



Business Collaboration

Table of
contents:

→ HTML → PDF This article: → HTML → PDF DOI: 10.1147/sj.454.0773

Copyright info

Following the sun: Case studies in global software development

by J. J.
Treinen

and
S. L.
Miller-
Frost

Advances in global network connectivity over the past 10 years have significantly reduced the effects that physical separation has on geographically distributed development teams. It is increasingly clear that time zones, rather than physical distances, are becoming the most significant factor which separates potential collaborators. Collaboration across time zones often involves modifications to the typical work day, with remote team members collaborating either very late at night or extremely early in the morning. Though this presents obvious problems, a natural question is whether it is possible to exploit this diversity as a competitive differentiator, instead of treating time zones as an impediment to productivity. In two case studies, we examine whether it is possible to create a development environment in which tasks can “follow the sun,” allowing teams to work during extended local business hours and assign or hand off tasks at the end of their day to teams that are just starting their day, effectively yielding a 24-hour development clock. We examine the factors that influenced the success or failure of the respective projects and conclude with a discussion of best practices for using this approach successfully.

Introduction

Global telecommunications networks have matured in the past decade, providing a vehicle for relatively inexpensive communication between teams that are separated by great physical distances. This new reality provides an opportunity for teams to collaborate in a much more effective manner than ever before. Because the teams are now able to communicate effectively and economically, they can work together on common goals with groups who are located not only in different regions of their native countries but also across the entire globe. Many new tools that enable this type of collaboration have emerged, and although these tools greatly reduce the effects of physical distance, a new separating factor has been exposed; that presented by significant differences in time zone.

For example, in order to communicate in a form other than e-mail, leaders of teams (if not the whole team itself) are often forced to work well outside of their normal working hours. This is inconvenient at best and completely unworkable at worst. Although problems such as this exist, it may be possible to exploit the phenomenon of global software development in a positive manner, giving the distributed team a competitive advantage.

In this paper, in the context of two globally distributed software development

projects, we examine whether it is possible to create a development environment in which tasks can be assigned in such a manner that work can “follow the sun.” We examine the effects of moving work across the globe so that each team is working during their local normal business hours and assigning or handing off tasks at the end of their day to teams that are beginning their normal work day, effectively yielding a 24-hour development clock. There are advantages and disadvantages to using this method, and we explore these in the context of two case studies. In the first case study, it was possible to use this approach successfully; in the second, a series of projects met with greater challenges. We examine, in depth, which factors were most influential regarding the success of the respective efforts and which factors limited it. Because the workforce is becoming increasingly global in nature, we conclude with a discussion of best practices for using this global workforce to its maximum potential, thus creating higher value for customers and increasing the ability to compete in this new marketplace.

Related work

Although little has been published regarding a “follow the sun” approach to global software development, the topic of global software-development practices has been the subject of much discussion in recent years. The general consensus has been that globalization introduces a great deal of complexity to an already complex process. Specifically, the use of globally distributed development teams has been found to extend time lines and to require more development resources in order to complete a project of the same complexity when compared to projects staffed with collocated teams.^{1,2} A major factor that has been repeatedly discussed is the loss of informal communication channels, which occur naturally in collocated teams.^{1,3-7} The loss of these channels, which are often taken for granted when all team members are in the same location, has been found to make it increasingly difficult to find experts who can answer questions,¹ and can lead to a loss of trust between team members in different locations.^{3,8} In addition to these social challenges, the effect that the loss of communication has on the requirements management process is discussed in Reference 9, whose authors are among the few to note that time zone diversity can be viewed as an advantage.

Suggestions for mitigating these problems are discussed in Reference 3, whose authors propose the inclusion of face-to-face kickoff meetings, recurring face-to-face meetings among the team leaders when possible, increased cultural awareness training for the teams in general, and the implementation of tools that help to re-enable informal communications. A discussion on how to address communication issues from an organizational perspective is presented in References 9 and 10, and it is argued that for distributed teams in particular soft skills (such as those related to social interaction) might be even more critical than technical skills (such as those related to tool deployment).

Iterative development methodologies have been found to be helpful in mitigating some of the risks inherent to global software development, as they help build trust between distributed teams by forcing synchronization by requiring frequent deliverables.^{3,11} A case study evaluating this approach in terms of the “eXtreme” programming model is presented in Reference 12.

An interesting set of patterns and “anti-patterns” (i.e., methodologies that consistently produce unsuccessful projects) for effective global software development is proposed in Reference 13. The problem of globally distributed code control systems is discussed in Reference 14. A study of global software

development from an anthropological viewpoint is presented in Reference 15, whose authors agree with previous work⁹ that most failures are not due to tools, but to a web of social factors, not one of which can be blamed exclusively.

Case study: Business intelligence

This case study focuses on a project whose goal was to deploy a second instance of a custom-built business intelligence solution. The original solution was designed, developed, and deployed in the United States. The second instance was to be deployed at an IBM location in Australia. While some of the components of this project were commercially available software products, the middleware layer was comprised of custom-built application code which required significant generalization and rework before the second instance could be deployed.

The project was broken into four major phases. The first phase was the infrastructure set up at the data center in Sydney. The second phase consisted of creating the required installation scripts for the data warehouse and later building the warehouse on the Australian infrastructure. The third phase comprised the writing of the front-end Web page code. The fourth phase consisted of the design, test, and deployment of the middleware code used to collect the data from the source systems and populate the data warehouse so that the front-end Web pages could display the results.

The teams

In order to complete this project, we were required to merge two geographically distant development groups into one cohesive team. The local development team was based in Boulder, Colorado. The Boulder team also had a management sponsor and an executive sponsor who assisted with the resolution of any business-related issues. The remote team was distributed across three locations in Australia. The discussion in this paper is limited to the interactions between the teams as they occurred over international boundaries and does not discuss the dynamics between the three teams in Australia.

Challenges

Because this was a custom-built solution and not a packaged software product, there were no ready-made installation scripts. Much of the code was specifically tailored for the Boulder environment and as such, required significant modification before it would function properly in the two data centers simultaneously.

The two teams had never met each other face-to-face. When the project was first proposed to the sponsoring executives, the two teams, with no previous experience working together, had to present a unified front in order to secure funding. The lack of previous experience working together between the two teams and the resulting lack of mutual trust made promoting the business case difficult. When a development team is collocated, trust generally develops early in the project. Unfortunately, before we could secure funding for face-to-face meetings, we had to convince our sponsors that the project itself was worth doing. As such, the early phases of the project had to be completed before the two teams met in person. Funding for future phases was contingent upon successful completion of the proof of concept.

Our approach

After funding was secured for the initial phase, the lead developer of the Australian team flew to Boulder for two weeks of training on the various components of the solution. These two weeks served a dual purpose. The primary purpose, as perceived then, was to bring the development team in Australia up to speed on the inner workings of the code that would be deployed in Australia. The secondary goal was for this developer to meet the team in Boulder so that he could associate faces of team members with their names when questions arose. Building personal knowledge about the team and building mutual trust, however, was actually more important than resolving the technical issues. It was, in fact, this trust which allowed us to resolve future technical issues efficiently by conference calls that stretched halfway around the world. This foundation of trust resulted in increased efficiency.

After training was completed in Boulder, the lead project manager for the overall effort was assigned. With this team member in place, we outlined high-level milestones and specified the tasks that would allow us to meet those objectives. Because we only had funding for the initial proof of concept, it was critical that we achieve these milestones. Each milestone resulted in a review with the architectural control board in Australia and a request to approve funding for subsequent phases of the project. As such, failure to meet target dates would jeopardize the project, delaying it or halting it completely.

After we defined the milestones and the detailed task list for the first phase, the actual development began. Note that we did not attempt to define the complete end-to-end project plan before beginning work on the first phase. This was primarily because we were working within a compressed time frame. Furthermore, we were aware that the project was to be completed in the context of a dynamic service delivery environment. This type of environment differs from that of traditional software development projects in that the development iterations tend to be very compressed due to constantly changing circumstances driven by real-time changes in requirements. Thus, we initially concentrated on high-level tasks and on managing the details only of the phase that we were developing at the time.

Managing the milestones

Once the project plan was defined, we approached the actual development in the following manner. Every Monday at 2:00 p.m. or 4:00 p.m. Boulder time, we held a 1-hour project status meeting. The start time varied because the United States moved from Standard Time to Daylight Savings Time, and the clocks moved ahead one hour. At the same time, the exact opposite occurred in Australia, where the clocks move back one hour, resulting in a time zone difference of two hours. These meetings involved two main work products—the agenda, which was sent out to all participants the day before the meeting, and the meeting minutes, which were sent to each participant immediately following the meeting, detailing who was present and who had agreed to which action items, with deadlines to be completed before the next meeting. The minutes were the more important of the two, as they clearly defined who had signed up for which tasks in the next seven-day iteration. It was well known early in the project that if you agreed to a time frame for a specific task and took ownership of its completion, then you were expected to get it done on time, barring something which was completely out of your control. We worked together to ensure that the goals we set were realistic, and once everyone agreed that they were, failure to meet those goals on any individual's part almost never occurred.

The structure of each review meeting was as follows. The minutes of the previous week's meeting were reviewed, including a description of each action

item and its status. Any outstanding issues were discussed. The project plan was updated based on status reported, and any changes (additions or deletions) of tasks were made. High-level milestones were reviewed to ensure that we were on track. New actions were reviewed and assigned based on the requirements of the project plan. Once a month, a status call was held with the executive sponsors of the project. If an emergency arose that required executive intervention, appropriate escalation meetings were scheduled as needed.

There are a large number of similarities between the process just described and the Scrum¹⁶ approach to agile software development. Agile software development is a relatively new paradigm for the production of high quality software in shortened time frames. The goal is to be able to quickly respond to changes in requirements and to provide frequent releases of the product, ensuring that the highest priority requirements are met in a timely manner. Effectively, the process we employed was a watered-down version of the Scrum methodology, which we tailored to meet our specific needs. Rather than use a more traditional development approach in which requirements were gathered in detail at the onset, we chose to use this more flexible methodology. The reasons are twofold: first, we needed to show incremental results quickly to ensure that we could secure funding for future phases of the project; second, we were working in a very dynamic service delivery environment. Because of this, we had to be able to adapt to the changing needs of our customers, and at the same time build the actual system that would fulfill our commitments.

When this type of approach is used, discipline is critical. Our project maintained a delicate balance between keeping a good-natured, amicable atmosphere among team members and making them aware that failure to complete tasks in a timely manner could have very serious implications to the project. The only way we were able to meet aggressive time lines and simultaneously remain flexible was by requiring that once a set of tasks was agreed to, those tasks would be executed by the specified time. There were instances where delays did occur, but these were almost exclusively due to outside forces.

A high-performance project culture is important in any development effort, but even more so when teams are globally distributed for the following reason: If the team that is beginning their work day is expecting a set of inputs based on a certain set of tasks and those tasks are not completed, then that team effectively loses an entire day waiting for the other team to sleep, wake up, complete their tasks, and hand them over 24 hours behind schedule. In a more traditional environment, the team that is waiting for behind-schedule tasks is sleeping at the same time as the other team. Clearly, for work across time zones, timely completion of tasks is crucial.

About halfway through the project, we received some unpleasant news. We were forced to give up our lead project manager due to needs in other parts of the business. The loss of a key leader in a complex project such as ours introduces significant risks. Had the change not been handled appropriately, it could have sent the project into a severe decline from which we might never have recovered. The geographically dispersed teams had only just become accustomed to working together when the key leader change occurred. However, given the gradual manner in which the transition occurred, coupled with the maturity and experience of the rest of the team, we were able to get through it with relatively little disruption.

Coincidentally, at roughly the same time that this transition was happening, the lead architect and lead developer from the Boulder team were scheduled to

conduct two weeks of training for the development team in Australia. Aware of the change in leadership, we scheduled the project transfer activities when both project managers and the two key technical leaders from the Boulder team were present in Sydney. By actively adjusting our schedule, we enabled many of the transfer activities to be handled face to face.

The cultural factor

Given that the two teams came from opposite sides of the globe, culture was something that required special attention. Because both teams came from English-speaking countries, it might have been assumed that culture would play a minimal role in our success. In truth, cultural understanding was something that had to be practiced by both teams. For example, a comment intended as a small joke could potentially derail the project if it were misunderstood as an insult. In our case, culture did cause some minor friction in the early phases, but we were able to resolve this by being honest with each other and working out issues as they arose.

Another aspect of culture that is easily overlooked is how executive communications are handled in different parts of a globally diverse organization. We learned early in the project that it was important that the senior project managers and architects on each team have the responsibility for coordinating any communications regarding political or management-related issues locally. Both sets of team leaders were accustomed to managing executive communications with their respective management teams, but to avoid inadvertently sending a message which could be misinterpreted by the executives on the other team, it was beneficial to have each team read any communication before sending it out.

Results

After completing the initial proof-of-concept phase of the project, we were authorized to begin the full development and deployment of the global business intelligence solution. The same geographically dispersed team worked on the follow-on project and completed the roll-out of the project over a 12-month time period. The implementation was a success, and funds were made available for the implementation of a second project to expand on the work that we completed in the first 12 months. The solution is now functioning in a production steady state, with incremental functionality additions being made in much the same manner as the process described previously.

Lessons learned

Anyone who has tried to schedule a truly global meeting knows that it is impossible to find a time that is acceptable for all participants. If one divides the globe into only three geographical areas, as is commonly done, it is still quite difficult. If this is reduced to one more level of granularity and an attempt is made to accommodate each individual time zone in each of the three geographical areas, the problem becomes impossible. A simple way of illustrating this is to attempt to schedule a meeting using a Web site such as *timeanddate.com*.¹⁷

The main opportunity for project optimization came in the following form. The time difference between Boulder, Colorado and Canberra, Australia (which we will use because this is where the primary development team was located) is either 16 or 18 hours, depending on the time of year, as previously noted. Also, because Australia is so far ahead, our Sunday evening was Monday morning in

Australia, and when the team in Boulder was working on a Friday, the team in Australia was away on their weekend. Surprisingly, this was actually a benefit rather than a detriment—the team leaders from Boulder were able to get an early start on any issues for the upcoming week by signing on briefly on Sunday evenings and either reviewing e-mail or conducting brief online chats with the team in Australia. Because it was Monday morning in Australia, we were able to get this early information without causing any of the rest of the team, in either Boulder or Australia, to work outside of normal hours. If any pressing issues arose, we were able to prepare any special meetings or action plans and have them ready before the Boulder team arrived on Monday morning, local time. As such, we did not have to waste any of the Boulder team's time catching up with the day's work that had already been completed in Australia. Conversely, this gave the team in Australia an extra day to prepare any issues, and have the receiving team in Boulder work on them with effectively one extra day, because working on Friday (Boulder time), we could resolve these issues on what was a weekend day (Saturday) in Australia. Thus, we were effectively working to resolve any outstanding issues with the code during Australian weekend hours, while allowing the development team in Boulder to maintain a standard work week (Monday through Friday) in Boulder.

The net effect was that the project plan gave the appearance that the team was working six out of seven days, each of these six days being effectively 16 hours in length, without the exhaustion resulting from having any single individual work 96-hour weeks. The result was that each team member was really working a normal schedule (aside from some variable work hours on the part of the team leaders), but the project plan looked as though we were working roughly double that.

On occasion, an issue would arise which required attention by the team in the other hemisphere while they were asleep. When this happened, the issue was logged and an e-mail was sent by the initiating team to the support team, who dealt with it when they came online. Because the two teams were effectively opposites with respect to geography, once the support team received the e-mail and any crossover communications were handled, they could work to resolve the problem while the initiating team was sleeping. In most cases, when the initiating team came back online, they had an answer waiting for them. They no longer had to waste their working hours waiting for the support team to fix the problem; the issue was resolved while they slept.

Case study: Web application development

This section discusses a case study in which the follow-the-sun approach was not as successful as in the previous case study. We discuss several projects, all of which involved distributed Web application development efforts with the local team based in the United States and the remote team based in India. After a brief discussion of the projects, we highlight critical factors that contributed to the lack of success encountered with this model. As in the first case study, the projects involve the deployment of both packaged software products and significant custom coding efforts. Before the projects described here, the two teams had never met face to face, nor had they worked together.

The teams

The location of the two teams in this case study resulted in time zone differences of 10.5 to 13.5 hours, depending on the customer location. These projects involved larger teams than the previous case study: the local teams had from 20

to 40 team members; the remote teams were somewhat smaller.

The local teams were brought together many times specifically for the project. They did not necessarily know each other before the project, but they did have experience in the technologies that were being implemented, and, they did have the advantage of constant communication and face-to-face meetings to help build the trust in the local team very quickly.

Challenges

The remote and local teams were often working on different configurations of the development environment. This occurred primarily when the projects involved the customer's code configuration and deployment system. Many times the remote teams could not gain access to the customer's system and therefore could not store and configure their code directly on the customer's source control system.

The next challenge was related directly to the time zone issues; if the remote team had a question regarding the specifications for the code they were writing, they could not seek immediate resolution of these issues. As such, they often made assumptions that were based on their local preference and culture. These choices often clashed with the assumptions that were made by the team in the United States and in some cases, caused a great deal of rework. For example, because there was no specification calling for left justification of all Web content, the remote team assumed content was to be centered and developed their code based on this assumption.

In order to mitigate the impact of this challenge, the design specifications should have included a significant amount of low-level detail, but this level of detail was not provided. This shortcoming of the specification made it impossible for the remote team to work in complete autonomy.

Another challenge involved the cost estimates for the project. These estimates did not take into consideration the time required for the overall coordination of the local and remote development environment. With such large teams on both sides of the project, situations arose where rework was required due to an incorrect understanding of the requirements. This often had an effect on the overall project schedule. Given that our project plans were comprised of thousands of tasks, one project manager was not enough to manage the complexities of the two development environments.

Finally, shortcomings in each team's cultural awareness of the other team led to project challenges in the areas of respect and overall cohesiveness. For example, the local team would have benefited from being aware of cultural aspects such as the holidays and work week for the remote team. We expand our discussion of the challenges in the following sections.

Our approach

All of the projects we describe in this section followed a similar development process. For these projects, the remote teams were primarily comprised of developers. In a typical project of this size, the development staff is engaged later in the life cycle, but in two of the projects, the staff in India was engaged earlier than necessary in the project time line in order to reserve them. To ensure the time that the remote teams were billing to the projects was used effectively, they took on the tasks of setting up development environments and coding the common modules which would be required for the projects. Unfortunately,

because the specifications were not available yet for the common modules, the remote team was forced to code with very little direction. This led to a large rework effort later in the project.

During the design phase, the local teams developed the design specifications to a level of detail that was adequate for use by the local development team. Unfortunately, because the design team had never worked with the remote team in India, the specifications were often based on assumptions that were easily misinterpreted by the remote team. This was generally due to the specification not being detailed enough to allow the remote team to do development effectively. In many cases, the local team needed to develop designs almost to the point of code specifications before handing off the code to the Indian team. Unfortunately, due to time constraints, the local team would often simply take on the rework and development themselves, regardless of project budget. As one can imagine, this had several ripple effects. First, members of the remote team were sitting idle while local team members were working double time. Second, the project schedule was placed at risk, as it was impossible for the local team to maintain the extended hours without extreme exhaustion. Third, the costs rose dramatically as the per-hour cost of local personnel was typically three to four times the cost of the remote personnel. Last, and most important, customer confidence in the global delivery environment began to erode as a result of these effects.

The three projects in this case study attempted a true follow-the-sun approach where several different developers were working on the same code base. This was extremely difficult and not successful for several reasons. When the local team developers would finish the code to a certain point, they would document their progress and leave instructions for the remote team to follow up. They tried to assign the same developers to work together on tasks, but due to scheduling conflicts, developers were constantly being reassigned, and the continuity of the code development was lost. All three projects resorted to assigning discrete components of the development to the remote team and the local team. For example, in most cases, the Web page development was assigned to the remote team. This assignment allowed the remote team to work on components of the overall system development that were not affected by deep system integration issues or a complete understanding of the business requirements.

One issue that was common to all of the projects was that the India team needed to connect to the customer's development environment in order to access the source control system. In two out of the three projects, this connectivity was never achieved. As such, the team in India could not effectively access and maintain their code in the customer's development environment. This led to the development of an ineffective temporary solution in which the team in India was forced to send their software updates by e-mail or FTP (File Transfer Protocol) to the team in the United States. Another method required the use of a Lotus Notes* TeamRoom (a Lotus Domino application that provides a tool for information sharing and collaboration) in which the code was stored as an intermediate step. The team in the United States would then load the code into the customer's development environment. The team in India would try to replicate the source control system with the help of the United States team, but often, these environments were out of sync. As expected, this led to inaccuracies, misunderstandings, and defective code development. If a build (configuration) was sent to the local team or remote team and it did not replicate to the development environment correctly, this often led to a delay of 12 hours until the other team came back online and could help to find the source of the newly appearing issue.

During the testing phase of the project, when certain defects were assigned to the remote teams, many of the same issues described previously occurred. If the problem was not specified in great detail, it was very difficult for the remote team to function effectively during the testing process. In addition, the remote team's environment was often not in sync with the customer's test environment, making it very difficult to work on problem determination and recovery.

Project structure

In all three projects, during the sales cycle, only one project manager was assigned in the overall project estimates. The projects were complex due to the custom coding requirements as well as the distributed development environment. Instead of managing issues, risk, and customer relationship, the project manager ended up spending the bulk of the time managing the detailed project plan. These projects involved thousands of lines of detailed tasks in the overall plan, and it took days to normalize the plan after the slightest change in schedule. Due to the challenges that were discussed in the previous section, the project manager's role became largely administrative in nature. Adding a full-time project executive and a full-time project administrator helped to redress this issue, but again, at the expense of cost and overall project schedule slippage. To generalize, globally distributed projects are inherently more complex than other projects, and their budget should include the appropriate number of governance personnel.

The cultural factor

As in the previous case study, the teams were from opposite ends of the globe, and the resulting cultural differences contributed to the overall challenges. Furthermore, the projects we describe here had the additional challenge of difficulties in forming a sense of camaraderie due to the troubled-project environment. Often, the remote teams were very polite and did not ask many questions. When asked if they had a good understanding of the requirements and if they would be able to meet a scheduled date for a task, the answer was often yes, but the local team did not have enough cultural awareness to understand that the responses given by the remote team might need additional probing.

Another issue was respect for the holidays and work week of the remote team. The local teams had deadlines and unhappy customers and were working significant amounts of overtime. They expected the remote teams to be available at all hours regardless of holidays or typical work weeks, which was not always possible. This eroded the team relationships even further. Some basic cultural awareness training (built into the project startup) could have gone a long way in helping both sides to understand and respect the various differences in working styles and schedules, potentially eliminating these issues. Cultural differences must be met with sensitivity, or the project will incur more risk due to a breakdown of relationships between geographically distributed teams.

Results

As challenges continued to arise in these projects, the overall effect was a slip in schedule and a significant decrease in profitability. In two of the three projects, the gross profit was negative. In the third project, there was a significant net loss. The project did manage a positive gross profit, but its percentage was in the single digits. In all cases, there was a major slip in schedule, unhappy customers, and exhausted project teams. As one can imagine, this also led to issues between the local and remote project teams, as it was easy to lay blame for overall project problems on the other team.

Lessons learned

The following lessons learned were common to all three projects in the second case study and apply to any global software development effort.

- Time should be spent in defining the source-code management processes and infrastructure. In many cases with global development teams, infrastructure accessibility can be an issue; this issue should be identified and mitigated early in the project.
- The level of detail needed for handoff activities (i.e., coding specifications, defect descriptions, etc.) should be well-understood. This becomes paramount for global development teams, as the next opportunity for the remote team to communicate for clarification could be as far off as 24 hours.
- The project structure should be built from the start to handle the complexities of managing in a globally distributed software development environment. This includes a project management team instead of one overall project manager, additional time for the setup of processes for managing code, and time to build and explain detailed specifications, if necessary.
- Cultural awareness training should be incorporated in the startup activities for the project kickoff. Strong communication and relationships should be built between the teams. The managers should get to know the teams, understand their cultural differences, and determine what they know and what they do not know. The interfaces and communication plan should be constructed accordingly.

Conclusion

Based on the previous discussion, we have formulated a set of characteristics that help make a distributed development effort successful, as well as some best practices that can help make follow-the-sun projects work successfully.

It is very important that a project which involves distributed development work spanning time zones allocate sufficient time and money in the project estimates so that appropriate oversight activities can be executed in an effective manner. These estimates must include time to build detailed specifications to accommodate the remote teams, because when they are developing the code, they will not be able to ask questions about details, as they would if they were collocated in similar time zones. Additional oversight time should be included in the daily schedule to facilitate the handoff of tasks from one team to the next, as well as to describe any outstanding issues.

If the source control system must be replicated due to connectivity issues, a process must be in place, ensuring that the two environments remain synchronized. Ideally, each team will use the same source control system and development environment. If this is not possible, it should be noted from the start that this situation could potentially impact not only the project estimates and time line, but also ongoing project activities required for synchronization.

Both teams must understand the level of the specifications that will be necessary in order to perform each task. Time and budget must be sufficient to generate detailed specifications if necessary. With time-zone-challenged projects, strong

communication between the two teams is mandatory to ensure there are no misunderstandings on the tasks and detail required. One way to accomplish this would be to have the remote team document their understanding of the requirements of each meeting that they attend. It is essential to ensure that the teams are in sync during the handoff calls or meetings, as the next checkpoint could be as long as 24 hours later. One of the goals of using the follow the sun approach is to compress the time line for a project to be completed. If misunderstandings arise that require resolution by the off-shift team, the efficiencies gained are easily lost.

Effective communication should be encouraged and required during the project. The remote team should be encouraged to always ask questions and not to make assumptions, at least not in the very early stages of the project. The team leader should try not to ask “yes or no” questions; instead, questions should be phrased so that they solicit a response that demonstrates a complete understanding of the task at hand. Another way to minimize the impact of assumptions made is to employ some of the recommendations of an agile methodology, such as Scrum, which includes in the development process daily status meetings to facilitate the handoff and assignment of tasks and to address any outstanding issues or questions. Ideally, these meetings are very brief, on the order of 15 minutes per day, but the time investment is well worth the advantage of not having to go back and fix defects further in the process. The length of the meetings may be extended if required, especially in early stages. This is an effective means of handing off tasks on a daily basis when employing a follow-the-sun methodology.

Mutual success criteria should be developed for the teams. When working in a distributed development environment, it is important to give both teams a mutual goal and mutual sharing in the overall success. Success should be recognized when it occurs, and factors that are causing failures should be addressed immediately as they occur. Mutual ownership of success criteria breeds commitment from all involved teams. If each team feels some ownership for the success of the project, it is much more likely to succeed. If the teams share an egalitarian environment, as opposed to relegating one of the teams to secondary status, the project is more likely to succeed.

In both of the case studies described in this paper, though many times the teams were working 12-hour days, we were coordinating across only two geographic areas, and as such, the methodology was not a follow-the-sun methodology in the strictest sense, which involves 24-hour development. Exploration still needs to be done regarding how this approach would work when extended to three geographic areas, thus providing a true 24-hour development clock. Many of the same recommendations made here would still apply, especially regarding building in time for coordination activities. We contend that in this case, the management structure would actually have to be designed specifically to facilitate this type of approach, not retrofitted from a model that was designed for traditional development methodologies.

In summary, although it does introduce new variables into the development process, we believe that using the follow-the-sun approach for software development can work. Given the project characteristics listed previously, successful communication between team members and an overall project structure designed to fit the complexities of these projects, this approach can be a successful way to decrease overall project costs and shorten schedules, provided the methodology is specifically designed to facilitate project success. We believe that as more experience is gained in this area, such advantages will be used more

and more as a competitive differentiator in the software development marketplace.

*Trademark, service mark, or registered trademark of International Business Machines Corporation.

Cited references

Accepted for publication April 19, 2006; Published online September 29, 2006.