Supporting Collaboration in Software Development Activities

Arnaud Lewandowski, Grégory Bourguin

Supporting Collaboration in Software Development Activities

Arnaud Lewandowski, Grégory Bourguin
Laboratoire d'Informatique du Littoral (LIL), France
lewandowski@lil.univ-littoral.fr; bourguin@lil.univ-littoral.fr

Abstract

Today, software development is intrinsically a collaborative activity and there is still a crucial need to provide adequate computer tools well supporting collaboration in such activity. Empirical studies have already identified some requirements to provide better collaboration-aware software development environments, and theories coming from human and social sciences still help researchers to better understand these activities. Founding our work on the Activity Theory, we present here some important issues that have been identified for creating better software development environments. Adding our experience, we particularly emphasize an aspect of human activity that has still not really been taken into account in creating these computer supports: the user's experience crystallization and sharing. Finally, we propose an implementation supporting the identified properties in an existing and widely used software development environment.

Keywords: Software Development Environments, Collaboration Support, Activity Theory, Experience Crystallization.

1. Introduction

Software Development (SD) is intrinsically a collaborative activity. Over the past years, many studies have shown that this dimension is still bad supported in Software Development Environments (SDEs). Recently, many propositions have been made to enhance the support for this particular dimension in existing or new tools. These new propositions take benefit from an approach largely developed in the frame of the CSCW (Computer Supported Cooperative Work) research field: using theories developed in Social and Human Sciences (SHSs) to better understand the cooperative human activities in which SD is realized. One of these theories that has a wide audience in CSCW is the Activity Theory (AT). AT has recently been used to propose better computer support for design activities in domains like architecture [28], education [9], and others. It also has provided very interesting results for studying SDEs [3]. However, we believe that this theory still has not delivered all its secrets as for helping us to better understand how to support design activities like SD.

Starting from this assumption, we will first try to summarize the main issues that have already been identified for creating collaboration-aware SDEs. After a brief presentation of the AT, we will particularly focus on the issues highlighted by using this particular theory. This study will lead us to propose a new focus that still has not really been taken into account in the design of SDEs. Finally, we will show our proposition trying to support the identified properties in a widely used SDE.

2. Supporting Collaboration in SDEs

It would be impossible to summarize here all the issues identified to better support the collaborative dimension in SDEs. However, we will underline the main results that are driving our own approach of the problem.

2.1. General issues

Adding collaboration support in SDEs means more than providing additional communication tools [8]. Even if efforts have been done to improve collaboration support, some collaborative aspects are still missing. For example, most Integrated Development Environments (IDEs), such as the widely used Eclipse, focus on code-producing activities, considering them as "individual" activities in the development process. The collaborative support is then generally limited to the use of a common repository – such as CVS – that supports documents sharing, but not the collaboration between developers [11]. In other cases, some collaboration support is provided, but is still disconnected from the development process: collaboration is supported by adding communication functionalities, without really connecting this dimension to the main activity. This is for example the case when a synchronous discussion tool is simply plugged into the environment. A real support for collaboration in SD supposes that the environment should be able to support this activity *as a whole*. Collaboration is a constituting part of the global activity, not an aside one. Thus, the tool we want to build can be called an *integrated collaboration environment*, as proposed by Sarma [27] that has defined a framework in which collaborative tools for software development have been classified according to their degree of supported col-

laboration. In this classification, *integrated collaboration environments* appear in the highest layer by supporting continuous coordination and cooperation through the whole development process. However, it clearly seems that such environments are currently very rare and research in this area is still in progress [5][29].

Tailorability is also a well-identified requirement for SDEs. Many empirical studies [10][15] and theoretical research [25] have demonstrated that human activity, and then SD, is reflective in the sense that the users needs towards their activity support emerge from and during this activity. In SD, this continuous evolution may for example affect the development process that has to change in order to take care of unexpected events [18]. Considering this issue, many workflow solutions have already been largely criticized for their rigidity [1]. The set of tools involved by the users may also evolve and we can notice that most of the widely used solutions are faced with this problem [20]. For example and even if they propose useful tools, Web portals like SourceForge integrate several components intended to support collaboration through the development process, but the tailorability of such environments is in most cases reduced since the available tools are defined *a priori*. The dynamic integration of new tools by end-users is generally not possible. One commonly accepted generic solution for such tailorability is to propose reflective properties in SDEs, thus allowing for example the dynamic redefinition of the enacted process model, or the dynamic integration of tool components. However, if tailorable solutions like Eclipse [13] exist for a particular dimension, none of them actually supports all the identified issues, and research on how to improve tailorability in collaborative SDEs is still an on-going work [16][18][30].

Finally, we should keep in mind that even if SD is collaborative, its goal is still to develop software, not just to collaborate. As demonstrated by the emergence of new organizational paradigms like *Extreme Programming* [23] with a process mainly organized around the coding activity, developers want to develop. Even if the support for collaboration is missing from existing SDEs, these SDEs have been used and developed for a long time, thus well supporting other dimensions of SD activities like coding. Through the years, developers have crystallized and shared their experience in these broadly used environments. It would not be appropriate to ask them to change their preferred code editor "just to benefit from collaborative features". Faced with this situation, we can try to improve the collaboration support in existing and widely used open environments. This approach may take benefit from a large audience that makes this environment continuously evolve. This is the choice made by Hupfer et *al.* [17] with Jazz and also by Sarma et *al.* with Palantír [26], for example.

2.2. The Activity Theory

We have presented some results coming from a general state-of-the-art in the SDEs field. These results, mainly coming from empirical studies, highlight what should be done to improve collaboration support in SDEs. Moreover, we strongly believe that studying these results about SD activity by using theories coming from the SHSs, like the Activity Theory (AT), can help to better understand these requirements. We will now briefly introduce the basic concepts of AT, but one can refer to [4] and [14] to get further information.

The AT takes the activity as the basic unit for analyzing human activities. The basic structure of an activity proposed by Engeström [14] presents the human activity as an interdependent system involving a *subject* that realizes the *object* of the activity, and the *community* of subjects who are concerned with this realization. Relations between the subject, the object and the community are mediated. In particular, the subject uses *tools* to realize the object of the activity. *Rules* determine what means belonging to the community, and a *division of labor* describes how the members of the community share the work up. Furthermore, activity is dynamic and continually *evolves* during its realization. For example, subjects may transform the mediating elements as new needs emerge in response to *contradictions* that rise between elements of the activity. Activity dynamics have been classified by Bardram [2] in three levels: the *coordination* level, where subjects concentrate on performing basic actions; the *cooperation* level, where subjects effectively act cooperatively towards their object; and the *co-construction* level, where subjects reconceptualize their activity. Finally, subjects themselves evolve during the activity by acquiring skills and developing some experience about its realization. Thus, when subjects transform the elements participating in their activity, their experience is *crystallized* in these elements. This experience, *written* in the transformed artifacts, becomes available for others that reuse them in other activities.

2.3. Previous studies using AT

Many studies have already been conducted by using AT in the field of software development. We will now present some results coming from these studies and that we find particularly close to our approach.

In [12], De Souza and Redmiles use the AT to study collaboration among developers in a particular software development activity. They focus on the many *contradictions* rising inside such an activity and underline how these tensions have an impact on the other elements constituting the activity, or even on the other connected activities. For example a change in the source code may make the documentation out-of-date. This is what they call an *inconsistency*. They also show how multiple in-

stances of an activity can increase the number of inconsistencies, e.g. when several developers simultaneously check-in the same part of code on the common repository. It is interesting to note that contradictions inside and even between activities hold a strong place in SD. Moreover, this *"focus on identifying tensions and conflict is useful […] for highlighting areas where software tools and practices might be improved"* [12]. These considerations help to better understand the above-mentioned issue about the need to support the SD activity as a whole: in order to manage the tensions existing between several activities, we should support the global activity they belong to.

Korpela and *al*. propose a framework to study information systems development as *"a real-life work activity in context"* [21]. They consider the development activity as part of a network of activities, taking care of others activities (such as the company's organizational management) and the way they are connected together. Activities are linked when, for example, the outcome of an activity is consumed by one or more other activities. This framework is basically intended to describe information systems development. However, the concept of activity network, originally proposed by Kuutti [22], remains very interesting and can be useful to manage the contradictions emerging between activities during the development process.

Barthelmess and Anderson [3] use the AT to undertake a full analysis and evaluation of Process-Centered Software Development Environments (PCSDEs). The study is conducted by analyzing how these environments support the three activity levels defined by Bardram, i.e. *coordination*, *cooperation*, and *co-construction*. They underline that PCSDEs aim at supporting collaborative activities by providing support for *division of labor* through enactment of process models. However, *"this emphasis on software process can result in 'blindness' with respect to other important aspects of work, in particular collaboration"* [3]. Actually, even if they provide a good support at the coordination level, PCSDEs suffer from their production-oriented philosophy and present a serious lack of adequate support for cooperation, and then also limit the support for co-construction: existing PCSDEs limit the co-construction support to the reconceptualization of process models. Unfortunately, co-construction may also imply the reconceptualization of the whole activity structure through cooperation between subjects: co-construction needs then a good support for cooperation.

These studies point out some important aspects of software development activities. Supporting these dimensions in SDEs is important and remains a non-trivial work. Moreover, we have been working for years in the CSCW research domain by using the AT [6][7]. We have identified some other properties that have not really been taken into account in developing such systems. We will then now introduce these results that we want to add in the issues for creating better SDEs.

2.4. Adding our experience

Some years ago, we have been working in a particular field of CSCW: the Computer Supported Cooperative Learning (CSCL) [6]. It is interesting to notice that this previous work led us to identify the same properties that we have just exposed considering SD activities, in particular, the need for better supporting co-construction through cooperation between subjects. This is not so surprising because learning and software development are both human activities. The approach we have developed has been synthesized under the *co-evolution* principle that we have defined in [7]. Co-evolution emphasizes the fact that human activity is reflective in the sense that any (cooperative) activity is closely linked to a (cooperative) meta-activity where the subjects co-construct their environment in response to contradictions emerging during the core activity. Then, the Activity Supports (AS) we create aim at supporting domain-specific activities, and also their closely related cooperative meta-activities.

Today, we apply the co-evolution principle in creating SDEs and the techniques we have used, even if a little different, are close to those we have just presented in other SDEs: for example, tailorability at the process level is realized by providing computational reflective properties and a particular process meta-model that allows the subjects to create/transform their own process models. In our case, the process meta-model is mainly inspired by the AT and is called a *Task*: i.e. an *Activity model* [8]. A task specifies the *Roles* of the subjects, thus defining how they can (or have to) use the *Resources* involved in the activity. Some of these Resources are *meta-tools* allowing the (re)definition of the task. One difference between our approach and the others is that we have developed a minimal kernel introducing a recursive approach in which the *meta-tools* are used in cooperative *(meta-) activities* that are defined in *(meta-) tasks*. Then, in our approach and according to the co-evolution principle, co-construction is realized during cooperative (meta-) activities themselves managed by particular and *tailorable* process models.

However, and as we would not have the space here to present all the differences between our realization and the others, we have decided to present in the rest of this paper a particular aspect of the co-evolution that has not been brought to the fore in other approaches. This work takes its roots in a fundamental idea developed in the AT while emphasizing the cultural and historical dimensions of human activity: the crystallization of the subjects experience inside their developed artifacts. We have briefly introduced how the subjects experience can crystallize at the end of Section 2.2. We will now present how this can happen in the SDEs we build.

This mechanism is illustrated in the Figure 1. The community of subjects realizes an activity in the real world. This activity is supported by the system. The *System Task* is the part of the real task that has been specified

inside the system to create an Activity Support (AS). The community acquires some experience while performing its activity. Making the System Task evolve can make this experience explicit through the system. For example, an evolution in the division of labor in the real activity may result in a new set of roles specified in the system task and that will affect the corresponding AS. This new evolved task corresponds to a new AS model that can also be instantiated for another community. The crystallized experience developed during an activity can then be transmitted through the tailorable system, thus supporting the experience crystallization and sharing through developed artifacts.

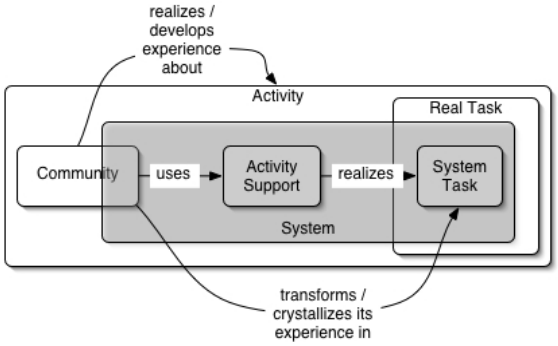

Figure 1. Crystallizing subjects' experience

One can notice that this property seems at least partially supported by any system proposing tailorability by such a model-driven approach. However, our work goes a little further because we try to really merge the results coming from the AT and the computational techniques used to support human activities. Tailorability in a cooperative and shared system poses the problem of stability. Of course, there is a stability problem from the computer system point of view: a reflective system is not easy to maintain, and access to reflection should be controlled, as stated in the Open Implementation proposed by Kiczales [19]. But we focus here on stability issues from the human point of view. Considering the AT, we believe that the subjects have to understand their system before to make it evolve. However, subjects will not be able to understand a continuously changing system. This is why evolution should only happen after stable phases and in a cooperative meta-activity where subjects may for example negotiate this evolution. Moreover, we cannot talk about experience crystallization if no experience has been developed. This is why, even if we believe that the use of shared model transformation is interesting for crystallizing and sharing experience, a model should also only evolve after discussion about an evolved prototype. With such a prototyping approach, subjects will be able to test new solutions before really modifying the system model. Following this idea, we now present how we have started to support this co-evolution principle in CooLDev.

3. CooLDev: Collaboration under Eclipse

In the previous parts of this paper, we have identified issues to improve collaborative support in SDEs. We have underlined that SDEs should consider the global activity as a whole, be tailorable in a cooperative fashion, and take benefit from experience developed during the activities they support. We have also underlined that even if most currently used SDEs do not well support the collaborative dimension of SD activities, an appropriate approach would be to enhance one of these SDEs. This is exactly what we have done by extending the Eclipse platform in the CooLDev (Cooperative Layer for software Development) project.

3.1. The Eclipse Platform

Eclipse is an open-source generic platform based on a powerful integration framework that supports the dynamic discovery, installation and activation of *plug-ins*. In other words, this kernel manages and controls a set of integrated tools working together to support specific tasks. Thus, our choice has mainly been driven by the fact that Eclipse has been conceived in terms of tailorability. The subject can adapt the environment according to his emergent needs by dynamically integrating tools made available on the network. In a few years, Eclipse has grown very quickly due to the great amount of developers using it, and making it evolve.

However, regarding the collaborative dimension of software development, Eclipse still presents some lacks. Indeed, as well as in other IDEs, collaboration support is limited to the use of a common repository such as *CVS* [11]. Developers using the platform have been faced with this limitation and, as a result, some collaborative plug-ins have been produced [17][26]. However, and even if needed, this kind of extensions providing collaborative functionalities like an IRC does not tend to consider the cooperation at a global level. Eclipse has not been designed in that orientation, and for example, it does not propose a role notion managing the status of a user in the global cooperative activity. As a result, each subject has to integrate the tools (plug-ins) he needs, and to configure them himself according to his role in the real supported activity.

3.2. Introducing global collaboration

In Section 2, we have mentioned that some researchers already try to support this kind of global collaboration in Eclipse. This is particularly the case in Jazz [17] where many existing plug-ins have been enhanced. Our approach is different because we do not modify existing plug-ins: we propose a context for their execution.

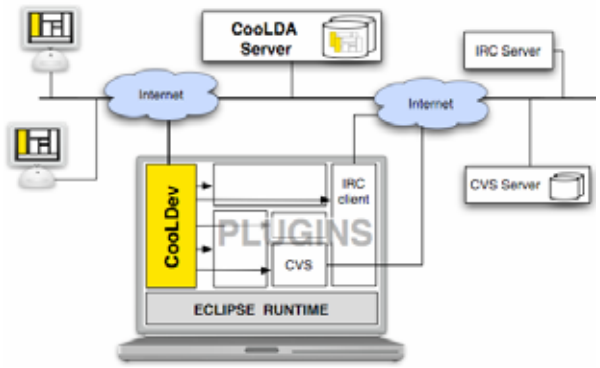

Figure 2. CooLDev's architecture

We consider that each plug-in supports a particular activity like coding, merging different sources, or even chatting. In this approach, supporting the global activity means supporting the links existing between these (sub-) activities, i.e. managing the *inter-activities* [8]. This is mainly inspired by the above-mentioned work (c.f. 2.3) of Kuutti [22] about relations between activities. To achieve this, we propose a plug-in called CooLDev whose role is to articulate the other plug-ins in the context of global cooperative activities. CooLDev's architecture is presented in Figure 2. After a basic identification phase, CooLDev retrieves on the CooLDA server the activities the subject is involved in, and configures his environment according to his role(s): subjects playing the same role retrieve an instance of the same CooLDev's perspective that will configure properly the user's environment. Moreover, CooLDev uses Java introspection mechanisms to dynamically retrieve public methods provided by the other plug-ins and to pilot them in response to activity state changes as defined in the task (activity model) that is running on the CooLDA server. These mechanisms are further described in [8] and we will now particularly focus on how experience can be crystallized and shared in CooLDev.

3.3. Experience crystallization

A first look at Eclipse shows that the concept of experience crystallization, even if not clearly identified, seems to be in tune with *Eclipse perspectives*. A perspective corresponds to a particular point of view on the working environment (and the activated plug-ins) during the realization of a task. It manages the plug-ins activation and arrangement at the user interface level. Eclipse lets the subject create and modify his own perspectives, thus saving his preferences for a task. From our viewpoint, the perspectives mechanism provides a powerful mean to crystallize some experience. However, this experience is not intended to be shared by users. Even if some people may work with the same perspective because it has been packaged with a specific plug-in, nothing is provided for sharing perspectives in the context of a particular global, evolving and cooperative activity. As a first step in trying to better manage some experience crystallization and sharing through CooLDev, we have developed some basic features over the perspectives mechanisms.

CooLDev associates roles in a given activity with particular perspectives. When a subject joins an activity, he retrieves a perspective instance that is defined according to its role. However, all the subjects playing the same role may not share exactly the same perspective since we let them adapt/modify it according to their role and emerging needs. These instances, originated from the same role model, can then evolve and be considered as prototypes reflecting the subject's experience he has developed while playing his role. We then have developed a plug-in allowing subjects to share their perspectives. This feature is presented in Figure 3. The view we developed shows the perspectives shared with others, and allows the users to *test* these shared perspectives[1]. Finally, CooLDev allows *generalizing* a perspective at the task level, i.e. in a role model, for example after some negotiations between the subjects. Following the co-evolution principle, this form of co-construction helps the subjects to develop a real experience that is written into the perspective prototype. This experience can be crystallized in the model that can benefit to the subjects playing this role, and can be reused later in similar activities. This demonstrates how a community of developers can make their environment co-evolve by sharing their experience. As CooLDev also supports transformations of the whole activity (process) model, we are currently extending this prototype-based approach to support experience crystallization in the other AS elements.

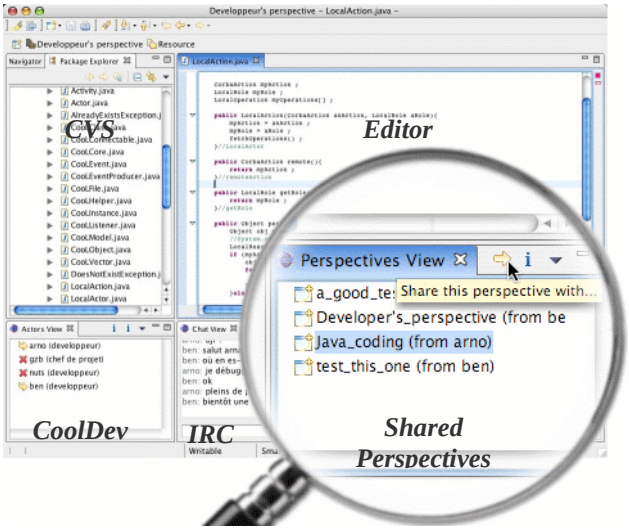

Figure 3. Zoom on the shared perspectives view

[1] Currently, only users playing the same role can share perspectives, mainly for reasons of rights on the plug-ins in a particular activity.

4. Conclusion

In this paper, we have shown that Software Development (SD) activities still need better computer supports. Indeed, SD is nowadays an intrinsically collaborative activity. The software development research field has identified some issues to improve collaboration supports in SDEs. We have presented those that we find most important. Furthermore, the Activity Theory highlights these issues by explaining the basic mechanisms ruling every human activity. In our words, these issues are synthesized in the co-evolution principle where computer tools should be tailorable while supporting their co-construction through cooperative meta-activities. We have presented how these identified issues can be taken into account and we have particularly focused on an aspect of human activity that has still not well been identified and supported in SDEs: the subjects experience crystallization and sharing. We have developed this concept in the CooLDev environment that tends to manage SD activities at a global level, while supporting experience crystallization and sharing in and through the system. CooLDev is implemented over the existing and widely used Eclipse platform. Currently, this proposition is still a prototype. Nevertheless, the basic presented features helped us to verify the feasibility of our approach and to illustrate how we can support some co-evolution through experience crystallization and sharing inside the platform.

We are pursuing our efforts in order to improve these mechanisms. We are particularly working on extending existing component models to provide a higher abstraction level that will ease introspection for the dynamic and fine integration of plug-ins in supported activities. Finally, a next step will consists in conducting evaluations of our approach by experimentations in real situations. We plan to test the platform in the context of a real commercial development team we are in contact with, which already uses an Eclipse compatible tool and which is faced with the issues presented in this paper.

Acknowledgements

The authors want to thank the organisms supporting this work, in particular the French Research ministry for the ACI Jeunes Chercheurs CooLDev, and the TAC (Advanced Technologies for Communication) program financed by the Région Nord/Pas-de-Calais and by the French State in the framework of the NIPO/MIAOU and the EUCUE projects.

References

[1] A. Agostini and G. DeMichelis, “A light workflow management system using simple process models”, *Computer Supported Coop. Work, 9*(3-4), 2000, pp. 335-363.

[2] J. Bardram, “Designing for the dynamics of cooperative work activities”, *Proceedings of CSCW98*, Seattle, Washington, United States, ACM Press, 1998, pp. 89-98.

[3] P. Barthelmess and K.M. Anderson, “A view of software development environments based on activity theory”, *Computer Supported Coop. Work 11*(1-2), 2002, pp.13-37.

[4] G. Bedny and D. Meister, *The Russian Theory of Activity: Current Applications to Design and Learning*, Lawrence Erlbaum Associates, Publishers, 1997.

[5] G. Booch and A. Brown, “Collaborative development environments”, *Advances in Computers, 59*, 2003.

[6] G. Bourguin and A. Derycke, “A reflective CSCL environment with foundations based on the Activity Theory”, *Proc. of the 5th Int. Conf. on Intelligent Tutoring Systems*, Montreal, Canada, 2000, pp. 272-281.

[7] G. Bourguin, A. Derycke and J.C. Tarby, “Beyond the interface: Co-evolution inside Interactive Systems – A proposal founded on Activity Theory”, *Proceedings of Human-Computer Interaction 2001*, in Blandford, Vanderdonc-kt, Gray (eds), *People and Computer vol. 15 – Interaction without Frontiers*, Springer Verlag, 2001, pp. 297-310.

[8] G. Bourguin and A. Lewandowski, “Inter-activities management for supporting cooperative software development”, *Proceedings of the Fourteenth International Conference on Information Systems Development (ISD'2005)*, Karlstad, Sweden, 15-17 August, 2005.

[9] R.K.E. Bellamy, “Designing educational technology: Computer-mediated change”, in B. Nardi (eds), *Context and Consciousness: Activity Theory and Human-Computer Interaction*, Cambridge: MIT Press, 1996

[10] D. Cubranic, G.C. Murphy, J. Singer and K.S. Booth, “Learning from project history: a case study for software development”, *Proceedings of CSCW04*, Chicago, Illinois, USA, 2004, pp. 82-91.

[11] CVS online manual, Section 1.2 “What is CVS not”. Available from URL: http://ximbiot.com/cvs/manual/

[12] C.R.B. de Souza and D.F. Redmiles, “Opportunities for extending Activity Theory for studying collaborative software development”, *Workshop on Applying Activity Theory to CSCW Research and Practice (in conjunction with ECSCW 2003)*, Helsinki, Finland, 2003.

[13] Eclipse Foundation, Eclipse.org Web Site. Available from URL: http://www.eclipse.org/

[14] Y. Engeström, *Learning by Expanding*, Orienta-konsultit, Helsinki, 1987.

[15] V. Folcher, “Appropriating artifacts as instruments: When design-for-use meets design-in-use”, *Interacting with Computers, 15*(5), October 2003, pp. 647-663.

[16] J. Grundy, R. Welland and H. Stoeckle, “Workshop on directions in software engineering environments”, *SIGSOFT Softw. Eng. Notes 29*(5), 2004, pp. 1-3.

[17] S. Hupfer, L.T. Cheng, S. Ross and J. Patterson, “Introducing collaboration into an application development environment”, *Proceedings of CSCW04*, Chicago, 2004, pp. 21-24.

[18] P.J. Kammer, G.A. Bolcer, R.N. Taylor, A.S. Hitomi and M. Bergman, “Techniques for supporting dynamic and adaptive workflows”, *Computer Supported Cooperative Work, 9*(3-4), 2000, pp. 269-292.

[19] G. Kiczales, “Beyond the black box: Open implementation”, *IEEE Software, 13*(1), 1996, pp. 8-11.

[20] T. Koch and W. Appelt, "Beyond Web technology – Lessons learnt from BCSCW", *Proceedings of the 7th Workshop on Enabling Technologies: Infrastructure for Collaborative Enterprises*, 1998, pp. 176-181.

[21] M. Korpela, A. Mursu and H.A.Soriyan, "Information systems development as an activity". *Computer Supported Cooperative Work, 11*(1-2), 2002, pp.111-128.

[22] K. Kuutti, "Notes on systems supporting 'Organisational Context' – An activity theory viewpoint", *COMIC European project*, D1.1, 1993, pp. 101-117.

[23] A. Mackenzie and S. Monk, "From cards to code: How extreme programming re-embodies programming as a collective practice", *Computer Supported Cooperative Work 13*(1), 2004, pp. 91-117.

[24] B. Nardi, *Context and Consciousness: Activity Theory and Human-Computer Interaction*. Cambridge: MIT Press, 1996.

[25] P. Rabardel, "From artefact to instrument", *Interacting with Computers, 15*(5), October 2003, pp. 641-645.

[26] R.M. Ripley, R.Y. Yasui, A. Sarma and A. van der Hoek, "Workspace awareness in application development", *Proceedings of the 2004 OOPSLA Workshop on Eclipse Technology Exchange*, Vancouver, Canada, 2004, pp. 17-21.

[27] A. Sarma, A. van der Hoek and L.T. Cheng, "A need-based collaboration classification framework", *CSCW 2004 Workshop on Eclipse as a Vehicle for Collaboration Research*, Chicago, USA, November 2004.

[28] K. Schmidt and I. Wagner, "Ordering systems: Coordinative practices and artifacts in architectural design and planning", *Computer Supported Cooperative Work, 13*(5-6), 2004, pp. 349-408.

[29] A. van der Hoek, D. Redmiles, P. Dourish, A. Sarma, R.S. Filho and C. de Souza, "Continuous coordination: A new paradigm for collaborative software engineering tools", *Workshop on Directions in Software Engineering Environments WoDISEE*, Scotland, 2004, pp. 29-36.

[30] M. Webster, "An end-user view of the collaborative software development market", *Market Research Report*, IDC #30608, Vol. 1, 2003. Available from URL: http://www.collab.net/