Global Software Development

James D. Herbsleb and Deependra Moitra, *Lucent Technologies*

The last several decades have witnessed a steady, irreversible trend toward the globalization of business, and of software-intensive high-technology businesses in particular. Economic forces are relentlessly turning national markets into global markets and spawning new forms of competition and cooperation that reach across national boundaries. This change is having a profound impact not only on marketing and distribution but also on the way products are conceived, designed, constructed, tested, and delivered to customers.[1]

Software engineers have recognized the profound influence of business globalization for some time, generating alarmist reactions in some quarters and moving others to try to capture and emulate models that have met with success. More recently, attention has turned toward trying to understand the factors that enable multinationals and virtual corporations to operate successfully across geographic and cultural boundaries.

Globalization and software

Over the last few years, software has become a vital component of almost every business. Success increasingly depends on using software as a competitive weapon. More than a decade ago, seeking lower costs and access to skilled resources, many organizations began to experiment with remotely located software development facilities and with outsourcing. Several factors have accelerated this trend:

- the need to capitalize on the global resource pool to successfully and cost-competitively use scarce resources, wherever located;

0740-7459/01/$10.00 © 2001 IEEE

- the business advantages of proximity to the market, including knowledge of customers and local conditions, as well as the good will engendered by local investment;
- the quick formation of virtual corporations and virtual teams to exploit market opportunities;
- severe pressure to improve time-to-market by using time zone differences in "round-the-clock" development; and
- the need for flexibility to capitalize on merger and acquisition opportunities wherever they present themselves.

As a result, software development is increasingly a multisite, multicultural, globally distributed undertaking. Engineers, managers, and executives face numerous, formidable challenges on many levels, from the technical to the social and cultural.

Is working at a distance really such a problem? Nearly everyone with GSD experience, it seems, has anecdotes illustrating difficulties and misunderstandings. While these stories are compelling, they do not give us a clear picture of its cumulative effects. However, we have strong evidence,[2] based both on statistical modeling of development interval and on survey results, that multisite development tasks take much longer than comparable colocated tasks and that communication and coordination play major roles in this delay.

While we focus primarily on the problems of GSD, we should not neglect the potential benefits of geographic dispersion. For example, if an organization can manage daily handoffs of work between remote sites and focus attention around the clock on critical-path tasks, it is possible to take advantage of widely dispersed time zones.[3] We could theoretically extend the productive hours of the day from the current 8- to 10-hour norm to somewhere near the limit of 24. This is perhaps a distant goal as a general model for development, but occasional benefits—for example, accelerated problem investigation or a distributed daily test-and-fix cycle—are possible.

Moreover, we probably think of "distributed" work in much too limited a way. Distances need not be global to be important.[4-5] In fact, being in another building or on a different floor of the same building, or even at the other end of a long corridor, severely reduces communication. Solutions that help globally distributed colleagues work together more effectively will often help those in the same zip code as well.

Dimensions of the problem

Physical separation among project members has diverse effects on many levels.

Strategic issues

Once a particular set of project sites has been determined (a decision outside the scope of this issue), deciding how to divide up the work across sites is difficult. Solutions are constrained by the resources available at the sites, their levels of expertise in various technologies, the infrastructure, and so on. An ideal arrangement would let the sites operate as independently as possible while providing for easy, flexible, and effective communication across sites. A number of models are possible and appropriate under different circumstances and require different coordination mechanisms.[6]

Another fundamental challenge is the organization's resistance to GSD. This resistance often surfaces because of misalignment between senior and middle management on the intent and perceived benefits of GSD. Many individuals might believe their jobs are threatened, experience a loss of control, and fear the possibility of relocation and the need for extensive travel.

Cultural issues

GSD requires close cooperation of individuals with different cultural backgrounds. Cultures differ on many critical dimensions, such as the need for structure, attitudes toward hierarchy, sense of time, and communication styles.[7] While many people find such differences enriching, they can also lead to serious and chronic misunderstandings, especially among people who do not know each other well. An email, for example, from someone in a culture where communication tends to be direct might seem abrupt or even rude to someone from a different background. A different sense of time can lead to acrimony over the interpretation and seriousness of deadlines.

Cultural differences often exacerbate communication problems as well. When people are puzzled as to how to respond to odd-

Cultures differ on many critical dimensions, such as the need for structure, attitudes toward hierarchy, communication styles, and sense of time.

Developers not located together have very little informal, spontaneous conversation across sites.

sounding messages, they often just ignore them or make uncharitable attributions about the sender's intentions or character.

Inadequate communication

Software development, particularly in the early stages, requires much communication.[8] In fact, software projects have two complementary communication needs. First, the more formal, official communications need a clear, well-understood interface. For crucial tasks like updating project status, escalating project issues, and determining who has responsibility for particular work products, a fuzzy or poorly specified interface loses time and lets problems fall through the cracks.

Disruption to a second, vital communication channel can be surprisingly crippling[9]: developers not located together have very little informal, spontaneous conversation across sites. Informal "corridor talk" helps people stay aware of what is going on around them, what other people are working on, what states various parts of the project are in, who has expertise in what area, and many other essential pieces of background information that enable developers to work together efficiently. One result is that the issues, big and small, that crop up on a nearly daily basis in any software project can go unrecognized or lie dormant and unresolved for extended periods. The absence of ongoing conversation can also lead to surprises from distant sites, potentially resulting in misalignment and rework. The more uncertain the project, the more important this communication channel is.[10]

These issues are even more complex in outsourcing arrangements. The fear of loss of intellectual property or other proprietary information about products or schedules leads to restricted or filtered communication, often seriously impairing this critical channel.

Knowledge management

Without effective information- and knowledge-sharing mechanisms, managers cannot exploit GSD's benefits. For example, they might fail to promptly and uniformly share information from customers and the market among the development teams. When project leaders disseminate status information inadequately, teams cannot determine what tasks are currently on the critical path. Needed expertise might be available but cannot be located and hence is not exploited. Also, owing to poor knowledge and information management, teams miss many reuse opportunities that otherwise would have saved cost and time.

Poor documentation can also cause ineffective collaborative development. The resistance to documentation among developers is well known and needs no emphasis. In GSD, however, in addition to documenting the various artifacts, updating and revising the documentation is equally important. To prevent assumptions and ambiguity and to support maintainability, documentation must be current and reflect what various teams are using and working on.

Project and process management issues

When teams hand off processes between sites, the lack of synchronization can be particularly critical—for example, if the development team at one site and the test group at another site define "unit-tested code" differently. Synchronization requires commonly defined milestones and clear entry and exit criteria. Though concurrent development process models have been suggested in the literature and used,[11-13] effectively implementing concurrent engineering principles in GSD often becomes difficult because of volatile requirements, unstable specifications, the unavailability of good tools that support collaboration across time and space, and the lack of informal communication. Some groups practice risk management in a traditional fashion, not taking into account the possible impacts of diverse cultures and attitudes.

Technical issues

Since networks spanning globally dispersed locations are often slow and unreliable, tasks such as configuration management that involve transmission of critical data and multisite production must be meticulously planned and executed. The need to control product changes and to ensure that all concerned hear about them is much greater in GSD. Other common issues include using incompatible data formats and different versions of the same tools.

The articles in this issue

Each article in this issue provides some answers to managers and engineers navigat-

ing these difficult waters. They describe new tactics and techniques as well as hard-won practical lessons from experience.

As we noted earlier, distance is a major issue in GSD leading to coordination, communication, and management problems. In "Tactical Approaches for Alleviating Distance in Global Software Development," Erran Carmel and Ritu Agarwal provide several approaches that can be applied across a range of geographically distributed projects. Audris Mockus and David M. Weiss, in "Globalization by Chunking: A Quantitative Approach," offer a method for using code change history to compute the degree of "relatedness" of the work items at two sites. They further propose a method for distributing work in a way that minimizes the need for coordination across sites.

In "Using Components for Rapid Distributed Software Development," Alexander Repenning, Andri Ioannidou, Michele Payton, Wenming Ye, and Jeremy Roschelle describe their experiences using a component architecture to support work distribution across sites. Based on their large, geographically distributed testbed for publishing educational software applications, the authors outline a rapid production process using components suited for distributed development. This enables each site to take ownership of particular components and work on them independently without much need for intersite communication and coordination.

In our experience, training programmers to think and behave like software engineers is an uphill task. Many educational programs now include team-oriented work, and with globalization so pervasive, they also need to train their students with geographically distributed development in mind. Jesús Favela and Feniosky Peña-Mora, in "Geographically Distributed Collaborative Software Development," describe a project-oriented software engineering course and show how students in two different countries collaborated using an Internet-based groupware environment. While their objective in designing the project was educational, their experiences are significant to the business community.

Synchronization requires commonly defined milestones and clear entry and exit criteria.

About the Authors

James D Herbsleb is currently a member of the Software Production Research Department and leader of the Bell Labs Collaboratory project. For the last three years, his work has focused on collaboration technology to support large, globally distributed projects. For the past 10 years, he has conducted research in collaborative software engineering, human–computer interaction, and computer-supported cooperative work. He holds an MS in computer science from the University of Michigan and a PhD in psychology from the University of Nebraska. Contact him at herbsleb@research.bell-labs.com.

Deependra Moitra is currently general manager of engineering at the Lucent Technologies India R&D Program. His interests are in software engineering management, management of technology and innovation, new-product innovation, R&D globalization, and entrepreneurship in software and high-tech industries. He serves on the editorial boards of *Research-Technology Management, Technology Analysis and Strategic Management, International Journal of Entrepreneurship and Innovation, Journal of Small Business Management, Journal of Knowledge Management,* and *IEEE Software.* He is a member of IEEE, IEEE Computer Society, IEEE Engineering Management Society, and ACM. Contact him at d.moitra@computer.org.

Software outsourcing is increasingly popular among corporations as an economically and strategically attractive business model. As companies outsource their software needs to software houses across national borders, the two independent organizations must interact—causing the dynamics of globally distributed software development to surface rather strongly. In a thought-provoking article, "Synching or Sinking: Global Software Outsourcing Relationships," Richard Heeks, S. Krishna, Brian Nicholson, and Sundeep Sahay report on three interesting case studies and capture their successful outsourcing strategies to maximize business value.

Finally, we offer three excellent articles summarizing real organizational experiences, lessons learned, and good practices. In "Surviving Global Software Development," Christof Ebert and Philip De Neve narrate Alcatel's experience in globally distributed software development and synthesize the good practices they observed in a large operation involving 5,000 engineers. Robert Battin, Ron Crocker, Joe Kreidler, and K. Subramanian, in "Leveraging Resources in Global Software Development," report on their experiences and approaches while working on Motorola's 3G Trial Systems project, which spans six countries. In "Outsourcing in India," Werner Kobitzsch, Dieter Rombach, and Raimund Feldmann capture their experiences and lessons learned with distributed software development at Tenovis, a German company. Interestingly, while these articles are from different companies operating in different cultural settings, there is a marked commonality in experiences gained and approaches that worked.

Given the global reach of today's large corporations and the global market for software products, few software engineers will remain unaffected as the globalization trend surges forward. As we increasingly work in virtual, distributed team environments, we will more and more face formidable problems of miscommunication, lack of coordination, infrastructure incompatibility, cultural misunderstanding, and conflicting expectations—not to mention the technical challenges of architecting products for distributed development. We hope the advances in collaborative tools and multimedia, Web technology,[14-15] and a refined understanding of concurrent-engineering principles will help us address these challenges.

Acknowledgment

We thank the reviewers who contributed their valuable time and expertise toward development of this special issue. Our sincere thanks also to Dawn Craig at *IEEE Software* for her meticulous efforts helping us put together the issue.

References

1. M. O'Hara-Devereaux and R. Johansen, *Globalwork: Bridging Distance, Culture, and Time*, Jossey-Bass, San Francisco, 1994.
2. J.D. Herbsleb et al., "An Empirical Study of Global Software Development: Distance and Speed," to be published in *Proc. Int'l Conf. Software Eng. 2001*, IEEE CS Press, Los Alamitos, Calif., 2001.
3. E. Carmel, *Global Software Teams*, Prentice Hall, Upper Saddle River, N.J., 1999.
4. T. J. Allen, *Managing the Flow of Technology*, MIT Press, Cambridge, Mass., 1977.
5. R.E. Kraut, C. Egido, and J. Galegher, "Patterns of Contact and Communication in Scientific Research Collaborations," *Intellectual Teamwork: Social Foundations of Cooperative Work*, J. Galegher, R.E. Kraut, and C. Egido, eds., Lawrence Erlbaum Assoc., Hillsdale, N.J., 1990, pp. 149–172.
6. R.E. Grinter, J.D. Herbsleb, and D.E. Perry, "The Geography of Coordination: Dealing with Distance in R&D Work," *Proc. Int'l ACM SIGGROUP Conf. Supporting Group Work*, ACM Press, New York, 1999, pp. 306–315.
7. G.H. Hofstede, *Cultures and Organizations: Software of the Mind—Intercultural Cooperation and Its Importance for Survival*, revised ed., McGraw-Hill, New York, 1997.
8. D.E. Perry, N.A. Staudenmayer, and L.G. Votta, "People, Organizations, and Process Improvement," *IEEE Software*, vol. 11, no. 4, July/Aug. 1994, pp. 36–45.
9. J.D. Herbsleb and R.E. Grinter, "Architectures, Coordination, and Distance: Conway's Law and Beyond," *IEEE Software*, vol. 16, no. 5, Sept./Oct. 1999, pp. 63–70.
10. R.E. Kraut and L.A. Streeter, "Coordination in Software Development," *Comm. ACM*, vol. 38, no. 3, Mar. 1995, pp. 69–81.
11. M. Aoyama, "Managing the Concurrent Development of Large-Scale Software Development," *Int'l J. Technology Management*, vol. 14, no. 6/7/8, 1997, pp. 739–765.
12. J.D. Blackburn, G. Hoedemaker, and L.N. van Wassenhove, "Concurrent Software Engineering: Prospects and Pitfalls," *IEEE Trans. Eng. Management*, vol. 43, May 1996, pp. 179–188.
13. F. Rafii and S. Perkins, "Internationalizing Software with Concurrent Engineering," *IEEE Software*, vol. 12, no. 5, Sept./Oct. 1995, pp. 39–46.
14. S. Murugesan, "Leverage Global Software Development and Distribution Using the Internet and Web," *Cutter IT J.*, vol. 12, no. 3, Mar. 1999, pp. 57–63.
15. M. Aoyama, "Web-Based Agile Software Development," *IEEE Software*, vol. 15, no. 6, Nov./Dec. 1998, pp. 56–65.