AGILE PRACTICES REDUCE DISTANCE IN GLOBAL SOFTWARE DEVELOPMENT

Helena Holmström, Brian Fitzgerald, Pär J. Ågerfalk, and Eoin Ó. Conchúir

This article explores how agile practices can reduce three kinds of "distance" — temporal, geographical, and sociocultural — in global software development (GSD). On the basis of two in-depth case studies, specific Scrum and eXtreme Programming (XP) practices are found to be useful for reducing communication, coordination, and control problems that have been associated with GSD.

HELENA HOLMSTRÖM is a research fellow with the University of Limerick, Ireland, and an assistant professor at the software engineering and management program at the IT University in Gothenburg, Sweden. She can be reached at helena.holmstrom @ul.ie.

BRIAN FITZGERALD holds the Frederick A. Krehbiel II Chair in Innovation in Global Business and Technology at the University of Limerick, where he also is a university research fellow and Science Foundation Ireland Principal Investigator.

PÄR J. ÅGERFALK is a research fellow with the University of Limerick, and an assistant professor at Örebro University, Sweden.

EOIN Ó. CONCHÚIR is a Ph.D. research student at the University of Limerick.

INTEREST IN GLOBAL SOFTWARE DEVELopment (GSD) is rapidly growing as the software industry experiences increasing globalization of business (Herbsleb & Moitra, 2001). In GSD, stakeholders from different national and organizational cultures and time zones are involved in developing software (Damian, 2002), and tasks at various stages of the software lifecycle may be separated and implemented at different geographic locations coordinated through the use of information and communication technologies (Sahay, 2003). As recognized by Sahay (2003), GSD allows for a range of new possibilities. For example, benefits such as the business advantage of having proximity to the market, the ability to exploit market opportunities through quick formation of virtual corporations and virtual teams, and the possibility to use time zone differences to achieve 'round-the-clock development have accelerated the interest in GSD (Herbsleb & Moitra, 2001). As a result, software development is increasingly a multisite, multicultural, globally distributed undertaking.

However, although GSD opens new opportunities, there is little doubt that it presents new challenges (Damian, 2002; Sahay, 2003). As recognized by Herbsleb and Moitra (2001), physical separation among project members has diverse effects on many levels. For example, strategic issues include how to divide work between sites and how to handle organizational resistance. Often, individuals believe their jobs are threatened, they experience a loss of control, and they fear the possibility of relocation. Moreover, cultural issues, such as attitudes toward hierarchy, sense of time, communication styles, and need for structure, are different. Although these differences can be seen as enriching, they can also lead to misunderstandings among people. Cultural differences often exacerbate communication problems, and because software development requires rich communication (Perry et al., 1994), the lack or absence of this can lead to misalignment and rework. Finally, coordination and control issues need to be tackled. Without effective information- and knowledge-sharing mechanisms, the benefits of GSD cannot be exploited.

All the issues mentioned above relate to temporal, geographical, and sociocultural distance, and their combination makes GSD a

As a result, software development is increasingly a multisite, multicultural, globally distributed undertaking.

complex task. In particular, processes such as communication, coordination, and control are challenged (Ågerfalk et al., 2005), and there is a strong need for methods that address these issues (Damian, 2002).

Recently, agile methods (Abrahamsson et al., 2003; Damian, 2002; Holtz & Maurer, 2002) have begun again to be focused on the question of how to address key problems in software development; namely, that software takes too long to develop, costs too much to develop, and does not work very well when eventually delivered. In emphasizing speed and simplicity (McCauley, 2001; Highsmith & Cockburn, 2001), those using agile methods seek to avoid prescribing cumbersome and time-consuming processes that add little value to the software product (Fowler & Highsmith, 2001). Instead, the focus is on individuals and interactions, working software, customer collaboration, and fast response to changes. Agile methods are basically an attempt to satisfy the industry quest for more lightweight and faster development processes.

To achieve this, many agile methods, such as eXtreme Programming (XP) (Beck, 2000) and Scrum (Schwaber & Beedle, 2002), include practices such as pair programming, planning game, sprints, and on-site customer collaboration. Although it is not a magic set of revolutionary new development techniques, it is a set of tried and trusted principles, well established as part of the conventional wisdom of software engineering, but taken to an extreme level. However, due to temporal, geographical, and sociocultural distance in GSD, key concepts in agile methods are more difficult to realize (Maurer & Martel, 2002; Kirscher et al., 2001; Turk et al., 2005). For example, the opportunity for pair programming, on-site customer collaboration, and face-to-face interaction is severely reduced, hence negatively influencing the way in which agile methods can be applied. Although there has been some preliminary research on how to apply XP in GSD (e.g., Ngo-The et al., 2005; Kirscher et al., 2001), the more common view is that agile methods are not applicable for GSD. Clearly, there is more to learn about how to apply agile methods in distributed settings. What is needed is an increased understanding of the characteristics of agile methods (Conboy & Fitzgerald, 2004) and how these can be applied to reduce the negative influence of distance in GSD (Maurer & Martel, 2002).

In this article, we present findings from two in-depth case studies in which we explore challenges associated with distance in GSD and how agile practices reduce these challenges and, hence, reduce distance. The specific research questions are elaborated further below.

GLOBAL SOFTWARE DEVELOPMENT

In recent years we have witnessed the globalization of many organizations. Consequently, globally distributed collaborations and virtual teams have become increasingly common (Sarker & Sahay, 2004). According to Carmel (1999), distributed development projects are projects consisting of teams working together to accomplish project goals from different geographic locations. More than a decade ago, the desire for lower costs and the possibility to capitalize on a global resource pool were the main drivers for companies experimenting with GSD (Herbsleb & Moitra, 2001). Although these remain important, other factors have only accelerated the trend. For example, there are business advantages of proximity to the market, including knowledge of customers as well as the good will engendered by local investment. Second, there is the possibility for quick formation of virtual corporations to exploit market opportunities. Third, there is 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 in which engineers, managers, and executives face numerous challenges on many levels — from the technical to the social and cultural (Herbsleb & Moitra, 2001).

Traditionally, literature on GSD has focused on technical aspects (Kotlarsky & Oshri, 2005), and previous research suggests that proper application of collaborative technologies is crucial for successful software development (Carmel, 1999). A related stream of studies has focused on issues relating to the dispersion of work and the constraints associated with this. In these studies, constraints such as temporal distance, geographical distance, and sociocultural distance are identified. Although these distances increase the scope of organizational operation (Sahay, 2003) and facilitate a broader skill and product knowledge base (Baheti et al., 2002), there is little doubt that each of them challenge project processes such as communication, coordination, and control (Herbsleb & Mockus, 2003; Damian, 2002).

Temporal distance is a measure of the dislocation in time experienced by two actors wishing to interact; geographical distance is a

TABLE 1 Impacts of Three Distance Dimensions on GSD Processes

Impact on GSD process	Distance Dimension: Temporal Distance	Geographical Distance	Sociocultural Distance
Communication	+ Improved record of communications	+ Potential for closer proximity to market and utilization of remote skilled workforces	+ Potential for stimulating innovation and sharing best practice
	– Reduced opportunities for synchronous communication	– Increased cost and logistics of holding face-to-face meetings	– Risk for misunderstandings
Coordination	+ Decreased coordination needs due to division of labor	+ Increase in size and skills of labor pool can offer more flexible coordination planning	+ Access to rich skill set and various practices
	– Increased coordination costs	– Reduced informal contact can lead to lack of task awareness	– Inconsistency in work practices can impinge on effective coordination, as can reduced cooperation through misunderstandings
Control	+ Opportunities for 'round-the-clock development	+ Communication channels often leave an audit trail	+ Access to rich skill set and authority
	– Management of project artifacts may be subject to delays	– Difficult to convey vision and strategy	– Different perceptions of authority/hierarchy can undermine morale

Note: + (plus sign) indicates an opportunity; – (minus sign) indicates a challenge.
Source: After Ågerfalk et al. (2005)

measure of the effort required for one actor to visit another; and sociocultural distance is a measure of an actor's understanding of another actor's values and normative practices (Ågerfalk et al., 2005). According to Ågerfalk et al. (2005), Table 1 provides an overview of opportunities and challenges in GSD by relating the dimensions of distance to the software development processes of communication, coordination, and control.

AGILE SOFTWARE DEVELOPMENT

Unlike plan-based methods, agile methods deal with unpredictability by relying on people and their creativity rather than formalized processes (Cockburn, 2002). Agile methods are characterized by short, iterative cycles of development driven by product features, periods of reflection and introspection, collaborative decision making, incorporation of rapid feedback, and continuous integration of code changes into the system under development (Highsmith, 2002). Thus, agile methods operate on the principle of "just enough method" because they seek to avoid prescribing cumbersome and time-consuming processes that add little value to the software product and elongate the development process.

Agile methods are explicitly value based. Whereas most traditional methods are unclear about their underlying philosophy, agile methods are characterized by their adherence to a set of agile values (Lindstrom & Jeffries, 2004). The change in emphasis from the traditional approaches is summarized in the "Agile Manifesto" (see Table 2).

Many different agile methods are in use (Abrahamsson et al., 2003; Lindstrom & Jeffries, 2004; Erickson et al., 2005). The two most well known are XP and Scrum. XP is basically a collection of well-known software engineering practices taken to their extreme (Beck, 2000). Scrum is a simple, low-overhead process for managing and tracking software development (Schwaber & Beedle, 2002). The two methods are highly compatible in that XP provides specific engineering techniques and Scrum essentially works as a wrapper for such techniques (Fitzgerald et al., 2006).

Agile Methods in GSD

Despite evidence of successful agile software development, its application in GSD is still to gain momentum. Here, distribution of project members makes many agile practices difficult to apply (Maurer & Martel, 2002; Kirscher et al.,

TABLE 2 Agile Manifesto

VALUES

Individuals and interactions over processes and tools
Working software over comprehensive documentation
Customer collaboration over contract negotiation
Responding to change over following a plan

PRINCIPLES

Our highest priority is to satisfy the customer through early and continuous delivery of valuable software.
Welcome changing requirements, even late in development. Agile processes harness change for the customer's competitive advantage.
Deliver working software frequently, from a couple of weeks to a couple of months, with a preference to the shorter timescale.
Business people and developers must work together daily throughout the project.
Build projects around motivated individuals. Give them the environment and support they need and trust them to get the job done.
The most efficient and effective method of conveying information to and within a development team is face-to-face conversation.
Working software is the primary measure of progress.
Agile processes promote sustainable development. The sponsors, developers, and users should be able to maintain a constant pace indefinitely.
Continuous attention to technical excellence and good design enhances agility.
Simplicity — the art of maximizing the amount of work not done — is essential.
The best architectures, requirements, and designs emerge from self-organizing teams.
At regular intervals, the team reflects on how to become more effective, then tunes and adjusts its behavior accordingly.

Source: Adapted from www.agilemanifesto.org

2001). This is mainly because one of the key requirements of agile methods is effective communication, and achieving this would require team members to be collocated. Still, physical collocation of developers is not always feasible, and therefore different solutions of how to apply agile methods in GSD have been suggested.

For example, Kirscher et al. (2001) recommend Distributed eXtreme Programming (DXP), which they suggest addresses all aspects of XP, although to varying degrees. In DXP, eight of the XP practices (small releases, metaphor, simple design, testing, refactoring, collective ownership, 40-hour week, and coding standards) are seen as independent of the locality of the team and thus are practices that can be applied also in GSD. The remaining four practices (i.e., planning game, pair programming, continuous integration, and on-site customers) are identified as dependent on collocated team members and thus require alternative solutions to work in GSD. With DXP, remote team members can be integrated into the development process; the authors suggest that DXP is a valuable extension to traditional XP.

In a similar vein, Ngo-The et al. (2005) discuss the use of XP to increase efficiency in communication when outsourcing software development projects. They emphasize that a major benefit of XP is that it can be deployed partially and that if some practice is not convincing, it does not prevent the application of the overall method. Interestingly, this is contrary to Beck's original claim that XP has to be used in its entirety due to the synergistic relationships between core practices. In their conclusions, Ngo-The et al. (2005) report on a number of positive experiences in applying XP in GSD. For example, they recognize a decrease in overtime, high morale among team members, and strong customer commitment.

As suggested in these studies, many benefits derive from using agile methods in GSD. However, future research is needed to better understand the benefits of XP (Ngo-The et al., 2005) and DXP (Kirscher et al., 2001). We believe that there is more to learn from existing practice about how to apply agile methods in GSD.

THIS STUDY

Previous research has identified both challenges and opportunities with increased distance in software development. Interestingly, most opportunities are found on the business level, whereas most challenges are introduced at the level of development practice. Distance has been identified as a major challenge to the use of agile methods. Agile methods, on the other

TABLE 3 Research Activities from March 2002–August 2005

Date	Research Activity
Mar 2002	Seminar and workshop at university on the topic of agile methods
Mar 2002–Apr 2003	Interviews at company sites E-mail survey on use of agile methods at Intel
May 2003	Seminar and workshop at Intel
Jun 2003	Seminar and workshop at university: "Silver Bullets/Lead Balloons: Software Solutions and How They Really Work" (http://www.b4step.ul.ie/db/dir/alt_page.php?d=events&item=86)
Jun 2003–Jan 2004	Interviews at company sites
Feb 2004	Seminar and workshop at university: "Globally-Distributed Software Development: Software Solutions and How They Really Work" (http://www.b4step.ul.ie/GDSD/)
Mar 2004–Nov 2004	Interviews at company sites
Dec 2004	Workshop at university: "Agile Approaches for Distributed Development"
Jan 2005–May 2005	Interviews at company sites
Jun 2005	Workshop at university: "A Framework to Analyze Global Software Development"
Jul 2005–Aug 2005	Interviews at company sites

hand, have the potential to improve communication and, as a result, reduce coordination and control overhead. This study therefore seeks to understand whether agile methods can be successfully used in a GSD context. More specifically, *can agile methods be used to reduce the negative influence of distance on communication, coordination, and control in a GSD context?*

Our data collection focuses on the experiences of software development teams in two global companies with headquarters in the United States: Intel and Hewlett Packard (HP). Both companies have development teams in Ireland who coordinate with other remote colleagues in, for example, India, Poland, China, and Malaysia. The interviews reported on here were conducted at the Irish company sites. Intel has been deploying a range of agile methods over the past five years: XP is used for the technical engineering aspects of software development and Scrum for project planning and tracking. HP has been deploying a range of agile methods over the past years: different practices of XP, such as pair programming. The research activities for this study are reported in Table 3; a fuller discussion of the research methods can be found in the appendix.

FINDINGS

This section presents the results from the interviews. First, we identify GSD challenges in relation to each distance factor. Second, we explore the use of agile methods in both companies.

Challenges in Global Software Development

Temporal Distance. According to our respondents, temporal distance is challenging when it comes to controlling projects that constitute different sites:

> Time zone distance is the biggest problem when organizing the different parties in projects. (Project manager, Intel)

Besides control issues, temporal distance challenges communication and coordination within and between teams. In particular, response delays are seen as frustrating:

> I received e-mails this morning from a conversation that kicked off after I left yesterday. Sometimes conversation jumps ahead, and you fall a bit behind. (Architect, HP)

> It is frustrating … sometimes there is a lag of a day in responses. You send an e-mail today and you get one back tomorrow. … People go out of their way to communicate late at night, depending on the intensity of the project at that point in time. It's okay to do that for a while, but it's hard to sustain it, that's the problem. There's burnout of people. (Manager, HP)

Establishing a feeling of trust and belonging, i.e., "teamness," within the teams can be difficult.

Clearly, communication and coordination are challenged by temporal distance. As recognized by our respondents, the main problem is the delay in responses. Also, project control is more difficult when overlap in time is reduced.

Geographical Distance. Although there is an advantage of "intellectual horsepower" (i.e., the ability to recruit the cream-of-the-crop students from top universities in countries where education and employment is more competitive), the companies experience problems related to geographical distance. Establishing a feeling of trust and belonging, i.e., "teamness," within the teams can be difficult:

> The feeling is that we remain two different teams. However, there is a good cross-site relationship at management level and between certain peers … in general, the developers have not met each other. (Software developer/team leader, Intel)

It seems that good cross-site relationships exist at higher levels within the organization, but the software developers at the coalface (doing the actual development) seldom meet. Although management expresses a desire to have developers meet, it is not always achieved. However, respondents in both companies agree that the opportunity to meet depends on the specific project — and the specific phase of the project.

> The degree of communication depends on the phase of the project. For example, during integration, when things are put together, there can be unexpected behavior. Usually, we fly people over in critical phases. Mostly, travel happens at front-end and back-end of projects. (Manager, Intel)

Despite communication technologies such as e-mail, IM, NetMeeting applications, virtual classroom applications, and the phone, there might still be the feeling of being two different teams.

Sociocultural Distance. Sociocultural distance is a complex dimension involving organizational culture, national culture and language, politics, and individual motivations and work ethics. Our study shows that language can be a barrier in many projects:

> We often experience minor language problems, especially when vocabulary is limited to technical subjects … even going out at night with them [nonnative English speakers], conversation can revert back to technical subjects because of their limited [English] vocabulary. (Software developer, Intel)
>
> Language … it's a really, really difficult problem. (Project manager, HP)

Besides vocabulary, interpretation and meaning can be different. Both managers and project participants experience this:

> Difficulties can arise in countries where it is considered impolite in saying "no" even when "yes" would be an inappropriate answer. I have heard people saying "yes — no problem, we will have it done by the weekend" and then 3–4 months later it is still not done and some of the developers might already have left the project. … I think it is due to pride — they'll obey when asked, without saying they can't do it within the given timeframe. (Project manager, Intel)
>
> The general understanding is not too bad. It is often the more subtle ones [cultural issues] that can trip you up the most. They're the ones that slip through. You interpret it one way, and they interpret it the other way. That gets worse the further away from native English speaking people you go. (Architect, HP)

In addition, cultural, political, and religious differences can challenge project work. Both companies in our study have experienced this:

> There are a lot of political and religious diversity. … if any element of that came into everyday work it could just blow everything apart and create lot of tensions … . (Architect, HP)
>
> When you have language difficulties initially causing confusion, I think cultural differences can actually drive further awkward situations, and it snowballs. … (Architect, HP)

Clearly, sociocultural distance is a complex dimension. Both companies express misunderstandings and confusion as a result of language and interpretation problems. This has implications for communication, coordination, and control and makes it a real challenge to create mutual understanding within and between teams.

Overall, the practice of pair programming was perceived as having a number of significant advantages.

Agile Methods Used

At Intel, a range of agile methods has been used over the past five years. In particular, parts of XP are used for the technical engineering aspects of software development and parts of Scrum for the project planning and tracking aspects. Both methods were introduced at a grassroots level, as optional techniques, and their adoption has grown over time. Only 6 of the 12 XP practices have been implemented: pair programming, testing, refactoring, simple design, coding standards, and collective ownership. As for the other practices, these were not found suitable by the companies in our study. For example, the practice of a 40-hour week was not achievable in a GSD environment, in which workers collaborate across different time zones. Also, the concept of on-site customers was difficult to implement in a development environment, where the early conceptual stages have no specific customers. Similar to Intel, HP uses parts of XP, particularly the practice of pair programming. Despite temporal distance, this practice works satisfactorily:

> At the moment I have a pair with one guy in Fort Collins and the other guy in Brussels. That is an eight-hour time difference. They have both shifted their working day, so they have a six-hour overlap per day. (Team manager, HP)

Overall, the practice of pair programming was perceived as having a number of significant advantages. Both companies found code quality high, and at Intel, the feeling was also that this quality was achieved earlier. As mentioned by one of the managers, one reason for this might be that the developers did not get stuck wondering what to do next. If one person was unsure, the other probably did know, and even though there was sometimes a delay in response, developers seemed eager to be flexible to create as much overlap in time as possible. Furthermore, having pairs proved useful for testing and debugging because someone with a fresh viewpoint could spot mistakes that were not obvious to the pair partner. The practice also ensured that more than one developer gained a deep understanding of the design and code, thus facilitating collective ownership of code. At Intel, this was perceived as important because changes in team composition were common. It also provided greater flexibility in relation to maintenance of code.

Despite these advantages, however, some aspects were problematic. For example, pair programming was found unsuitable for simple, well-understood problems and when doing small changes. Here, collocated developers tended to get frustrated, something that got only worse in GSD, where temporal distance makes response slow:

> If you're trying to progress something very quickly, there can be an issue with the time zones. … If there's any need for me to ask something or find an update, I can't really get hold of him [American colleague] until 3pm my time — maybe two o'clock at the earliest. (Team leader, HP)

Still, XP was perceived as having major benefits. At Intel, a "test-code development strategy" was implemented, something that helped developers get a better understanding of what functionality was required from a client point of view. Also, the practice of simple design was used. In this case, design was done on a whiteboard and the design document emerged in parallel with the code implementation. Interestingly, the XP practice of simple design is very similar to the concept of simplified planning in Scrum. This pre-game phase was initially piloted by one team at Intel and has grown to the extent that it is now used by almost all teams. As commented on by one of the managers, the reason for its popularity is probably the simplicity and the low-tech techniques that can be applied during project planning. For example, the daily Scrum meeting took place around a board covered with yellow Post-it® notes. The team recorded tasks for a 24-hour period on these notes. During the Scrum meeting the team moved completed tasks into the "done" area and the group achieved a shared group visualization of project progress. In this way, Scrum was made visible in the organization and curiosity from other groups helped the spread of the practice.

Although it created some overhead, this practice was easily converted to include the distributed development teams. In that setting, notes were published on a Web page and, as with the collocated teams, distributed team members could get a shared visualization of project progress. As a result, more distributed teams have commenced using shared spreadsheet and networked meeting software — and the technique has facilitated coordination and control within the distributed teams.

Other XP practices that were explored in our study were refactoring and coding standards. In relation to refactoring — i.e., restructuring of systems to improve nonfunctional

XP was found useful for the more technical and coding aspects of GSD projects, whereas Scrum practices provided a good framework for GSD planning and tracking.

aspects (e.g., duplication of code, simplicity, and flexibility) — our participants found it useful when this was done early in the project. In this way, it eliminated bugs that would have taken up a lot of debugging time in later stages of the project. In relation to coding standards, they were defined early in the projects. For example, a C-coding standard was defined at Intel very early in the project and was referred to during coding and code inspections. However, coding standards were already a strong feature in both companies and therefore cannot be thought of as unique to the application of XP or other agile methods.

Overall, several XP and Scrum practices were found beneficial for GSD and particular practices could be used to reduce distance. Specifically, XP was found useful for the more technical and coding aspects of GSD projects, whereas Scrum practices provided a good framework for GSD planning and tracking.

DISCUSSION OF FINDINGS

In accordance with the literature in the field, we have chosen to focus on temporal, geographical, and sociocultural distance and the way in which these distances challenge GSD practice. For a number of years, the International Workshop on Global Software Development has highlighted these distances and the impact they have on processes such as communication, coordination, and control (see, e.g., Damian et al., 2003). A number of other authors have also focused on one or more of these distances to see how they affect project processes (Carmel & Agarwal, 2001; Evaristo et al., 2004; Malone & Crowston, 1994).

Here, *communication* is seen as the formal and informal exchange of information between people. Communication is an essential process in all software development (Curtis et al., 1988), but it becomes even more critical in GSD. This is due to the fact that the distributed environment changes the communication context away from the "ideal" face-to-face setting (Clarke, 1996) into a technology-mediated and thus potentially more complex one (Ågerfalk, 2004). Furthermore, *coordination* is seen as the act of integrating each task with each organizational unit (Carmel & Agarwal, 2001). As found in our study, all software development requires coordination, but GSD increases this need because activities are distributed. Finally, *control* is seen as the process of adhering to goals, policies, standards, or quality levels (Carmel & Agarwal, 2001). The control process concerns the management and reporting mechanisms to ensure that development is progressing. Thus, control relates to project management and hence to the formalized structures required to ensure development of software in time, on budget, and of desired quality.

In our study, we see that the main challenges of GSD lie in the complexity of maintaining good communication, coordination, and control when teams are dispersed. In relation to temporal distance, our research reveals difficulties in achieving overlap in time between different sites. One disadvantage of being separated by temporal distance is that the number of overlapping hours during a workday is reduced; team members have to be flexible to achieve overlap with remote colleagues. As noted by one of the managers, the lag in response time brings with it a feeling of "being behind" and "missing out," which makes people frustrated. Our study reveals that limited overlap with colleagues — and delay of responses — make people lose track of the work process, something that can pose severe problems in distributed, yet time-critical, work. However, with the introduction of agile methods, especially the XP practice of pair programming, the issue of creating time overlap has become less problematic. Both Intel and HP found pair programming beneficial for encouraging commitment among developers. As mentioned by one of the managers, people were flexible and, even though there was a delay in response, individual developers tried hard to spend as much time as possible with the distributed pair programmer. Because XP was introduced at the grassroots level, this commitment was not enforced but rather resulted from individual interest. Having distributed pair programmers increased individual responsibility to create overlap in time and hence to reduce temporal distance and the negative influence it might have on team communication, coordination, and control.

A major challenge in relation to geographical distance is how to create a feeling of "teamness" among distributed project members. Previous research on distributed organizations shows that people at different sites are less likely to perceive themselves as part of the same team (Kotlarsky & Oshri, 2005). Our study shows that the Scrum planning practice can still be useful. By publishing Post-it notes on a Web page, the distributed team members could easily participate in the process. The overall feeling was that team communication

TABLE 4 Agile Practices and GSD Benefits

Agile Practices	Benefits
XP pair programming	High code quality (and code quality earlier in projects) Fresh viewpoint in testing/debugging Facilitates collective ownership of code
XP simple design	Design document in parallel with code implementation
XP refactoring	Early elimination of bugs
XP coding standard	Consistency in coding/code inspection
Scrum simple planning	Low-tech techniques Shared visualization of project activities

TABLE 5 Agile Practices, Benefits, and Impacts on Distance in GSD

Agile Practices	Benefits	GSD Distance
XP pair programming	Help increase time overlap	Reduce temporal distance
Scrum simple planning	Help increase "teamness"	Reduce geographical distance
XP pair programming and Scrum pre-game phase	Help increase mutual understanding and collaboration within and between teams	Reduce sociocultural distance

and coordination significantly improved and that the feeling of "teamness" among geographically distributed teams was improved.

In relation to sociocultural distance, the most widely experienced difficulty pertains to language and interpretation. Employees from both companies mention language problems as the primary reason for misunderstandings. It has been argued that informal communication plays a critical role in coordination activities for collocated software development, especially when size and complexity of development increase (Kotlarsky & Oshri, 2005); informal conversation allows team members to develop working relationships and allows a better flow of information (Herbsleb & Mockus, 2003). Although the need for informal conversation in GSD is extensive, people find it far more difficult to identify distant colleagues and communicate effectively with them (Herbsleb & Mockus, 2003).

We believe the agile practices mentioned here were useful for increasing communication. As discussed above, the practice of pair programming improved individual commitment for enhancing mutual understanding between team members. Agile practices, such as the Scrum pre-game phase, made project planning and tracking processes a collaborative activity, and the agile focus on individuals and interaction (over processes and tools) encouraged people with different cultural backgrounds to communicate. Although our respondents recognized the difficulty in overcoming sociocultural distance and did not believe that any single methodological approach would solve this problem, the developer-centric qualities of agile methods led to considerable benefits at the people and participation levels.

Table 4 provides a summary of the agile practices in our study and the benefits of using these in GSD. Table 5 provides a summary of the particular practices we found useful for reducing temporal, geographical, and sociocultural distance and hence illustrates the potential for agile methods to reduce the negative influences of these distance factors in GSD.

CONCLUSION

Temporal, geographical, and sociocultural distances impose several challenges to GSD practice, such as difficulty in creating overlap in time, difficulty in creating "teamness," and difficulty in creating mutual understanding between people with different sociocultural backgrounds. Although we believe that no single methodological approach may easily solve these challenges, the two companies we studied report agile practices to be valuable in re-

ducing some of them. In particular, XP and Scrum practices were found useful for improving communication, coordination, and control within GSD teams. Although they did not use a complete set of agile methods, the companies found that drawing from a palette of different methods helped them manage the complex task of GSD. Contrary to previous research, our findings suggest that agile methods may be more amenable to GSD than has been previously reported.

ACKNOWLEDGMENTS

This research was supported by grants from Science Foundation Ireland for the B4-STEP (Building a Bi-directional Bridge between Software ThEory and Practice) and Lero (the Irish Software Engineering Research Centre projects), and from the EU to the CALIBRE project. The authors would also like to thank all the interviewees at Intel and HP. Thanks are also due to the anonymous reviewers and the special issue guest editor for their valuable comments on the manuscript.

APPENDIX

Given that little research to date has been conducted on the use of agile methods in a GSD context, this study was concerned with achieving an increased understanding of this phenomenon. Bearing this in mind, an interpretivist approach, which sought to develop inductively a richer understanding based on in-depth case study analysis, was deemed appropriate (Yin, 1994; Walsham, 1993).

The case study data reported here covers a three-year period, beginning in March 2002, and is ongoing. In March 2002, the first phase of the research began with a workshop seminar on the topic (i.e., GSD), comprising researchers and industry practitioners. This workshop was followed by a series of interviews and site visits. The combination of on-site and university-hosted seminars and workshops has been greatly facilitated by the fact that the industry sites and the university are located less than one hour's drive from each other. The workshops have been hands on, with committed participation by both researchers and practitioners. Also, the seminars have involved leading researchers in this area worldwide (including, for example, Abrahamsson, Herblseb, Parnas, Raffo, and Succi).

The workshops and seminars have been complemented with qualitative interviews with managers and software developers at both companies. These interviews helped us gain a deeper understanding for the GSD context. Data collection involved a series of formal and informal personal interviews with the project managers and staff responsible for agile development. The interviews were generally of one- to two-hour duration, and informal interviews were used to clarify and refine issues as they emerged. In total, 20 interviews were conducted at the company sites. Interviews were transcribed according to the agile methods used: three distance factors (temporal, geographical, sociocultural) and three development practices (communication, coordination, control). Informal follow-up telephone interviews took place to clarify and refine emerging issues, and these emerging issues were also presented and discussed at the various workshops.

In terms of data analysis, a primarily qualitative grounded theory (GT) approach was adopted (cf. Corbin & Strauss, 1990; Miles & Huberman, 1994). A problem that has been identified in relation to qualitative research is that different individuals may interpret the same data in different ways (Kaplan & Duchon, 1988). This problem was addressed in two ways. First, the GT method of data analysis explicitly recognizes this problem of subjective data interpretation and, to address it, prescribes rigorous coding and memoing processes, which provide a traceable, documented justification of the process by which research conclusions were reached, thereby providing an audit trail of the process (Guba, 1981). Second, we used a *venting* method, a process whereby results and interpretations are discussed with professional colleagues (Goetz & LeCompte, 1984). The findings were presented and discussed with colleagues and expert practitioners in detail at the various workshops. ▲

References

Abrahamsson, P., Warsta, J., Siponen, M., and Ronkainen, J. (2003). New Directions on Agile Methods: a comparative analysis. In Proceedings of the 25th International Conference on Software Engineering, Portland, Oregon, pp. 244–254.

Ågerfalk, P. J. (2004). Investigating Actability Dimensions: A Language/Action Perspective on Criteria for Information Systems Evaluation, Interacting with Computers, 16(5), pp. 957–988.

Ågerfalk, P. J., Fitzgerald, B., Holmström, H., Lings, B., Lundell, B., and Ó. Conchúir, E. (2005). A Framework for Considering Opportunities and Threats in Distributed Software Development,

In Proceedings of the International Workshop on Distributed Software Development (DiSD 2005), Paris, 29 August 2005: Austrian Computer Society, pp. 47–61.

Baheti, P., Gehringer, E., and Stotts, D. (2002). Exploring the Efficacy of Distributed Pair Programming. *In Proceedings Extreme Programming and Agile Methods — XP/Agile Universe*, Chicago, USA, August 4–7, 2002.

Beck, K (2000). *Extreme Programming Explained: Embrace Change*, Addison-Wesley, Reading.

Carmel, E. (1999). *Global Teams: Collaborating Across Borders and Time Zones*. Prentice-Hall, Upper Saddle River: NJ.

Carmel, E., and Agarwal, R. (2001). Tactical approaches for alleviating distance in global software development, *IEEE Software*, Vol. 18, No. 2, pp. 22–29.

Clarke, H. H. (1996). *Using Language*, Cambridge University Press, Cambridge.

Cockburn, A. (2002). *Agile Software Development*. Boston: Addison-Wesley.

Conboy, K., and Fitzgerald, B. (2004). Toward a Conceptual Framework of Agile Methods: A Study of Agility in Different Disciplines. *In Proceedings of the ACM Workshop on Interdisciplinary Software Engineering Research (WISER)*, November 5, Newport Beach, CA, USA.

Corbin, J. and Strauss, A. (1990) *Basics of Qualitative Research: Grounded Theory Procedures and Techniques*, Sage, California

Curtis, B., Krasner, H., and Iscoe, N. (1988). A Field Study of the Software Design Process for Large Systems, *Communications of the ACM*, Vol. 31, No. 11, pp. 1268–1287.

Damian, D. (2002). Workshop on Global Software Development. *In Proceedings of International Conference on Software Engineering (ICSE)*, Orlando, Florida, USA, May 19–25, 2002.

Damian, D., Lanubile, F., and Oppenheimer, H. L. (2003). Addressing the Challenges of Software Industry Globalization: The Workshop on Global Software Development, In *Proceedings 25th International Conference on Software Engineering*, IEEE Computer Society, Los Alamitos, pp. 793–794.

Erickson, J., Lyytinen, K., and Siau, K. (2005). Agile Modeling, Agile Software Development, and Extreme Programming: The State of Research, *Journal of Database Management,* 16(4), pp. 88–100.

Evaristo, J. R., Scudder, R., Desouza, K. C., and Sato, O. (2004). A dimensional analysis of geographically distributed project teams: a case study, *Journal of Engineering and Technology Management*, Vol. 21, No. 3, pp. 175–189.

Fitzgerald, B., Harnett, G., and Conboy, K. (2006). Customizing Agile Methods to Software Practices. *European Journal of Information Systems, Vol 15, No. 2*.

Fowler, M., and Highsmith, J. (2001). Agile methodologists agree on something. *Software Development,* vol. 9, pp. 28–32.

Fowler, M., and Highsmith, J. (2001). The agile manifesto. http://www.agilemanifesto.org/

Goetz, J., and LeCompte, D. (1984). *Ethnography and Qualitative Design in Educational Research*, Academic Press, Orlando.

Guba, E. (1981). Criteria for assessing the trustworthiness of naturalistic inquiries. *Educational Communication and Technology*, 29, 75–92.

Herbsleb, J. D., and Mockus, A. (2003). An Empirical Study of Speed and Communication in Globally Distributed Software Development, *IEEE Transactions on Software Engineering*, Vol. 29, No. 6, pp. 481–494.

Herbsleb, J., and Moitra, D. (2001). Global software development. *IEEE Software*, March/April.

Highsmith, J. (2002). The great methodologies debate: Part 2, *Cutter IT Journal*, vol. 5.

Highsmith, J., and Cockburn, A. (2001). Agile Software Development: The Business of Innovation. *Computer,* vol. 34, pp. 120–122.

Holtz, H., and Maurer, F. (2002). Knowledge Management Support for Distributed Agile Processes. *In Proceedings of the Workshop on Learning Software Organizations (LSO),* August 6, Chicago, USA.

Kaplan, B. and Duchon, D. (1988). Combining qualitative and quantitative methods in IS research: a case study, *MIS Quarterly*, 12, 4, 571–587.

Kirscher, M., Jain, P., Corsaro, A., and Levine, D. (2001). Distributed eXtreme Programming. *In Proceedings of the International Conference on eXtreme Programming and Flexible Processes in Software Engineering*, May 20–23, Sardinia, Italy.

Kotlarsky, J., and Oshri, I. (2005). Social ties, knowledge sharing and successful collaboration in globally distributed system development projects, *European Journal of Information Systems*, 14, pp. 37–48.

Lee, A. S., and Baskerville, R. L. (2003). Generalizing Generalizability in Information Systems Research. *Information Systems Research*, 14 (3), 221–243

Lindstrom, L. and Jeffries, R. (2004) Extreme Programming and Agile Software Development Methodologies, *Information Systems Management,* 24(3), pp. 41–60.

Malone, T. W., and Crowston, K. (1994). The interdisciplinary study of coordination, *ACM Computing Surveys*, Vol. 26, No. 1, pp. 87–119.

Maurer, F., and Martel, S. (2002). On the Productivity of Agile Software Practices: An Industrial Case Study. *In Proceedings of the International Workshop on Global Software Development*, May 21, Orlando, FL, USA.

McCauley, R. (2001). Agile Development Methods Poised to Upset Status Quo. *SIGCSE Bulletin*, vol. 33, pp. 14–15.

Miles, M., and Huberman, A. (1994) *Qualitative Data Analysis: A Sourcebook of New Methods*, 2nd Ed. Sage, Beverley Hills.

Ngo-The, A., Hoang, K., Nguyen, T., and Mai, N. (2005). Extreme Programming in Distributed Software Development: A Case Study. *In Proceedings of International Workshop on Distributed Software Development,* August 29, Paris.

Perry, D. E., Staudenmeyer, N. A., and Votta, L. G. (1994). People, Organizations and Process Improvement, *IEEE Software*, vol. 11, No. 4, JULY/August, pp. 36–45.

Sahay, S. (2003). Global software alliances: the challenge of "standardization." *Scandinavian Journal of Information Systems,* Vol. 15, pp. 3–21.

Sarker, S., and Sahay, S. (2004). Implications of space and time for distributed work: an interpretive study of US-Norwegian systems development teams*, European Journal of Information Systems*, 13, pp. 3–20.

Schwaber, K., and Beedle, M. (2002). *Agile Software Development with Scrum.* Upper Saddle River, NJ: Prentice-Hall.

Turk, D., France, R., and Rumpe, B. (2005). Assumptions Underlying Agile Software-Development Processes, *Journal of Database Management,* 16(4), pp. 62–87.

Walsham, G. (1993). *Interpreting Information Systems in Organizations*, Wiley, UK.

Yin, R. (1994). *Case Study Research: Design and Methods*, 2nd Ed., Sage Publications, California.