

Enabling Collaboration in Distributed Requirements Management

Vibha Sinha and Bikram Sengupta, *IBM India Research Lab*

Satish Chandra, *IBM T.J. Watson Research Center*

The EGRET tool can support global software development teams in collaborating on requirements management.

Requirements management, one of the most collaboration-intensive activities in software development, presents significant difficulties when stakeholders are distributed, as in today's global projects. Because of inadequate social contact, geographically distributed practitioners without appropriate tool support have trouble gaining a consistent understanding of requirements or managing requirement changes. We can alleviate many of these difficulties by integrating

collaboration support in practitioners' work environments.¹

With these needs in mind, we developed a new collaborative tool named EGRET (Eclipse-based global requirements tool) for distributed requirements management. From a tools perspective, we wanted to explore how to weave in collaboration services for distributed requirements management. We were equally interested in how the practitioners with whom we had interacted to derive our scenarios would react to such a tool—in particular, which features they'd most appreciate. Here, we present EGRET's design motivations, a preliminary evaluation of its usability, and suggested directions for further enhancement.

The problem

For more than a year, we interacted extensively with IBM practitioners engaged in dis-

tributed development to understand and propose solutions for their pain points. We spoke to approximately 30 practitioners in different roles—both onsite people in the US and the Netherlands and remote team members based in India—across 14 projects. For data collection, we primarily used semistructured interviews conducted through face-to-face meetings and teleconferences.

The practitioners reported a wide range of issues, from the well-known difficulties of cultural and time-zone differences to challenges related to data privacy, application knowledge migration, project management, and quality metrics. These findings, and a research agenda for global software development that we proposed based on them, are documented elsewhere.² Other researchers have also reported some of these challenges.³

Communicating and managing require-

ments in a distributed setting was one of the concerns the practitioners expressed most often. Enterprises are increasingly outsourcing software development and maintenance to service providers who operate globally to maximize operational efficiency. In most of these projects, business analysts (and sometimes architects) are located close to the customer; this helps the analysts elicit the initial set of business needs and build trust and relationship with the customer. At the same time, development work takes place in an offshore center, where it's significantly more cost-effective. The onsite and offshore teams must work jointly on the requirements to translate a customer's needs into system-level specifications. Requirements thus flow back and forth between locations prior to, and often during, development. More complex engagements might involve requirements from multiple customers who are based in different locations and have varying stakes in the system. There might be multiple development teams (often from different organizations) who must work together to draw up interface requirements and keep their respective components in sync.

In all these scenarios, geographic separation makes it much more difficult to hold effective discussions around requirements, to manage requirement changes across several sites, and to preserve and harness project knowledge. As a result, requirement defects creep in and can go undetected until late into the project, when they're expensive to fix. In fact, in some of the projects we studied, we traced a high proportion (in one case, more than a third) of defects found during user-acceptance testing to requirements defects.

Such challenges aren't specific to any one organization. Other studies have noted that reactions to requirements-related issues are propagated much more quickly locally, through informal communication, than across sites.⁴ Taken together, these challenges indicate a widespread problem—that stakeholders in today's global projects can't collaborate effectively over requirements using existing tools and methodologies.

Tool considerations

Why doesn't the work environment in typical distributed projects support requirements management adequately? What functionalities might a work environment support to correct this deficiency?

Communication and discussions

Many distributed projects share requirements via emailed text documents. Stakeholders might communicate further by phone, email, or chat to resolve ambiguities in the document and refine it. Some projects that we came across use commercial requirements management tools such as IBM Rational RequisitePro and Telelogic's DOORS but primarily for storing requirements—even though the tools have incorporated some basic collaboration mechanisms. The lack of deep integration between the requirements and communication environments leads to frequent context switching and information fragmentation across several media, ultimately leading to a lack of common understanding of requirements.

We believe developers need a more intuitive and seamless integration of informal collaboration facilities in the requirements management environment. Tools must offer facilities for not only documenting requirements but also holding rich contextual discussions around these requirements. Developers should be able to easily initiate a conversation on a requirement from within the tool or navigate from a discussion to an associated requirement. Moreover, just as colocated team members often come to know about important project-related conversation through word of mouth, the tool should support easy dissemination of ongoing discussions, thereby enabling remote team members to participate.

Figure 1 shows this proposed scenario as a sequence diagram: a test analyst who needs to create test cases for selected requirements first identifies the right stakeholders to discuss the requirements with, looks up their online status through a presence-awareness service, then initiates communication (synchronous or asynchronous, depending on availability) with them. To convey context and ease navigation, the analyst can embed links to the requirements in these messages. The system logs the conversation in the communication repository and visually highlights the associated requirements, so that other remote members become aware of the ongoing discussions and participate. Finally, when the test analyst has sufficiently fleshed out the requirements, he or she creates the test cases.

Managing changes

As stakeholders collaborate over requirements, they often detect gaps and ambiguities.

Reactions to requirements-related issues are propagated much more quickly locally, through informal communication, than across sites.

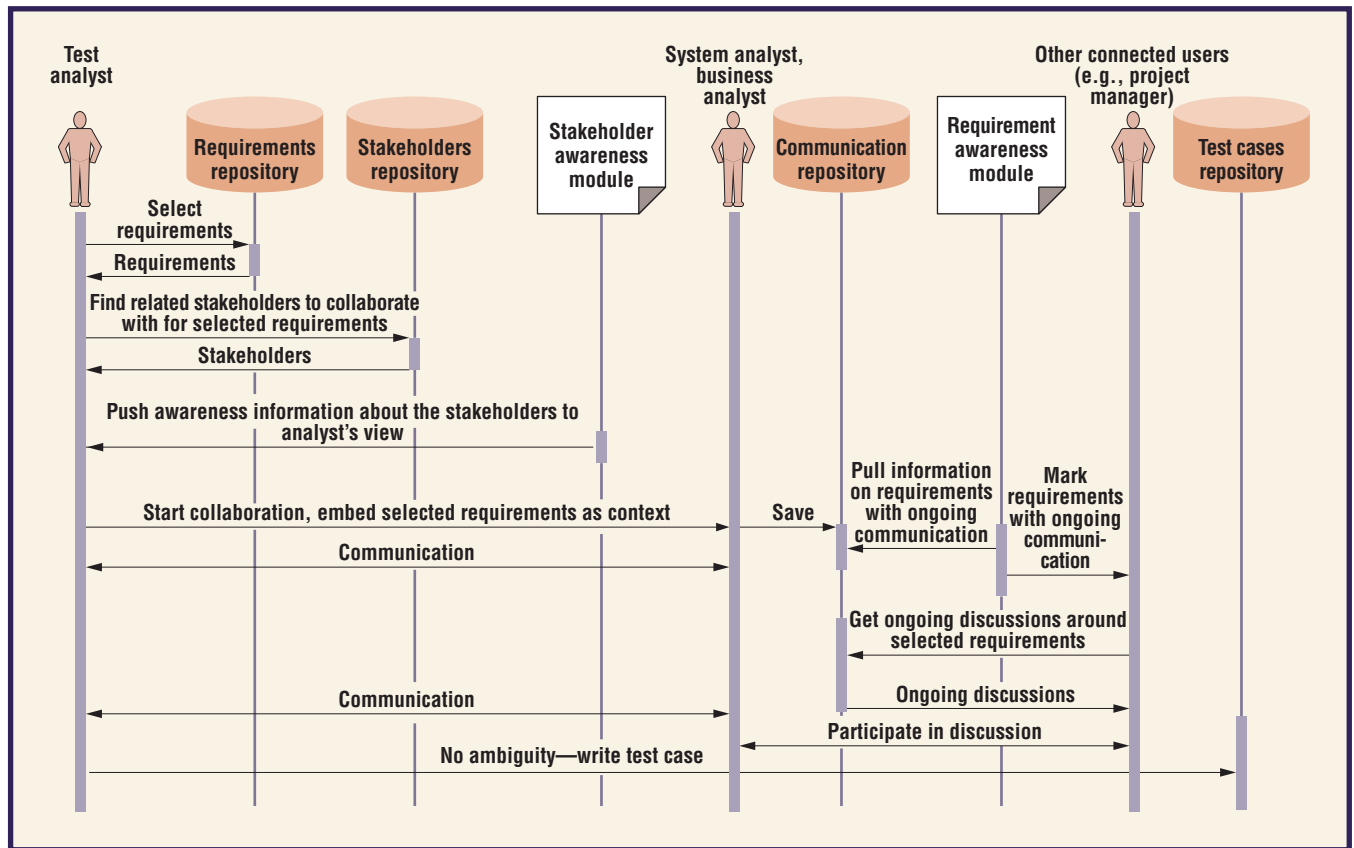


Figure 1. Stakeholders' interaction around requirements to resolve ambiguities. This typical scenario demonstrates informal contextual collaboration and peripheral awareness.

Changing customer needs might also modify existing requirements. Unlike in colocated projects, relying on informal communication to propagate requirement changes in a distributed setting is simply unrealistic (although this is often the case in practice). Important information pertaining to a change often doesn't reach the right people at the right time; moreover, the status of a change isn't globally visible, making it difficult for project managers to track whether all necessary follow-up actions have been taken.

This suggests that change management must be a crucial component of any requirements management solution meant for distributed teams. Just letting developers edit requirements in a shared database isn't enough. Rather, the support must be comprehensive—allowing change requests to be raised, proactively identifying and notifying potentially impacted stakeholders about the change, and keeping a globally visible record of subsequent actions. Figure 2 shows such a proposed scenario, in which a change request submitted by a business analyst is routed to an appropriate stakeholder, who edits the requirement. The system records the

context of the change by linking the edit to the change request and marks the request's status as completed. Furthermore, the system identifies other requirements that might be affected (for example, using traceability information). The system sends change notification messages to appropriate stakeholders and tracks the messages' status.

Managing project knowledge

Another challenge distributed teams face is that knowledge is often fragmented, making preservation and reuse difficult. In particular, offshore teams often grow or shrink depending on funding from the customer. When team members leave a project, crucial domain knowledge is lost. New team members might not have easy access to this knowledge; they end up spending considerable time relearning what some of the requirements mean, why they were changed, and how to handle future changes.

A collaborative tool with facilities for informal communication and change management as we've described has the advantage that all requirements-related information can

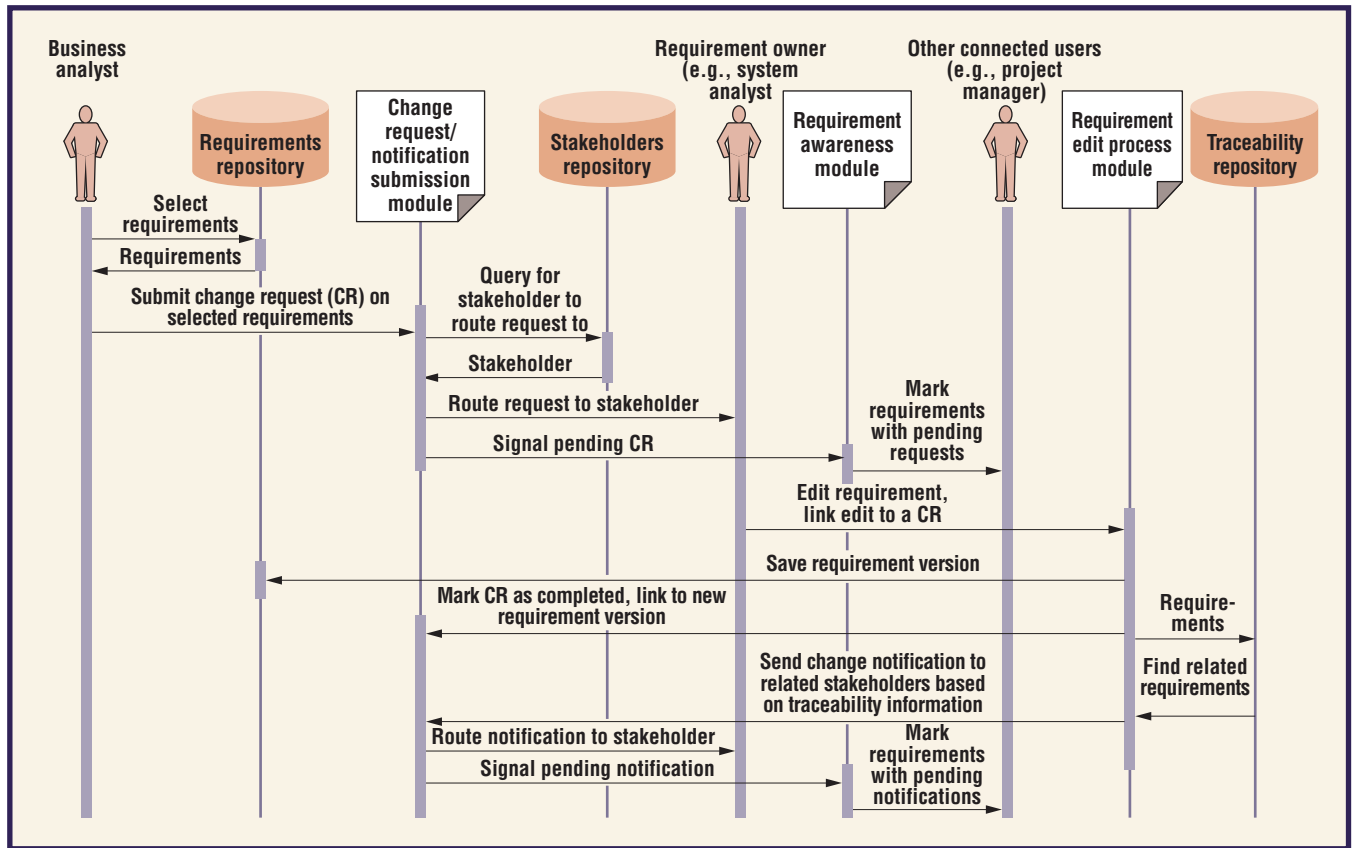


Figure 2. Managing requirement changes in a collaborative environment. This scenario demonstrates change propagation and peripheral awareness.

be logged in the tool repositories for future reference. However, users would need help so that they can navigate the data efficiently and locate relevant information. For example, a new system engineer who is asked to handle a change request on a requirement can run a search query and retrieve the history of previous change requests on that requirement, associated discussions, and subsequent changes, if any; the engineer is then better informed to take appropriate action. Moreover, users can run different kinds of analytics on the persistent data to monitor the project's health—for example, to see if there's too little communication across sites or if change requests are being processed quickly. These analytics can then inform management decisions.

A collaborative development environment

Thus, to address the challenges of distributed requirements management, teams building next-generation requirements management solutions must make collaboration a centerpiece of the tool architecture. Such a tool would have a judicious mix of

- informal collaboration services that facilitate ad hoc conversation around requirements;
- structured (or “formal”) collaboration services to support the requirements phase's critical and regular processes (such as change management) and reduce the need for human interaction;
- awareness features that facilitate these collaboration mechanisms (for example, visual cues for online information and pending change requests); and
- knowledge management techniques to navigate and make sense of project content.

In essence, such a solution would be an adaptation of the emerging concept of collaborative development environments, customized to the needs of geographically distributed stakeholders who manage requirements. The purpose of a collaborative development environment is “to create a frictionless surface for development by eliminating or automating many of the daily, noncreative activities of the team and by providing mechanisms that encourage creative, healthy, and high-bandwidth modes of communication among a project's

Related Resources on Collaboration Tools

Collabnet: www.collabnet.net

A collaborative development environment offering facilities for configuration management, bug tracking, task management, and discussions

CVS: www.nongnu.org/cvs

Eclipse: www.eclipse.org

Expertise Browser and Hipikat:

Eclipse plug-ins that help users identify artifacts and people pertinent to a given task^{1,2}

IBM Rational RequisitePro: www-306.ibm.com/software/awdtools/reqpro
Jazz:

Supports rich synchronous communication and awareness of coding activities within a team of developers working in proximity³

Milos: www.informatik.uni-bremen.de/uniform/gdpa/director/products/jm009.htm

A research collaborative development environment that focuses primarily on software process workflow automation⁴

MySQL: www.mysql.com

Sangam: <http://sangam.sourceforge.net>

Includes a shared editor and chat for pair programming; implemented as plug-ins in Eclipse

Stellation: <http://domino.watson.ibm.com/synedra/synedra.nsf>

An open source effort that introduces “activity”-oriented source control to simplify collaboration and provide awareness of changes⁵

Telelogic Doors: www.telelogic.com/corp/Products/doors/doors

References

1. A. Mockus and J. Herbsleb, “Expertise Browser: A Quantitative Approach to Identifying Expertise,” *Proc. 24th Int’l Conf. Software Eng. Workshop Open Source Software Eng.*, IEEE CS Press, 2002, pp. 503–512.
2. D. Cubranic and G. Murphy, “Hipikat: Recommending Pertinent Software Development Artifacts,” *Proc. 25th Int’l Conf. Software Eng. (ICSE 03)*, IEEE CS Press, 2003, pp. 403–418.
3. L. Cheng et al., “Building Collaboration into IDEs,” *ACM Queue*, vol. 1, no. 9, 2004, pp. 40–50.
4. M. Maurer et al., “Software Process Support over the Internet,” *Proc. 21st Int’l Conf. Software Eng. (ICSE 99)*, IEEE CS Press, pp. 642–645.
5. M. Carroll, J. Wright, and D. Shields, “Supporting Aggregation in Fine-Grained Software Configuration Management,” *Proc. ACM Symp. Foundations of Software Eng. (SIGSOFT FSE 00)*, ACM Press, 2000, pp. 99–108.

stakeholders.”¹ Many research efforts, both open source and commercial, have brought elements of collaboration into their software development activities (see the “Related Resources on Collaboration Tools” sidebar). Most of these tools, however, focus on the collaboration needs of distributed coding, bug tracking, and project management and give relatively little attention to a distributed requirements process. The few exceptions deal only with groupware for distributed requirements negotiation^{5,6} in support of specific interaction models rather than the overall management of requirements across multiple development sites.

EGRET, our prototype collaborative requirements management tool, addresses this gap, following the design we just outlined.

EGRET: A case study

EGRET aims to support the requirements communication and management across distributed teams. Potential users of the tool include business analysts and architects who interface with the customer and elicit high-level requirements as well as system engineers, testers, and other members of distributed development teams who help refine these requirements and define validation criteria for them. EGRET isn’t tied to any particular development methodology, although we believe it will be more useful in iterative-style development, where requirements change frequently and stakeholders need to maintain a high level of collaboration.

In terms of infrastructure, we based EGRET on an Eclipse client that communicates with back-end repositories (see figure 3). We used

- MySQL, a popular open source database, as the repository for requirements, stakeholder information, and communication;
- CVS as the version-controlled repository for figures and tables and for lower-level design elements that might need to be linked to requirements; and
- an experimental collaboration server,⁷ developed at IBM Research, for synchronous communication.

Tool architecture details are available elsewhere.⁸

The EGRET interface consists of a set of views, as the example in figure 3 shows. The main views are as follows:

- Artifact Explorer shows the hierarchical structure of project requirements, stored in a backend MySQL database. You can define requirements from scratch using EGRET or import them from an existing repository.
- Communications Record, the EGRET “inbox,” lets users access all the discussions they have participated in and initiate new conversations with other stakeholders.
- Project Stakeholders lists all stakeholders along with their roles in different modules and their online status.
- Traceability shows the requirements’ traceability relationships.

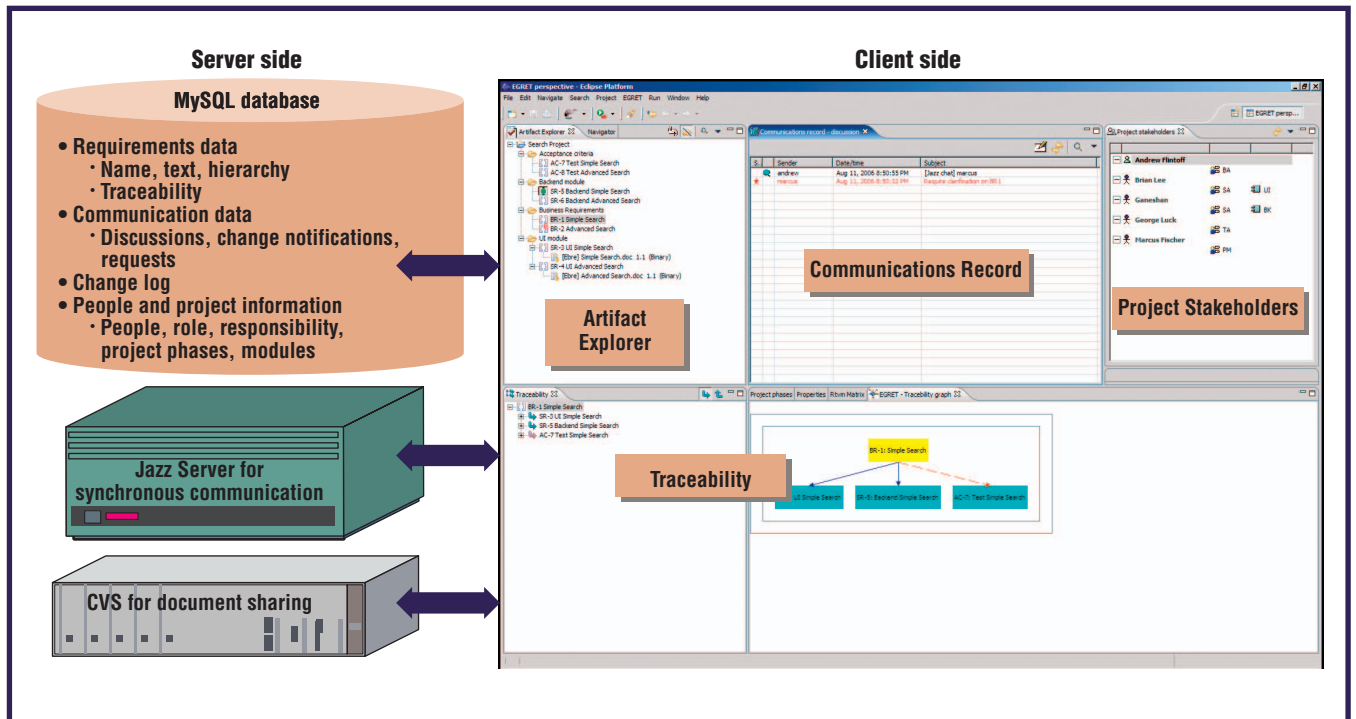


Figure 3. EGRET infrastructure.

During project initiation, an administrator creates an EGRET project and defines related information such as types of requirements in this project, proposed project phases and modules, stakeholder roles, and repository locations, as required. These can be defined using the tool itself or imported through a template. The administrator then creates user accounts and assigns roles to users. Then, registered users log in and create an initial set of requirements, and collaboration begins.

Supporting informal collaboration

EGRET supports synchronous (chat-like) and asynchronous (email-like) *contextual collaboration* around requirements. The idea is to provide a shared context to stakeholders separated by distance, in a way that makes their discussions smoother and more transparent. This involves a change of perspective: instead of looking at email and chat facilities as stand-alone applications external to the requirements management tool, we use them as collaboration facilitators inside the tool itself, with direct access to the requirements repository.

EGRET captures a conversation's context through easily navigable links to relevant artifacts. The tool automatically inserts these links into messages when the context is obvious, or users can insert them manually. For example,

when a user selects a requirement and then chooses to compose an email message, EGRET automatically embeds a link to the requirement to convey the context. The EGRET Read Discussion window in figure 4 shows such a message carrying a link to the BR-2 Advanced Search requirement. The synchronous communication window to its right (which opened up when the user clicked a chat button in the email) carries two links that were automatically embedded: one pointing to the email and the other to the attached requirement, which together constitute the chat conversation's immediate context. This integration of synchronous and asynchronous communication also means that participants can have an extended conversation on a requirement, sometimes through email and sometimes through chat, without having to create separate discussion threads for the same topic. All discussions persist in the communication repository, with links to associated requirements.

Managing changes

Central to EGRET's change management process is the concept of a requirement *owner*, a stakeholder who manages all changes to a given requirement. EGRET automatically routes change requests that are submitted on a requirement to the owner. Again, when a requirement changes, the tool sends automatic notifications to owners

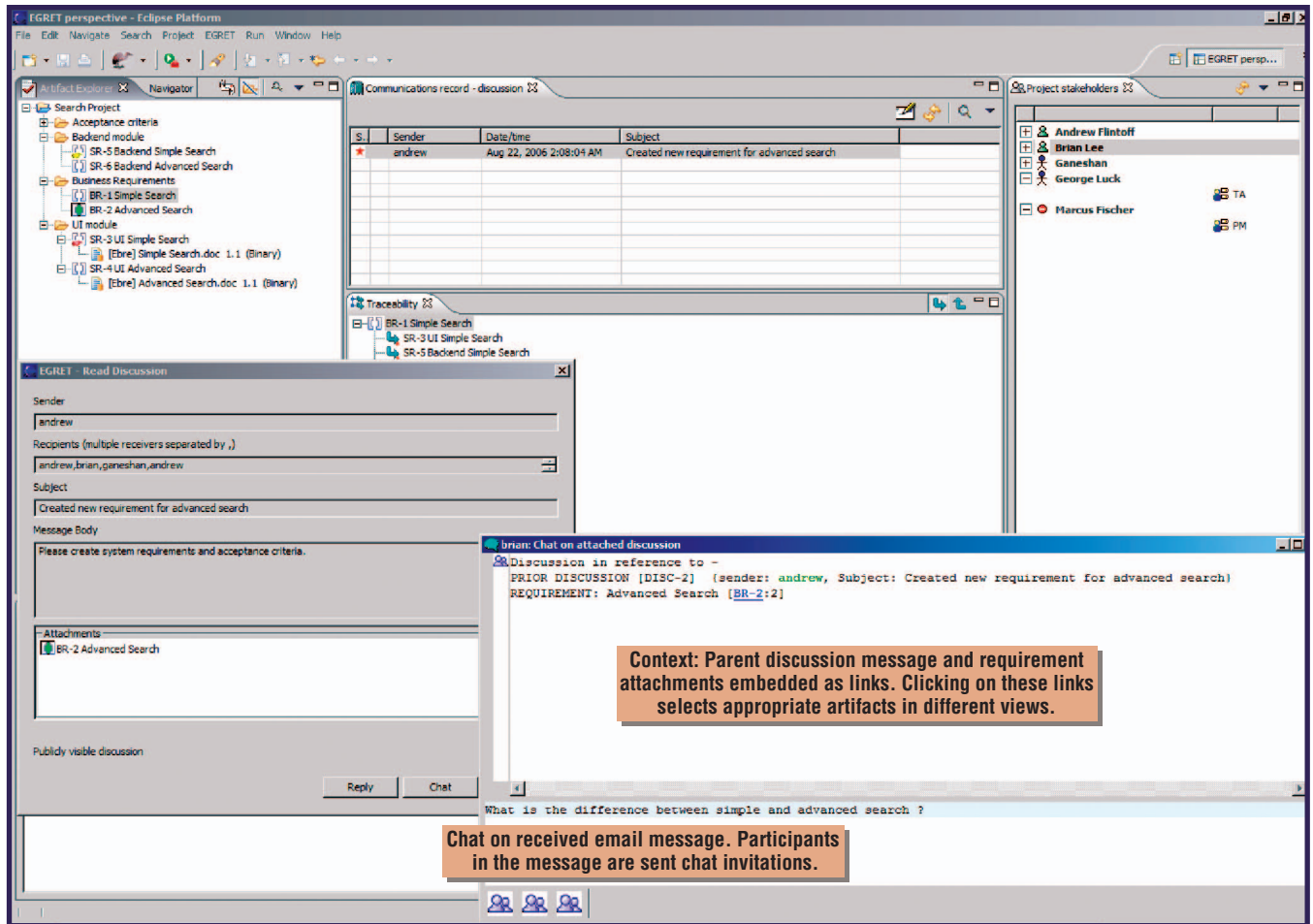


Figure 4. Informal communication, both synchronous and asynchronous, on requirements.

of related requirements (identified using the traceability graph), because these requirements might also need to change. This has two advantages. First, the upstream owner making a change doesn't have to identify and notify stakeholders who might be impacted. Second, potentially impacted downstream stakeholders are notified quickly. Moreover, notifications carry rich contextual information regarding the change—for example, what caused it, links to appropriate requirements, details of other notifications that have been sent out, and so on. This helps remote stakeholders get the big picture of the change and decide what they must do as follow-up. The tool tracks the status of all change requests and notifications automatically to ensure that appropriate follow-up actions are indeed taken.

Although EGRET lets only one stakeholder at a time actually own a requirement (thereby clearly allocating responsibility), other stakeholders might want to know about the requirement or in changes that might affect it. To han-

dle this, EGRET lets any number of users subscribe to a requirement and automatically notifies them of activities pertaining to the requirements they have subscribed to. Also, if traceability relationships in a project aren't rigorously maintained, users might choose to use subscription as the only basis for change propagation. The ownership-and-subscription model ensures that change information goes to all (and only those) stakeholders who would benefit from it. This avoids redundant notifications, often a major irritant in automated systems.

Promoting awareness

One of the challenges in distributed projects is a general lack of awareness of activities at the other sites. Several researchers have explored awareness needs in distributed projects, suggested visual cues to share such information, and studied the positive effects of awareness on group collaboration.^{7,9,10} EGRET's user interface provides *peripheral awareness* information through a set of visual cues—for example, the

green icon on BR-2 in figure 4 indicates that it was the focus of recent discussions. The visual information is also personalized to some extent: a red icon on a requirement indicates that the user has been assigned a task, while pending tasks assigned to others appear as identical yellow icons.

EGRET also provides people-centered presence awareness in the Stakeholders View (see figure 3)—for example, various icons indicate that the user is online, is currently away, or should not be disturbed. In a distributed environment, this information gives a sense of colocation, as if members are in close proximity and have natural knowledge about others' availability.

Managing knowledge

Knowledge management aims to help users navigate, manage, and make sense of the data that builds up in the various EGRET repositories. EGRET supports a comprehensive search on individual repositories as well as global search over all of them. To explore inter-requirement dependencies, users can also view a number of visualizations of traceability data (for example, tree, graph, and matrix views).

However, we see several opportunities to expand the tool to more sophisticated knowledge management techniques. For example, we plan to incorporate routines that will enable users to perform various metadata analytics on project content—for example, to determine high-activity requirements, compute average time for requirements change closure, or identify users who are causing bottlenecks or experiencing communication overload. Such feedback will help project managers discover areas of concern and take the necessary corrective action. We might also adapt traditional knowledge management techniques in information retrieval to provide advanced services, such as analyzing requirements for ambiguity or discovering hidden traceability links.

EGRET evaluation

We demonstrated an early EGRET prototype to various communities within IBM, including practitioners and tool builders. The tool got positive reviews. In particular, reviewers felt that the persistence of ad hoc discussions with remote team members would enable “knowledge logging” while the use of traceability to communicate requirement changes would help

“enforce accountability.” The sessions also saw lively discussions on specific EGRET features. Some practitioners wanted notifications to be sent to their normal email account as well. A few disagreed, saying that this would “take people out of the shared workspace” and thereby defeat the purpose of contextual collaboration. Some expressed concern over our platform choice (“Will managers use Eclipse?”), but many felt the choice was appropriate because demand for Eclipse-based tools is increasing. Some tool builders pointed out that we must address how to efficiently manage the high volume of ad hoc conversation in real projects. In particular, they were unsure if EGRET should save chat transcripts automatically (“it is often too informal”). They suggested that, at the very least, people should be able to edit transcripts and save just the truly relevant information. Some practitioners wanted integration with audio, so that we could also capture telephone discussions. One manager noted that parallel development sometimes takes place across multiple releases in a project, with many requirements being shared across releases, so the tool should provide ways to manage this situation.

Trial runs

Twelve practitioners from three projects (all based in India) then volunteered to conduct trial runs, each ranging from a couple weeks to a month. The testers found the overall user experience good. One team, which used EGRET to capture and refine several high-level requirements in an actual project, noted in their evaluation report that they “found the tool very useful for capturing requirements, having discussions, and tracking requirement changes.” However, the testers also raised some performance issues—in particular, they found synchronous communication not robust enough for heavy use. One of the teams didn't find any use for EGRET after they finished the requirements phase and started with implementation because it doesn't enforce a process connection between the requirements captured and the subsequent design and implementation.

User survey

Of the 12 practitioners who tested EGRET, nine also responded to an online usability survey. The first few survey questions explored communication necessities in distributed re-

Knowledge management aims to help users navigate, manage, and make sense of the data that builds up in the various EGRET repositories.

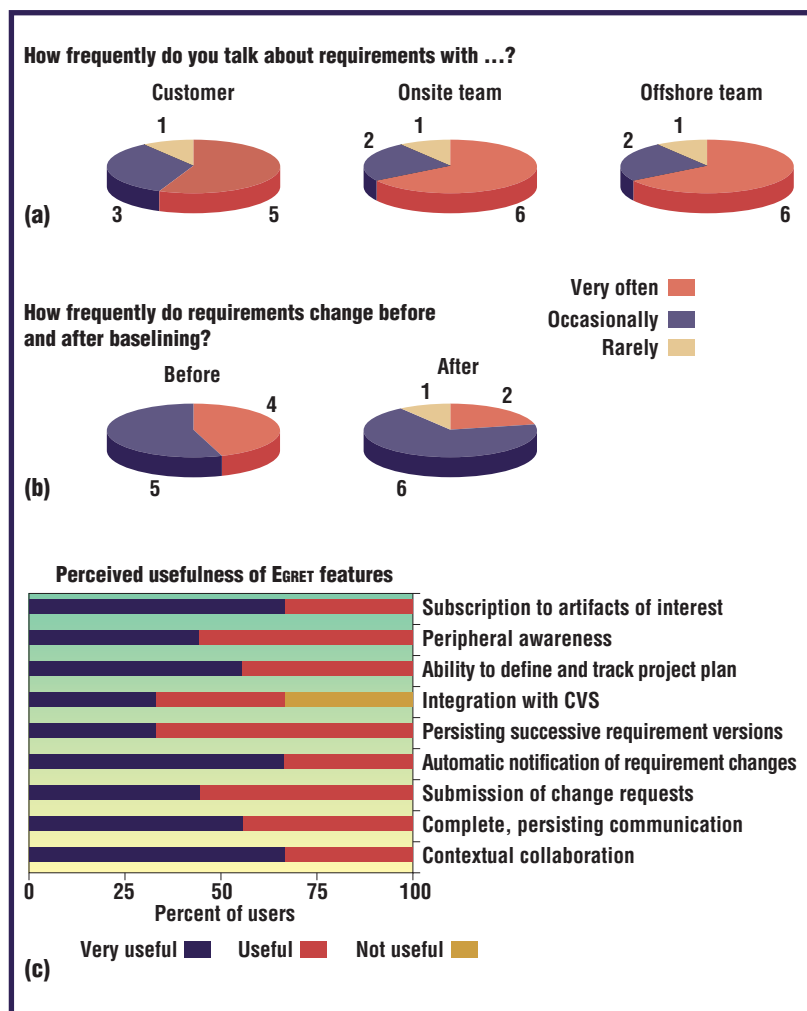


Figure 5. Distributed requirements management issues and EGRET usability as reported by nine survey respondents: (a) frequency of discussions about requirements with various team members, (b) frequency of requirements change before and after baselining, and (c) perceived usefulness of EGRET's features.

quirements management. All the participants reported that they had to communicate with team members in various roles to understand project requirements (see figure 5a). As we expected, we found that requirements frequently change before baselining (that is, before signing off on requirements). But we also discovered that requirements change at least occasionally (in some cases, very frequently) even after baselining (see figure 5b). Opinions were more or less evenly divided regarding the reason for a requirements change:

- actual change in customer needs,
- imprecise requirements gathering by the onsite team, and

- imprecise communication of requirements from the onsite team to offshore teams.

Five of the nine participants reported personal experience with not being notified in time of a requirement change that needed follow-up actions on their side. This led to rework and delays in change closure. The participants attributed this largely to inaccurate impact analysis of the change and insufficient team communication.

The second part of the survey asked participants to rate various EGRET features (see figure 5c). The most popular features were integration of email and chat with requirements (contextual collaboration), subscription to requirements of interest, and automatic notification of requirement changes. All the participants found these features useful: peripheral awareness, persisting discussions on requirements, defining the project plan (phases, deliverables, roles, and so on), preserving successive requirements versions, and submission of change requests. Three participants, however, reported that they didn't find integration with CVS particularly useful. They noted that linking requirements only to CVS documents was restrictive; ideally, it should be possible to link requirements to documents stored in other kinds of repositories as well.

While this preliminary evaluation of EGRET is encouraging, it involved only a few practitioners. We expect to conduct a full-scale pilot of the tool in the future. We also plan to enhance EGRET, providing advanced analysis capabilities on project content, policies for archiving or retiring artifacts, and support for disconnected use.

Primarily, this research validates our belief that a deep integration of appropriate collaboration support with a requirements management tool can greatly aid distributed teams. Several practitioners instantly resonated with the idea when they saw a prototype embodying this. While this supports the arguments that Grady Booch and Alan W. Brown put forth in favor of collaborative development environments,¹ feedback from practitioners has given us further insight into some important design considerations that we must remember when we build such tools:

- Although ad hoc collaboration services are essential, we must design tools such that we can plug in existing collaboration mecha-

nisms (that users are comfortable with) rather than inventing new messaging services.

- User authentication and access to various collaboration services should be uniform. Using multiple IDs, passwords, and sign-on mechanisms isn't acceptable; rather, we must have a unifying directory service.
- Interoperability is another important issue. For users to have a seamless development experience, a collaborative requirements management tool such as EGRET should be able to interoperate with tools belonging to subsequent life-cycle stages, so that there's a natural continuum of work. Similarly, the tool should not be tied to a single repository or version management solution.
- While Eclipse is a natural choice for tool builders because of its rich support for user-interface design and extensibility, a Web interface is essential for the tool to be widely accessible, especially when potential users might include stakeholders outside the immediate development team—for example, customers and business analysts.

We also observed a need to actively promote a tool-centric culture in the management of requirements in distributed projects. Ad hoc manual processes that suffice for requirements management in colocated projects don't scale when development is distributed—a fact that's not yet fully appreciated. In addition, distributed projects need more investment upfront to make requirements precise (for example, through early adoption of test-driven approaches¹¹). Otherwise, changes that result from misinterpretation of requirements can be difficult to manage across sites. Finally, collaborative technologies such as EGRET can make a true impact only when a healthy culture of collaboration exists. Fostering this culture is ultimately the key to making distributed projects work. ☞

Acknowledgments

We thank John Patterson and Li-Te Cheng of IBM Research in Cambridge, Mass., for their help in integrating the Jazz server with EGRET. We also thank Sharath Sampath, Guru K. Prasad, and Suman Poonacha of IBM Global Services in Bangalore, India, for helping us conduct trial runs of EGRET. Finally, we thank all our colleagues who used EGRET, shared their experiences, and participated in the survey.

References

1. G. Booch and A. Brown, "Collaborative Development Environments," *Advances in Computers*, vol. 59, chapter 1, Academic Press, 2003, pp. 1–26.

About the Authors



Vibha Sinha is a technical staff member at IBM Research, India. Her research interests include software engineering and computer-supported collaborative work. She received her MS in electrical engineering from Stanford University. Contact her at IBM India Research Laboratory, Block 1, Indian Inst. of Technology at Delhi, Hauz Khas, New Delhi-110016, India; vibsinha@in.ibm.com.

Bikram Sengupta is a research staff member at IBM Research, India. His research interests include formal methods, software engineering, and systems management. He received his PhD in computer science from the State University of New York, Stony Brook. Contact him at IBM India Research Laboratory, Block 1, Indian Inst. of Technology at Delhi, Hauz Khas, New Delhi-110016, India; bsengupt@in.ibm.com.



Satish Chandra is a member of the Programming Languages and Software Engineering Department at IBM's T.J. Watson Research Center. He received his PhD in computer science from the University of Wisconsin, Madison, and is a member of the ACM. Contact him at IBM T.J. Watson Research Center, 19 Skyline Dr., Hawthorne, NY 10532; satishchandra@us.ibm.com.



2. B. Sengupta, S. Chandra, and V. Sinha, "A Research Agenda for Distributed Software Development," *Proc. 28th Int'l Conf. Software Eng. (ICSE 06)*, ACM Press, 2006, pp. 731–740.
3. J. Herbsleb and D. Moitra, "Guest Editors' Introduction," special issue on global software development, *IEEE Software*, vol. 18, no. 2, 2001, pp. 16–20.
4. D. Damian and D. Zowghi, "Requirements Engineering Challenges in Multi-Site Software Development Organizations," *Requirements Eng. J.*, vol. 8, no. 3, 2003, pp. 149–160.
5. D. Herlea and S. Greenberg, "Using a Groupware Space for Distributed Requirements Engineering," *Proc. 7th IEEE Int'l Workshop Enabling Technologies: Infrastructure for Collaborative Enterprises (WETICE 98)*, IEEE CS Press, 1998, pp. 57–62.
6. B. Boehm, P. Grunbacher, and B. Briggs, "Developing Groupware for Requirements Negotiation: Lessons Learned," *IEEE Software*, vol. 18, no. 3, 2001, pp. 46–55.
7. L. Cheng et al., "Building Collaboration into IDEs," *ACM Queue*, vol. 1, no. 9, 2003, pp. 40–50.
8. V. Sinha, B. Sengupta, and S. Chandra, *EGRET: A Collaborative Tool for Distributed Requirements Management*, tech. report RI06001, IBM Research, 2005.
9. C. Gutwin and S. Greenberg, "Effects of Awareness Support on Groupware Usability," *Proc. Conf. Human Factors in Computing Systems (CHI 98)*, ACM Press, 1998, pp. 511–518.
10. D. Damian et al., "Awareness Meets Requirements Management: Awareness Needs in Global Software Development," *Proc. ICSE Workshop Global Software Development*, 2003; <http://gsd2003.cs.uvic.ca/upload/Damian.pdf>.
11. B. Sengupta et al., "Test Driven Global Software Development," *Proc. 26th Int'l Conf. Software Eng. Workshop Global Software Development*, 2004; <http://gsd2004.uvic.ca/camera/sengupta.pdf>.

For more information on this or any other computing topic, please visit our Digital Library at www.computer.org/publications/dlib.