

Dynamic Community: A New Conceptual Framework for Supporting Knowledge Collaboration in Software Development

Yunwen Ye^{1,3}

¹*Department of Computer
Science
University of Colorado
Boulder, CO80309-0430, USA
yunwen@cs.colorado.edu*

Yasuhiro Yamamoto²

²*Knowledge Interaction Design
Laboratory
RCAST, University of Tokyo
Tokyo, 153-8904, Japan
yxy@computer.org*

Kouichi Kishida³

³*SRA Key Technology
Laboratory, Inc.
3-12 Yotsuya, Shinjuku
Tokyo 160-004, Japan
k2@sra.co.jp*

Abstract

Software development is a knowledge intensive activity and software developers are knowledge workers. Knowledge needed for software development is often distributed among different developers. Supporting efficient knowledge collaboration and transfer is thus essential for software development organizations to remain competitive. This paper proposes a new theory—dynamic community to support knowledge collaboration and discusses its application in software development. Dynamic community integrates the traditional knowledge management approach in which knowledge is formalized and accumulated in knowledge repositories, and the community-based knowledge collaboration approach in which knowledge is transferred through informal community participation and human contacts. The dynamic community theory takes into full consideration individual knowledge workers' ever-changing needs for new knowledge as well as the role of social relationship of knowledge workers in effective knowledge collaboration. This paper introduces the dynamic community theory, a general system architecture of sociotechnical environments in support of dynamic community, and its application in software development.

1. Introduction

Software development is a knowledge construction process and software developers are knowledge workers who apply programming knowledge and domain knowledge to construct new knowledge artifacts through the process of understanding problems, designing solutions, and coding systems. As software systems are getting increasingly complicated today, the knowledge

needed for the implementation is vast and unlikely to be held by any individual software developers [2]. In other words, software development should be viewed as a distributed cognitive activity [10] and the knowledge required for software development is distributed among four different dimensions: *spatial, temporal, technological and social* [5]:

Spatial distribution. With the advancement of communication technology, developers of a project are often not co-located and are distributed geographically in different locations, and even different countries.

Temporal distribution. Several issues in software development contributed to the temporal distribution of knowledge across time. First, most complicated software systems span many years of development. During the period, due to inevitable personnel turnouts and intra-organization job reassignments, some developers have to work on portions developed by others. Second, during the long maintenance phase, maintainers are often not the original developers. Third, when software developers reuse, they may need to access the knowledge of the developer of the reused code.

Technological distribution. Software development can be described as a reflective conversation between developers and systems they create. Developers use tools to construct, and then listen to the “back-talk of the situation” that is created [21]. Unlike passive design materials, such as pen and paper, computational design materials are able to interpret the work of designers and actively talk back to designers.

Social distribution. Knowledge required for a complicated software system comes from many people and multiple domains. The group members need to collaborate with each other through shared understanding and mutual learning.

Due to the above-mentioned inherent distributedness of software development, software development should be

reframed as a knowledge collaboration process during which software developers collaborate with tools and each other along the four dimensions.

Supporting efficient knowledge collaboration and transfer among software developers is therefore essential for software development organizations to remain competitive. This paper proposes a new theory—dynamic community to support knowledge collaboration, and discusses its application in software development. *Dynamic community* integrates the traditional knowledge management approach in which knowledge is formalized and accumulated in knowledge repositories [6], and the community-based knowledge collaboration approach in which knowledge is transferred through informal community participation and human contacts [13]. The dynamic community theory takes into full consideration individual knowledge workers' ever-changing needs for new knowledge as well as the role of social relationship of knowledge workers.

After having discussed the problems with existing approaches to knowledge collaboration support in Section 2, and described the dynamic community theory in Sections 3, the paper presents the general system architecture of sociotechnical environments in support of dynamic community in Section 4, followed by discussions on its application in software development in Section 5.

2. Approaches to Knowledge Collaboration Support

Two major approaches to knowledge collaboration support exist: the knowledge repository approach and the community-based approach.

Deeply influenced by research in artificial intelligence, the knowledge repository approach believes that expertise or knowledge can be externalized and formalized with logics or rules for sharing. The research focus, therefore, is on extracting knowledge by interviewing experts; formalizing extracted knowledge; storing formalized knowledge in knowledge repositories; and developing retrieval mechanisms to locate from repositories the knowledge relevant to the task at hand [6, 14].

This knowledge repository approach completely overlooked the fact that a great amount of human knowledge is inherently tacit [19]. Even if a knowledge repository can capture and store externalized knowledge, the tacit context is always lost. Due to the loss of the context, many users are unable to apply the knowledge that they located from the knowledge repository [18].

Realizing the difficulties in the knowledge repository approach, a community-based approach was proposed in the 90s [13]. This approach focuses on understanding and supporting knowledge transfer and collaboration through natural and informal human communications because it

believes that learning takes place naturally when people from a community of practice engage in practices together. In contrast with the knowledge repository approach in which knowledge is managed, this approach advocates promoting informal communication in communities through the management of social network and the creation of expert lists, such as Expert Finders [22], Expertise Browser [16], and ContactMap [17], so that the user knows whom to ask when they have a problem.

Simply put, the knowledge repository approach manages “know-how” while the community-based approach tries to manage “know-who”. In the former approach, users search knowledge repository to find knowledge applicable to their problems; in the later approach, users search their social network or expertise list to find the experts to answer their questions.

Current community-based knowledge approach has several problems. First, it is founded on the assumption that in a community, there are experts and novices. Knowledge transfer flows in one direction from experts to novices through the process of *Legitimate Peripheral Participation* [13]. This assumption is not true in many cases where absolute experts do not exist [3]. In software development where knowledge from multiple domains is involved and technology changes rapidly, expert is at best a relative attribute of a specific context, rather than an absolute personal attribute. In other words, a person is an expert in one context may become a novice in another. Consequently, who is an expert for a particular task can only be known after the task itself is clearly defined.

Second, human relation in the community is treated rather uniformly. It assumes that experts will always rush to assist novices in need of help, without taking into consideration that experts need motivation to volunteer help because they have their own workload too. Furthermore, whenever a novice has a request for help, all members of the whole community will get the request no matter whether they are experts on this particular issue or not. This creates considerable waste of attention.

3. Dynamic Community

Each of the above two currently dominant approaches focuses on only one aspect of knowledge collaboration: either the provision of formalized knowledge or the provision of informal social support. The concept *dynamic community* that we propose is an attempt to integrate both approaches seamlessly. A dynamic community is a subgroup of knowledge workers that forms ad hoc in support of a particular *user* and a particular *task*, and dissembles as the task is finished. Unlike a static community which forms around a particular domain and exists for a long time, a dynamic community forms for a particular task, exists for a short

period. A dynamic community is a small group of people in a community, and the members of the dynamic community include only those people who have expertise on the particular task. Therefore, other non-experts of the community are not disturbed. Furthermore, because only those experts who already have social contacts with the particular user are included in the dynamic community, it is reasonable to believe that those experts are more motivated to provide help based on the social norm of generalized reciprocity [7].

3.1 A Motivating Example

Before we move on to give a formal definition of dynamic community, this section gives a motivating example to illustrate the necessity and benefits of dynamic community.

Jack is a novice Java programmer, and he is asked to implement the following task:

Traditionally, Chinese write numbers with a comma inserted at each fourth number from the right. For example, 1,000,000 is written as 100,0000. Implement a program that transforms the Chinese writing format (100,0000) to the Western format (1,000,000).

This task can be implemented in several different ways by using different components in the standard Java API library. The simplest solution is to use the `format` method from the `NumberFormat` class in the `java.text` package. Not even knowing the existence of the `java.text` package, Jack does not know how to write the program in an efficient way. He needs help from external resources.

Jack can proceed in two ways. He can spend time browsing through the huge Java API library or mailing list archives to find what he needs. This is typically what the knowledge repository approach supports. For a novice programmer who is not familiar with the Java API library and does not know how to pose the question properly, this could be a time-consuming pursuit. Even if Jack might be able to find the `format` method, he might not be able to understand how to use it in his program.

A more efficient way is to ask a colleague who is an expert Java programmer, by using for example an expert list. This is typically what the community-based approach aims to support. The problem is that not every expert Java programmer knows this particular method because few programmers, if any, would know every method of the 2,723 classes contained in the Java 1.4 API library. He needs to find the one that happens to know `format`. Even if he is able to find an expert, the expert might not be able to help him because he or she is busy with his or her own task, or Jack might hesitate to interrupt the expert because Jack does not know the expert well enough.

A dynamic community-based approach can provide more efficient support for Jack. We have been extending a

system called CodeBroker [24] to provide dynamic community-based sociotechnical support for Java library reuse. CodeBroker takes Jack and Jack's programming task as input and provides layered support to Jack according to his needs.

Layer 1. It autonomously locates Java library methods that are relevant to Jack's task, in this case the `format` method.

Layer 2. If Jack is not able to understand and apply the method by himself, an example program that uses the `format` method will be located by the system from his colleagues for his reference.

Layer 3. If Jack is still not able to use the method, a list of experts who have used the method in their programs will be identified. From the list, experts who Jack knows will be selected so that Jack can readily get help from someone he knows.

Support provided by layers 1 and 2 is basically the knowledge repository-based technical solution, and support provided by layer 3 is community-based social solution. Economically, technical solution is much cheaper than social solution. The layered support in dynamic community aims to maximize the benefits of technical solution while using seamlessly integrated social support as a backup to cover the shortcomings of technical solution. Such an approach can strike a balance between accommodating the needs of people who ask for help and the economic use of the time and attention of experts. More details about the CodeBroker system will be discussed in Section 5.1.

3.2 A Formal Definition

A dynamic community, or *DynC*, is dynamically formed in a knowledge work space, or *KWS*. A *knowledge work place* consists of a group of knowledge workers, and the knowledge that the workers hold. The group can either be a formal organization such as a company or an informal community such as an mailing lists. Three kinds of relations exist in a knowledge work space: the relation between people, the relation between people and information¹; and the relation between information. For different application domains, the definition of each relation might be different. Taking the relation between people and information as an example, in one application domain, the relation can be defined as those who create the information; and in another one, it can be defined as those who are affected by the information. A knowledge work space can be formally defined as follows.

¹ We start to use the term information here instead of the term knowledge because we are discussing from an implementation perspective.

Definition 1 (Knowledge Work Space, KWS).

Let Ψ be a group of people, i.e.,

$$\Psi = \{A, B, C, D, E, \dots\};$$

and Φ be a set of information, i.e.,

$$\Phi = \{\alpha, \beta, \gamma, \delta, \varepsilon, \dots\}.$$

Let PP be a set of binary relation between people in Ψ , i.e.,

$$PP = \{(A, B) \mid A \in \Psi \wedge B \in \Psi\};$$

II be a set of binary relation between information in Φ , i.e.,

$$II = \{(\alpha, \beta) \mid \alpha \in \Phi \wedge \beta \in \Phi\};$$

and PI be a set of binary relation between people in Ψ and information in Φ , i.e.,

$$PI = \{(P, \varepsilon) \mid P \in \Psi \wedge \varepsilon \in \Phi\}.$$

A KWS is a 3-tuple of II , PI , and PP over the domains of Ψ and Φ , i.e.,

$$KWS = (II, PI, PP)$$

or

$$KWS = (\{(\alpha, \beta)\}, \{(P, \varepsilon)\}, \{(A, B)\})$$

where $(\alpha, \beta) \in II \wedge (P, \varepsilon) \in PI \wedge (A, B) \in PP$

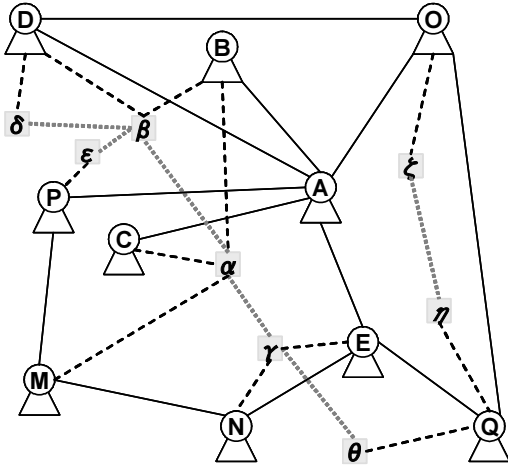


Figure 1: A KWS (Knowledge Work Space)

Figure 1 shows an example of a knowledge work space, in which $\Psi = \{A, B, C, D, E, M, N, O, P, Q\}$ and $\Phi = \{\alpha, \beta, \gamma, \delta, \varepsilon, \zeta, \eta, \theta\}$. The relation between people, PP (solid line), is $\{(A, B), (A, C), (A, D), (A, E), (A, O), (A, P), (D, O), (E, N), (E, Q), (M, P), (M, N), (O, Q)\}$; the relation between information II (dot line) is $\{(\alpha, \beta), (\alpha, \gamma), (\beta, \varepsilon), (\beta, \delta), (\gamma, \theta), (\zeta, \eta)\}$, and the relation between people and information PI (dash line) is $\{(B, \alpha), (C, \alpha), (M, \alpha), (B, \beta), (D, \beta), (E, \gamma), (N, \gamma), (D, \delta), (P, \varepsilon), (O, \zeta), (Q, \eta), (Q, \theta)\}$. Therefore, the knowledge work space is

$$KWS = (PP, II, PI)$$

$$= (\{(\alpha, \beta), (\alpha, \gamma), (\beta, \varepsilon), (\beta, \delta), (\gamma, \theta), (\zeta, \eta)\},$$

$$\{(B, \alpha), (C, \alpha), (M, \alpha), (B, \beta), (D, \beta), (E, \gamma), (N, \gamma),$$

$$(D, \delta), (P, \varepsilon), (O, \zeta), (Q, \eta), (Q, \theta)\},$$

$$\{(A, B), (A, C), (A, D), (A, E), (A, O), (A, P), (D, O),$$

$$(E, N), (E, Q), (M, P), (M, N), (O, Q)\}.$$

A dynamic community ($DynC$) is a subset of people in a KWS . $DynC$ is a function of person A and information α . A $DynC$ consists of those people who are related to both information α and person A , and those people who are both related to person A and other information that is related to α . A $DynC$ can be formally defined as:

Definition 2 (Dynamic Community, DynC).

$$DynC(A, \alpha) = \{P \mid P \in \Psi \wedge (P, A) \in PP \wedge$$

$$(P, \alpha) \in PI \vee$$

$$(\exists \beta \in \Phi ((\alpha, \beta) \in II \wedge (P, \beta) \in PI))\}$$

3.3 The Formation of a DynC

A dynamic community can be formed in the following process. Suppose a person A needs information α for a particular task in the KWS defined by Figure 1. Information β and γ is related to α . The group of people who are related² to α is B, C , and M . The group of people who are related to either β or γ is B, D, E , and N .

Because a dynamic community only includes those people who are related to A , only B, C, D , and E who have interaction history with A in Figure 1 are selected as members of the dynamic community. Although M and N are also related to α , because they do not have an interaction history with A , they are excluded from the dynamic community. Furthermore, person O who is related to A is also excluded from the dynamic community because he or she is not related to information α .

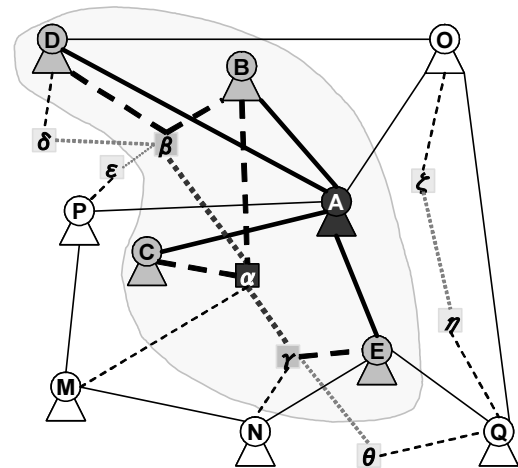


Figure 2: A dynamic community $DynC(A, \alpha)$

² As we mentioned before, for different domain, the relation can be defined differently. For example, it can be “knows”, “interested in”, “uses” or “creates”.

3.4 Characteristics of Dynamic Communities

A dynamic community $DynC(A, \alpha)$ is *dynamically* created when user A needs help for the task that requires information α , and dissembles when A 's task is done. It is formed *ad hoc* by selecting members who can potentially provide expertise to help user A use information α in his or her specific task.

A dynamic community is task-specific. For the same user A , $DynC(A, \alpha)$ that is created to support his or her task that requires information α is different from $DynC(A, \beta)$ that is created to support the task that requires information β , because the lists of experts on information α and β are different. Therefore, A does not belong to a single, static community; he or she belongs to different dynamic communities depending on the task at hand.

A dynamic community is also user-specific. For the same task that requires information α , the dynamic community $DynC(A, \alpha)$ that is formed to support user A is different from $DynC(B, \alpha)$ that is formed to support user B , because A and B have interacted with different people in the knowledge work space.

4. The General Architecture of Dynamic Community-based Systems

Creating computational environments that support the

creation of dynamic communities as the platform for knowledge collaboration entails not only technical issues such as locating relevant information and task-specific experts, but also social issues such as the motivation to participate in dynamic communities.

This section discusses the general architecture of sociotechnical environments that support dynamic community-based knowledge collaboration. As shown in Figure 3, such a system consists of five major elements: knowledge repository; event detection; dynamic community formation; social-aware communication mechanisms; and information extraction and accumulation.

4.1 Knowledge Repository

A knowledge repository is a model of a knowledge work space. Similar to most repositories in knowledge management systems, the repository stores information and its relation extracted and formalized from a domain. The main difference is knowledge repositories of dynamic communities also store relation between information and people as well as social relation between people.

Existing community theories focus mainly on the relation between members and community. Its locus of discussion is on whether a member belongs to a community or not, and counts only interactions inside the specific community. It assumes all members in a

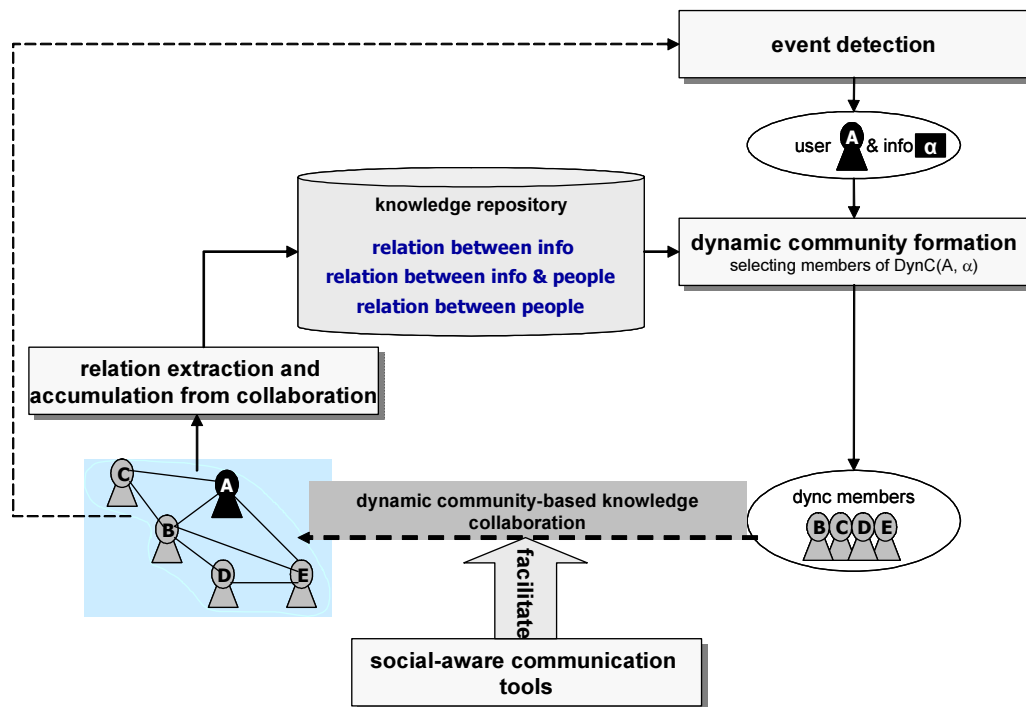


Figure 3: System architecture of sociotechnical environments in support of dynamic community-based knowledge collaboration

community have a uniformed relation, and every member is ready to help other members. Dynamic communities focus on the relation between members in a community by taking into full consideration the proximity of members: how a member is related to others in their past interactions. While considering the relation between community members, dynamic community counts all interactions both in and out of the community. For example, suppose that A has a Java question that B has the expertise to help with. Although A and B have never interacted with each other in the Java domain, there is a high possibility that B would be very glad to help A about the Java question if A and B played soccer together.

To create such an all-embracing social relation between people, the environment needs to extract member interactions out of the supported application domain, through such as analyzing emails of its members, in addition to tracking interactions within the domain.

4.2 Event Detection

The formation of a dynamic community is triggered by a specific event in which a user working on a task needs help. The event detection subsystem monitors user activities in a knowledge work space and identifies those events that trigger the needs of creating dynamic communities. Event detection can be either autonomously performed by the system or activated by users.

In the autonomous detection case, rule-based mechanism can be used to define triggering events. Or, software agents can be developed and deployed to monitor user's work space and autonomously infers the needs for the creation of dynamic communities. For example, in the CodeBroker system, which we will discuss in the next section in detail, an agent monitors programmers' editor, infers their needs for reusable library components and then leads programmers into the creation of dynamic communities.

In the user activated case, a user interface is provided for users to specify who they are and what tasks they are doing that needs the support of a dynamic community.

In both cases, the event detection subsystem analyzes the event and produces two output items: *who* is in need of help and *what* information is needed for the task.

4.3 Dynamic Community Formation

The dynamic community formation subsystem accepts *who* and *what* as inputs and produces a list of members that will participate in the dynamic community.

Figure 4 shows the process of forming a dynamic community for a specific user A and information α . The process consists of three steps: locating relevant information; identifying experts; and selecting experts.

In the first step, information relevant to α from the knowledge repository is located. The definition of relevance depends on the application domain of the supporting environment. For example, for software reuse support, relevant information is those library components that have the similar functionality to the programming task the programmer is developing. For software maintenance support, relevant information will be those program modules that are affected by the module of concern.

The second step identifies experts on the specific information (both the original information and relevant information). Experts are identified based on the relation between information and people in the knowledge repository. Similar to the definition of relevance of information, the relation between information and people is defined differently in different domains. For software reuse support, related experts are those people who created the library components or who have used the components before because those experts can provide help on understanding and using the components. For software maintenance support, related experts are those people who have originally developed or maintained the modules, and those people who have developed or maintained the modules that are affected by the module of concern, because they could provide insights on issues such as how the module was developed and what tacit assumptions were made.

The experts identified in step 2 are derived from the relation between information and people. Step 3 selects from this group those experts that are likely willing to provide help to user A . Selection of experts is based on the member relation stored in the knowledge repository. In other words, experts who have ever interacted with user

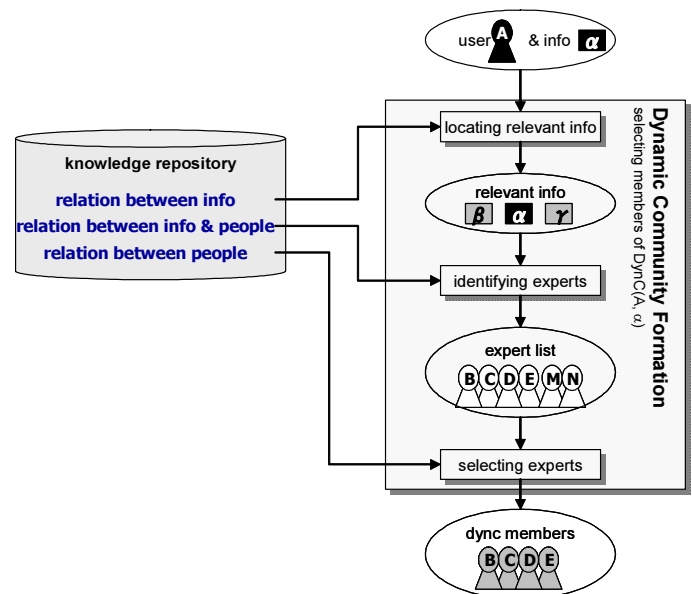


Figure 4: The dynamic community formation process

A are selected as members of the dynamic community. For example, suppose members B , C , and D are experts on α . If B has got help from A a week ago on another problem, C has worked together with A in a previous project, and D and A do not know each other, then B and C should be selected over D because they are more likely to help A .

Another problem needs to be considered in the selection of experts is the balance of workload on experts. Each expert has their own workload and cannot afford participating in too many dynamic communities. Therefore the selecting process needs to consider how many times a member has participated in different dynamic communities recently and chooses those members who are less involved.

4.4 Social-Aware Communication Mechanisms

Effective knowledge collaboration in a dynamic community depends on the active participation of each member. To motivate active participation, the following social issues have to be carefully considered.

4.4.1 Unobtrusive Notification Mechanism

Members who are selected to a dynamic community need to be notified. When a notification is sent to selected members, it should not create obtrusive distraction from those members' work; otherwise those members might be too irritated to be motivated to participate. Peripheral notification mechanisms are desirable because they reduce the possibility to distract users from their own primary tasks while remaining perceptible [1].

4.4.2 Excuse Space

The members who are selected to the dynamic community $DynC(A, \alpha)$ should not be made known to A , so that A would not know even if a member declined to participate for some reason. This could reduce the social pressure that the selected members feel because they do not need to worry that A would know that they declined to help A . In short term, it seems a good thing that the selected members get pressed to participate. In long term, however, if members always feel pressed to participate in any dynamic community no matter what their own situation is, their overall motivation would be gradually chipped away. It is very important to give members the *excuse space* for them to decide not to participate without worrying about potential social backlash of non-participation, so that each participant is truly voluntarily and members can choose to participate freely when they feel like it. Long-term success of knowledge collaboration relies on the motivated and active participation of expert

members; therefore respecting experts' choice should be the priority.

4.4.3 Motivating Mechanism

When the participation notification is presented to the selected members, certain information can be added to exert social pressure in motivating participation. For example, if a notification of participating in $DynC(A, \alpha)$ is sent to B with added information such as reminding B that A has previously helped B in other situations, B will feel obliged to participate, following the social norm of generalized reciprocity [7]. However, such explicit exertion of social pressure must be carefully tempered with excuse space. Otherwise, the explicit representation of social pressure will likely create negative effects on motivation in the long run.

4.5 Information Extraction and Accumulation

A dynamic community forms when a user needs help for a specific task and disassembles when the task is finished. During the lifecycle of the dynamic community, members collaborate with each to create new knowledge. Such newly created knowledge should be captured, extracted and accumulated in the knowledge repository for later use.

The accumulated knowledge should be stored together with link to those who contributed to its creation so that later comers know to whom they can turn for help when the knowledge is needed in new tasks.

The communication and collaboration in the ad hoc formed dynamic community might also lead to the formation of new dynamic community if the system detects the happening of certain triggering events.

5. Application in Software Development

We are currently conducting research on applying the dynamic community theory to various domains. This section describes our efforts in extending the CodeBroker system [24] (a software reuse support environment) to incorporate the support for dynamic community as a way of facilitating and promoting software reuse. This section will also discuss the possibility of applying the theory to other activities of software development.

5.1 Software Reuse Environment

A software reuse process consists of three steps: location, comprehension, and modification [4]. Programmers first have to locate potentially reusable components, comprehend the component, and modify it according to their current needs. Each step is accompanied with difficulties and needs tool support.

Software reuse research has mainly focused on the development of various locating mechanisms and locating support systems [8, 11, 15, 25]. Automated computer support for comprehension and use is very hard because they are too knowledge intensive.

The dynamic community theory provides a framework to design comprehension support systems through the mutual support of peer programmers. This section describes our effort in applying dynamic community theory to incorporate comprehension support into the reuse environment CodeBroker [24] that we have developed. The description of the system follows the process depicted in Figure 3 and Figure 4.

Event Detection. CodeBroker has a software agent [12] running behind program editor, *Emacs*, to monitor programmers' development activities. When it detects the input of a doc comment and/or a signature, it extracts the contents of the doc comment and signature. In the example shown in Figure 5, CodeBroker detects that "programmer *A* wants to create a random number between two integers."

```

/** This class simulates the process of card dealing. Each card is
    represented with a number from 0 to 51. And the program produces
    a list of 52 cards, as it is resulted from a human card dealer */
public class CardDealer1 {
    static int [] cards=new int[52];
    static {
        for (int i=0; i<52; i++) cards[i]=i;
    }
    /** Create a random number between two limits */
    public static int getRandomNumber (int from, int to) {

```

```

--:## CardDealer1.java (JDE)--L10--A11-----
1 0.63 getInt Generate a random number using the default generat
2 0.64 getLong Generate a random number using the default generat
3 0.59 getFloat Generate a random number using the default generat
4 0.59 getDouble Generate a random number using the default generat
--:## *RCI-display* (ReusableComponentInfo)--L1--Top-----
com.objectspace.jgl.util.Randomizer::static int getInt(int lo, int hi)

```

Figure 5: A screen shot of CodeBroker

Locating Relevant Information. CodeBroker extracts the contents of doc comment and signature as a search query. It then uses information retrieval techniques (in particular, probabilistic information retrieval [20]) and signature matching [26] to compute the similarities between the components in Java API library and the search query. The most similar library components are autonomously delivered, without any user operation, into the delivery buffer, which is the bottom part in Figure 5. The first component in the delivery buffer `getInt` can be directly reused to implement *A*'s program. Clicking on the delivered component, the full documentation for the component will be displayed in a web browser. If *A* still has trouble in understanding and using the component, the system will automatically locate an example that uses `getInt` to assist comprehension [23].

Identifying and Selecting Experts. Even with the documentation and use example, *A* might still not know how to use the component. Programmer *A* needs to get help from those programmers who used `getInt` before. CodeBroker searches programmers' directories in the same knowledge work space as *A*, finds all the Java programs that use `getInt`. The programmers who wrote those programs are considered experts for *A*'s task. Suppose four programmers *B*, *C*, *D*, and *E* are identified as experts. Using the relation between programmers in the knowledge work space, CodeBroker finds that except *E*, all the remaining three programmers have interacted with *A* in one occasion or another. Therefore, programmers *B*, *C*, and *D* are selected as the member of the dynamic community for programmer *A* and component `getInt`.

Sending Participation Notification. The identification and selection processes are automatically conducted in the background, and *A* does not know who are selected. When *A* activates the command `get-peer-help` in the pop-up menu associated with `getInt` in the delivery buffer, CodeBroker starts sending a message of participation notification to *B*, *C*, and *D* without letting *A* know to whom the messages are sent.

Knowledge Collaboration. Upon receiving the request for help, *B* decides to help *A*, he or she can reply to *A* by sending a message. This message exchange will be recorded in the knowledge repository to reflect the updated relation between *A* and *B*. Subsequent knowledge collaboration can proceed either online or offline, depending on the choice of *A* and *B*. If conducted online, the message exchanges among the members will be captured and stored in the knowledge repository so that other programmers who needs help for using `getInt` later can benefit.

5.2 Other Possible Applications

The dynamic community theory can be applied to support different tasks in software engineering such as project estimate, open source software development, and software maintenance. Currently, we are still at the stage of theoretical exploration. To illustrate its far-reaching potential, this section sketches its possible application in maintenance support.

After a software system has been developed and deployed, it is not rare that the original developers are assigned to other projects and the maintenance work is handed over to other members. Furthermore, many software development and maintenance are now conducted in geographically distributed sites [9]. Under such conditions, maintainers often do not know who the original designers and developers of the module under

maintenance are and do not know who to approach for design rationales. We believe that dynamic community-based system can provide a solution.

Suppose maintainer A needs to modify a module α during the maintenance phase. 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 [16]. Because those original programmers have new assignments as their current work, they might not be readily available to help A . Coupling this expert list with the informal social relation of A , a list of experts who are most likely to assist A can be selected to become a dynamic community for this task.

6. Conclusions

In this paper, we proposed the dynamic community theory as a new approach to supporting knowledge collaboration that seamlessly integrates the traditional knowledge management approach and the community-based approach. The dynamic community theory takes into full consideration the ever-changing needs for new knowledge of individual knowledge workers as well as the role of social relationship of knowledge workers in effective knowledge collaboration. A general system architecture of sociotechnical environments in support of the formation of dynamic communities was presented, with the description of an example system CodeBroker that supports the location, comprehension and modification of reusable software components. The research is still undergoing, and we are currently trying to apply the dynamic community theory to different tasks of software development as well as in other application domains such as E-learning and application use support. We hope that the notion of dynamic community can stimulate more research interest in exploring the seamless integration of social support into software development environments to reflect the fact that efficient and effective knowledge collaboration is an essential factor for the success of large-scale software projects and software development organizations.

Acknowledgements. This research is supported by (1) the National Science Foundation, Grants CCR-0204277; (2) the Ministry of Education, Culture, Sports, Science and Technology of Japan, Grant 15103 of the Open Competition for the Development of Innovative Technology program; and (3) Japan Science and Technology Corporation, Research and Development for Applying Advanced Computational Science and Technology. We thank Prof. Kumiyo Nakakoji of University of Tokyo, Prof. Katsuro Inoue of Osaka

University, and Prof. Ken'ichi Matsumoto of Nara Institute of Science and Technology, who have made major contributions to the dynamic community theory described in the paper.

References

- [1] Carroll, J.M., D.C. Neale, Philip L. Isenhour, M.B. Rosson, and D.S. McCrickard, Notification and Awareness: Synchronizing Task-Oriented Collaborative Activity. *International Journal of Human-Computer Studies*. **58**(2003):605-632, 2003.
- [2] Curtis, B., H. Krasner, and N. Iscoe, A Field Study of the Software Design Process for Large Systems. *Communications of the ACM*. **31**(11):1268-1287, 1988.
- [3] Fischer, G., User Modeling in Human-Computer Interaction. *User Modeling and User-Adapted Interaction*. **11**(1&2):65-86, 2001.
- [4] Fischer, G., S. Henninger, and D. Redmiles, Cognitive Tools for Locating and Comprehending Software Objects for Reuse, in *Proceedings of 13th International Conference on Software Engineering (ICSE'91)*, Austin, TX, 318-328, 1991.
- [5] Fischer, G. and J. Ostwald, Knowledge Communication In Design Communities, in *Barriers and Biases in Computer-Mediated Knowledge Communication*, R. Bromme, F. Hesse, and H. Spada, (eds.) Kluwer Academic: Dordrecht, Netherlands, (to appear), 2004.
- [6] Fischer, G. and J. Ostwald, Knowledge Management — Problems, Promises, Realities, and Challenges. *IEEE Intelligent Systems*. **2001**(January/February):60-72, 2001.
- [7] Fischer, G., E. Scharff, and Y. Ye, Fostering Social Creativity by Increasing Social Capital, in *Social Capital*, M. Huysman and V. Wulf, (eds.), (in press), 2004.
- [8] Frakes, W.B. and B.A. Nejme, An Information System for Software Reuse, in *Software Reuse: Emerging Technology* IEEE CS Press, 142-151, 1990.
- [9] Herbsleb, J. and A. Mockus, An Empirical Study of Speed and Communication in Globally-Distributed Software Development. *IEEE Transactions on Software Engineering*. **29**(3):1-14, 2003.
- [10] Hollan, J., E. Hutchins, and D. Kirsch, Distributed Cognition: Toward a New Foundation for Human-Computer Interaction Research, in *Human-Computer Interaction in the New Millennium*, J.M. Carroll, (ed.) ACM Press: New York, 75-94, 2001.
- [11] Inoue, K., R. Yokomori, H. Fujiwara, T. Yamamoto, M. Matsushita, and S. Kusumoto, Component Rank: Relative Significance Rank for Software Component Search, in *Proceedings of 2003 International Conference on Software Engineering (ICSE'03)*: Portland, OR, 12-24, 2003.
- [12] Klusch, M., ed. *Intelligent Information Agents: Agent-Based Information Discovery and Management on the Internet*. Springer-Verlag: Berlin, Germany. 1999.
- [13] Lave, J. and E. Wenger, *Situated Learning: Legitimate Peripheral Participation*, Cambridge University Press: Cambridge, UK. 1991.
- [14] Lindvall, M. and I. Rus, Knowledge Management for Software Organizations, in *Managing Software*

- Engineering Knowledge*, A. Aurum, et al., (eds.) Springer, 73-94, 2003.
- [15] Mili, A., R. Mili, and R.T. Mittermeir, A Survey of Software Reuse Libraries, in *Systematic Software Reuse*, W. Frakes, (ed.) Baltzer Science: Bussum, The Netherlands, 317-347, 1998.
 - [16] Mockus, A. and J. Herbsleb, Expertise Browser: A Quantitative Approach to Identifying Expertise, in *Proceedings of 2002 International Conference on Software Engineering (ICSE'02)*: Orlando, FL., 503-512, 2002.
 - [17] Nardi, B.A., S. Whittaker, and H. Schwarz, It's Not What You Know, It's Who You Know: Work in the Information Age. *First-Monday: Peer-reviewed Journal on the Internet*. 5(5), 2000.
 - [18] Nonaka, I. and H. Tkeuchi, *The Knowledge-Creating Company: How Japanese Companies Create the Dynamics of Innovation*, Oxford University Press. 1995.
 - [19] Polanyi, M., *The Tacit Dimension*, Doubleday: Garden City, NY. 1966.
 - [20] Robertson, S.E. and S. Walker, Some Simple Effective Approximations to the 2-Poisson Model for Probabilistic Weighted Retrieval, in *Proceedings of the 17th International ACM-SIGIR Conference*, Dublin, Ireland, 232-241, 1994.
 - [21] Schön, D.A., *The Reflective Practitioner: How Professionals Think in Action*, Basic Books: New York. 1983.
 - [22] Vivacqua, A., Agents for Expertise Location, in *Proceedings of 1999 AAAI Spring Symposium on Intelligent Agents in Cyberspace*, Stanford, CA, 9-13, 1999.
 - [23] Ye, Y. and G. Fischer, Information Delivery in Support of Learning Reusable Software Components on Demand, in *Proceedings of 2002 International Conference on Intelligent User Interfaces (IUI'02)*, San Francisco, CA, 159-166, 2002.
 - [24] Ye, Y. and G. Fischer, Supporting Reuse by Delivering Task-Relevant and Personalized Information, in *Proceedings of 2002 International Conference on Software Engineering (ICSE'02)*, Orlando, FL., 513-523, 2002.
 - [25] Ye, Y., G. Fischer, and B. Reeves, Integrating Active Information Delivery and Reuse Repository Systems, in *Proceedings of ACM SIGSOFT 8th International Symposium on Foundations of Software Engineering (FSE8)*, San Diego, CA, 60-68, 2000.
 - [26] Zaremski, A.M. and J.M. Wing, Signature Matching: A Tool for Using Software Libraries. *ACM Transactions on Software Engineering and Methodology*. 4(2):146-170, 1995.