RC23484 (W0501-026) January 7, 2005
Computer Science

IBM Research Report

Reinventing Team Spaces for a Collaborative Development Environment

Susanne Hupfer, Li-Te Cheng, Steven Ross, John Patterson
IBM Research Division
One Rogers Street
Cambridge, MA 02142

Research Division
Almaden - Austin - Beijing - Haifa - India - T. J. Watson - Tokyo - Zurich

Reinventing Team Spaces for a Collaborative Development Environment

Susanne Hupfer, Li-Te Cheng, Steven Ross, John Patterson
IBM Research
Collaborative User Experience Group
1 Rogers St., Cambridge, MA
{Susanne_Hupfer, Li-Te_Cheng, Steven_Ross, John_Patterson}@us.ibm.com

INTRODUCTION

Software development is a collaborative endeavor. From open source projects to corporate products, programmers engage in conversations with colleagues around the world about the complex software they are designing, building, testing, and fixing. Besides face-to-face interaction, developers use an assortment of collaborative tools inside and outside their integrated development environments (IDEs). These include formal tools accessible from the IDE, such as source code management and bug tracking systems, and also ad hoc ones available outside the IDE, such as email, instant messaging, and threaded conversation tools [10,19]. Non-collocated software teams, especially, face collaboration challenges. Indeed, as software teams become increasingly distributed, there is an increasing need for tools to support both structured and unstructured communication and coordination of work [11].

Booch and Brown postulate that a rich collaborative development environment (CDE) arises from the collection of many apparently simple collaborative components that support coordination, collaboration, and community building – the "Three C's" of CDEs [2]. Moreover, they assert that IDEs equipped with team-centric features are superior to those merely enhanced with collaborative support. We have been building a prototype CDE known as Jazz, an IBM Research project that embeds collaborative capabilities into an application development environment (i.e. by extending the Eclipse Java IDE) to enable small teams of software developers to work together more effectively. [4,5,7]. Other companies are also interested in team-enhanced development environments [14,21].

Jazz is based on an "open office" approach to development: A small team works in close proximity at their workstations, with a shared space available for collaborating at whiteboards, sharing materials, or having meetings [4,5]. Communication is vital: Teammates shout out questions or information, or call colleagues over to consult. Team awareness is also key: Even while focused on their own work, developers have a peripheral sense of the work, activities, and discussions around them. Our goal with Jazz is to elevate the team to a first-class object in the development environment, and to facilitate *people awareness* (who's present and what they're doing), *resource awareness* (who's working on code that I depend on), communication, and coordination among the team members. Jazz provides the means to initiate chats, VOIP calls, or screen-sharing sessions with teammates. Another feature that we have begun to explore is a type of asynchronous team space: Conversation transcripts may be saved to the team's space, and other events and artifacts (e.g. code check-ins, check-outs, build results, documents) may be posted. We also envision members contributing to asynchronous team-wide conversations in the space.

SPACES FOR SOFTWARE DEVELOPMENT

The Good of Team Spaces

In the realm of collocated software development teams, there is little doubt that physical conversation spaces are useful – to have meetings, review design, APIs, or code, negotiate, resolve conflicts, disseminate information, share materials, or even converse informally [6,16]. Sawyer and Guinan have studied software development and reported the positive impact that team-level social processes have on product quality and team performance [18], and proponents of agile development affirm that "the most efficient and effective method of conveying information to and within a development team is face-to-face conversation" [1].

As teams become increasingly distributed and chances for face-to-face discussion diminish, they turn to virtual team spaces for some of their communication needs. Spaces may incorporate tools such as shared text editors, chat utilities, discussion forums, wikis, whiteboards, and document repositories [3,9,13,17,19,20]. Booch and Brown call for "a virtual space in which 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" [2].

The Bad of Team Spaces

Despite the benefits of team spaces as a communication and collaboration tool, they have disadvantages:

Not contextual. Programmers' main work environment is the IDE, not a web browser or email client. Team spaces are external to the main environment and lack links to the work context, artifacts, and team activities. Moreover,

programmers use version control systems to manage their code, so linking code artifacts to discussions is further complicated by versioning issues.

Not easy to monitor. Individuals may belong to multiple spaces and need to leave their work environment to check each one periodically. There is no automatic way to keep up-to-date, and spaces may stay "out of sight, out of mind."

Not necessarily relevant or interesting. Team spaces may collect many discussions and artifacts, not all of which are significant to each team member, and disinterest in the space or information overload may result.

Not easy to search. Content is captured in individual team spaces and not available for cross-project search. Also, defunct workspaces are not generally visited or searched, and members may disperse, so knowledge is lost.

Not easy to share. Information is isolated in team spaces, accessible only to members. To share information, outsiders may be let in, but that may be too extreme; access control may be desirable to protect sensitive information.

The Potential of Team Spaces

Recognizing both the great potential of team spaces to support distributed software teams, and also their drawbacks, we are focusing our research on "reinventing" them. Our goal is that these improved spaces provide:

- Information that's contextual, relevant, interesting, & timely for individual members
- Easy monitoring of what's new in multiple spaces
- Sense of the team's current activities and progress & sense of team cohesion
- Search capability across spaces (even retired ones)
- Means for multiple teams to share information, and for letting outsiders in on some information

We intend to improve team spaces by:

Making spaces contextual. Jazz illustrated the value of *contextual collaboration*: Users collaborate without leaving their core work environment [12]. One example consists of discussions anchored around code: Users can highlight code, right-click, and start a chat with teammates. From the chat UI, participants see the code snippet and can click on a hyperlink that opens the pertinent code in their IDE. Participants can opt to save a chat transcript, which will appear as an annotation to the code in the IDE. Other team members will see the annotation, and can revisit and even continue the conversation later. Similarly, one can imagine a requirements analyst initiating a chat from a requirement in a requirements document, or a developer starting a chat from an API in an API specification [6]; all participants are automatically informed of the context of the conversation. Contextual collaboration can enhance teamwork by establishing a persistent, shared context and facilitating capture and retrieval of collaborative artifacts.

We contend that the discussion taking place in a team space should also be contextual – grounded in the team's work artifacts. One should be able to select part of an artifact and post a comment or query to the space. Teammates automatically see what artifact the post refers to, and can click to examine it in the IDE. Discussions should be available from the space, and also from the artifacts around which they revolve.

Making spaces activity-centric. There is an emerging view that software is increasingly being developed not by static teams suggested by corporate organizational structures, but rather by self-organizing or dynamic teams that assemble based on the expertise needed for projects [1,15]. Accordingly, we contend that spaces should shift their conventional focus on teams to a focus on what participants are trying to accomplish together, i.e. a shared *activity* [8]. A shared space should come into existence when there's a substantive activity to perform. The semantics of the activity can inform the semantics of the shared space, e.g. the space can provide information about the activity lifecycle, sub-activities and their assigned team members, who is working on what sub-activity, and the progress being made. When the work is done, the space can be retired into a content repository based on the work that was accomplished, rather than the team being supported.

Making spaces subscribable and searchable. We plan to augment our reinvented spaces with a subscription mechanism (possibly RSS-based), so that users will be delivered "feeds" containing only the relevant items from their spaces. Each user can read these personalized feeds using a form of feed aggregator. Once spaces are activity-aware and have knowledge of which sub-activities are important to each member, it also becomes possible to make inferences about what information is important for each, and the system should be able to make automatic "subscription recommendations." Once we introduce the notion of feeds, we can envision a team publishing or exporting some of the items in their space. Then it becomes possible for other teams or individuals to subscribe to a particular team's output and keep up with the information deemed appropriate for external consumption.

The flip side of getting too much information is not being able to retrieve the right data. Thus, we claim that it's also important to have a powerful cross-space search capability.

CONCLUSION

Our group has carried out some studies of software development team dynamics that will help inform our work on improving team spaces [6]. We anticipate that there will be interesting UI and visualization challenges for our improved team spaces. We expect that the new team spaces will be useful not only for developers, but also for other user groups (e.g. analysts, QE teams, even users outside software development).

REFERENCES

1. Agile Alliance. Manifesto for Agile Software Development, 2001, http://www.agilemanifesto.org.
2. Booch, G. and Brown, A. Collaborative Development Environments. *Advances in Computers, 59*, Academic Press, Aug. 2003.
3. Bugzilla, http://www.bugzilla.org.
4. Cheng, L., de Souza, C., Hupfer, S., Patterson, J., and Ross, S. Building Collaboration into IDEs. *ACM Queue, 1*, 9 (2003-2004), 40-50.
5. Cheng, L., Hupfer, S., Ross, S., and Patterson, J. Jazzing Up Eclipse with Collaborative Tools. In *Proc. of 2003 OOPSLA Workshop on Eclipse Technology eXchange*. ACM Press (2003), 45-49.
6. De Souza, C., Redmiles, C., Cheng, L., Millen, D., Patterson, J. Sometimes You Need to See Through Walls: A Field Study of Application Programming Interfaces. In *Proc. of 2004 ACM Conference on Computer-Supported Cooperative Work*. ACM Press (2004), 63-71.
7. Eclipse.org, http://www.eclipse.org.
8. Geyer, W., Vogel, J., Cheng, L., and Muller, M. Supporting Activity-centric Collaboration through Peer-to-Peer Shared Objects. In *Proc. of 2003 ACM SIGGROUP Conference on Supporting Group Work*. ACM Press (2003), 115-124.
9. Groove Networks, http://www.groove.net.
10. Gutwin, C., Penner, R., and Schneider, K. Group Awareness in Distributed Software Development. In Proc. of *2004 ACM Conference on Computer-Supported Cooperative Work*. ACM Press (2004), 72-81.
11. Herbsleb, J., and Grinter, R. Architectures, Coordination, and Distance: Conway's Law and Beyond. *IEEE Software*, *16*, 5 (1999), 63-70.
12. Hupfer, S., Cheng, L., Ross, S., and Patterson, J. Introducing Collaboration into an Application Development Environment. In *Proc. of 2004 ACM Conference on Computer-Supported Cooperative Work*. ACM Press (2004), 21-24.
13. IBM Lotus Team Workplace (QuickPlace), http://www.lotus.com/quickplace.
14. Java Studio Enterprise 7, http://www.sun.com/software/products/jsenterprise/.
15. Malone, T. *The Future of Work: How the New Order of Business Will Shape Your Organization, Your Management Style and Your Life*. Harvard Business School Press, 2004.
16. Rising, L., and Janoff, N. The Scrum Software Development Process for Small Teams. *IEEE Software, 17*, 4 (2000), 26-32.
17. Roseman, M. and Greenberg, S. TeamRooms: Network Places for Collaboration. In *Proc. of ACM 1996 Conference on Computer Supported Cooperative Work*. ACM Press (1996), 325-333.
18. Sawyer, S. and Guinan, P. Software Development: Processes and Performance. *IBM Systems Journal*, *37*, 4 (1998), 552-569.
19. SourceForge.net, http://sourceforge.net.
20. TWiki, http://www.twiki.org.
21. Visual Studio 2005 Team System, http://lab.msdn.microsoft.com/vs2005/teamsystem/.