Activity-based Computing Support for Agile and Global Software Development

Jakob E. Bardram
IT University of Copenhagen
Rued Langgaards Vej 7, DK-2300 Copenhagen S.
bardram@itu.dk

ABSTRACT
As part of globalization, offshoring and outsourcing, many software organizations are adopting global software development (GSD). Studies, however, show that among the top ten risk factors to GSD, four of them is related to change management and developer-user cooperation. In order to embrace change and to foster close cooperation between developers and the users, state-of-the art within software engineering is to recommend an agile approach. Agile software development, however, rely heavily on co-located cooperation and mutual awareness based on physical artifacts and e.g. pair programming. This is especially difficult to achieve in a globally distributed setting. In this position paper, I suggest to approach this conflict using the principles from activity-based computing, including creating support for activity centered computing, activity suspend/resume and roaming, activity sharing, and activity integration of the physical and digital artifacts. As a hypothesis, we do not offer any evidence that this should be a successful approach, but based on our experience of developing activity-based computing technologies for e.g. hospital work – similarly is distributed in time and space – we hope this research path will reveal new insight in supporting GSD in particular, and global processes in general.

Author Keywords
Activity-Based Computing, Global Software Development, Agile, eXtreme Programming, Globalization

ACM Classification Keywords
H.5.2 Information Interfaces and Presentation: User Interfaces

INTRODUCTION

As part of the current trend towards globalization, offshoring and outsourcing, many software organizations are adopting global software development (GSD) as a new mode of software production [3, 1]. A core characteristic of GSD is that software is developed by a geographically dispersed team of software developers, who may belong to the same or different organizations. For example, it is becoming increasingly common for US based and European companies to offshore the development of software to low-wage countries, such as India. Another core characteristic of GSD is that users and the development team is equally distributed. For example, the users being located in the US or Europa, and the developers in India.

According to a recent survey, among the top 10 risk factors to offshored and outsourced software development project, four out of ten is related to requirement and change management, user involvement, and communication [13]:

2 Original set of requirements is miscommunicated

4 Inadequate user involvement

6 Failure to manage end-user expectations

7 Poor change control

For researchers within software engineering, you feel a sense of *deja-vu* when you see this list of challenges. These were the exact same kinds of challenges which are associated with the waterfall model of software development [15].

In order to mitigate these challenges and to create a better environment for on-going requirement specification, change control, and user-developer collaboration and calibration, state-of-the art within software engineering has been to advocate and adopt more light-weight, agile software development processes [15]. Agile methods, such as eXtreme Programming (XP) [8] and Scrum [16], operate on the principle of just enough method. They emphasize the importance of prioritize working code over comprehensive documentation, responding to change, and they advocate close co-located collaboration between people, including developers, stakeholders, and users. To a great extend, agile methods have evolved in response to increasingly changing business environments and volatile (user) requirements.

Since agile software engineering practices advocate co-located and close collaboration between developers and users, there seems to be an inherent conflict between agile software engineering and GSD. The many challenges of GSD are, perhaps obviously related to the effects of increased distance between people. Distance is, however, not only to do with spatial separation and geographical distance, but also with

Submitted to the CSCW 2008 Workshop on Supporting Distributed Team Work.

temporal distance (i.e. the dislocation in time experienced by two actors wishing to interact) and socio-cultural distance (i.e. a distance between actors understanding of other actors values and normative practices) [1].

A core aim of CSCW is to design and research technologies that enables distributed teams to collaborate. In this context, is appropriate to ask how such collaborative technologies can help distributed teams to collaborate in a global software development process while maintaining an agile approach. In other words, how collaborative technologies can be designed to foster *agile global software development*.

In this position paper, I will outline our current work in supporting agile global software development. I will do this by first reviewing existing knowledge on agile software development and XP, paying special attention to studies showing what makes agile processes work in practice. The I will discuss our ideas of using the activity-based computing approach to GSD. By doing so, I hope on the one hand to contribute to the workshop an outline of a very important problem which we – as CSCW researchers – are in a position to do something about, and on the other hand outline some ideas for how to addresses the challenges. I hope this may be of inspiration to others, and a source of discussion at the workshop.

AGILE SOFTWARE DEVELOPMENT AND XP

The basic principles behind agile methods have been summarized by the leading agile method thinkers in what has become known as the agile manifesto[1]. The agile manifesto is presented as a set of values and associated principles, where especially two of them seem to be at odd with DSD:

- Business people and developers must work together daily throughout the project.
- The most efficient and effective method of conveying information to and within a development team is face-to-face conversation.

The most widely used agile development practice is eXtreme Programming (XP). In XP 'User stories' are the mechanism used to communicate user requirements to the development team. They represent "units of customer-visible functionality", and have become a central focus of many agile teams. User stories are captured on 'Story Cards' (an index card typically no bigger than 5 by 7 in on which the users requirements are written) and an area of physical space where the story cards are organised and displayed which is called 'the Wall.

The efficacy of story cards is supported by the teams environment. In explaining the Informative workspace primary practice, Beck [8] states "Make your workspace about your work" and goes on to say that "An interested observer should be able to walk into the team space and get a general idea of how the project is going in 15 sec". He mentions the fact that many teams do this by putting story cards on the wall.

[1]http://agilemanifesto.org/

An 'Information Radiator' "displays information in a place where passersby can see it". It should have two characteristics: information changes over time; and it takes very little energy to view the display. Thus, a wall where story cards are displayed in a public place conforms to the notion of an information radiator

XP has also been tried in a GSD context and a 'Distributed eXtreme Programming (DXP) version has been proposed [14]. DXP suggests that eight of the XP practices (small releases, metaphor, simple design, testing, refactoring, collective ownership, 40-hour week and coding standards) are independent of team locality and can thus be applied also in GSD. The remaining four practices (the planning game, pair programming, continuous integration, and on-site customer), on the other hand, depend on co-located team members and users, and thus prove challenges to applying XP and agile methods in GSD.

Studies of GSD projects, however, suggest that communication and coordination are substantially disrupted across distant sites [12]. And, based on interview with distributed programmers, Gutwin et. al [11] concludes that explicit communication methods, like email and chat, were only effective when programmers made a significant effort to stay committed to those tools.

It is also interesting to note that detailed studies of teams successfully applying XP show, that co-ordination and collaboration activities are highly inter-related and co-located [2]. The kind of co-ordination that is undertaken by a team results in a situation where collaboration is made easy because team members are very aware of others work, overall project progress, and the state of the code base. Co-ordination and collaboration are supported by two key artefacts: the Story Card and the Wall. Hence, two physical objects work inside a special team room in a sophisticated and complementary manner and their physical nature is significant in underpinning the highly collaborative and self-organising style of agile teams.

ACTIVITY-BASED COMPUTING FOR GSD

Summing up on the issues discussed above, there seems to be a range of issues and/or challenges associated with the creation of an agile environment in global software development. These seems to be:

- Agile and XP teams rely on co-located work for ad-hoc communication, coordination and co-located collaborative work. For example, to easy settle open issues or questions and to allow for pair programming. Precisely communication and coordination is substantial disrupted across sites [12].
- Teams rely heavily on shared artifacts on public display for mutual awareness. For example, the Story Cards are displayed on the shared Wall, along with other important design and project artifacts.
- Teams adapt their physical environment to fit the work, c.f. the XP statement on "Make your workspace about

your work". Hence, local adaptation and change is essential while the project evolve.

- Close contact with clients, customers, and users is essential. This issues entails two things. First, the project should work with regular releases of working code which is deployed to the users for use or assortment. Second, clients and users should work together with the developers by either visiting the team in the team room or by having the developers come to the client site.
- The integration of physical artifacts and the physical space with the digital environment will be essential. Even though we might image that e.g. Story Cards might become digital, there is still a lot of material which offers great affordances on paper of cardboard, which cannot be replace with digital representations. For example, sketches and mock-ups made together with users.

Based on our prior research, we have a couple of hypothesis for creating support for distributed, global software development teams. This is ongoing research in its initial phase, and we hence do not have any evidence that these hypothesis hold in reality. The aim in this context of the CSCW workshop is to outline these hypothesis for further discussion. Our hypothesis are based on our work with activity-based computing (ABC), which to a large degree has been focused on creating ubiquitous computing support for clinicans working in large hospitals. Focus has especially been on support for mobile, collaborative, and time-critial work that involve a lot of physical artifacts which cannot be digitalized (e.g. specimens, drugs, surgical instruments, and the patient him- or herself). Clearly there is big difference between the working conditions inside a hospital and software developers in a globally distributed team. Issues like mobility, for example, seem to play a minor role in GSD. But there is also a set of issues which seems to be similar. For example, support for intense collaboration, social awareness on the progress of work, co-located work on shared artifacts, and close integration between the digital and physical artifacts.

For this reason we will investigate activity-based computing support for GSD. Our overall hypothesis is that activity-based computing support will help GSD. In order to investigate this hypothesis, let us consider the six ABC principles and relate them to GSD.

P1: Activity-centered aggregation of resources

The principle of 'Activity-centered resource aggregation' in ABC entails two thing: First, it suggests that the computer should model and maintain an understanding of the real world human activities taking place. For example, instead of just managing files and documents, the computers models the human activity, which include information on the purpose of the activity, who is participating, the history, and other relevant information. Second, the principles suggest that resources and services that are used in performing the activity is bundled 'into' or linked to the activity. This include both digital resources and services like documents and applications, as well as physical resources like real-world artifacts, tools, and documents.

In a GSD the principle of activity-centered aggregation would help us model the activities involved in the distributed work, and all participants would share the motives, goals, and resources of the different activities. In a GSD, a set of activities will continuously be created and maintain which would cover work in the agile process, including the inception, elaboration, construction, and transitions phases. The activity would specify the overall objective of each activity, a description of it, and list all the participants involved in it. It would also bundle and link to all the artifacts which are part of the activity. Resources include typical software development documents, like the Unified Process artifact of vision statement, use case model, UML diagram, glossary, requirement model, etc. But it would also contain links to more ad-hoc material like pictures and video from the customer site, design sketches, user interviews, persona descriptions, etc.

The core benefit to GSD of this activity-centered principle is primarily two thing. First, all participants of the different activities share the same description of the overall activity and is hence able to align their work to a shared representation of the work. Second, all participants have an overview of the resources and artifacts associated with the activity, is provided with an awareness of the status of the activity and the resources (e.g. the state of the requirement specification), and have easy access to all relevant resources and artifacts for this activity.

P2: Activity Suspend/Resume

Activity suspend/resume was designed to help multi-tasking in work, i.e. the ability to work on several activities at the same time, and to be able to alternate between which activity is in focus. Activity suspend/resume is particular important in a hospital environment where clinicians juggle many concurrent patient cases simultaneously. In a GSD environment, this seems less urgent – after all, most developers often work concentrated on a more focused development activity for longer time period. Nevertheless, when looking at agile methods, a core ideas is to prevent using the waterfall model of development, but encourage developers to move back and forth in the design and development process in order to constantly 'embrace change' and to continuously update documents and other artifacts with these changes. Hence, it is essential that developers can easily get access to the different activities and resources which belong to other phases of the development process. In this case, easy activity suspend and resume would help developers to easily go back and forth in the different activities. Thus, the argument is that activity suspend and resume will help an agile approach to software development. But this principle has little to offer the distributed nature of GSD. This is the target for the next ABC principle; activity roaming.

P3: Activity Roaming

Activity roaming enables the user to move activities and their related resources between devices, including mobile tablet PCs, desktop computers, and large interactive horizontal or

vertical displays. Activity roaming is essential in supporting mobility and enable users to use appropriate devices in the environment. In a GSD context, activity roaming is especially useful for two things. First, activity roaming allow the different participants in the activity to retrieve and resume the activity and its associated resource on their preferred work station. In a local environment, activity roaming also support users to move an activity to a shared display for shared problems solving or discussion. Second, activity roaming enable distributed team members to access and resume the activity. For example, activities and resources created and used in one location like the team room in Europa can be moved and resumed in the team room in Asia.

P4: Activity Sharing

An activity is shared among the GSD team. It has a list of participants who can access and manipulate the activity. Consequently, all participants of an activity can resume it and continue the work of another user. In ABC we distinguish between three modes of activity sharing:

- In *Asynchronous Activity Sharing*, users take turn in working on the activity. Hence, only one team member has resumed the activity and works on it.

- In *Synchronous Activity Sharing*, two or more users resume the same activity at the same time on different devices, potentially distributed in space. In this case, the team members will engage in an on-line, real-time activity sharing session which mean that the state of the activity and its associated resources are synchronized in real time.

- In *Temporal Activity Sharing*, users can share an activity across distribution in time. One (or more) participant(s) is able to capture their work in an activity and store if for other members of the team to later access it.

Asynchronous and temporal activity sharing can help distributed team members to collaborate across time distribution. For example, team members in one time zone can work on an activity, update and create artifacts and resources in the activity, document their work as part of the activity log, and suspend the activity at the end of their work shift. Other members in a different time zone can resumed this activity and continue the work. In temporal activity sharing the team members can capture their work, which later can be accessed by others for general review. For example, team members in one time zone may host a workshop with end-users. This workshop and the creation of artifacts may be captured using e.g. video, audio, and the creation of different physical and digital sketches and artifact. This captured activity record may later be replayed and reviewed by other participants of the activity.

Synchronous activity sharing can help bridge physical or geographical distances. For example, team members located in different locations may be able to engage in pair programming, where they have real-time sharing of an IDE like Eclipse. With a video link, team members can engage in same-time editing of source code.

P5: Activity Awareness

Maintaining an awareness of the progress of work within a software development team is at the core of the recommendations from the agile approach to software development. The use of Story Cards and the Wall is a prime example of this. The principle of activity awareness is targeted toward helping the participant of an activity to maintain an awareness of the overall progress and status of the activity. This includes status information on the different artifacts and resources included in the activity, as well as status information on the participants of the activity (e.g. information on name, current role, location, status, online status, etc.). For example, in a GSD setup awareness information would include content and status changes of the resources such as source code, changes to documents and artifacts, and changes and status information on participants.

Existing work on creating shared awareness within a development team has focused on visualizing and tracking changes to the shared code base (e.g. FASTDash [9] and Augur [10]). The principle of activity awareness suggest to extend support for awareness to include all parts of the activity, including all artifacts, resources, documents, participants, etc.

P6: Activity Integration

Based on studies of successful XP teams and how they use e.g. Story Cards and the Wall, Sharp & Robinson [2] argue that

> "A key danger of translating these mechanisms into an electronic form is that activity may become hidden, and less easily accessed by all." (p. 516)

Activity integration denotes the principles that the computational activity co-exists and integrated closely with its real-life counterpart, including the activity's parts such as resources, tools, participants, state, etc. Philosophically speaking, the computational activity and the real-world activity exist in a dialectical relationship. More specifically, this means that ABC tries to maintain a linkage between the digital and physical world. For example, in the hospital environment, the current ABC platform supports linking physical artifacts like drugs and surgical instruments to relevant digital activities [5]. Similarly, we envision to be able to link physical artifacts like Story Cards and design sketches to relevant activities in the digital space. This would support developers to continue to use physical artifacts as part of the development process.

WRAPPING UP

In this position paper I have outlined some of the problems which are associated with distributed, global software development (GSD). The state-of-the art within software development recommend more agile methods, like e'Xtreme Programming (XP). Since agile methods encourage co-located collaboration in a shared physical space, this approach is not compatible with GSD.

I have proposed that activity-based computing (ABC) may help mitigate the challenges of running a distributed soft-

ware development project. This includes supporting modeling the development project as a set of activities which evolve as the project progress, and which constantly bundle the different resources and artifacts involved in the activity. It also includes support for activity suspend/resume and roaming, which would help the participants to juggle several concurrent activities, and move these activities and their resources between distributed sites. The support for synchronous, asynchronous, and temporal activity sharing seems to be especially well-suited for bridging physical and temporal distances,

This paper has provided a very overall discussion of activity-based computing support for global software development, and has provided very little concrete examples or technologies of how this would actually take place. However, based on out previous work on building technologies for activity-based computing [5], we believe that the visions outlined above is achievable. For example, we already have concepts and technologies in place for activity-centered computing, activity suspend/resume, and activity roaming in place for the Windows XP operating system [7]; we have the underlying concepts and protocols in place for synchronous activity sharing [4], and we have recently developed and tested a activity-based computing support for large interactive displays [6]. This said, however, there is still much research to be done – part of it related to technologies for activity awareness, sharing, and the integration of the physical and digital artifact; and part of it related to investigating how activities, tools, resources, and artifacts related to software development may fit into the ABC approach.

Another important aspect to address in this research, is the physical and digital equipment of the team room – or more specifically the team rooms. We envision to create a virtual XP team room, where the distributed team is virtually co-located. This would include technologies for experiencing virtual co-locate, i.e. that the room is divided into two halves, each one in different physical location. The ABC technology will be designed to run in an environment like this.

REFERENCES

1. P. Ågerfalk. Towards Better Understanding of Agile Values in Global Software Development. In *Exploring Modeling Methods for Systems Analysis and Design – EMMSAD*. Via Nova Architectura, 2006.

2. H. S. andHugh Robinson. Collaboration and co-ordination in mature eXtreme programming teams. *Int. J. Human-Computer Studies*, 66:506–518, 2008.

3. W. Aspray, F. Mayadas, and M. Y. Vardi. Globalization and Offshoring of Software: A Report of the ACM Job Migration Task Force. Technical report, Association for Computing Machinery, 2006.

4. J. E. Bardram. Activity-Based Computing: Support for Mobility and Collaboration in Ubiquitous Computing. *Personal and Ubiquitous Computing*, 9(5):312–322, July 2005.

5. J. E. Bardram. Activity-based computing for medical work in hospitals. *ACM Transactions on Computer-Human Interaction*, 2008. Accepted for publication.

6. J. E. Bardram, J. Bunde-Pedersen, A. Doyrab, and S. Sørensen. MPAD - Activity-Based Support for Distributed Multiple Display Environments. Technical Report, IT University of Copenhagen. 2008.

7. J. E. Bardram, J. Bunde-Pedersen, and M. Soegaard. Support for activity-based computing in a personal computing operating system. In *CHI '06: Proceedings of the SIGCHI conference on Human Factors in computing systems*, pages 211–220, New York, NY, USA, 2006. ACM Press.

8. K. Beck. *Extreme Programming Explained*. Addison-Wesley, 2000.

9. J. T. Biehl, M. Czerwinski, G. Smith, and G. G. Robertson. Fastdash: a visual dashboard for fostering awareness in software teams. In *CHI '07: Proceedings of the SIGCHI conference on Human factors in computing systems*, pages 1313–1322, New York, NY, USA, 2007. ACM.

10. J. Froehlich and P. Dourish. Unifying artifacts and activities in a visual tool for distributed software development teams. In *ICSE '04: Proceedings of the 26th International Conference on Software Engineering*, pages 387–396, Washington, DC, USA, 2004. IEEE Computer Society.

11. C. Gutwin, R. Penner, and K. Schneider. Group awareness in distributed software development. In *CSCW '04: Proceedings of the 2004 ACM conference on Computer supported cooperative work*, pages 72–81, New York, NY, USA, 2004. ACM.

12. J. Herbsleb 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.

13. C. L. Iacovou and R. Nakatsu. A risk profile of offshore-outsourced development projects. *Commun. ACM*, 51(6):89–94, 2008.

14. M. Kirscher, P. Jain, A. Corsaro, and D. Levine. Distributed extreme programming. In *Proc. International Conference on eXtreme Programming and Flexible Processes in Software Engineering*, 2001.

15. C. Larman. *Applying UML and Patterns : An Introduction to Object-Oriented Analysis and Design and the Unified Process*. Prentice-Hall, Upper Saddle River, NJ, 2nd edition, 2002.

16. K. Schwaber and M. Beedle. *Agile Software Development with Scrum*. Prentice-Hall, Upper Saddle River, NJ, 2002.