Collaboration should become a first-class citizen in support environments for software engineers

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

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

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

***Abstract*—Much work has been done in developing Integrated Development Environments (IDEs) for supporting software engineers in their isolated programming tasks. Software Engineering however, is primarily a collaborative activity in which communication, coordination and cooperation with colleagues is essential. Supporting this collaboration is often overlooked in support environments while it is in fact highly beneficial for Software Engineering teams in general and distributed teams in particular. Progress has been made in extending existing IDEs with functionality for supporting the collaborative activities in Software Engineering, however such environments are focused primarily on the programming task with collaboration functionality added to that. In this paper we argue the case that collaboration should be at the core of Integrated Collaborative Development Environments by showing exhibits that Software Engineering is primarily a collaborative activity, discussing limitations in the support for this in existing solutions and discussing our own approach in dealing with these limitations.**

I. Introduction

In 2002 Booch and Brown [1] emphasized the importance of a Collaborative Development Environment (CDE) defined as: *"a virtual space wherein all the stakeholders of a project - even if distributed by time or distance - may negotiate, brainstorm, discuss, share knowledge, and generally labor together to carry out some task, most often to create an executable deliverable and its supporting artifacts"*. They argue that [1]: (i) effective teamwork is an essential part of every nontrivial Software Engineering effort and (ii) collaborative capabilities are essential to support these teams, particularly as team sizes get smaller while team interaction becomes more geographically dispersed. They conclude that CDEs are an ideal way to get people to work together effectively.

Since Booch and Brown's paper, the importance of CDEs has only increased because software engineers work more and more dislocated from each other due to the globalization of business [2], [3], [4] and the rising popularity of working from home [5]. However, advances have also been made in the area of CDEs, for example by IBM with Rational Team Concert [6] which extends the IDE with awareness information and collaboration functionalities such as status indicators of team members and direct communication options. However, while providing support for distributed collaboration, such an approach is inherently focused on the programming perspective and merely adds collaborative features to existing individual functionalities. In other words the core of the solution revolves around the individual programming task while the team and the collaboration within the team is secondary. According to Sengupta et al. [7]: *"Most of the work in the CDE domain till date has focused on collaborative coding"*. We argue the next step should be focusing on Collaborative Software Engineering.

However, achieving collaboration support for software development by extending existing solutions is difficult. Sarma agrees and states [8]: *"To create intrinsic collaboration support for software development, researchers need to reevaluate the traditional development practices to design development tools from the ground up to truly support collaboration"*. Next to this, Gerald Weinberg [9] has noted that programming is ultimately a human activity and studies by DeMarco and Lister [10] suggest that on large projects typical developers spend about 70% of their time working with others. So, effective communication plays a key role in software development [11]. Therefore we argue useful insights into how to best support Software Engineers can be identified by creating a CDE which has support for inter-personal collaboration at its core as opposed to support for the individual programming task. Because such an approach is contradictionary to how most solutions attempt to support collaborative Software Engineering at the moment, the goal of this paper is:

To argue that collaboration should become a first-class citizen in support environments for Software Engineers.

To do this we have structured the paper as follows. First, in section II we discuss how Software Engineering is an inherent collaborative and human activity and why this implies the most important thing to support is collaboration. Following this in section III we give an overview of a selection of existing solutions to support collaborative Software Engineering by looking at a number of reviews of the current state of practice. In this analysis we identify a number of limitations of the current solutions and in section IV we discuss our approach in resolving the limitations by the creation of a collaborative environment called Iris: an extensible communication platform bootstrapped and created by and for a group of fully distributed Software Engineers. Finally, we summarize our work and discuss further research possibilities in section V.

II. Software Engineering: a collaborative activity

In this section we present and defend the thesis: *"Collaboration is essential for Software Engineering teams and therefore also for support environments for them"*

We first discuss existing research on the collaborative and social aspects of Software Engineering. A number of sources have reported on this. Firstly, Weinberg [9] is generally considered a pioneer in recognizing that software development is something done by human beings and the implications of that. A significant portion of his work is focused on re-engineering the software development processes from a *"people empowering point of view"*. Secondly, DeMarco and Lister [10], Perry et al. [12] and Jones [13] are examples of studies on the amount of time spent on communication and collaboration with others. In the study of DeMarco and Lister [10] they report that on large projects typical developers spend about 70% of their time working with others. Perry et al. [12] report that in their study over half of developers' time was spent in interactive activities. Jones [13] reports that team activities account for about 85% of the costs of large software systems. Thirdly, Strubing [14] has also performed a study in this area in which he peformed two series of experiments. In the first series, he conducted 10 open-ended interviews and one group discussion; in the second series, he conducted 25 interviews with programmers and two other experts. He found that programmers perform four activities: (i) coding, (ii) organizing their working space and process, (iii) representing and communicating design decisions and ideas, and (iv) communicating and negotiating with various stakeholders. So three of the four core daily activities Strubing found are activities with a significant collaborative aspect. He concludes [14]: *"Being a sociologist, I have found that designing software is a highly cooperative process"*. Finally, Booch and Brown [1] have also performed an experiment to obtain a snapshot of the daily life of a developer which confirms these findings. So, based on these studies, it seems clear that social activities represent a considerable portion of the average day of programmers.

Exhibit 1
Social activities represent a considerable portion of the average day of software engineers **Based on [9], [10], [14], [12], [1], [13]**

Next we show further implications of this by discussing how it is reflected in two movements in Software Engineering which are currently quite popular: Global Software Engineering and Agile Software Development. In Global Software Engineering (GSE) the software development process is distributed between several geographically dispersed locations [15], [16], [17]. GSE is becoming more popular both due to the globalization of business [2], [3], [4] and the rising popularity of working from home [5]. The main challenges that arise due to working dislocated from each other, have to do with difficulties with communication, coordination and control of the development process [18]. Extensive literature exists which reports on the challenges. The most important of these are: lack of informal communication [2], [19], [3], [20], reduced hours of collaboration [21], [22], [23], [24], communication delay [20], [25], [4], [15] and loss of cohesion [2], [26], [4]. As a commonality between these challenges we see that they all have to do with difficulties in collaboration, which confirms our thesis that collaboration is essential in Software Engineering.

Exhibit 2
The main challenges experienced in Global Software Engineering are difficulties with collaboration **Based on [2], [19], [18], [3], [21], [22], [26], [20], [25], [15], [23], [4], [24]**

The second movement in Software Engineering we want to use to illustrate the collaborative nature of Software Engineering is the rising popularity of agile methods. In the last twenty years many sources have reported on the development of lightweight methodologies as a reaction to the inflexibility of existing heavyweight methods [27], [28], [29], [30], [31], [32], [33], [34]. Examples of these new methodologies are:

- *Scrum* [35], [36]
- *eXtreme Programming* [37], [38], [39], [40]
- *Crystal methods* [41], [42]
- *Adaptive software development* [43], [44], [28]
- *Feature driven development* [45], [46]
- *Kanban* [47]

After the development of these methodologies, in 2001, seventeen prominent process methodologists held a meeting to discuss future trends in software development. They noticed their methods had many commonalities and defined a name for methods of this kind: *"agile methods"*. They formed the *"Agile Alliance"* and wrote *"The agile manifesto"* [48] in which they defined four core values:

1) *Individuals and Interactions over processes and tools*
2) *Working software over comprehensive documentation*
3) *Customer collaboration over contract negotiation*
4) *Responding to change over following a plan*

We can see that these core values as well as a set of principles (see [48]) quite strongly have to do with collaboration and communication. Agile methodologies focus on people and their collaboration, both internally (with team-members) and externally (with the end users). One of the core reasons Agile Software Development has been so popular in practice [49], [50] is this focus on collaboration. So, the popularity and success of agile methods also help build our case that collaboration is an important aspect of Software Engineering.

Exhibit 3
The popularity in practice of Agile Software Development is grounded on its emphasis on people and their collaboration **Based on [27], [28], [29], [30], [48], [31], [32], [33], [34]**

These three exhibits underline that collaboration is essential in Software Engineering because (i) it represents a significant portion of the daily activities of software engineers, (ii) a lack of it leads to significant challenges (GSE) and (iii) strengthening it is the main benefit of Agile Software Development. Because of the value in good collaboration and harm caused by having insufficient collaboration it can be argued that adequate support for collaboration is indispensable.

III. Support environments for software engineers

In the previous section we discussed how Software Engineering is inherently a collaborative activity. It is precisely the collaboration between team members through communication, coordination and cooperation [51] which is the core of the activity and therefore essential. In this section we discuss existing support environments for software engineers, discuss how these solutions include support for collaborative activities and identify gaps (limitations) in the existing support for collaborative Software Engineering.

Today, many environments exist to support software engineers and most of these include at least some support for collaboration as well. However, they emerged with different initial goals in mind. Firstly Integrated Development Environments (IDE) exist of which Eclipse[1] and Microsoft Visual Studio[2] are probably the most well-known. IDEs evolved from code editors which were extended with functionalities software developers frequently use such as code repositories and support for functional testing. Such environments however are primarily individual productivity tools [1]. They have also been combined with collaborative technologies such as code-repositories and Instant Messaging platforms. Examples of environments with these extensions are: Rational Team Concert [6], Merlin Toolchain[3], Team Foundation Server (TFS) and Visual Studio Team System[4]. Similar to environments which evolved from code editors, environments also exist which evolved from being a host for projects. Examples are SourceForge[5], Launchpad[6], Google Code[7] and Microsoft CodePlex[8]. These technologies have been extended with all sorts of functionality regarding collaborative development such as: code review support, build systems and bug tracking. Thirdly, there exist systems created to centralize the building of the system. Examples of such systems are Apache Continuum[9], CruiseControl[10] and Tinderbox[11]. These systems, in turn, have also been extended with collaboration supporting technologies such as scheduling software.

Some of the support environments discussed above are considered by some to be Collaborative Development Environments (CDEs). Lanubile et al. [52] discuss and compare support of nine support environments they consider CDEs. In another work Lanubile [53] indicates SourceForge is the most popular CDE with over 170.000 hosted projects and 1.800.000 registered users. Looking at the functionality of the discussed CDEs however, it can be seen that the "integration" of collaborative functionalities is mostly limited to making the functionality available in a single platform. Providing a mailing list, message board or Instant Messaging is one thing, but integrating it in the process is another. An example of this is being able to start conversations based on and linked to your current work context. With regard to integrations like this there is still quite some work to be done in the field of CDEs.

Limitation 1
Existing support environments for software engineers focus on the individual programming task

In the CSCW (Computer Supported Collaborative Work) community much research has been done about collaboration and what information is essential to do this effectively. One of the primary drivers in this research is the notion of awareness, defined as: *"An understanding of the activities of others which provides a context for your own activity"* [54]. Researchers in the Software Engineering community have leveraged the work done in the CSCW community to provide support for software engineers as well. A number of studies has been done to research the existing tool support for collaborative Software Engineering (e.g. [55], [56], [52], [57], [7], [8]). Most of these studies have focused on Global Software Engineering which is appropriate because collaboration issues using tools are much more apparent in an environment where the tool is the primary (and sometimes only) way to carry out the collaboration. One of the most striking outcomes of these studies is that there exist many separate tools to support collaborative Software Engineering but very little has been done to integrate them. Portillo-Rodríguez et al. [55] for example concludes: *"although there are sufficient tools to support most areas or processes in the software life cycle, there is a lack of connection between the tools. Almost only when using tools from the same company (i.e. IBM or Microsoft tools), and only in some areas, is it possible to integrate the different tools."*. Steinmacher et al. [56] agrees: *"In general, most part of the primary studies (79%) focuses on introducing a new tool with some awareness support to GSE."*. In fact, most of the integration in collaborative tools for Software Engineering has been done in IDEs, code hosts and build servers discussed in the previous paragraphs. Next to this, a significant portion of these isolated single-purpose tools are academic research tools and therefore less likely to be usable in a industrial setting.

Limitation 2
Collaborative functionalities are often only available in isolated (special purpose) environments

The second thing that stands out when looking at these studies is that most of the surveyed tools are related to code specific tasks [7]. Steinmacher et al. [56] states: *"The main focus is given to studies gathering information from source code version management repositories, used to provide awareness supporting both coordination and cooperation"*. As we just discussed in the previous section, Software Engineering is primarily a collaborative activity so focusing more on support for this is an important next step in the research in this area.

[1] http://www.eclipse.org
[2] http://msdn.microsoft.com/en-us/vstudio
[3] http://merlintoolchain.sourceforge.net
[4] http://www.microsoft.com/visualstudio/en-us/products/teamsystem
[5] http://sourceforge.net
[6] https://launchpad.net
[7] http://code.google.com
[8] http://www.codeplex.com
[9] http://continuum.apache.org
[10] http://cruisecontrol.sourceforge.net
[11] http://www.mozilla.org/projects/tinderbox

Limitation 3
Tools which support collaboration mainly focus on code related collaboration

Finally, Steinmacher et al. [56] have also surveyed what portion of the tools support coordination, cooperation and communication. They conclude that the vast majority of the work has revolved around supporting coordination and that in particular support for communcation in CDEs is a: *"fruitful research topic"* [56].

Limitation 4
Tools which support collaboration mainly focus on easing coordination. Support for communication and cooperation is supported less

It is important to try to resolve the limitations discussed in this section because Software Engineering is a highly collaborative activity as has been discussed in the previous section. Additionally, providing improved support for collaborative Software Engineering is even more important for supporting Global Software Engineering and Agile Software Development, because of an increased dependence on supporting technologies for collaboration in the first case and because of the increased focus on collaboration in the latter.

IV. Our approach

In this section we discuss our own approach in trying to resolve the issues identified in the previous section through the creation of Iris[12]. We do this to show our first steps in actually implementing a CDE with support for inter-personal collaboration at its core. First, we discuss the main objectives we want to achieve and the reasoning behind these objectives. Following this we discuss how we aim to reach the objectives by describing the setting in which the solution will be evaluated and the process we employ in developing and evaluating the solution in that setting. Finally, we also discuss initial results.

A. Objectives

The core idea behind of our approach is that Software Engineering is primarily a collaborative activity and therefore support for communication, coordination and cooperation should be the core of a support environment for Software Engineering. The first limitation of existing support we discussed in the previous section is that existing support primarily focuses on the individual programming task. In our approach we will focus primarily on creating a collaboration platform and only secondarily support the individual programming task to try and target this limitation. So, the environment we are creating focuses on providing support for communication, coordination and cooperation first. Subsequently, additional functionality can be added to support the individual programming task which can build on top of the collaborative functionality which is the core of the system.

Secondly, we found that many of the solutions that support sharing awareness between the members of the development team are not integrated. This integration is in fact valuable. Sillito et al. [58] for example reports on an empirical study on how programmers resolve change tasks and how tools support them in answering questions they have in the process of carrying out these tasks. They report that most of the tools that they researched treat questions as if they are asked in isolation while they often are, in fact, part of a larger process. Examples are asking questions on different levels of abstraction and asking questions involving different information sources. Because awareness questions are often part of a larger process, involving a series of questions and activities that provide context, it is often difficult for distributed software engineers to obtain sufficient contextual awareness [59]. Therefore our second objective is to provide all awareness needs of software engineers in a single platform.

Thirdly, Sillito et al. [58] also state that even programs that do support asking different questions generally fail at combining the information in a useful way and merely report the information in isolation as largely undifferentiated and unconnected lists. Because of this we also find it important to enable the integration of the awareness information from different sources. Therefore our third objective is to provide support for truly integrating information from different sources and to be able to utilize this information to support the software engineer.

To summarize, our core objectives are the following:

1) Make collaboration the core of the support environment
2) Support all awareness needs of software engineers in a single platform
3) Enable integration of the awareness information from different information sources

Out of these objectives, Objective 1 is intended to deal with Limitation 1 from the previous section while Objectives 2 and 3 are intended to deal with Limitation 2. Limitations 3 and 4 have to do with restrictions to code related tasks and coordination respectively. We will deal with these limitations by not imposing such restrictions.

B. Setting

The development and evaluation of the solution we are creating is relatively unique because it is done at a company called IHomer, a Dutch Software Engineering company founded in August of 2008, which is distributed in the true sense of the word. In the company everyone is responsible for all business decisions like the strategy, vision and core values, in contrast with employees at 'regular' companies who are mainly responsible for the specific role they fulfill. At the moment the company employs 19 participants.

In the company, physically distributed collaboration is common since home is the default location to work from (see Figure 1 for a map showing the different primary working locations of the IHomers). As we mentioned before, Software Engineering settings where the software engineers are physically dislocated from each other is a particularly appropriate setting for researching support for software engineers because collaboration issues using tools are much more apparent in an

[12]Named after the Greek goddess Iris: messenger of the gods

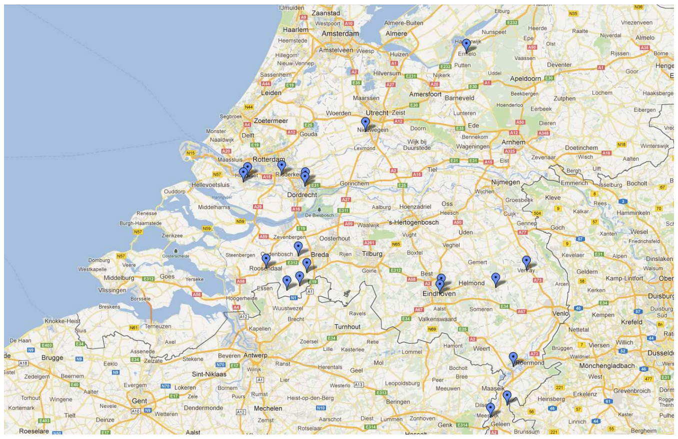


Fig. 1: Distribution of the IHomers

environment where these tools are the primary (and sometimes only) way to carry out the collaboration. This makes the company a particularly suitable setting for performing our research. Additionally, the people are quite experienced with dealing with the difficulties of working distributed from each other and therefore have quite a good understanding of what is needed to improve this situation. Because of this we can closely collaborate with them to determine what functionality is most beneficial to support first and what are good ways to do this. Next to this, they also collaborate with us in realizing the actual technical implementations which improves the quality of the solutions and reduces the time it takes to realize these. Finally, because they encounter the issues we are attempting to solve in their daily work, the solutions will also benefit them directly.

C. *Process*

We have chosen to use Scrum [35], [36] to implement the Iris platform. This decision is based on the specific characteristics of the project, the setting we are conducting research in and our own experiences in the past. Firstly, the process should be able to cope with uncertainty and changing requirements since we are creating a genuinely novel product and projects creating genuinely novel products are often faced with uncertainty regarding both requirements and implementation technologies. Secondly, the process should involve the intended users of the system as strongly as possible because they are quite experienced with working in a distributed setting since this is something they encounter on a daily basis. Finally, the process should stimulate the usage of the solution by all users by providing value as soon as possible. This is important because in earlier research [60] we found that the value of awareness sharing technology (CSCW groupware) is higher when a larger portion of the team uses it and that this is often a problem when introducing such tools in industry.

The Scrum agile methodology fulfills all these requirements. Firstly, it is better able to cope with uncertainty and changing requirements in projects than plan-driven approaches [61]. The main ways in which this is accomplished is by acquiring rapid feedback from the actual users of the system by using short iterations, rapid deployment and working closely with the end users. Secondly, Scrum advocates working closely with the end users both to acquire feedback and to discover how value can be created as quickly as possible resulting in increased satisfaction and commitment.

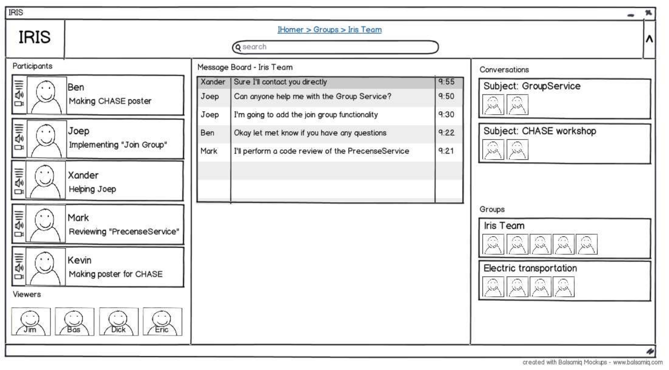
Basically we are bootstrapping a solution for supporting collaborative Software Engineering with a group of people that have a particular need for this solution. We are using an emergent process in doing so because we do not know up front exactly what we need and we use the process to find out what that is. The starting point we have in this approach to reach the objectives we discussed earlier, is the following: Firstly, we create native support for communication, cooperation and coordination which can be utilized directly by the end users of the system but also by extensions. We propose to use an extensible plugin architecture, as can be found in the Jazz and Eclipse IDEs, to facilitate this because using such an approach will provide scalability and enable the utilization of the large amount of existing supporting tools for software engineers. Secondly, we create support for integrating these different extensions with each other.

D. *Initial results*

The project has been running for half a year and the current version is used by the majority of the IHomers in their daily work. We have experienced we are using a suitable approach by creating the solution iteratively in close collaboration with the people actually using the software. A consequence of working iteratively however is that functionality is added in slices, we (together with our stakeholders) choose to add what is most valuable to the users first. Therefore the core of the system is still under development. We are however constructing it bit by bit, each time utilizing the new functionality immediately to provide value to the users.

In the rest of this section we will describe the current version of the system (see figure 2 for a design impression and in figure 3 for a screenshot). Firstly, all colleagues of a user are available in a list with their personal photo. In this list it is possible to see: (i) whether or not that colleague is available, (ii) what his current activity is and (iii) what location he is planning to work tomorrow. The reason the stakeholders asked to support seeing tomorrow's location is that with that knowledge they can decide whether or not to work physically co-located on the next day. They did not see the need to also support the current location but we noticed that users often embed this in their display name by appending *"@ [Location]"*. So, it is likely support for this will be added as well.

Secondly, we support communication between two or more users using three different media: (i) Instant Messaging, (ii) Audio and (iii) Audio/Video. Ongoing conversations are visible also to people not participating in the conversation in a list on the right hand side of the screen. This is done to give users insight in the conversations their colleagues are


IRIS
IRIS
IHomer > Groups > Iris Team
search
Participants
Message Board - Iris Team
Conversations
Groups
Viewers
Jim
Bas
Dick
Eric
created with Balsamiq Mockups - www.balsamiq.com

Ben
Making CHASE poster

Joep
Implementing "Join Group"

Xander
Helping Joep

Mark
Reviewing "PrecenseService"

Kevin
Making poster for CHASE

Xander	Sure I'll contact you directly	9:55
Joep	Can anyone help me with the Group Service?	9:50
Joep	I'm going to add the join group functionality	9:30
Ben	Okay let met know if you have any questions	9:22
Mark	I'll perform a code review of the PrecenseService	9:21

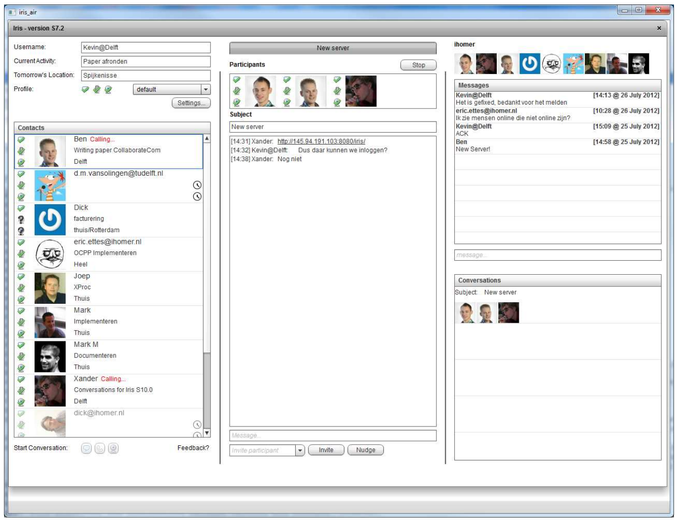
Subject: GroupService

Subject: CHASE workshop

Iris Team

Electric transportation

Fig. 2: Design impression of Iris



Username: Kevin@Delft
Current Activity: Paper afronden
Tomorrow's Location: Spijkenisse
Profile: default

Ben Calling...
Writing paper CollaborateCom
Delft

d.m.vansolingen@tudelft.nl

Dick
facturering
thuis/Rotterdam

eric.ettes@ihomer.nl
OCPP Implementeren
Heel

Joep
XProc
Thuis

Mark
Implementeren
Thuis

Mark M
Documenteren
Thuis

Xander Calling...
Conversations for Iris S10.0
Delft

dick@ihomer.nl

Subject: New server

[14:31] Xander: http://145.94.191.103:8080/iris/
[14:32] Kevin@Delft: Dus daar kunnen we inloggen?
[14:38] Xander: Nog niet

Kevin@Delft [14:13 @ 26 July 2012]
Het is gefixed, bedankt voor het melden
eric.ettes@ihomer.nl [10:28 @ 26 July 2012]
Ik zie mensen online die niet online zijn?
Kevin@Delft [15:09 @ 25 July 2012]
ACK
Ben [14:58 @ 25 July 2012]
New Server!

Subject: New server

Fig. 3: Screenshot of Iris

having (for more about technological support for overhearing conversations see [62], [63], [64], [60]). Additionally users can also configure how they like to be contacted on each of these three mediums which is also shown in the user list. Firstly, it is possible to block a medium altogether which will deny all incoming conversation requests on that medium. Secondly, it is possible to allow all incoming conversation requests on a certain medium. This will result in the conversation automatically starting when another users starts a conversation on that medium. For instance when Bob has this setting enabled for audio/video conversations and Alice starts such a conversation with him Alice will automatically hear and see Bob without going through a calling mechanism which is common in existing tools for communication. Bob will also immediately hear and see Alice. Finally, it is also possible to allow conversations on a medium but only after explicit acceptance. If Bob has this setting enabled and Alice contacts him he will be able to hear and see Alice but she won't be able to see and hear him until he accepts the conversation request. It is an interesting anecdote how this system came to be. Initially we implemented a system in which on all media all incoming conversation requests were automatically accepted. We chose to do this because in a traditional co-located office setting it is also normal for people to automatically see and hear the people they are attempting to start a conversation with, even if this attempt fails. However, after we deployed this system, two users reported they were uncomfortable using this system and would like the possibility to block incoming conversations. The main issue they reported was that one of their colleagues could look at them in their living room without needing permission. It is noteworthy however, that since we deployed the new system the majority of the users are using the "allow all conversations"-setting for each of the media during normal working hours.

Finally, the system supports the notion of groups. Within an organization different groups exist, for example a group of people working on a certain project. We use this notion to filter the information a user sees to prevent information-overload. In the current version of the system this is still quite limited as it only influences the users and ongoing conversations you see and gives access to a group-specific message board. When you indicate you are currently active on a specific group (you can only be active on one group at a time) you only see the other members that are currently active on that group as well and get access to a group specific message board. Groups are hierarchal, so by being active on the company group a user has access to the information of the entire company as well.

V. Summary and Future Work

In this paper we have argued that *while most of the work in the CDE domain till date has focused on collaborative coding, the next step should be collaborative software engineering.* This is the main take-away from this paper.

We have argued that collaboration should be at the core of support environments for software engineering by discussing how Software Engineering is a human activity. We have shown that two popular current movements in Software Engineering, Global Software Engineering and Agile Software Development, confirm the importance of collaboration in Software Engineering as well. Subsequently, we discussed existing support for software engineers, discussed on what areas the solutions focus and identified limitations of these solutions. Finally, we have described our own approach in targeting these limitations in the Iris project and discussed the core ideas behind our approach.

It is clear there is still a lot of work to be done for us to reach the goals we have with our approach. At the moment the most promising extensions to the system we have identified are the following. Firstly, we are planning on incorporating the concept of *"virtual office walls"* in the system. In a normal office, office walls are used to control the information which is available to a developer. A developer has different awareness needs based on his activity and is able to control this, for instance by moving to a different office. We feel this notion is valuable in a support system for collaborative Software Engineering as well. By allowing users to control their context and, in turn, give them access to contextualized information and support, we can remove friction and make their work more efficient. The environment could for instance automatically open the right IDE when a developer chooses to start working on a specific code task or automatically open a document if the users starts working on a task involving documentation. Next to contextualizing the interface of the user based on his own activity, providing access to the context of others is valuable as well. For instance seeing who, at this very moment, is working on a task which is related to yours, will help in identifying additional opportunities for cooperation and help in communicating with your colleagues in an unobtrusive fashion.

The second direction for further work we have in mind, concerns optimizing the users' discovery of changes in their work environment. If a developer is working in a traditional co-located setting, he is not continuously scanning the entire environment to stay up-to-date; doing so would be cognitively exhausting. He will however likely notice when someone enters the room: a change in the state of the environment. We would like to explore the value of using a *"delta mechanism"* in communicating environment related information to the user, to keep the user up-to-date in an unobtrusive fashion.

Overall, we foresee a future in which support environments for software engineers support collaboration as well as they support the individual programming task today, resulting in the *"frictionless surface"* envisioned by Booch and Brown [1] which enables software engineers to unobtrusively carry out their creative work.

References

[1] Booch, G. and Brown, A.W, "Collaborative Development Environments," in *Advances in Computers*. Elsevier, 2003, vol. 59, pp. 1 – 27.

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

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

[4] 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.

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

[6] I. B. M. Corporation, "Rational team concert," 2008. [Online]. Available: http://www-01.ibm.com/software/awdtools/rtc/

[7] B. Sengupta, S. Chandra, and V. Sinha, "A research agenda for distributed software development," in *Proceedings of the 28th international conference on Software engineering*, ser. ICSE '06. ACM, 2006.

[8] A. Sarma, "A Survey of Collaborative Tools in Software Development," University of California, Irvine, Tech. Rep., 2005.

[9] G. Weinberg, *The Psychology of Computer Programming*. Dorset House Publishing, 1989.

[10] T. DeMarco and T. Lister, *Peopleware: productive projects and teams*. Dorset House Publishing, 1987.

[11] N. Ahmadi, M. Jazayeri, F. Lelli, and S. Nesic, "A survey of social software engineering," in *Automated Software Engineering - Workshops, 2008. ASE Workshops 2008. 23rd IEEE/ACM International Conference on*, 2008, pp. 1 –12.

[12] D. Perry, N. Staudenmayer, and L. Votta, "People, organizations, and process improvement," *Software, IEEE*, vol. 11, no. 4, pp. 36 –45, Jul. 1994.

[13] C. Jones, *Programming productivity*. McGraw-Hill, Inc., 1986.

[14] J. Strubing, "Designing the working process: What programmers do besides programming," in *User-Centered Requirements for Software Engineering Environments*. Springer, 1994.

[15] E. Conchúir, H. Holmström Olsson, P. Ågerfalk, and B. Fitzgerald, "Exploring the Assumed Benefits of Global Software Development," in *Proceedings of the IEEE 2006 International Conference on Global Software Engineering*. IEEE Computer Society Press, 2006, pp. 159–168.

[16] D. Damian and D. Moitra, "Guest Editors' Introduction: Global Software Development: How Far Have We Come?" *IEEE Software*, vol. 23, no. 5, pp. 17–19, 2006.

[17] R. Sangwan, M. Bass, N. Mullick, D. Paulish, and J. Kazmeier, *Global Software Development Handbook*. Auerbach Publications, 2007.

[18] E. Carmel and R. Agarwal, "Tactical approaches for alleviating distance in global software development," *IEEE Software*, vol. 18, no. 2, pp. 22–29, 2001.

[19] J. Herbsleb and R. Grinter, "Architectures, coordination, and distance: Conway's law and beyond," *IEEE Software*, vol. 16, no. 5, pp. 63–70, 1999.

[20] P. Ågerfalk, B. Fitzgerald, H. Holmström Olsson, B. Lings, B. Lundell, and E. Conchúir, "A Framework for Considering Opportunities and Threats in Distributed Software Development," in *Austrian Computer Society*, August 2005, pp. 47–61.

[21] R. Battin, R. Crocker, J. Kreidler, and K. Subramanian, "Leveraging resources in global software development," *IEEE Software*, vol. 18, no. 2, pp. 70–77, 2001.

[22] L. Kiel, "Experiences in distributed development: a case study," in *Proceedings of the 2003 International Workshop on Global Software Development*, 2003, pp. 44–47.

[23] H. Holmström Olsson, E. Conchúir, P. Ågerfalk, and B. Fitzgerald, "Global Software Development Challenges: A Case Study on Temporal, Geographical and Socio-Cultural Distance," in *Proceedings of the IEEE 2006 International Conference on Global Software Engineering*. IEEE Computer Society Press, 2006, pp. 3–11.

[24] B. Fitzgerald, P. Ågerfalk, H. Holmström Olsson, and E. Conchúir, "Benefits of Global Software Development: The Known and Unknown," in *Proceedings of the 2008 International Conference on Software Process*. Springer, 2008, pp. 1–9.

[25] J. Herbsleb, D. Paulish, and M. Bass, "Global software development at siemens: experience from nine projects," in *Proceedings of the IEEE 2005 International Conference on Software Engineering*. ACM Press, 2005, pp. 524–533.

[26] J. Herbsleb and A. Mockus, "An empirical study of speed and communication in globally distributed software development," *IEEE Transactions on Software Engineering*, vol. 29, no. 6, pp. 481–494, 2003.

[27] M. Fowler, "The new methodology," 2000. [Online]. Available: http://martinfowler.com/articles/newMethodology.html

[28] J. Highsmith, *Agile Software Development Ecosystems*. Addison Wesley, 2002.

[29] P. Abrahamsson, O. Salo, J. Ronkainen, and J. Warsta, *Agile software development methods. Review and analysis.* VTT Publications, 2002.

[30] J. Kalermo and J. Rissanen, "Agile software development in theory and practice," Master thesis, University of jyväskylä, 2002.

[31] D. Cohen, M. Lindvall, and P. Costa, "An introduction to agile methods," *Advances in Computers*, vol. 62, pp. 2–67, 2004.

[32] L. Williams, "A survey of agile development methodologies," 2004. [Online]. Available: http://agile.csc.ncsu.edu/SEMaterials/AgileMethods.pdf

[33] M. Awad, "A comparison between agile and traditional software development methodologies," Honours program thesis, University of Western Australia, 2005.

[34] T. Hataria, "The confounding world of process methodologies," in *Proc. CCEC 2006 Symposium*, 2006.

[35] K. Schwaber, "Scrum development process," in *OOPSLA'95 Workshop on Business Object Design and Implementation*, 1995.

[36] K. Schwaber and J. Sutherland, "Scrum guide," *Scrum Alliance*, 2011.

[37] K. Beck, "Embracing change with extreme programming," *Computer*, vol. 32, no. 10, pp. 70–77, 1999.

[38] B. Beck, *Extreme Programming Explained: Embrace Change*. Addison-Wesley, 1999.

[39] R. Glass, "Extreme programming: The good, the bad, and the bottom line," *IEEE Softw.*, vol. 18, no. 6, p. 112, 2001.

[40] K. Beck, *Extreme Programming Explained: Embrace Change, Second edition.* Addison-Wesley, 2004.

[41] A. Cockburn, *Writing effective use cases, The crystal collection for software professionals*. Addison Wesley, 2000.

[42] ——, *Agile software development*. Addison-Wesley, 2002.

[43] J. Highsmith, "Messy, exiting, and anxiety-ridden: Adaptive software development," *American Programmer*, vol. 10, no. 1, 1997. [Online]. Available: http://www.jimhighsmith.com/articles/messy.htm

[44] ——, *Adaptive Software Development: A Collaborative Approach to Managing Complex Systems*. Addison Wesley, 2000.

[45] P. Coad, E. Lefebvre, and J. De Luca, *Java Modeling in Color with UML*. Prentice Hall, 1999. [Online]. Available: http://csis.pace.edu/ marchese/cs615sp/L2New/jmcuch06.pdf

[46] S. Palmer and M. Felsing, *A Practical Guide to Feature-Driven Development*. Pearson Education, 2001. [Online]. Available: http://goo.gl/FwXco

[47] D. Anderson, *Kanban : Successful Evolutionary Change for Your Technology Business*. Blue Hole, 2010.

[48] K. Beck, M. Beedle, A. van Bennekum, A. Cockburn, W. Cunningham, M. Fowler, J. Grenning, J. Highsmith, A. Hunt, R. Jeffries, J. Kern, B. Marick, R. C. Martin, S. Mellor, K. Schwaber, J. Sutherland, and D. Thomas, "The agile manifesto," *The agile alliance*, 2001. [Online]. Available: http://www.agileAlliance.org

[49] O. Salo and P. Abrahamsson, "Agile methods in European embedded software development organisations: a survey on the actual use and usefulness of Extreme Programming and Scrum," *Software, IET*, vol. 2, no. 1, pp. 58–64, 2008.

[50] Z. Hussain, W. Slany, and A. Holzinger, "Current state of agile user-centered design: A survey," in *HCI and Usability for e-Inclusion*, ser. Lecture Notes in Computer Science. Springer Berlin / Heidelberg, 2009, vol. 5889, pp. 416–427.

[51] H. Fuks, A. B. Raposo, M. A. Gerosa, and C. J. P. Lucena, "Applying the 3c model to groupware development," *Int. J. Cooperative Inf. Syst.*, 2008.

[52] F. Lanubile, C. Ebert, R. Prikladnicki, and A. Vizcaino, "Collaboration tools for global software engineering," *Software, IEEE*, vol. 27, no. 2, pp. 52 –55, 2010.

[53] F. Lanubile, "Software engineering." Springer-Verlag, 2009, ch. Collaboration in Distributed Software Development, pp. 174–193.

[54] 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.

[55] 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.

[56] I. Steinmacher, A. Chaves, and M. Gerosa, "Awareness support in global software development: A systematic review based on the 3c collaboration model," in *Collaboration and Technology*, ser. Lecture Notes in Computer Science. Springer Berlin / Heidelberg, 2010, vol. 6257, pp. 185–201.

[57] M. Jiménez, M. Piattini, and A. Vizcaíno, "Challenges and improvements in distributed software development: a systematic review," *Adv. Soft. Eng.*, vol. 2009, pp. 3:1–3:16, Jan. 2009.

[58] J. Sillito, G. Murphy, and K. De Volder, "Asking and answering questions during a programming change task," *Software Engineering, IEEE Transactions on*, vol. 34, no. 4, pp. 434–451, 2008.

[59] I. Omoronyia, J. Ferguson, M. Roper, and M. Wood, "A review of awareness in distributed collaborative software engineering," *Software: Practice and Experience*, vol. 40, no. 12, pp. 1107–1133, 2010.

[60] K. Dullemond and B. van Gameren, "An industrial evaluation of technological support for overhearing conversations in global software engineering," in *Proceedings of the 2012 International Conference on Global Software Engineering*. IEEE Computer Society Press, 2012, pp. 65–74.

[61] ——, "Technological support for distributed agile development," Master thesis, Delft University of Technology, 2009.

[62] K. Dullemond, B. van Gameren, and R. van Solingen, "Virtual Open Conversation Spaces: Towards Improved Awareness in a GSE Setting," in *Proceedings of the 2010 International Conference on Global Software Engineering*. IEEE Computer Society Press, 2010, pp. 247–256.

[63] ——, "Overhearing Conversations in Global Software Engineering - Requirements and an Implementation," in *Proceedings of the 2011 International Conference on Collaborative Computing: Networking, Applications and Worksharing*. IEEE, 2011.

[64] ——, "An Exploratory Study on Open Conversation Spaces in Global Software Engineering," in *Proceedings of the 2011 International Conference on Collaborative Computing: Networking, Applications and Worksharing*. IEEE, 2011.