, *information delivery* proactively provides information by watching what the developer is writing, inferring what his/her information needs are, and then recommending the needed information without user initiated search activities. Information delivery is able to make developers access external expertise in the repository whose existence they are not even aware of [41].

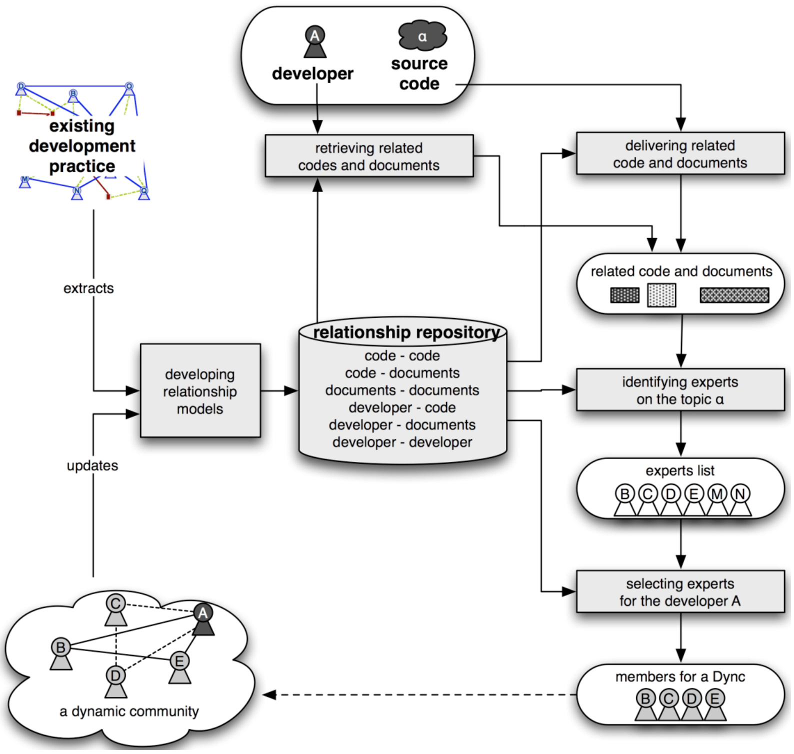


Fig. 4. The Dynamic Community Framework

When the relevant information retrieved or delivered from the repository is not sufficient for the developer to obtain the expertise, he/she need to access knowledgeable peers. In the Dynamic Community framework, a developer can post a question about the topic he/she is currently interested in, and a sub-network of developers is dynamically formed by activating the links in the actor-network of Fig. 5 through two processes: *expert identification* and *expert selection*.

The expert identification process traces the link between a developer and information, and identifies peer developers that are related to the set of relevant information nodes (i.e. α, β and γ in Fig. 3 where β and γ are related to α). Depending on the definition of the relation, those peer developers might have expertise or hold special interest in the set of information nodes. For brevity, we refer them as experts. The experts list obtained in this phase is *{B, C, D, E, M, N}* because they are linked to either α, β or γ in Fig. 3.

From the above experts list, the expert select process selects those who have good social relations with the developer *A*, which is *{B, C, D, E}*. The relationship between developers is derived from their previous interaction history and represents the affinitive relationship existing among them. A link from developer *B* to *A* indicates a high possibility that *B* is likely to help *A* when *B*'s expertise is needed for *A*'s task.

An ephemeral mailing list (called a DynC) is then dynamically created for the selected experts and *A* on the topic α (noted as *DynC(A, α) ={A, B, C, D, E}*), and *A*'s question is sent to the members of DynC(A, α). DynC(A, α) members who reply to the question posted by A is also sent to all the members. When the developer A thinks there is no more need to discuss about the topic, he/she needs to terminate the DynC, and the associated dynamic mailing list dissolves. All the discussions, however, are archived in the repository so that other developers who have similar questions can benefit by either browsing or searching the repository.

4.3 Cost Reduction Strategies

The Dynamic Community approach attempts to reduce the overall communication cost in knowledge collaboration in a globally distributed project by utilizing the following strategies.

First, it considers social support as a costly transaction, and encourages software developers to explore the technical support afforded by the rich knowledge repository that weaves together the code, document and previous discussions. All the discussions in DynC mailing lists are archived and linked with the related information so that repeated DynCs can be avoided. The combination of sophisticated search, browsing and delivery mechanisms is employed to make locating relevant information easier for software developers. The Dynamic Community framework requires a developer to initiate a DynC from the search results, ensuring he/she has at least spending some time exploring the related information. Social support is very costly and should not be used as the main resources for expertise.

Second, the automatic identification of experts relieves a software developer from gaining an awareness of who the experts are, and thus reduces the cost of finding the location of expertise and asking the question. Knowing the experts is one of the major obstacles faced by developers in offshore outsourcing projects due to the lack of informal and spontaneous communication available in collocated projects.

Third, it reduces 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. Other developers who either do not have the necessary expertise or whose relationships with the expertise seeker are not strong enough to be motivated to engage in knowledge collaboration with the seeker are not disturbed. The strong social relationship also increases the intensity of the engagement and therefore the effectiveness of knowledge collaboration among participants [6].

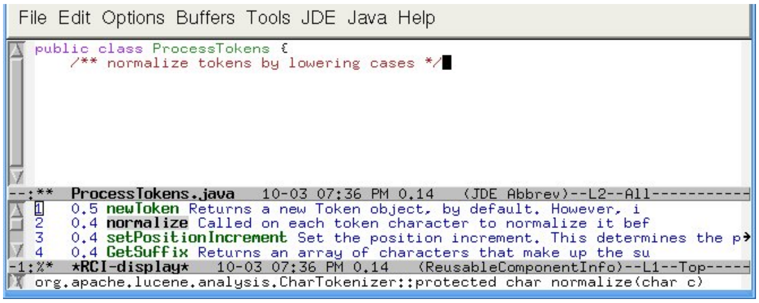
Fourth, the DynC mailing list follows the principle of *asymmetric disclosure of information* [26] to conserve further the attention and good will of experts. On one hand, when the question is posted to a DynC, the members selected to the DynC are not made public either to the expertise seeker or to other members; only a receiver of the question message knows that he/she is selected as a member of the DynC. Only when a DynC member sends a reply message, his/her identity is revealed. A DynC member, therefore, may leave the DynC (a social equivalent of saying "no") at any moment without being publicly known. Due to this principle, no participation does not constitute the violation of social norms, which is punishable by the "iron hand of social pressure" of enforcing required individual behavior in a social unit [31]. On the other hand, because replying to the DynC reveals the identity of the sender of the message, the DynC members' contribution is publicly acknowledged and can lead to the improvement of motivation [9]. This socially aware communication mechanism that allows unwilling peer developers exit socially safely has two implications. The remaining peers are the participants of willing, and hence the expertise sharing becomes more effective. From the perspective of the expertise seeker, knowing that other developers could easily exit, he/she feels less pressured to post a question because the choice of participation is controlled by the experts.

5 System Development

To illustrate how the Dynamic Community framework supports knowledge collaboration in distributed software projects while reducing the overall cost of communication and collaboration, we describe two systems: CodeBroker [40] and STeP_IN [28] that we have developed. The two systems in combination provide continuous support for accessing external expertise. In the following usage scenario, which illustrates the functionality of the two systems, we use the Lucene-java (http://lucene.apache.org/java/docs/index.html) project as the data (the source code and its mailing list archive from 2001 to Aug. 2006) to populate the repository.

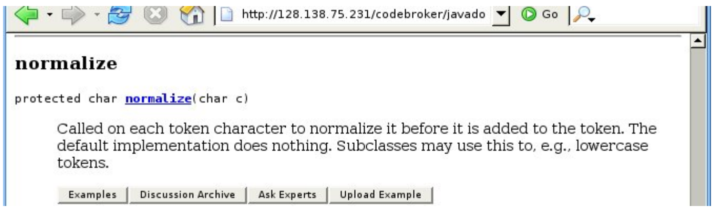
Suppose a developer (*lu1283*) needs to write a program that processes a stream of token extracted from a document in an information retrieval system, which uses the third-party open source library (Lucene-java). He first needs to normalize each token by lowering its cases, but he is not aware that a method already exists in the library. He sets to create his own program and writes a doc comment in the editor to describe his task (Fig. 5). As soon as the doc comment is written in the editor, CodeBroker automatically delivers a set of task-relevant library methods in the lower buffer of the editor. *Lu1283* finds the second method probably does what he needs, and clicks the method name in the buffer.

The document for the method is shown (Fig. 7). Now he knows this method is what he wants but he is not sure how it can be used. So he clicks the *Examples* button, and looks at the example code (Fig. 8). Now he wonders if this method does more than lowering the case. He clicks the *Discussion Archive* link and reads previous discussions on this method (Fig. 9) but could not find answers to his question. He thinks that other developers in the team might have used it before, so he clicks the *Ask Experts* link and posts a question (Fig.10).



```
public class ProcessTokens {
    /** normalize tokens by lowering cases */
--:**  ProcessTokens.java   10-03 07:36 PM 0.14   (JDE Abbrev)--L2--All-----------
1    0.5 newToken Returns a new Token object, by default. However, i
2    0.4 normalize Called on each token character to normalize it bef
3    0.4 setPositionIncrement Set the position increment. This determines the p
4    0.4 GetSuffix Returns an array of characters that make up the su
-1:%*  *RCI-display*   10-03 07:36 PM 0.14   (ReusableComponentInfo)--L1--Top-----
org.apache.lucene.analysis.CharTokenizer::protected char normalize(char c)
```

Fig. 5. CodeBroker: An Enhanced Emacs Editor for Java Programming

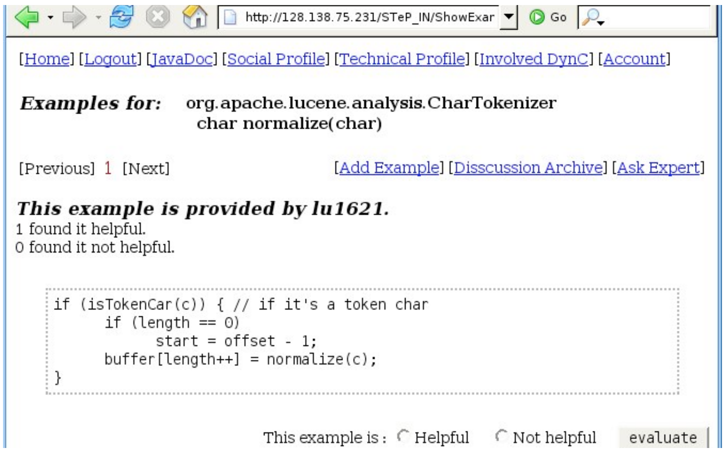


normalize

```
protected char normalize(char c)
```

Called on each token character to normalize it before it is added to the token. The default implementation does nothing. Subclasses may use this to, e.g., lowercase tokens.

Fig. 6. Enhanced Javadoc documentation


http://128.138.75.231/STeP_IN/ShowExar
Go
[Home] [Logout] [JavaDoc] [Social Profile] [Technical Profile] [Involved DynC] [Account]

Examples for: org.apache.lucene.analysis.CharTokenizer
char normalize(char)

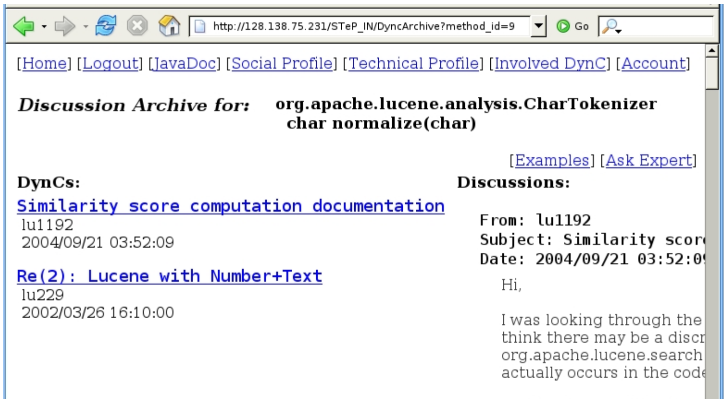
[Previous] 1 [Next]
[Add Example] [Disscussion Archive] [Ask Expert]

This example is provided by lu1621.
1 found it helpful.
0 found it not helpful.

```
if (isTokenCar(c)) { // if it's a token char
      if (length == 0)
            start = offset - 1;
      buffer[length++] = normalize(c);
}
```

This example is : ◯ Helpful ◯ Not helpful

Fig. 7. Example code


http://128.138.75.231/STeP_IN/DyncArchive?method_id=9
Go
[Home] [Logout] [JavaDoc] [Social Profile] [Technical Profile] [Involved DynC] [Account]

Discussion Archive for: **org.apache.lucene.analysis.CharTokenizer**
char normalize(char)

DynCs:

Similarity score computation documentation
lu1192
2004/09/21 03:52:09

Re(2): Lucene with Number+Text
lu229
2002/03/26 16:10:00

Discussions:

```
From: lu1192
Subject: Similarity scor
Date: 2004/09/21 03:52:0
```

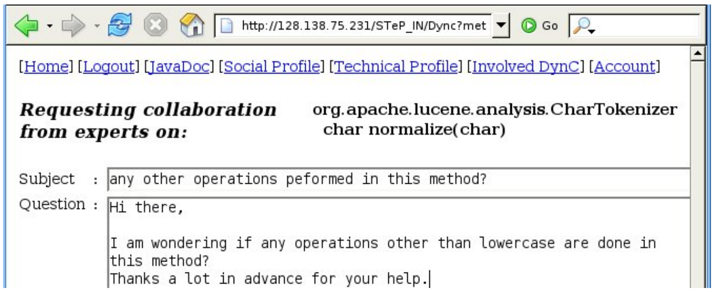
Hi,

I was looking through the
think there may be a discr
org.apache.lucene.search
actually occurs in the code

Fig. 8. Discussion Archive

Upon the submission of the question, a DynC mailing list is created by the STeP_IN system. Five members (*lu292, lu1192, lu229, lu953* and *lu1953*) are selected, regardless of their physical locations. They all have used this method before (5, 4, 2, 2, 1 times respectively) in their previous programs and have expertise on this method. In addition, all five members have affinitive social relationships with *lu1283* and the community, and they are very likely to help *lu1283*. *Lu292* and *lu1953* have sent emails to *lu1293* before, so they should have known *lu1283* by certain degree. *Lu1192* and *lu229* have got helped in the community by others more than they have helped others; therefore, it is their turn to fulfill their social obligations to reciprocate the favor they have received. *Lu953* is an eager helper [15], and had helped others more than 101 times, so he might also offer help this time.

The members, however, are not forced to help because lu1283 as well as other members do not know that they received this question due to the design principle of asymmetry of information disclosure. If some of the members are currently busy and do not have time to offer help for *lu1283*, no body would notice; and they will not face social consequences of being non-cooperative in this case.



[Home] [Logout] [JavaDoc] [Social Profile] [Technical Profile] [Involved DynC] [Account]

Requesting collaboration from experts on: org.apache.lucene.analysis.CharTokenizer char normalize(char)

Subject : any other operations peformed in this method?

Question : Hi there,

I am wondering if any operations other than lowercase are done in this method?
Thanks a lot in advance for your help.

Fig. 9. Ask Experts

6 Discussions

The two systems introduced in the paper are meant to illustrate how the conceptual framework of Dynamic Community can be applied to support knowledge collaboration in globally distributed software development while reducing the cost of communication. The conceptual framework can be applied to support different tasks in distributed offshore projects. To illustrate its potential, we briefly sketch its possible application in maintenance support and agile development.

After a software system has been developed and deployed, the original developers are often assigned to other projects and the maintenance work is handed over to other members. Under such conditions, maintainers often do not know who are the original designers and developers of the module under maintenance and do not know who to approach for design rationales. The dynamic community can be applied to deal with this situation. Suppose a maintainer *A* needs to modify a module *α*. It is quite possible that many programmers have used or changed module *α* during its lifecycle. All those programmers can be considered experts on *α* and they can be identified from the configuration management systems such as CVS used during the development phase [23]. Because those original programmers have new assignments as their current work, they might not be readily available to help *A*. Using the two-phase selection of experts in the Dynamic Community framework, a list of experts who have knowledge and are most likely to assist *A* can be selected to form a DynC for this task.

Communication can be limited to those selected members and the results archived for later use.

In agile development, document-based formal coordination mechanisms are replaced with frequent, intensive, and informal communications. As systems are incrementally developed, the dependency of code changes accordingly; and the related developers that need to be involved in communication and coordination change, too [37]. Currently, developers have to decide by themselves who they should engage in collaboration. If we apply the Dynamic Community framework to this, a system can be developed to identify automatically the subgroup of developers that should be involved based on the social dependency derived from the dependency of code that each developer is developing, and create a DynC mailing list for their communication. As a developer moves his focus of development, different DynC mailing lists can be created accordingly in an automatic manner to reduce the cost of communication by limiting the number of communicants to the concerned members and by reducing the cost of determining communicants.

7 Concluding Remarks

In this paper, we conceptualize a distributed software project as a distributed knowledge ecosystem, and model it as an actor-network. This modeling enables us to view software artifacts produced in the development process and developers as knowledge actants, which constitute the organizational knowing of the project, and which should be engaged equally as knowledge resources for the indispensable knowledge collaboration in software development. Based on this conceptualization and modeling, we proposed the Dynamic Community framework as a new communication mechanism for knowledge collaboration. The framework reduces the cost of communication in offshore outsourcing software development by (1) using information delivery and search mechanisms to allow developers locate relevant knowledge from a knowledge repository that consists of code, documents and discussions in order to reduce the frequency of collaboration with other developers; (2) automatically selecting experts to mitigate the difficulty of finding the experts and initiating contacts; and (3) forming an ephemeral DynC mailing list that consists only of developers who are both technically capable and socially willing to engage in collaboration with a particular developer on a particular topic.

The ephemeral DynC mailing list resulted from the Dynamic Community approach is neither a direct emails, nor a mail list, it is something in between with persistent storage similar to discussion forums. It is similar to mailing lists in that the email is sent to unspecified members, and the participation in knowledge collaboration is completely controlled by its recipients. It is not mailing lists in that the recipients are not determined by their own subscriptions but by their social relationships with the initiator and their technical expertise on the topic. The latter point makes DynC mailing list similar to direct emails because they are intentionally targeted recipients who have already established social ties with the sender. However, it differs from direct emails in that recipients remain anonymous to the sender and other members, leaving the control of participation to the recipients, and in that the recipients are automatically identified and chosen.

In offshore outsourcing software development, many development activities need to be coordinated and collaborated through communication channels. To reduce the communication cost, it is important for a project team to be able to operate within a communicative economy with a variety of communicative resources at its developer's disposal [33]. Both the unique structure of each communication channel and the socio-technical relationships among developers determine the collective cost and benefits of each communicative act. To reduce the cost of communication and coordination, developers should be able to choose the most appropriate channel for their needs. The Dynamic Community framework provides a new communication mechanism that has its special niche. It is not meant to replace any of the currently dominating communication channels such as face-to-face, direct emails or mailing lists, but as a complimentary one. For example, if a developer happens to know who are the experts on a topic of interest, and is socially comfortable to directly approach the experts, he/she can use the face to face or direct emails (if not collocated). If the developer feels that the topic is important enough to be known by all members of the project, he/she can send it through project-wide mailing lists. If the developer thinks that his/her question only concerns a few, but does not know who they are, the DynC mailing list is a perfect match for that.

References

1. Allen, T.J.: Managing the flow of technology. MIT Press, Cambridge, MA (1977)
2. Arguello, J., et al.: Talk to me: Foundations for successful individual-group interactions in online communities. In: Proceedings of conference on human factors in computer systems (chi06)., pp. 959–968. ACM Press, Montréal, Canada (2006)
3. Brooks, F.P.J.: The mythical man-month: Essays on software engineering, 20th edn. Addison-Wesley, Reading, MA (1995)
4. Brown, J.S., Duguid, P.: Organizing knowledge. Society for Organizational Learning Journal 1(2), 28–44 (1999)
5. Coleman, J.C: Social capital in the creation of human capital. American Journal of Sociology 94, S95–S120 (1988)
6. Cross, R., Borgatti, S.P: The ties that share: Relational characteristics that facilitate information seeking. In: Huysman, M., Wulf, V. (eds.) Social capital and information technology, pp. 137–161. MIT Press, Cambridge, MA (2004)
7. de Souza, C.R.B., et al.: How a good software practice thwarts collaboration: The multiple roles of apis in software development. In: de Souza, C.R.B., et al. (eds.) Proceedings of the 12th acm sigsoft twelfth international symposium on foundations of software engineering (fse04), pp. 221–220. Newport Beach, CA (2004)
8. Dorina, C.G: Distribution dimensions in software development projects: A taxonomy. IEEE Software 23(5), 45 (2006)
9. Fischer, G., Scharff, E., Ye, Y.: Fostering social creativity by increasing social capital. In: Huysman, M., Wulf, V. (eds.) Social capital, pp. 355–399 (2004)
10. Goldberg, A.: Collaborative software engineering. Journal of Object Technology 1(1), 1–19 (2002)
11. Goldhaber, M.H.: The attention economy. First Monday, vol. 2(4) (1997)
12. Herbsleb, J., Grinter, R.E.: Splitting the organization and integrating the code: Conway's law revisited. In: Proceedings of international conference on software engineering (icse99), pp. 85–95 (1999)

13. Herbsleb, J., Mockus, A.: An empirical study of speed and communication in globally-distributed software development. IEEE Transactions on Software Engineering 29(3), 1–14 (2003)
14. Herbsleb, J.D., Grinter, R.E: Architectures, coordination, and distance: Conway's law and beyond. IEEE Software, 63–70 (September- October 1999)
15. Hoff, B.v.d., Ridder, J.d., Aukema, E.: Exploring the eagerness to share knowledge: The role of social capital and ict in knowledge sharing. In: Huysman, M., Wulf, V. (eds.) Social capital and information technology, pp. 163–186. MIT Press, Cambridge, MA (2004)
16. Hollan, J., Hutchins, E., Kirsch, D.: Distributed cognition: Toward a new foundation for human-computer interaction research. In: Carroll, J.M. (ed.) Human-computer interaction in the new millennium, pp. 75–94. ACM Press, New York (2001)
17. Illich, I.: Deschooling society. Harper and Row, New York (1971)
18. Kraut, R.E., et al.: Informal communications in organizations: Form, function, and technology. In: Oskamp, I.S., Spacapan, S. (eds.) Human reactions to technology: The claremont symposium on applies social psychology, Sage Publications, Beverly Hills, CA (1990)
19. Kraut, R.E., Streeter, L.: Coordination in software development. CACM 38(3), 69–81 (1995)
20. Latour, B.: Reassembling the social: An introduction to actor-network-theory. Oxford University Press, Oxford (2005)
21. McDonald, D.W., Ackerman, M.S.: Just talk to me: A field study of expertise location. In: McDonald, D.W., Ackerman, M.S. (eds.) Proceedings of conference on computer supported cooperative work (cscw'98), pp. 315–324. Seattle, WA (1998)
22. Meyer, B.: The unspoken revolution in software engineering, pp. 121–124. IEEE Computer Society Press, Los Alamitos (2006)
23. Mockus, A., Herbsleb, J.: Expertise browser: A quantitative approach to identifying expertise. In: Proceedings of 2002 international conference on software engineering, Orlando, FL, pp. 503–512 (2002)
24. Mumford, E.: Socio-technical system design: Evolving theory and practice. In: Bjerknes, P.G., Ehn, P., Kyng, M. (eds.) Computers and democracy, pp. 59–76. Averbury, Aldershot, UK (1987)
25. Nahapiet, J., Ghoshal, S.: Social capital, intellectual capital, and the organizational advantage. Academy of Management Review 23, 242–266 (1998)
26. Nakakoji, K.: Supporting software development as collective creative knowledge work. In: Nakakoji, K. (ed.) Proceedings of ase workshop on supporting knowledge collaboration in software development, Tokyo, (in press) (2006)
27. Nakakoji, K., Fischer, G.: Intertwining knowledge delivery and elicitation: A process model for human-computer collaboration in design. Knowledge-Based Systems 8(2-3), 94–104 (1995)
28. Nishinaka, Y., et al.: Please step_in: A socio-technical platform for in situ networking. In: Proceedings of the 12th Asia-Pacific Software Engineering Conference, pp. 813–820. IEEE CS Press, Taipei (2005)
29. O'Reilly, C.A.: Variations in decision makers' use of information sources: The impact of quality and accessibility of information. Academy of Management Journal 25(4), 756–771 (1982)
30. Orlikowski, W.J.: Knowing in practice: Enacting a collective capability in distributed organizing. Organization Science 13(3), 249–273 (2002)

31. Pentland, A.: Socially aware computation and cmmunication. Computer 38(3), 33–40 (2005)
32. Perlow, L.A.: The time famine: Toward a sociology of work time. Administrative Science Quarterly 44(1), 57–81 (1999)
33. Reder, S.: The communication economy of the workgroup: Multi-channel genres of communication. In: Proceedings of cscw1988, pp. 354–368. ACM Press, New York (1988)
34. Sengupta, B., Chandra, S., Sinha, V.: A research agenda for distributed software development. In: Proceedings of 2006 international conference on software engineering, Shanghai, pp. 731–740 (2006)
35. Szoestek, A.M., Markopoulos, P.: Factors defining face-to-face interruptions in the office environment. In: Proceedings of conference on human factors in computer systems, pp. 1379–1384 (2006)
36. Tyler, J.R., Tang, J.C: When can i expect an email response? A study of rhythms in email usage. In: Proceedings of the eighth european conference on computer supported cooperative work (ecscw2003), pp. 239–258. Helsinki (2003)
37. Wagstrom, P., Herbsleb, J.: Dependency forecasting. CACM 49(10), 55–56 (2006)
38. Walz, D.B., Elam, J.J., Curtis, B.: Inside a software design team: Knowledge acquisition, sharing, and integration. CACM 36(10), 63–77 (1993)
39. Weinberg, G.M.: The psychology of computer programming. Van Nostrand Reinhold, New York (1971)
40. Ye, Y., Fischer, G.: Information delivery in support of learning reusable software components on demand. In: Proceedings of 2002 International Conference on Intelligent User Interfaces (IUI'02), pp. 159–166. ACM Press, San Francisco (2002)
41. Ye, Y., Fischer, G.: Supporting reuse by delivering task-relevant and personalized information. In: Proceedings of 2002 international conference on software engineering (icse'02), pp. 513–523. Orlando, FL (2002)