Auto-Erecting Virtual Office Walls

Ben van Gameren
Delft University of Technology
IHomer
b.j.a.vangameren@tudelft.nl

Kevin Dullemond
Delft University of Technology
IHomer
k.dullemond@tudelft.nl

Rini van Solingen
Delft University of Technology
Prowareness
d.m.vansolingen@tudelft.nl

***Abstract*—Collaborative Software Engineering is increasingly carried out from multiple, physically separated, locations around the globe. Software engineers are no longer tied to a fixed workplace and have the opportunity to work from the location of the customer, their home and even from their holiday location. When working in such a distributed setting, software engineers also need information about the context in which they are working to be able to collaborate effectively with their colleagues. In the last decades multiple technological solutions were developed by the Software Engineering community to fulfill this need. However, the majority of these solutions only support a single aspect of the development process, so each developer has to manually analyze, filter and combine the available information in order to acquire a sufficient level of awareness. Manually analyzing, filtering and combining available information can however be quite time-consuming and therefore we focus on how to automate this process. In this paper we present our vision on how auto-erected virtual office walls can help distributed software engineers to relatively passively and unobtrusively accomplish this automation.**

***Index Terms*—Activity Theory, Collaborative Software Engineering, Global Software Engineering, Iris, Virtual Office Walls**

I. Introduction

In collaborative work, awareness information is essential to properly cooperate with your colleagues [1], [2]. With awareness information we mean the information which is necessary to provide you with the context in which you are working. Examples of such information items are: information about the other members in the project team, their activities, and information about the current state of the project. Dourish and Bellotti more formally define awareness as [3]: *"An understanding of the activities of others which provides a context for your own activity"*. For software engineers it is essential to have a sufficient level of awareness, because Software Engineering is a collaborative activity which requires engineers to coordinate their efforts to be able to produce a functional system.

However, both due to the globalization of business [4], [5], [6] and due to the fact that people work from home more and more [7], people no longer share a physical work environment and as a consequence cannot exchange information without technological support. So, in order to collaborate with colleagues in a distributed setting, technological support is required to be able to acquire and maintain awareness.

In the last decades the (Global) Software Engineering community has developed many technological solutions to support globally dispersed teams in performing their tasks. Portillo-Rodríguez et al. [8] provide a systematic mapping review of available tools in the field of Global Software Engineering and what functionality these tools offer. Several of the tools discussed are widely adopted by distributed development teams and provide the team members with information. Most of these solutions only support a single aspect of the development process and as a consequence many diverse tools are needed to provide the user with all the information he or she needs. Accordingly, all this information needs to be analyzed, combined and filtered manually by each developer to acquire the information necessary to create the context of his current activity. However, this process can be quite time-consuming, therefore we focus on how to automate this process. In this paper we focus on how to provide distributed software engineers with the awareness information they need. As such the research question of this paper is:

"To which extent can the introduction of virtual office walls help provide distributed software engineers with the context of their current activity?"

We will answer this research question as follows. In section II we define virtual office walls and introduce two prerequisites of the construction of these. Next, in section III, we look at the first prerequisite and discuss that both access to data from a wide variety of tools is needed and the means to integrate it to create valuable information. Subsequently, in section IV, we look at the second prerequisite and discuss a way to describe the context of a software engineer. In section V, we validate our representation of the context of a software engineer in an industrial setting. Finally, we present conclusions and discuss opportunities for future work in section VI and VII.

II. Virtual Office Walls

As discussed in the introduction, Software Engineering is a collaborative activity which requires potentially many developers to coordinate their actions to be able to produce a system. In order to coordinate their actions developers need to distribute awareness information among each other. In the traditional co-located setting all information is available in a single place, the office building, and is accessible by all employees present at that location. In such a co-located setting awareness information is exchanged relatively passively and unobtrusively [1], [9]. But, how are the developers capable of abstracting useful information without experiencing an

COLLABORATECOM 2012, October 14-17, Pittsburgh, United States
Copyright © 2012 ICST
DOI 10.4108/icst.collaboratecom.2012.250440

overload of information? Probably this has mostly to do with the design of the office building [10]. In general an office building consists of several rooms, for example a foyer, a kitchen, meeting rooms and offices. All of these rooms have their own characteristics; the meeting room, for example, has several attributes which facilitate group discussions such as a white board, a beamer and the room's size. By moving around in the building and selecting a room which characteristics match the developer's needs, a developer is able to change the context of his activities. Another example is that people who work on related tasks are often seated in close proximity to each other. By organizing their work environment in such a way, they can easily exchange awareness information between all involved stakeholders. So, when working in a co-located setting, developers are continuously aware of information related to their current task.

However, in a distributed setting developers no longer share a physical work environment and as a consequence cannot exchange information without technological support. So, in order to collaborate with their colleagues developers need to use technological solutions to be able to retrieve information relevant to their current task. The Software Engineering community has developed several solutions to fulfill this need, but most of these solutions only support a specific type of information and this information cannot be processed by other solutions directly [8]. Therefore, developers need to manually analyze the available information to be able to construct the information they need. This increased complexity of information analysis may result in misunderstandings, inconsistencies, incompatibilities and duplicated information [8].

To be able to acquire awareness in a relatively passive and unobtrusive fashion, such as in the co-located setting, we need to automate this analytical process of accessing, combining and filtering the available information. In essence we need to automate the process of restricting the available information to the information that an actor needs to carry out his current activity. We propose to call this mechanism a *'virtual office wall'* and define it as: *"A mechanism which regulates information based on the context it encloses"*. The remainder of this paper will address the design of such a mechanism. To do this we first discuss the prerequisites: (i) access to a data set which at least contains the required data at a certain time and (ii) a method to differentiate between required and not required information. When these prerequisites are met it is straightforward how the mechanism of a virtual office wall can be constructed. For the second prerequisite we discuss a specific method and we test the validity of our approach in a practical case setting as well.

III. Selecting and Combining Information

The first prerequisite of a virtual office wall concerns having access to a data set which at least contains the required data at a certain time. In this section we discuss that to fulfill this prerequisite both access to data from a wide variety of tools is needed and the means to integrate this to create valuable information.

In a co-located setting all information is available in a single place and developers are able to gather all required information in a relatively passive and unobtrusive fashion. In a distributed setting, however, all required information is scattered across multiple sites and technological solutions are needed to exchange this information. It is even impossible to collaborate effectively without some kind of technological support when people do not share a physical work environment. Therefore, in order to collaborate effectively with distributed colleagues, the Software Engineering community has developed a wide variety of tools. Several of these tools are widely adopted by Global Software Engineering teams. Example are: configuration management systems, bug trackers and Instant Messaging solutions. However, the majority of these technologies only supports a single aspect of the development process. So to be able to provide the developers with sufficient information during the entire development process many specialized tools are needed [8]. Because the majority of these solutions focuses on managing a specific type of information, this information cannot be processed by other solutions directly. As a consequence developers need to manually analyze, filter and combine the available information to acquire the information they need to perform their current task. Therefore, access to a wide variety of tools is needed to fulfill the prerequisite of having access to a data set which at least contains the required data at a certain time. Because of the wide variety of systems a wide variety of access mechanisms is needed as well. Additionally, data from the different systems often needs to be combined to create valuable information. The process of combining information from different sources is often referred to as integration and we will further illustrate its value, origins and future by discussing the Coordination Pyramid defined by Sarma et al. [11].

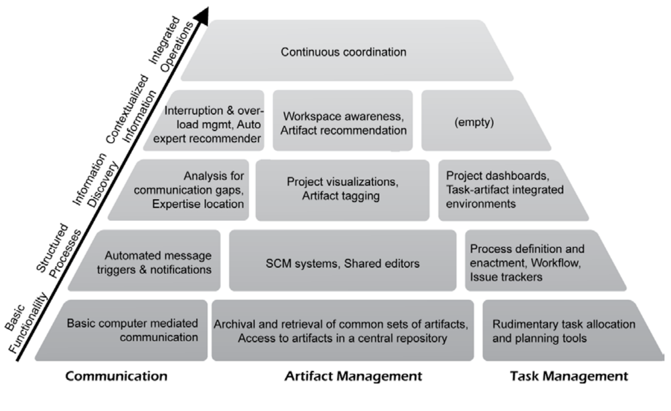
Integrated Operations
Contextualized Information
Information Discovery
Structured Processes
Basic Functionality
Continuous coordination
Interruption & overload mgmt, Auto expert recommender
Workspace awareness, Artifact recommendation
(empty)
Analysis for communication gaps, Expertise location
Project visualizations, Artifact tagging
Project dashboards, Task-artifact integrated environments
Automated message triggers & notifications
SCM systems, Shared editors
Process definition and enactment, Workflow, Issue trackers
Basic computer mediated communication
Archival and retrieval of common sets of artifacts, Access to artifacts in a central repository
Rudimentary task allocation and planning tools
Communication
Artifact Management
Task Management

Fig. 1. The Coordination Pyramid [11]

Sarma et al. [11] have reviewed several software tools which assist developers in coordinating their efforts and proposed a framework to categorize these (see figure 1). This framework organizes types of existing and emerging tools in a hierarchy of paradigms of coordination shifts (the vertical axis of the framework) that have historically emerged. These paradigms are categorized along three strands: communication, artifact management and task management. These three strands represent the basic coordination activities in software development. Developers need to (i) communicate with each other, (ii) coordinate their individual access and changes to a common set of interdependent artifacts and (iii) manage their tasks. Now we briefly discuss and summarize each of the five paradigms.

The first layer in the Coordination Pyramid is the *'Basic Functionality'* layer which focuses on enabling computerized coordination. Technology at this layer allows a team to move from purely manual coordination strategies to strategies that involve automated tools. These tools, however, only focus on a specific aspect of coordination and only automate the minimal functionality needed to support this. Examples of tools in this first layer are: email, scheduling tools and shared file systems. When using such tools developers still make (time-consuming) decisions such as when to coordinate and with whom.

The second layer, *'Structured Processes'*, focuses on guiding the developers in their engagement with the product and their team members. The underlying goal of tools from this layer is to enforce a specific procedure for editing, managing and relating changes to the different project artifacts. Examples of technological support that can be mapped to this layer include shared editors, issue trackers and work flow systems. These tools all reduce the coordination effort per developer because many coordination decisions, which take effort, are now captured by these tools. However, it can be time-consuming to explicitly model and set up the desired processes.

Subsequently, the *'Information Discovery'* layer aims to support informal practices of coordination. Informal coordination relies on users gaining information that establishes a context in which they perform their individual tasks. Tools at this layer try to provide the users with the information necessary to build this context. Examples of these tools are project dashboards, visualization systems and tool support for finding expertise. Tools from this layer combine and visualize information that is already specified by developers as part of other tasks (e.g. commit logs, personal information, work item status) in order to automate tasks that otherwise have to be performed manually. In this layer the benefits of integrating the information from different information sources becomes clear. By combining, for example, information about artifacts that usually are modified together and information about who most frequently modified the related source code files, it becomes possible for a developer to pro-actively determine who best to contact in case of doubt.

Fourthly, the *'Contextualized Information'* layer, tools at this level focus on automatically predicting and providing useful coordination information to create a context in which only relevant information is exchanged in a relatively unobtrusive manner. An example of such a technology is a workspace awareness tool, such a tool provides its users with information about potential conflicting activities undertaken by other users of the system. In this layer it is essential to focus on the interplay of awareness cues presented by the tools and the responses of the developers to these cues to be able to provide a stronger context of one's activities. Because, the stronger a context for one's activities the stronger the opportunity for developers to self-coordinate with their colleagues to swiftly resolve any emerging coordination problems.

Finally, Sarma et al. [11] leave the top of the pyramid open as they believe new paradigms of coordination will emerge as technology and organization practices continue to evolve. However, they do define the ultimate goal (the top of the pyramid) of coordination technologies: to achieve continuous coordination. In other words, the goal is to achieve *"flexible work practices supported by tools that continuously adapt their behavior and functionality so coordination problems are minimized in number and impact"* [12]. In this scenario developers no longer need to use specific coordination tools since coordination and work activities are integrated in a single environment providing its users with all the necessary information. We completely agree with this, since when all necessary information is integrated into a single environment and such an environment provides the necessary information and functionality in a seamless and effective manner it can be used to collaborate effectively with your distributed colleagues.

IV. Context of an Actor

The second prerequisite of a virtual office wall concerns a method to differentiate between required and not required information. In this section we argue that a valid representation of the context of an actor is sufficient to achieve this. Therefore we introduce Activity Theory as a means to represent the context of an actor and argue it is an appropriate representation of Software Engineering activities as well.

Tell et al. [13] propose the use of Activity Theory in order to both structure and describe the context in which distributed software engineers perform their tasks. The origins of Activity Theory are threefold: (i) classical German philosophy, (ii) the writings of Marx and Engels and (iii) the Soviet Russian cultural-historical psychology of Vygotzky, Leont'ev and Luria [14]. The theory was further improved by Leont'ev, one of the three Russian psychologists [15] and became popular after Engeström introduced it to the western world. One of Engeström's main contributions is a systematic representation of the theory; the activity system (see figure 2). This model consists of six elements:

Object	The objective of the activity
Subject	The actor engaged in the activity (either an individual or a group)
Community	The social context of the activity (all actors involved in the activity system)
Instrument	The artifacts or concepts used by the subject of the activity
Division of Labor	The hierarchical structure of the activity
Rules	The laws, rules and regulations that govern the subject inside a community

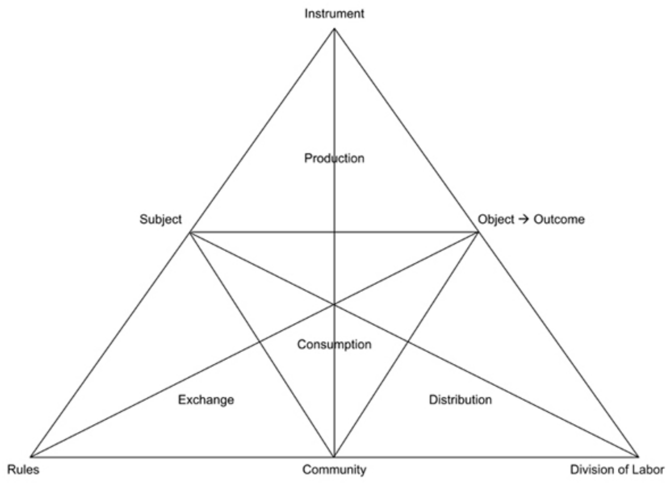

Fig. 2. Activity System [16]

All these elements together represent a single human activity resulting in a single outcome.

In addition to modeling a human activity, it is also necessary to describe the hierarchical structure of that activity. Because, such a structured overview is needed to be able to relate a single human activity to activities carried out by the rest of the team. Leont'ev defined a hierarchical model of human activity which consists of three levels [15]:

Activity	is driven by its motive (e.g. a man participates in a communal hunt because he wants to feed his family)
Action	is driven by its goal (e.g. a man scares away the prey from himself and toward the other members of the hunt)
Operation	is driven by its conditions (e.g. how the man carries out the various tasks involved in his role will depend upon the weather, the terrain etc.)

In [13] Tell et al. discuss how to use Activity Theory to describe different activities and processes of developing software in the context of GSE. They do this by describing, detailing and decomposing Software Engineering activities and map these to the activity system and the hierarchical model of human activity. In this discussion they use the software architecture design process as leading example. They show how this activity can be mapped to the activity system and explain each of the six elements and the outcome of this model. Subsequently, they decompose the software architecture design activity into its composing actions; architecture analysis, architecture synthesis and architecture evaluation. Next, they change the subject to the evaluation manager and emphasize on his/her motive to evaluate a candidate solution. For this activity, from the perspective of the evaluation manager, the same two mappings are applied and discussed. This decomposition can be further applied to a point at which the activity is performed through actions facilitated by technology [13].

Finally, Tell et al. [13] conclude that the introduction of Activity Theory into the field of Global Software Engineering makes it possible to determine what information is needed when performing a specific collaborative task. Because, on the one hand, applying the activity system to Software Engineering activities results in a uniform and detailed description of the current activity. This can be used to determine what information is directly related to the activity. On the other hand applying the hierarchical model of human activity to Software Engineering activities, results in an overview of the hierarchical structure of that activity. This can be used to determine the degree of relatedness between activities. In our opinion using Activity Theory is an appropriate way to describe the context of an actor and as such also to determine which information is required while performing a activity.

V. Industrial Evaluation

In the previous section we concluded that contexts described by Activity Theory are an appropriate way to represent activities of software engineers. To be able to use such contexts to differentiate between required and not required information, the contexts of actors performing the different activities which are common in Software Engineering need to be sufficiently different. To determine whether this is true in practice we have performed an industrial evaluation which is discussed in this section. We conducted a Focus Group in which we attempted to answer the following three questions:

1) Which activities are carried out most frequently?
2) Which instruments are used while carrying out an activity?
3) Which actors are involved while carrying out an activity?

A. Site

Participants in this study are a group of software engineers at IHomer, a Dutch Software Engineering company founded in August of 2008. The company currently employs 19 employees and is fully distributed (see figure 3), since the default location from which the employees work is their home. As a consequence, all employees are experienced with dealing with the difficulties of developing software when working physically separated from each other. This makes this company a suitable setting to conduct this evaluation.

B. Data gathering and analysis

To answer the three questions, we performed a Focus Group [17], [18] to gather the qualitative data we needed. We conducted a Focus Group to gather the insights, ideas, viewpoints and opinions of the people participating because such a setting enables the participants to build on the responses and ideas of others. This process increases the richness of the information gained [19]. We used this method because of its ability to discover new insights and because it is a cost efficient way of obtaining practitioner experience. The Focus Group we conducted lasted approximately 45 minutes and we

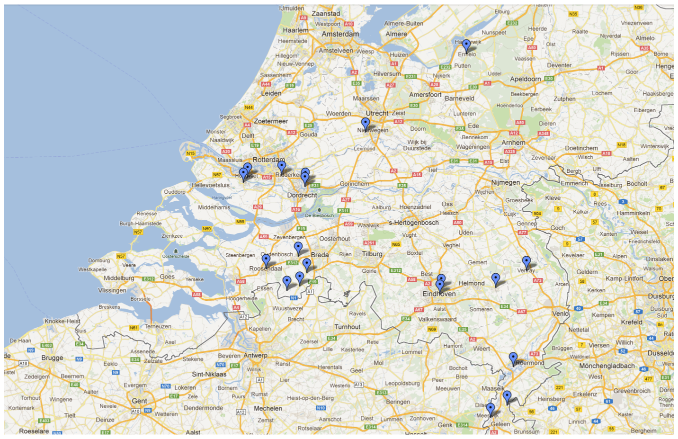

Fig. 3. Geographical distribution of the employees of IHomer

selected six employees of IHomer based on their availability. The Focus Group itself was carried out in a separate office to minimize the influences from outside. During the Focus Group the first author took the role of moderator. As a moderator he (i) explained what a Focus Group entails, (ii) explained what was expected of them, (iii) explained the goal of the Focus Group, (iv) kept the discussions on topic, (v) tried to ensure that all participants contributed to the discussion, and (vi) made sure that the predefined structure of the Focus Group was followed. The identification of the most frequently carried out activities was performed as follows: Firstly, the moderator handed out sticky notes and asked each of the participants to write down what they thought were the five most frequently carried out activities. Following this the moderator gathered the sticky notes from all individuals and discussed each of them with the entire group. In the discussion of each sticky note the group determined what was meant by it and grouped it together with other notes when appropriate, trying to create an overall group consensus. This process eventually resulted in the most frequently carried out activities. Finally, for each of these activities the moderator asked both which instruments are used and which actors are involved while carrying out that activity. We reached consensus in a similar fashion as with the activities.

C. Findings

In this section we discuss and present the findings of the industrial study. The four most frequently carried out activities are shown in figure 4. We only show the four most frequently carried out activities because the participants of the Focus Group unanimously decided those are the most important ones.

Activity 1: Coordination of tasks within the development team to be able to collaborate effectively	
Community	*Instruments*
Team	Communication technologies
Organization	Face-to-Face meetings
	Documentation
	Issue Management System
	Software Repository
	Agenda

Activity 2: Coordination of tasks between the development team and the customer to be able to collaborate effectively	
Community	*Instruments*
Team	Communication technologies
Organization	Face-to-Face meetings
Customer	Documentation
	Issue Management System
	Software Repository

Activity 3: Creation of new software to be able to add new functionality to the system	
Community	*Instruments*
Team	Communication technologies
	Face-to-Face meetings
	Issue Management System
	Software Repository
	Development Environment
	Requirement Management System
	Testing Framework

Activity 4: Testing of new or existing software to be able to guarantee the quality.	
Community	*Instruments*
Team	Communication technologies
	Face-to-Face meetings
	Documentation
	Development Environment

Fig. 4. The four most frequently carried out activities

Next, we discuss the differences between the contexts of the four most common activities and reflect on these differences. Firstly, we can see that in the four most common activities we already identified three different communities, namely: (i) team, (ii) team and organization, and (iii) team, organization and customer. These differences in actors involved in the activity make it possible to regulate information based on the social context. So, for example, when testing a new or existing piece of software you do not need to share this information with all your colleagues and the customer, only direct team members need to be aware of your activities. Secondly, we can also see that the artifacts and concepts used by the subject of the activity differ between the four activities. This makes it possible to filter the information based on the current activity of an actor. When creating new software, for example, an actor does not need information about the upcoming appointments of his colleagues. These two results both confirm the conclusion that applying Activity Theory to the field of GSE can help provide distributed software engineers with relevant information.

D. Limitations

In this section we discuss the limitations of using a Focus Group to gather empirical data as well as the limitations of the empirical evaluation we performed. The first limitation is that all participants worked for a single company; because of this we can only draw conclusions which are internally valid. The internal validity is significant however, since six of the nineteen employees participated in this study (32%). In order to draw more externally valid conclusions the empirical study should be repeated with a sample which more accurately represents the total population of software engineers.

There are also Focus Group specific limitations which have to do with group dynamics, communication styles and the social acceptability of certain topics and opinions which can all influence the discussion and therefore introduce bias [17], [18]. We dealt with these limitations by defining and following a predefined structure to be able to control the overall content of the Focus Group and to make sure that group dynamics did not steer the discussion in an undesirable direction. In order to minimize the negative effects of social acceptability we emphasized the importance that everyone should contribute to the discussion. We also used sticky notes to force everyone to think about the question in advance to reduce the temptation to agree with the loudest person or the first person to give his opinion. The final limitation is the possibility that participants have hidden agendas [17].

VI. Conclusion

In this paper we have discussed how the the introduction of virtual office walls can help provide distributed software engineers with the context of their current activity. We discussed that in essence a virtual office wall is an automation of restricting the available information to the information that an actor needs to carry out his current activity. Subsequently, we discussed the two prerequisites: (i) access to a data set which at least contains the required data at a certain time and (ii) a method to differentiate between required and not required information. As a way to implement the second prerequisite we discussed Activity Theory, how this can be applied to represent the context of software engineers and validated this application in a practical setting. As such, the main contributions of this paper are the following:

- The definition of a virtual office wall as *"A mechanism which regulates information based on the context it encloses"*
- The prerequisite of a virtual office wall that both access to data from a wide variety of tools is needed and the means to integrate this to create valuable information
- The prerequisite of a virtual office wall that a method is needed to differentiate between required and not required information
- The applicability of Activity Theory as a way to represent the context of a software engineer
- The validity of the applicability of Activity Theory in Software Engineering in an industrial setting
- By having access to data from a wide variety of SE tools, a means to integrate this information and Activity Theory based representations of contexts, it becomes possible to automatically erect virtual office walls and as such help to provide distributed software engineers with the context of their current activity

VII. Future Work

The next step in our research is to apply the concept of virtual office walls. We are planning to apply this concept on Iris (see [20]); a cross-platform, web-based, extensible communication framework we are working on. With this platform we aim to both support all awareness needs of software developers in a singe solution and enable integration of the awareness information from different information sources. The current version of Iris, supports the following: (i) Instant Messaging between two or more users, (ii) audio conferencing between two or more users, (iii) video conferencing between two or more users, (iv) showing the availability of the other users, (v) showing the current activity of the other users, (vi) showing tomorrow's work location of the other users, (vii) showing how other users like to be contacted, and (viii) showing the ongoing conversations of all users.

We have chosen to start by supporting standard synchronous communication since our stakeholders identified this as highly valuable because they spend a large portion of their time in communicating. Subsequently we added some information about the actors: (i) their availability because they want to know who is available to decide whether or not to try and contact someone, (ii) their current activity to be able anticipate on this, (iii) tomorrow's work location to be able to more easily identify possibilities for working co-located and (iv) their approachability to be able to know how to contact someone. Finally, we added a list of ongoing conversations to provide users insight in the conversations their colleagues are having.

Currently, we are using Iris to put the ideas discussed in this paper into practice. We are designing a way to incorporate the concept of virtual office walls in this platform to provide distributed software engineers with relevant information related to their current context. We propose to do this by providing the users with a mechanism to contextualize the awareness information. An example of such a mechanism is that a user can define a project group, in which all information about a specific project can be clustered, such as project members, project description, project message board, outstanding project issues and related conversations. Subsequently, the context of a user is confined to one of these projects based on his current activity. We are planning to implement this functionality in the upcoming iterations. One of the main challenges in this is visualizing the contextualization of the awareness information.

Next to designing and implementing this concept in Iris, it should also be evaluated in an industrial setting. We will evaluate the concept of virtual office walls at IHomer because on the one hand, physically distributed collaboration is common since the default location to work from is the employee his home and as such the people have experience with dealing with the difficulties of working distributed form each other. On the other hand, all the employees of IHomer are using the system in their daily activities which enable us to acquire all the information we need to evaluate this concept.

References

[1] K. Schmidt, "The Problem with 'Awareness': Introductory Remarks on 'Awareness in CSCW'," *Computer Supported Cooperative Work*, vol. 11, no. 3-4, pp. 285 – 298, 2002.

[2] A. Syri, "Tailoring cooperation support through mediators," in *Proceedings of the 1997 European Conference on Computer Supported Cooperative Work*. Kluwer Academic Publishers, 1997, pp. 157–172.

[3] P. Dourish and V. Bellotti, "Awareness and Coordination in Shared Workspaces," in *Proceedings of the ACM 1992 Conference on Computer Supported Cooperative Work*. ACM Press, 1992, pp. 107–114.

[4] E. Carmel, *Global software teams: collaborating across borders and time zones*. Upper Saddle River: Prentice Hall PTR, 1999.

[5] J. Herbsleb and D. Moitra, "Guest Editors' Introduction: Global Software Development," *IEEE Software*, vol. 18, no. 2, pp. 16–20, 2001.

[6] J. Herbsleb, "Global Software Engineering: The Future of Socio-technical Coordination," in *Proceedings of the IEEE 2007 Workshop on the Future of Software Engineering*. IEEE Computer Society Press, 2007, pp. 188–198.

[7] The Dieringer Research Group Inc., "Telework Trendlines 2009: A Survey Brief by WorldatWork," 2009.

[8] J. Portillo-Rodríguez, A. Vizcano, M. Piattini, and S. Beecham, "Tools used in global software engineering: A systematic mapping review," *Information and Software Technology*, vol. 54, no. 7, pp. 663 – 685, 2012.

[9] J. Fogarty, S. Hudson, C. Atkeson, D. Avrahami, J. Forlizzi, S. Kiesler, J. Lee, and J. Yang, "Predicting human interruptibility with sensors," *ACM Transactions on Computer-Human Interaction*, vol. 12, no. 1, pp. 119–146, 2005.

[10] T. Allen and G. Henn, *The organization and architecture of innovation: Managing the flow of technology*. Butterworth-Heinemann, 2007.

[11] A. Sarma, A. Van der Hoek, and D. Redmiles, "The coordination pyramid: A perspective on the state of the art in coordination technology," *Computer*, vol. PP, no. 99, p. 1, 2010.

[12] D. Redmiles, B. Al-Ani, T. Hildenbrand, S. Quirk, A. Sarma, R. Silveira, S. Filho, C. de Souza, and E. Trainer, "Continuous Coordination A New Paradigm to Support Globally Distributed Software Development Projects," *Wirtschaftsinformatik*, vol. 49, pp. 28–38, 2007.

[13] P. Tell and M. Babar, "Activity theory applied to global software engineering: Theoretical foundations and implications for tool builders," in *Proceedings of the IEEE 2012 International Conference on Global Software Engineering*, 2012, pp. 21–30.

[14] Y. Engeström and R. Miettinen, *Perspectives on activity theory*. Cambridge Univ Pr, 1999.

[15] A. Leont'ev, "Activity, consciousness, and personality," 1978.

[16] Y. Engeström, "Learning by expanding. an activity-theoretical approach to developmental research," 1987.

[17] J. Kontio, L. Lehtola, and J. Bragge, "Using the focus group method in software engineering: Obtaining practitioner and user experiences," *Empirical Software Engineering, International Symposium on*, vol. 0, pp. 271–280, 2004.

[18] K. D. Bailey, *Methods of Social Research*. New York: Free Press, 1978.

[19] J. Langford and D. McDonaugh, *Focus Groups: Supporting Effective Product Development*. Taylor and Francis, 2003.

[20] K. Dullemond, B. van Gameren, and R. van Solingen, "Supporting distributed software engineering in a fully distributed organization," in *Cooperative and Human Aspects of Software Engineering (CHASE), 2012 5th International Workshop on*. IEEE, 2012, pp. 30–36.