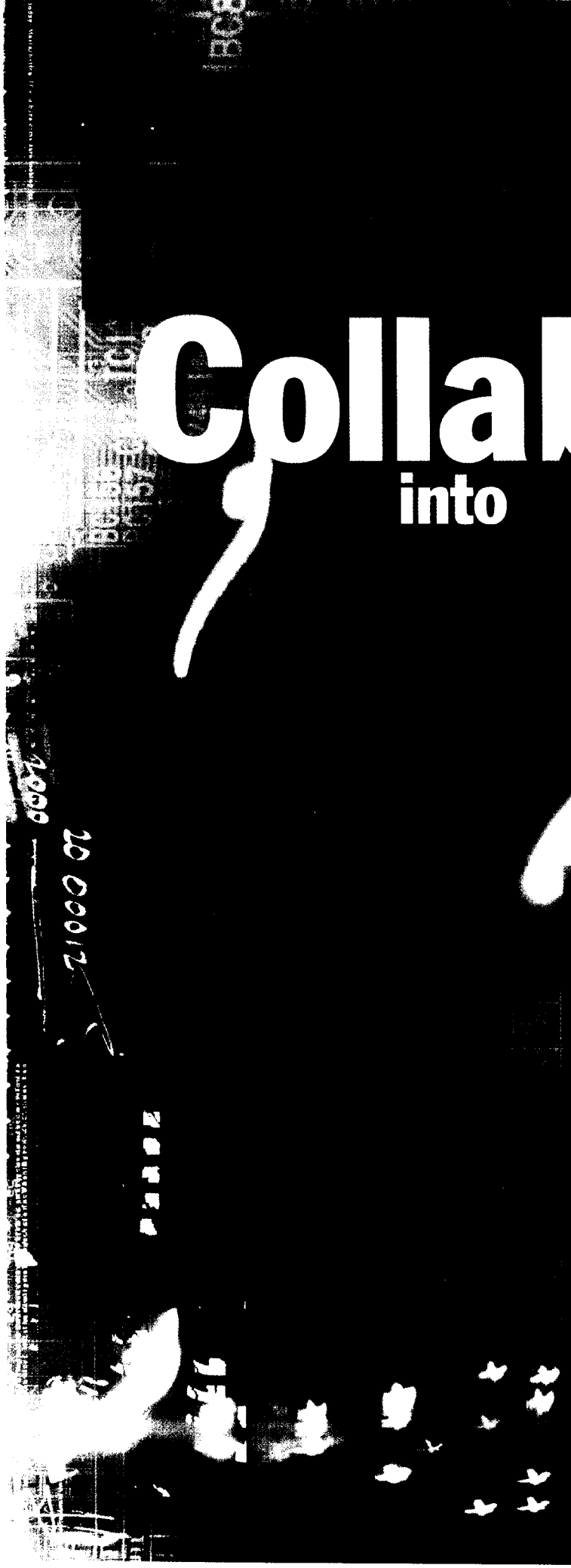
Collaboration into



Software development is rarely a solo coding effort. More often, it is a collaborative process, with teams of developers working together to design solutions and produce quality code. The members of these close-knit teams often look at one another's code, collectively make plans about how to proceed, and even fix each other's bugs when necessary. Teamwork does not stop there, however. An extended team may include project managers, testers, architects, designers, writers, and other specialists, as well as other programming teams. Programmers also interact with the community of developers outside their organization to obtain advice, code snippets, and a general understanding of what works and what doesn't.

Despite its benefits, collaboration can be time-consuming and problematic. Studies show that the escalating number of meetings, e-mails, discussions, source-control management tasks, and other coordination efforts are leaving less than half of the workday to do any real coding. But we cannot simply walk

Edit ↓ Compile ↓ Run ↓ Debug ↓ Collaborate?

LITE CHENG, IBM RESEARCH
CLEIDSON R. B. DE SOUZA, UNIVERSITY OF CALIFORNIA, IRVINE, AND FEDERAL UNIVERSITY OF PARÁ, BRAZIL
SUSANNE HUPFER, IBM RESEARCH
JOHN PATTERSON, IBM RESEARCH
STEVEN ROSS, IBM RESEARCH

away from meetings, turn off our e-mail, and hide in our cubes. We cannot ignore the trend toward an increasingly interconnected world where development teams are distributed around the globe. Indeed, a major problem in software development is the breakdown in communication and coordination among developers.

There are ways to confront these concerns: Better managerial processes, team-building training, careful architectural design, disciplined approaches to coding together, and agile software development practices can help significantly. (For more information on collaboration and software development, see Resources on p. xx of this issue.) Tools can help us collaborate without being dragged off to the meeting room. Examples include configuration management and bug tracking systems, as well as e-mail. Web sites such as SourceForge[1] integrate these tools into a single integrated online experience and are the hub of many distributed development projects.

From the individual developer's perspective, the IDE (integrated development environment) is where coding takes place and is the home of many different development tools. If coding is a team effort, then why not add collaborative capabilities to the IDE toolset alongside the editor, compiler, and debugger? In this article, we explore this notion of integrating collaboration into the IDE.

WHY INTEGRATE COLLABORATION?

Collaborative tools such as e-mail and instant messaging (IM) have existed outside the IDE for a long time. So, what is the payoff of folding such tools into the IDE? Let's examine this by considering examples using configuration management, screen sharing, and e-mail/IM.

Configuration Management. Tools such as CVS (Concurrent Versions System) are the central repositories of the software development team. They are also very structured collaborative tools: They allow developers to exchange, modify, mark, and merge files in a coordinated manner. While powerful, these tools are complex, especially for large development projects. By integrating configuration management into the IDE, we eliminate an extra step to perform shared file management operations. A good integration would give the developer the illusion that using configuration management is no more difficult than using the local file system to manage local files. The payoff here is similar to the benefits of integrating tools such as the debugger and the linker into the same IDE: Integration saves time and effort spent switching over to other tools and reduces the learning curve, only requiring familiarization with a new feature in the same IDE rather than learning an entire new stand-alone tool. Grady Booch and Alan Brown call this *reducing friction* in the software development process.[2]

Screen Sharing. A common occurrence in the course of a developer's workday is to ask a colleague to come over to help figure out a problem with some code. Although this usually involves a lot of explaining and discussing, the end result is that the code gets fixed more efficiently than if the programmer had pored over the problem alone. Getting another person's viewpoint is codified in practices such as code reviews, pair programming, and—on a larger scale—open source software development. Screen-sharing tools such as VNC (Virtual Network Computing) have been used to facilitate this kind of consultation for distributed teams.

If the screen sharing were integrated into the IDE, the developer could "reduce the friction" by shortcutting through the usual startup overhead of finding the IP address of the teammate, configuring the screen-sharing service, and so forth. Perhaps more interestingly, an integrated solution could save the screen-sharing session (and an electronic transcript of the accompanying IM chat or phone call) with the file of interest into the code repository. Thus, another payoff of integration is *context*: The collaboration is initiated, run, and saved where the action is—in the IDE—and collaborative artifacts immediately can be associated with the code that concerns us. The conversation in context never leaves the IDE, stays focused on the teamwork between the participants, and can be accessed in the future from the context in which it occurred.

E-mail and IM. Taking context up another notch, ad hoc communication tools such as e-mail and IM are often used for development. Messages might be announcements about the latest check-in to the source control system or a discussion about a particular bug. The messages might contain project-specific references (e.g., URLs, package and file names, code repository locations) or pasted code fragments. Composing such messages in an IDE that integrates e-mail and IM could automatically tie developers' informal discussion to formal source code and repository branches. Thus, the mere act of copying and pasting code can form a two-way link between the

discussion about code and the interconnected network of project files and documents. The third payoff of integration is thus *traceability*: The questions and answers about a piece of code, which would normally have been hidden away on an e-mail server or lost in a transient IM, would be a form of code annotation that supplements formal documentation. E-mail or chat discussions between two team members may often be of interest to the entire team. This informal "writing on the wall" can be helpful when new developers are spelunking through undocumented legacy code. Lines of code can also be associated with whoever discussed them, which is invaluable when tracking down leads.

CHALLENGES

Integrating collaborative capabilities into the IDE holds great potential for easing programmers' development activities. This integration introduces a number of technical and design challenges. Major issues include:

- Building for extensibility, interoperability, and flexibility
- Choosing and designing the "right" set of collaborative features
- Supporting transitions between individual and group work

Let's take a look at these challenges.

Building for Extensibility, Interoperability, and Flexibility. Integrating software is often a less-than-straightforward task, and there are challenges particular to integrating collaborative capabilities. For starters, it helps if the IDE being extended has an architecture that supports extensibility to begin with. Many commercial systems are extensible, with the intention of opening the door to third-party vendors whose components can increase the value of the IDEs. Being able to plug a collaborative extension easily into an IDE helps reduce friction for tool developers.

Integrating collaborative capabilities that enables the benefits of context and traceability, however, requires deeper extension mechanisms in an IDE than simply making an extension show up in the IDE's user interface. Context and traceability require access to models underlying the IDE: the file system, the compiler and syntax-checking mechanisms, networking, and the source control system. To assist developer-related problems, integrated collaborative tools need to understand the models and artifacts the developer is using in the IDE. Extensions to the IDE should be first-class citizens, and as capable and flexible as any feature shipping with the IDE. For example, enabling the chats (and chat transcripts) to properly highlight code syntax and hyperlink to source-code modules requires access to mechanisms used by the IDE to manage its syntax and modules.

Another consideration related to this requirement is that the IDE's search facility should be extensible enough to include whatever artifacts are generated by collaborative add-ons. You have only to note how useful it is to search newsgroups, discussion boards, and Web sites for tips and hints from fellow developers to realize how powerful this capability would be. Integrating IDE search with collaborative artifact search reduces friction (file-based and knowledge-based search become the same). If closely integrated with the IDE's core functions, a search can become even more contextual and traceable. For example, clicking on a word in the editor or a file in the file viewer would bring up a search context menu to locate related documents and discussions.

Ad hoc communication
tools such as e-mail and instant messaging are often used for development.

Interoperability is another issue to consider. Developers use different vendors' tools, even within the same IDE. Also, a collaborative IDE may be used to interact not only within the same organization, but also with external teams or customers. Interoperability issues can arise with vendor-neutral server-side services for collaboration (e.g., IM, e-mail, source control, screen sharing, name directories) and abstract APIs (application program interfaces) offered by the IDE. Open standards and protocols are helpful here, or the development team must agree upon the same set of messaging and collaboration protocols. The IDE's vendor needs to provide a flexible and extensible API—not tied to any specific implementation—for access to services. For example, accessing the source-control system from within the IDE would ideally be virtualized in a vendor-neutral API.

Building for flexibility is another challenge that should not be overlooked. The collaborative add-ons to an IDE should themselves be extensible by outside contributors—ideally, the developers who use them. The norms, practices, and needs of any development team vary and can change over time. Thus, collaborative capabilities need to be just as flexible as the social fabric of the team.

For example, a chat that can be linked to source code in the IDE may also need to be linked to design requirements stored on an external server running a project management system. Customization can come from user preferences, of course, but situations may arise beyond the scope of the collaborative add-on. A documented API enabling tool extensions, preferably consistent with the IDE's API, would be helpful.

An IDE augmented with collaboration requires some kind of supporting network infrastructure. Designing such an infrastructure raises the interoperability issues mentioned earlier (e.g., support for directory services, standards for messaging, etc.). Also, software supporting collaboration must be flexible enough to include artifacts and models specific to the IDE environment. For example, metadata that links source code with e-mail needs to be stored somewhere—perhaps with a special header field or with an URL. Where to store collaborative artifacts is another issue: You could try to juggle multiple stores (e.g., one for e-mail, one for source code, one for discussion forums, one for chat transcripts, one for bug tracking, etc.), or you could attempt to consolidate everything into a single store (e.g., the source-control repository). The former might complicate the administration and configuration of such a system. The latter case is therefore very appealing, but the question then becomes how flexible and extensible the single store is and whether it will be an acceptable solution when the development team starts interacting with outside organizations.

Choosing and Designing Collaborative Features. Choosing the "right" set of collaborative features to integrate with an IDE is a difficult task. Development groups are distinctive and work in unique ways, with their own methodologies, processes, and conventions. Different groups will find different collaborative features useful, because each has its own history, culture, working style, and organizational demands. For example, one particular group in an organization might adopt the practice of using chat to convey some types of information and e-mail for others. Some teams may use CVS for source-code control, whereas others may use IBM's Rational ClearCase. A team's requirements and development processes are also fluid over time, and their collaborative needs can change.

The choice of collaborative features might also change as a result of organizational requirements and external influences. For example, today more and more software development organizations are seeking certifications such as ISO 9001 and CMM (Capability Maturity Model), which means that changes in the software must be documented, reviewed, and authorized before they are integrated into the code. The underlying idea is that changes must be accountable. This affects the choice and/or usage of collaborative features. For example, recording chat conversations may be mandatory in such organizations requiring accountability.

The individuality of development teams and their enterprises thus suggests some guiding principles for adding collaborative capabilities to an IDE. Ideally, these collaborative features should:

- Avoid embedding or enforcing a rigid notion of what the "correct" development process should be, as this varies from team to team and enterprise to enterprise. Instead, the features should accommodate a variety of processes, both formal and informal.
- Provide a flexible collection of ways that developers can collaborate, so that groups can choose which features to use in their particular situations.
- Be configurable and extensible by the developers themselves.

While not an exhaustive list, the following capabilities can flexibly accommodate many of the ways programmers work together:

- Provide peripheral awareness of other programmers and their activities (who is doing what around our code, especially code that you depend on).
- Support a variety of communication mechanisms (text, voice, visual).
- Integrate with the team's source-code control system and bug tracking system.
- Support "in context" communication, both synchronous (chat, screen sharing) and asynchronous (code annotations, persistent chats, team documentation).
- Support searching through saved team artifacts and the development history.

Supporting Transitions Between Individual and Group Work. One important aspect of any collaborative effort is that it is often composed of a web of individual and collaborative activities. In software development settings, this distinction is desirable and indeed often enforced by formal practices and tools. For example, a developer may choose to work in a private branch in the

source-control repository and contribute to the main development stream only for integration milestones. The private branch thus becomes a "sandbox" for personal experimentation and testing closed off from the busy "outside world" of the team.

A natural consequence of this distinction between individual and collaborative work is the need to support the transitions between those aspects—that is, to assist in deciding *when* and *how* to move from the individual to the collaborative activities and vice versa. These transitions are particularly important because of the interdependencies inherent to any software development effort. Indeed, some tool support has already been provided for these transitions—for example, through source control tools' merging mechanisms, which allow several versions of the same file modified by different developers to be easily integrated. These tools are based on syntactic information (lines of code), however, and cannot process semantic information; they are limited in the support they can provide to help software developers understand the impact of the individual work on the larger collaborative effort of the whole team. Tying in less formal collaborative tools here provides mechanisms that allow developers to coordinate and alleviate this problem.

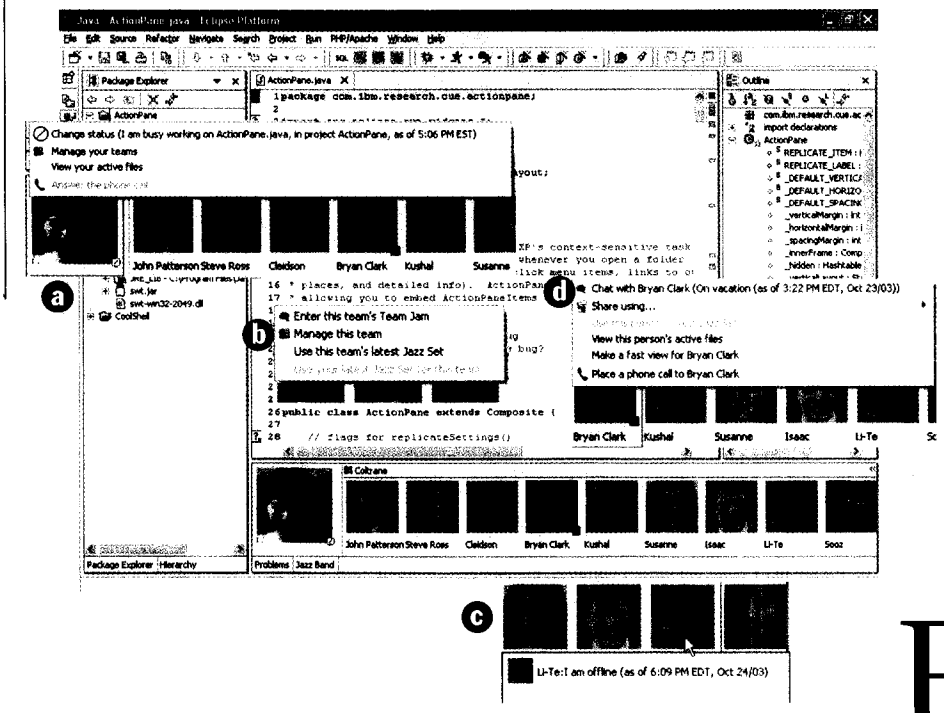
Another challenge of supporting the individual and group work dichotomy relates to attention management, interruptions, and information overload. If a collaborative feature is too distracting, it can interfere with everyday work rather than support it and may be summarily rejected by developers. Each user needs to have complete control over how a feature should deliver alerts, how to specify the appropriate times for interruptions, and how to quickly filter and sort relevant from irrelevant information. Collaborative tools need to take into account that developers will sometimes feel more interruptible and collaborative, and other times will be "in the zone" and want to shut out all distractions.

JAZZ: A CASE STUDY

A working example of integrating collaboration into an IDE is Jazz, a research project at IBM focusing on a specific set of collaborative features for the Eclipse IDE.[3] The objective is to nurture the "immediate team" of developers as a thriving social group, while capturing the team's artifacts to provide a useful backdrop and context for communication.

In the Jazz-enhanced IDE, everyone on the team is a first-class member of the environment, on par with files,

Jazz's Person-Centric Collaborative Features in the Eclipse IDE



(a) Pop-up menu to set the user's status message and manage all teams. On the bottom right of each portrait is the user's IM status. Offline team members have dark grayscale portraits. (b) People are grouped in user-defined teams. (c) Hovering over portraits reveal their online status messages. (d) Pop-up menu for a team member reveals options to start a chat and a screen share.

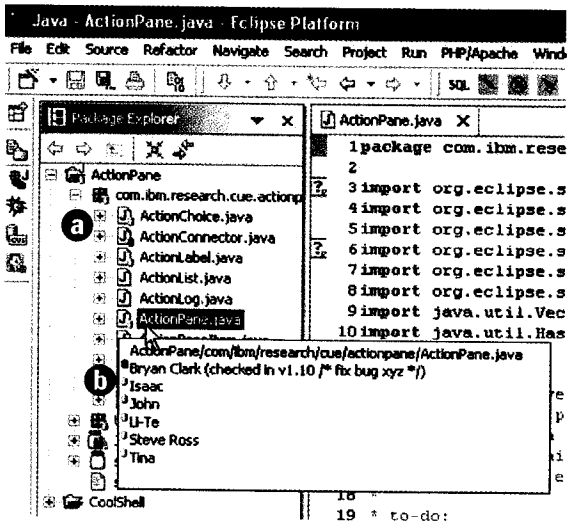
FIG 1

folders, and libraries. We provide a facility similar to an IM buddy list to monitor who is online and coding or not (see figure 1). The IM status message can automatically incorporate contextual information such as what file the developer is currently working on. Developers can initiate chats, which can be saved as code annotations or into a discussion forum, or use other communication modes such as screen sharing and VoIP (voice-over-IP) telephony, without any additional setup overhead (i.e., setting up servers, configuring IP addresses, etc). Thus, these capabilities reduce friction by being readily available within the coding environment and provide context and traceability by enabling developers to converse around code and link messages and status information with code artifacts.

Jazz also provides resource-centered awareness. As shown in figure 2, files and other resources in the file viewer are decorated with colored icons to indicate what other developers are doing with their local copies of the files (e.g., indicating that a file is in focus and being edited at this very moment or that a file has been locally saved but not checked back into the code repository). Tooltips on the resources reveal who is responsible for these changes. Merging these indicators into the IDE's file viewer reduces friction by saving the developer from having to go outside the IDE and manually dig for such information. These indicators provide the developer with the same kind of peripheral awareness of the activities of others on the team as would be available if the team were all working in proximity. Also, these indicators appear in context, where the developer normally manages files, and this additional information is blended with cues normally associated with files (e.g., file size, last modified date). Moreover, the ability to discover at a glance who was responsible for the changes incorporates traceability.

For more information on Jazz, see our workshop paper, "Jazzing up Eclipse with Collaborative Tools,"[4] from the Eclipse Technology Exchange Workshop at OOPSLA 2003. For additional examples of collaborative capabilities integrated with the Eclipse IDE, turn to "Eclipse: An Example of Collaborative Tools in the IDE" on page 47 of this *Queue* article.

Jazz's Resource Awareness FIG 2



(a) Files with local changes are decorated with colored icons on the bottom right of each file icon. (b) Tooltips over a file reveal who made the local change (e.g., file size, last modified date). The ability to discover at a glance who was responsible for the changes incorporates traceability.

LESSONS LEARNED

Our experiences with the Jazz Project have pointed out some interesting implementation issues to keep in mind when integrating collaboration into an IDE. These are the major lessons that we learned:

Open or Standardized Protocols Increase Choice and Ease Deployment. To accomplish screen sharing in Jazz, we chose to integrate a TightVNC client and server;[5] this screen-sharing software uses the standard RFB (remote framebuffer) protocol used in VNC.[6] Given this choice, we could experiment with plugging in different brands of VNC clients and servers. A non-Jazz user can even host a screen-sharing session to Jazz members or join a session taking place within Jazz. For example, a quality-assurance analyst who is working within a different software-testing environment may collaborate with a developer working within the IDE. IM could benefit from the flexibility of using open standards as well: Developers can chat within their IDE, as well as with managers and others external to the IDE. Deployment becomes easier if the IDE's IM system works with the

company's existing messaging infrastructure.

Open, Portable, Extensible APIs Allow Customization Beyond the IDE's Specification. In the case of our TightVNC server, we are using a Windows DLL (dynamic link library). Despite following the VNC protocol, this component is not immediately portable to a different language and platform. If the TightVNC community were to publish an open Java API for screen sharing, the job of porting our JNI (Java Native Interface)-based screen-sharing capability to an alternate platform—say, Linux—would be much easier.

APIs can be "open" on two levels: open source (the developer can directly read the source code) or open APIs (the developer can easily access an extensible API). In the case of Jazz, we were fortunate that Eclipse had both of these attributes. IDE makers probably can't anticipate all the needs of a developer integrating collaboration (or some other unforeseen functionality), but they can help by providing an extensible API.

Consider the resource-centered awareness we added to Jazz: Eclipse allows us to define an extension for file decorators (normally used to put indicators like compilation error markers on files in the file manager) that lets us put colored icons on the files to indicate what developers are doing with them. The ability to examine the open source code proved even more helpful: Eclipse provides a generic API for source control, but it was too limited for our aim of enabling resource awareness. Because we could read

Eclipse: An Example of Collaborative Tools in the IDE

Adding collaborative features into IDEs is not a brand new idea. Commercial products, open source, and research work already exist and continue to evolve. IDEs such as Eclipse (http://www.eclipse.org), NetBeans (http://www.netbeans.org), and IntelliJ (http://www.intellij.com) seamlessly integrate configuration management tools—an example of collaborative tools used by software developers in different contexts to coordinate their work. Here we reflect on Eclipse-related technologies for examples of collaborative tools integrated within the IDE.

COMMERCIAL PROJECTS

CodeBeamer and CodePro Studio

Commercial products that enhance IDEs with collaborative capabilities such as messaging, project management, and shared data.

http://www.intland.com/

http://www.instantiations.com/codepro/

OPEN SOURCE

Stellation

An open source effort (led by IBM Research) that introduces fine-grained source control—tied to the notion of activities—to simplify collaboration and provide awareness of changes to team members. It features lightweight activity authoring and file associations, enabling developers to manage relevant work, notify the team of their current work, be informed of changes pertaining to their own activities, and provides a context for persistent conversations.

http://www.eclipse.org/stellation

RESEARCH PROJECTS

GILD (Groupware-enabled Integrated Learning and Development)

A research project investigating how a collaboration-enabled IDE can help students learn programming more effectively.

http://gild.cs.uvic.ca/docs/overview/innovate.pdf

Hipikat

A research project that ties Eclipse's search with newsgroup and Bugzilla information.

http://www.cs.ubc.ca/labs/spl/projects/hipikat/

PLUG-INS

Thanks to its extensible architecture, the Eclipse IDE has a thriving community of collaborative plug-ins. The Eclipse 3.0 milestone integrates a "CVS blame" feature into the editor: Developers can select a line in the editor and find out who was responsible for writing the line of code.

Sangam

A plug-in that features a shared editor and chat for pair programming.

http://sangam.sourceforge.net

Composent

A plug-in that provides a variety of collaborative capabilities, including group chat, file sharing, co-browsing, and team awareness.

http://composent.com/code/eclipsesite/

Koi

This project is building a collaborative infrastructure for Eclipse applications.

http://www.eclipse.org/koi

the source code, however, we were able to locate the CVS implementation of the generic API, and it serendipitously provided public APIs that supplied the functionality we needed.

The open source code also came to the rescue when we were triggering chats around selected source code in the editor. Using a context-menu API, we were able simply to add a new right-click menu item to start a chat around a selected line of code. When we wanted to dynamically display a list of possible chat partners in a submenu cascading off the main menu (to enable "one-click" chatting), however, we found the API didn't support dynamic menus. In this case we examined the source code of the code editor, discovered where we could obtain the actual menu handle, and then inserted code to form our dynamic menu. Once again, extensible APIs enabled us to get our "foot in the door," and open source let us go even further—all without bothering the original developers of the IDE.

Build IDE Add-Ons for a Seamless User Experience. Much of the work in these examples relates to the UI (user interface) model of the IDE (e.g., making items appear in a context menu or putting icons on the files in the file explorer). The better the UI model is exposed to the developer, the easier it is to achieve these UI features, and the smoother the user experience will be. To the user, these new collaborative contributions should feel as if they are native to the IDE; they should be straightforward to use and not require a mental context shift.

One example of a seamless user experience is our "Jazz Band," or buddy list. A standard IM buddy list can occupy as much space as it wants on a desktop or can live minimized in the system tray. In the IDE where space is at a premium, the developer might want the list to be as compact as possible. Our solution was to build the Jazz Band UI so that it automatically exposes more (or fewer) details depending on how much space the user allots to it. This is a more continuous, finer level of UI granularity than in a typical stand-alone application: Names, images, status messages, and decorators appear, scale, or disappear, depending on the available space.

User IDs for Different Collaborative Features Need to be Coordinated. Considering that collaboration involves people interacting, it's not surprising that a major technical hurdle is reconciling user IDs among the different collaborative tools. This is especially compounded in an IDE because you're mixing and matching many tools: IM, source control, e-mail, and screen sharing can all be using different sets of IDs. We encountered this problem when implementing awareness of individuals using shared resources; we chose to map IM user IDs to/from source-control user IDs on the client side, as all of the user information was stored locally. This may not be true for all systems, so in other cases a unifying directory such as LDAP (Lightweight Directory Access Protocol) is useful.

Having multiple user IDs from multiple collaborative tools also means multiple passwords and sign-on mechanisms. Ideally, sign-on should be kept automatic, or at least down to just one prompt for a user ID and password. Again in Jazz, sign-on data is available locally, but this gets more complicated for other systems dependent on central usernames or user domains.

Also, collaborative tools such as IM may be used simultaneously by the same user, both inside and outside of the IDE, and therefore need to accommodate multiple sign-ons of the same user from the different contexts. These IM sessions need to be reconciled and operate in harmony. In the case of Jazz, we sidestepped this issue because we had our own IM system. IM systems need to support either parallel sign-ons of the same account on the same machine or a notion of personas—where a user ID may have several personas representing different contexts, or the IM system centralizes requests from different contexts. This problem seems to be specific to synchronous applications such as IM and screen sharing.

Realtime Awareness Needs Support for Push-Based Notification. One requirement that our implementation exposed early on is the need for servers that can signal state changes to clients. In the case of integrating information from the source-code control system, we discovered that CVS cannot notify clients; we were forced to make clients periodically poll the server to determine if there had been any relevant changes in the repository. We take good advantage of the signaling capability of our messaging system. Indeed, IM systems typically signal state changes, but other collaborative stores such as e-mail and Web-based systems often rely on automatic or even manual polls from the client; this doesn't fit well with the notion of realtime awareness that we aim to achieve in the IDE.

Address Software Developer Concerns Before Deployment. To assess potential problems in our proto-

type, we conducted 14 interviews during summer 2003 with professional software developers at IBM. Although our prototype was still under development, we used storyboards with mock screenshots (which we had used to plan our design earlier) as conversation pieces during the interviews. Information overload was an important concern that emerged. Although the ability to record chats sounded extremely useful as a means of documenting important design decisions that would otherwise be lost, some developers pointed out that certain conversations—for instance, ephemeral ones—should not be recorded because doing so would make finding relevant information in Jazz difficult. Privacy was another concern raised by the software engineers during the interviews. Some users expressed apprehension that the resource-centered awareness feature might be used by unethical managers to monitor their work, instead of being used as a coordination aid to avoid conflicting changes. Those concerns will be addressed in our continued work on Jazz.

CONCLUSION

Collaboration plays an integral role in software devel-

RESOURCES

Much has been studied in the area of collaboration in software development. Here is a small sample of introductory articles related to the discussion presented here:

Architectures, Coordination, and Distance: Conway's Law and Beyond
J. Herbsleb and R. Grinter
IEEE Software (Sept.-Oct. 1999), 63–70.
Reports findings of a study of various distributed software development teams at Lucent, with interesting discussions on the role of informal communication, as well as cultural factors in distributed multi-site software development.

Breaking the Code: Moving Between Private and Public Work in Collaborative Software Development
C. de Souza, D. Redmiles, and P. Dourish
Proceedings of the ACM GROUP (Nov. 2003).
Describes a set of formal and informal work practices adopted by the members of a software development team to minimize the impact of an individual's work when made available to the other team members.

Collaborative Development Environments
G. Booch and A. W. Brown
Advances in Computers 59 (Aug. 2003)
Presents the motivations for collaborative development environments in general, including the notion of "reducing friction." Also surveys various solutions for software development teamwork.

A Field Study of the Software Design Process for Large Systems
B. Curtis, H. Krasner, and N. Iscoe
Communications of the ACM 31(11) (Nov. 1988), 1268–1287.
Describes the three main problems of designing large software systems identified by the authors, namely: the thin spread of application domain knowledge, fluctuating and conflicting requirements, and communication bottlenecks and breakdowns.

People, Organizations, and Process Improvement
D. E. Perry, N.A. Staudenmayer, and L.G. Votta
IEEE Software 11(4) (July-Aug. 1994), 36–45.
Presents a study measuring how much developers spent their time on coding versus noncoding activities, and describes issues, such as a reluctance to use e-mail, that can impact the development process.

Palantir: Raising Awareness Among Configuration Management Workspaces
A. Z. Sarma, A. Noroozi, and A. van der Hoek
Proceedings of the ICSE (May 2003), 444–453.
Presents a tool, Palantír, that augments individual configuration management workspaces with information about other members of the software development team, therefore facilitating the coordination and reducing conflicting changes.

Using a Configuration Management Tool to Coordinate Software Development
R. E. Grinter
Proceedings of the COOCS (1995), 168–177.
Describes the essential role played by the configuration management system in coordinating a team of software developers.

opment, and developers can benefit greatly from the integration of collaborative features with the IDE. While collaborative tools can certainly be used alongside the IDE, integration brings the payoff of reduced friction in the development process, a greater sense of context, and immediate traceability between collaborative artifacts and code artifacts. Integrating collaboration into the IDE raises a number of challenges, including requirements for extensibility, access to the IDE's underlying models, interoperability, and network infrastructure. Collaborative features need to be chosen carefully to meet the needs of the team and should take into account issues of individual versus collaborative work. While our discussion centered on coding, collaborative tools can also contribute to other aspects of development—including project management, modeling, testing, and documentation. Q

REFERENCES

1. SourceForge: see http://www.sourceforge.net.
2. Booch, G., and Brown, A. Collaborative development environments. *Advances in Computers* 59 (Aug. 2003).
3. Eclipse IDE: see http://www.eclipse.org.
4. Cheng, L., Hupfer, S., Ross, S., and Patterson, J. Jazzing up Eclipse with collaborative tools. *Eclipse Technology Exchange Workshop at ACM OOPSLA* (Oct. 2003).
5. TightVNC: see http://www.tightvnc.com.
6. VNC: see http//www.realvnc.com/docs/rfbproto.pdf.

LOVE IT, HATE IT? LET US KNOW

feedback@acmqueue.com or www.acmqueue.com/forums

LI-TE CHENG is a research scientist in the Collaborative User Experience group at IBM Research. He is currently working on the Jazz project, and has interests in human computer interaction, mobile computing, and augmented reality. His background is in computer vision, graphics, image processing, artificial intelligence, and software design. Cheng holds a Ph.D. in electrical engineering from the Multimedia Communications Laboratory at Memorial University of Newfoundland.

CLEIDSON R. B. DE SOUZA is a Ph.D. student in the Interactive and Collaborative Technologies group in the School of Information and Computer Science at the University of California, Irvine, with research interests in the field of computer-supported cooperative work as it applies to software engineering. His focus is on understanding how software engineers work together to develop software, what problems they encounter during their daily work, and what tools can help. Cleidson is on leave-of-absence from the Department of Informatics at Federal University of Pará, Brazil, where he is a faculty member, and holds an M.S. in computer science from the University of California, Irvine, and from the Institute of Computing at University of Campinas, Brazil.

SUSANNE HUPFER is a research engineer working on Jazz in IBM Research's Collaborative User Experience group. Before joining IBM, she cofounded a technology spinoff of Yale University based on Lifestreams, a system pioneering one of the first alternatives to the desktop metaphor. Her interests include novel software architectures and interfaces for information management and collaboration, and distributed systems. She holds a Ph.D. in computer science from Yale University, where she focused on loosely coupled distributed programming and software for coordination. Hupfer coauthored *JavaSpaces Principles, Patterns, and Practice* (Addison-Wesley, 1999).

JOHN PATTERSON is a distinguished engineer in the Collaborative User Experience group at IBM Research. His research at Lotus/IBM has embraced a range of groupware projects, including an Internet-based state synchronization capability for synchronous groupware and alternate visualizations for Lotus Notes. He is currently directing the Jazz project, introducing collaborative tooling into the Eclipse application development environment in an effort to understand contextual collaboration and the componentry needed to enable it. Patterson received his Ph.D. in experimental psychology from the University of Michigan and has held research and development positions at Decisions & Designs, Bell Laboratories, Bellcore, and SunSoft.

STEVEN ROSS is a senior technical staff member in the Collaborative User Experience group at IBM Research. Prior to his work on Jazz, he was chief architect of a project that used speech recognition and synthesis technology to develop a conversational user interface. He also spent many years as an architect on the Lotus 1-2-3 spreadsheet. He was a founder of Reasonix, where he worked on a highly optimizing Fortran compiler and a mixed initiative expert system, and a software engineer at Verbex. Ross has S.M. and S.B. degrees in computer science from MIT.