"Follow the Sun" Workflow in Global Software Development

ERRAN CARMEL, J. ALBERTO ESPINOSA, AND YAEL DUBINSKY

ERRAN CARMEL is a professor in the Information Technology Department at the Kogod School of Business at American University. Professor Carmel's area of expertise is globalization of technology work. He studies global software teams, offshoring of information technology, and emergence of software industries around the world. His 1999 book *Global Software Teams* (published by Prentice Hall) is a pioneering book on the topic. His second book, *Offshoring Information Technology* (coauthored with Paul Tjia), published in 2005 by Cambridge University Press, is used in many global sourcing courses. He has written over 80 articles, reports, and manuscripts. He consults and speaks to industry and professional groups. In 2008–9, while writing this paper, he was the Orkand Chaired Professor at the University of Maryland University College. He has been a Visiting Professor at Haifa University (Israel) and University College Dublin (Ireland).

J. ALBERTO ESPINOSA is Associate Professor of Information Technology at the Kogod School of Business at American University. He received his Ph.D. in Information Systems from the Tepper School of Business at Carnegie Mellon University. His research focuses on coordination and performance in global technical projects across global boundaries, particularly across spatial and temporal distances, and coordination in large-scale technical collaboration tasks such as within-enterprise architecture.

YAEL DUBINSKY is affiliated with the Software and Services group at the IBM Haifa Research Lab (HRL). She is also a visiting member of the human–computer interaction research group in the Department of Computer and Systems Science at La Sapienza, Rome, and for more than ten years has been an instructor of project-based courses in the Department of Computer Science at Technion–Israel Institute of Technology. Dr. Dubinsky received her B.Sc. and M.Sc. in Computer Science and Ph.D. in Science and Technology Education from Technion, Israel. Her research interests center on software engineering and information systems. She has significant experience guiding Agile implementation processes in both industry and academia. She has presented her research at the ICSE (International Conference of Software Engineering), Agile, and XP (Extreme Programming) conferences since 2002. Her book *Agile Software Engineering,* coauthored with Orit Hazzan, was published by Springer in 2008.

ABSTRACT: Follow the sun (FTS) has interesting appeal—hand off work at the end of every day from one site to the next, many time zones away, in order to speed up product development. Although the potential effect on "time to market" can be profound, at least conceptually, FTS has enjoyed few documented industry successes because it is acknowledged to be extremely difficult to implement. In order to address this "FTS challenge," we provide here a conceptual foundation and formal definition of FTS. We then analyze the conditions under which FTS can be successful in reducing duration in

Journal of Management Information Systems / Summer 2010, Vol. 27, No. 1, pp. 17–37.

0742–1222 / 2010 $9.50 + 0.00.
DOI 10.2753/MIS0742-1222270102

software development. We show that handoff efficiency is paramount to successful FTS practices and that duration can be reduced only when lower within-site coordination and improved personal productivity outweigh the corresponding increase in cross-site coordination. We also develop 12 research propositions based on fundamental issues surrounding FTS, such as calendar efficiency, development method, product architecture and handoff efficiency, within-site coordination, cross-site coordination, and personal productivity. We combine the conceptual analysis with a description of our FTS exploratory comparative field studies and draw out their key findings and learning. The main implication of this paper is that understanding calendar efficiency, handoff efficiency, within-site coordination, and cross-site coordination is necessary to evaluation—if FTS is to be successful in reducing software development duration.

KEY WORDS AND PHRASES: calendar-efficient software development, global coordination, round-the-clock development, software development, software handoff efficiency, time to market, 24-hour development.

FOLLOW THE SUN (FTS) IS A RATHER INTUITIVE IDEA: hand off work at the end of every day from one site to the next many time zones away (e.g., United States to India) so that the work can be advanced while one's team rests for the night. The effect can be profound, both theoretically and for practice. Theoretically, *n* sites can increase their development speed by organizing the work tasks to work sequentially on a daily basis by optimizing coordination costs. For practice, FTS is appealing because of the potential to reduce "time to market."

Despite such temptations, FTS has had few documented industry success cases. As was acknowledged more than a decade ago, "Follow the sun with daily handoff is very difficult" [6, p. 34]. This difficulty has not changed noticeably in the ensuing years despite improved technologies and methodologies [3]. In this paper, we investigate this "FTS challenge"—the gap between promise and reality—with a comprehensive conceptual examination.

FTS (also called *24-hour development* and *round-the-clock development*) is one form of global software development [6] with all its corresponding challenges of coordination barriers, cultural differences, and communication difficulties [14]. However, we contend that FTS is uniquely focused on *speed* improvement in that the project team configuration is designed to reduce cycle time (also known as *time-to-market reduction* or *duration reduction*).

We position our research as a conceptual foundation to study FTS for the express purpose of accelerating software work in order to reduce time to market. FTS requires formidable daily handoff coordination—a time and effort cost—which is very much at the heart of its difficulty. However, creative practices may reduce coordination costs, which leads to our research question:

> *RQ: Can FTS be more effective in improving development speed—and if so, in which ways?*

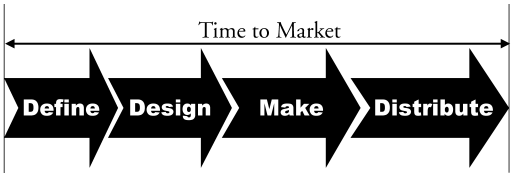


Figure 1. Timeline of Time to Market, Measured from Inception to Use/Sale

Our research question can be further decomposed into more specific unresolved FTS issues, which we begin to address in this paper: (1) *FTS definition*—the term is used inconsistently; in order to make progress in this research area, it is important to adopt and employ a consistent definition so that we can adequately compare and contrast findings; (2) *development speed*—we dissect its theoretical basis and introduce a related concept of calendar efficiency; (3) *development method*—the particular development method employed is likely to drive FTS success; (4) *product architecture*—the architecture chosen will likely drive FTS success (software products can be partitioned into subsystems, modules, features, etc.); and (5) *coordination costs*—the highly interdependent work that FTS imposes has different coordination costs relative to both collocated and conventional global work configurations. Our goal in this paper is to begin to address these issues and provide a conceptual framework to guide further studies of FTS.

Background on Time to Market and FTS

IN ORDER TO UNDERSTAND FTS, one needs first to understand development speed and its associated concept of time to market. *Time to market* is the length of time it takes from product conception until the product is available for use or sale [40] (Figure 1). Time to market is most important in industries where products become outmoded quickly, such as mobile telephone handsets and their corresponding software. Time to market is also important for strategic information systems (IS) such as competitive e-commerce systems or innovative supply chain management systems. There are other managerial reasons for duration reduction—avoiding contract creep, schedule slippages, and budget overruns.

A desire for rapid development—a sense of urgency—is shared by most firms and projects in a competitive marketplace, but most efforts to reduce project duration are reactive, utilizing overtime hours or work speedup (e.g., work faster, skip steps, set aggressive deadlines). All these reactive efforts have real costs due to burnout and fatigue [32]. Adding personnel for speedup is of little interest in software development because of the wisdom gained long ago from the seminal Brooks's law—"adding manpower to a late software project makes it later" [4, p. 25]. Rather than reactive tactics, proper time-to-market reduction requires a deliberate design around the objective of speed that is based on high awareness of achieving this goal within the existing development team [32, 37, 40].

The first well-documented global software team specifically set up to take advantage of FTS was at IBM in the mid-1990s [6]. This team was set up from inception to employ FTS, spread out across five sites around the globe. However, FTS was unsuccessful. It was uncommon to move the software artifacts daily as had been hoped. Finally, management decided to abandon the effort of frequent daily handoffs (tight coupling) and to reduce collaboration between the sites to the loose coupling that is common in the majority of today's global collaborations.

The first researchers to examine FTS were Gorton et al. [18]. They conducted a series of small controlled experiments in the mid-1990s but did not continue their line of inquiry beyond this. Cameron [5] claimed some limited FTS success at the global American firm EDS (now Hewlett-Packard) but did not continue his efforts either. Gupta has also written extensively about the promise of FTS or, more specifically, the 24-hour knowledge factory [20].

During the past decade, some have claimed successful FTS practices, but on closer inspection, even though these projects were indeed dispersed, they did *not* practice the daily handoffs of FTS (e.g., [43]). We note that this is consistent with the authors' experience in industry: the *FTS* term is used loosely, and on closer inspection there is no—or very little—FTS. For example, contrary to myth, Indian offshore firms do little FTS [7].

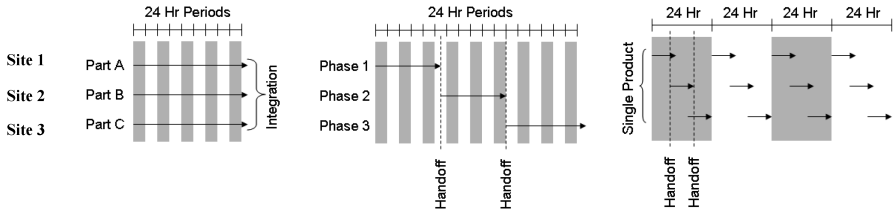
In summary, in the ensuing decade since the much-publicized IBM FTS project, there has been little progress to address and understand the FTS challenge, either in the research literature or in practice. With limited progress in empirical field research, the FTS research literature has recently moved in another trajectory—mathematical modeling [26, 39, 41, 42]. We will return to these models later in this paper.

As illustrated in Figures 2a and 2b, globally distributed configurations involve decomposing tasks and allocating them to multiple sites in a way that minimizes dependencies across sites [6, 21]: parallel work[1] or development phase (we note that there are other considerations besides minimizing dependencies, such as location of expertise). For our own shorthand notation, we refer to these two configurations as "conventional global configurations." The key difference between FTS (Figure 2c) and conventional global configurations is that FTS focuses on daily handoffs from site to site, whereas the *opposite* is true for conventional global configurations, in which an attempt is made to reduce interdependencies and handoffs as much as possible.

Defining and Disambiguating "Follow the Sun"

IN THIS SECTION, WE PROPOSE A FORMAL DEFINITION OF FTS based on the foregoing discussion. A definition is critical because progress in FTS research requires that researchers use the same frame of reference to compare results. Before we propose our definition, we posit that FTS requires satisfying all four of these criteria:

1. *The main objective of FTS is duration reduction.* This criterion distinguishes FTS from other popular global software development configurations and practices (e.g., offshoring is often conducted for cost objectives; parallel develop-



(a) Parallel (b) Development Phase (c) Follow the Sun

Figure 2. FTS Compared with Other Globally Distributed Configurations
Note: We refer to parallel- and phase-based as "conventional global" configurations.

ment is a more manageable configuration). FTS is clearly difficult and offers no other advantages over other configurations besides speed.

2. *Production sites are far apart in time zones.* This criterion differentiates FTS from other production acceleration tactics.
3. *At any point in time there is only one site that owns the product.* This criterion differentiates FTS from conventional global configurations in which various sites may own different parts of the product.
4. *Handoffs are conducted daily at the end of each shift.*[2] This criterion differentiates FTS from conventional global configurations that minimize dependencies and handoffs between sites.

Building on the above criteria, we define FTS as

> a round-the-clock work rotation method aimed at reducing project duration, in which the knowledge product is owned and advanced by a production site and is then handed off at the end of each workday to the next production site several time zones west.

Our definition is flexible in a number of respects. First, FTS applies to any type of knowledge work in which a knowledge product is being developed (not just software development). For example, Gupta [19, 20] describes other knowledge-based applications that claim to do FTS—at General Motors and at OfficeTiger. Second, the definition is consistent with broader definitions of global collaborative software development across global production sites [25]. Third, it allows us to expand our thinking of how FTS work is organized. For example, we usually envision FTS with two or three sites, but assuming six hours per site of intensive software development per day ("task time"), it is theoretically possible to manage FTS with even four sites spread out across time zones of the globe and perhaps even more. Fourth, our definition allows for work time overlap between sites, if desired, because many time-separated teams plan for such overlap, at the beginning/end of a shift, to allow for synchronous coordination. Fifth, in cases where some workdays involve parallel work (and there is no FTS handoff), then these cases could be labeled *mixed* FTS-parallel.

Moreover, we state four key *assumptions* necessary for our definition to be robust: (1) each production site works during its day as a "subteam" and it needs within-site

coordination, (2) a subteam can consist of one or more members, (3) the handoff from one site to the next can occasionally be empty in the case of holidays or emergencies, and (4) there is a common digital product repository (such as a software configuration management system), which allows all sites to "commit" the code/objects at the end of the workday.

As a final step in clarifying FTS, it is important to disambiguate FTS and to state clearly what FTS is *not.* We delineate four types of similar concepts that are not FTS:

1. *Global knowledge work.* Global knowledge work is a general label for geographically dispersed knowledge workers working collaboratively across multiple global boundaries [14]. However, in most cases, these knowledge workers have little task dependency and do not hand off work in order to reduce duration. Therefore, although FTS is one instantiation of global knowledge work, most global knowledge work is not FTS, because it tends to fail one or more of the four criteria of the FTS definition above.
2. *24-hour business processes.* Such work arrangements are quite familiar in modern call centers because they can automatically route calls to workers who are on active shift somewhere else in the world (usually in daylight hours). However, in most cases, these knowledge workers have little task dependency and do not hand off work in order to reduce duration. Global help desks, for example, are set up to provide continuous service coverage around the clock. Twenty-four-hour business processes are not the same as FTS, because they fail criteria 1 and 4 of the FTS definition above.
3. *24-hour manufacturing.* In a continuous production line, workers assemble products until the end of their shift. Shifts are employed to fully utilize expensive production/factory resources that could not produce more by simply enlarging the production crew in a single shift. In software development, however, expensive production resources (e.g., testing labs, hardware platforms) are not usually the driver of the project configuration. Rather, the resource that is shared across shifts is the software code along with its meta-data.
4. *Collocated multishifts.* A reasonable alternative to FTS is to choose one location where labor is cheap and run several eight-hour shifts of software developers. In addition to cost advantage, the shifts can be timed to overlap at the shift transfer times to allow for synchronous face-to-face handoff coordination. Such a configuration is feasible, but our interest in FTS rests on the premise that distributed global work is *a given* (an endogenous factor), and hence our challenge is to understand how to do it optimally. After all, globally distributed software development is more difficult to manage and coordinate than collocated development, and yet it is ubiquitous—despite its difficulties.

Speed, Duration, and Calendar Efficiency

TIME TO MARKET AND THE RELATED CONCEPT OF TASK DURATION are important areas of inquiry because they are relatively understudied in the disciplines of IS and software

engineering. The IS literature has devoted some attention to the time domain but has largely focused on subjective perceptions of time [38] rather than approaches to increasing speed. In global software engineering/development there has been some tangential interest in speed, and some studies [21, 22] have found that multisite software teams take longer than collocated teams.

In our own research stream, beginning in 2003, we studied the effects of time separation on speed. Cummings et al. [10] studied global teams in the field and found that the time zone difference between two software developers increases delay, but this increase is significant only when team members have no overlapping work time. When there is some time overlap, such as with synchronous handoff, the effect on delay is negligible. Espinosa et al. [13] experimented with time zone variations in a computer lab and found that small increases in time zones (compared with collocated configurations) reduced speed, but as more time zone separation was added, speed increased, suggesting that there are speed advantages to working across time zones. Although these studies may point to the potential benefits of FTS, they do not specifically address FTS work in which the workflow is synchronized to take advantage of time zones.

In order to fully understand how FTS may affect speed and duration, we analyze the efficient usage of the entire *calendar* time available for production. We introduce the term *calendar efficiency* to focus our discourse in the rest of this section and in Table 1. We define calendar efficiency as the percentage of all of the calendar time (e.g., $24 \times 7 = 168$ hours available per week) that is used productively for work. Thus, a 40-hour workweek utilizes 23.8 percent of the calendar workweek (40/168). Therefore, the calendar efficiency is only 23.8 percent efficient, showing that there is a lot of room for calendar efficiency improvement. One simple way to increase calendar efficiency is to work overtime. Our usage of the term *calendar efficiency* is analogous to Cameron's [5] compression or improvement factor in his treatment of FTS.

In Table 1 we compute the calendar efficiency in different modes (note our assumptions at the bottom of the table; also note that we do not introduce any notions of labor units or productivity yet). The key numbers appear in the fourth column, beginning with a typical one-site team (the Baseline), which uses only a dismal 17.9 percent of the overall calendar time after taking into account nontask activities. This rather low figure proves the high potential for FTS. One simple way to increase calendar efficiency is to work overtime, but the typical *overload* mode (overtime) only raises calendar efficiency to 23.8 percent. A very heavy overload mode of 20 hours of weekly overtime raises calendar efficiency to 29.8 percent but is not sustainable over a long time period because of employee burnout.

The significant FTS potential for calendar efficiency gains becomes evident in the bottom rows of Table 1. An optimal FTS configuration can raise calendar efficiency as high as 71.4 percent. The four-site FTS approach reduces duration by nearly four times relative to the baseline.

At this point we begin to introduce our 12 research propositions. The first proposition is based only on the concept of calendar efficiency. Later we introduce other factors into subsequent propositions. In regard to calendar efficiency, our discussion suggests that:

Table 1. Calendar Efficiency in Different Work Modes

Shifts/sites	Descriptor	Calendar efficiency net (task time only)		Calendar efficiency (task + nontask time)		Maximum calendar time per week
		Hours	Percent	Hours	Percent	
1	Baseline	30	17.9	40	23.8	168 hours
1	Overload (light) = + 10 hours/week overtime	40	23.8	50	29.8	168 hours
1	Overload (heavy) = + 20 hours/week overtime	50	29.8	60	35.7	168 hours
2	Follow the sun	60	35.7	80	47.6	168 hours
2	Follow the sun (+ 10 hours/week overtime)	80	47.6	100	59.5	168 hours
3	Follow the sun	90	53.6	120	71.4	168 hours
4	Follow the sun	120	71.4	120	71.4	168 hours

Assumptions and basis for calculations:

1. *Days per week* = 7; workdays per week are only 5.
2. *Hours per day* = 8 hours per shift except for the overload modes.
3. *Overload time* assumes that all additional hours are devoted to task (rather than nontask) activities.
4. *Task activities* include all software development work, including all meetings and all coordination time.
5. *Nontask activities* = 25 percent of workday. These are activities such as staff recruiting, writing a memo about the previous project, filling out time sheets, or fixing the copy machine. These are not coordination activities, which we discuss later in this paper. We use 25 percent here, building on the rules-of-thumb in the Agile community; task activities are called “ideal working hours” and are typically estimated at 50–75 percent of the workday [9].
6. *Coordination time losses or gains* are ignored in this table and will be introduced in the next section.
7. *Time off.* Company holidays at each site are ignored here; generally, they represent about 6 percent of annual workdays across nations; individual absenteeism, as well as individual vacations, is assumed to have negligible effect.
8. *Multitasking* is ignored.
9. *Clarification for FTS with four sites.* At this level, 24 hours, or 100 percent, of task time, is utilized per workday, because nontask activities can be conducted separately and do not detract from calendar usage (i.e., 6 hours of task time and 2 hours of nontask time per site).

Proposition 1: Compared with conventional global configurations, FTS increases calendar efficiency substantially, and this efficiency increases as the number of shifts/sites increases.

Structural Considerations

IN THIS SECTION, WE MOTIVATE THE FOLLOWING ISSUES AROUND FTS—phase specificity, choice of development method, and product architecture. We then illustrate some of these concepts with exploratory observations.

Phase Specificity

There is substantial anecdotal evidence in industry that FTS can be effective in reducing duration within *specific phases* (Figure 3). Testing can work well in FTS: one team searches for bugs and documents these bugs in a database, which is then accessed and worked on by the software team at another site many time zones west. For example, EDS claims to do this between Argentina and India [17]. Testing is a good fit because the handoff is structured, granular, and, with trained staff, usually does not suffer much from miscommunication. Short spurts of prototyping have also been successful in FTS. For example, PortalPlayer, an early maker of embedded software for Apple's iPod, with research and development in India and Silicon Valley, claimed that it performed rapid prototyping using FTS [36]. These anecdotes are consistent with the authors' industry observations. For example, software engineers claim benefits of FTS, but these instances of FTS are brief spurts of several days or, at most, a few weeks.

In contrast, work that spans more than one phase may not be suitable for FTS because of the amount of communication that is necessary to move from one phase to the next. Consequently, our next proposition is:

Proposition 2: Relative to work that spans multiple systems development life cycle (SDLC) phases, the work within a particular SDLC phase is more suitable for FTS development because its specificity allows for more structured and granular handoffs.

Development Method

Phase specificity means that FTS is achieving only partial, limited improvements in overall development speed. So, those who have examined FTS closely have recognized the importance of selecting an FTS software development methodology that spans the entire development process and supports the special needs of daily handoffs. IBM's classic FTS team of the 1990s constructed a unique organization structure and process [6]. Similarly, Cameron [5] at EDS crafted a special methodological adaptation for FTS.

This leads to considering the advantages and disadvantages of linear-sequential approaches (e.g., waterfall, incremental) versus iterative models (e.g., Unified Process,

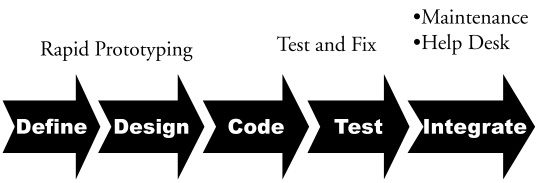


Figure 3. Phase-Specific Activities That Fit FTS Displayed Above a Generic Waterfall SDLC

Agile UP, Agile [2, 21, 24, 44]) and how they apply to FTS. The limitations in phase specificity, which we noted above, suggest that linear-sequential approaches are unlikely to be optimal for FTS (except for work within a single phase) and therefore we should turn to iterative models. Each iterative model includes all the activities of the SDLC phases. Hence, those models that use longer iterations resemble the linear-sequential approaches, whereas models that use short iterations blur the borders between SDLC activities. In the latter case, FTS handoffs contain artifacts that cover all activities—requirements, design, code, and test.

In order to move forward, we chose a specific iterative approach. We argue that the Agile approach is the most promising of the iterative approaches for FTS—and use it in exploratory studies described later in this paper—for several reasons. First, it has the enabling property of using short time-boxed iterations of two to four weeks each. The customer requirements for each iteration are feature based; thus, all the SDLC activities are merged in each iteration, and features are designed, tested, developed, and presented. Second, with all activities intertwined, the Agile method introduces continuous integration that enables granular and structured daily handoffs. Continuous integration (while using an automated integration environment) enables each team to develop in its own code base in its own time period. Yet each team maintains an updated, testable code base to be used by the next production site. The policy of keeping the integration *green* (i.e., all tests pass) at the end of the workday is common in Agile teams. It ensures high-quality handoffs and thus fits nicely with FTS requirements. Third, Agile inspires a sustainable pace that fits the notion of working mostly during one's daylight hours. Fourth, Agile promotes exhaustive automated testing, which should achieve a duration reduction.

Also note that Denny et al. [11] and Gupta [20] have attempted to conceptually marry FTS with elements of Agile.[3] All of which leads to the following proposition:

> *Proposition 3: FTS is more suitable than conventional global configurations for Agile development when some core Agile practices are used—small time-boxed iterations, exhaustive automatic testing, continuous integration, and sustainable pace.*

Product Architecture

How the software product is architected and how the work is partitioned across sites may have an effect on the extent to which FTS helps increase speed. In general, FTS

may seem somewhat paradoxical because it violates one of the foundational principles of software management—that software should be decomposed and dependencies (coupling) between development groups be minimized [33] (see Figure 2). Once decomposed, the work may then be assigned to different sites based in part on where expertise resides [6, 34], regardless of locations and time zones. This type of decomposition and partitioning is unsuitable for FTS development because there would be little work to be handed off at the end of each day.

However, we posit that there are exceptions to the traditional architecture that manifest at some optimum point of *complexity-granularity* that suits FTS. Our example is from large complex systems that work around modification requests (MRs). Large, complex modules and subsystems are typically developed and modified by large groups of developers, often geographically dispersed [22]. Top-priority MRs involve the development of new features or modifications that have a severe effect on client service, either in the form of new critical functionality or repair of critical client services [15]. Because of the complex interactions of the new and existing code in such modifications, it is best to employ small teams and develop the code sequentially rather than to use a larger team that would need to develop the software in parallel and then integrate the various parts. Thus, these are cases where the complexity is high, the granularity is low, and there is a need to keep the team small in order to reduce the number of communication links between individuals. We posit that it is precisely in these situations that FTS can help accelerate development speed. The following propositions summarize the key points we surfaced:

Proposition 4: FTS will be more successful for product architectures that partition the software into smaller, relatively independent components (e.g., features, MRs, modules).

Proposition 5: FTS is more suitable for the development of product components than for integration of components.

Proposition 6: FTS suitability increases when a product component is more functionally cohesive and more well defined.

Field Study and Preliminary Observations

In order to address the "FTS challenge," we also conducted an exploratory comparative field study. This part of our FTS study derives from the design science research paradigm [23]. Here, we not only observe the phenomenon but also work, in a utilitarian sense, on a build–evaluate loop. Design science seeks to create new and innovative artifacts, where the artifact can be a physical artifact, a software algorithm, or, as in this case, a method/process.

Our exploratory study is described in more detail in a prior article [8] and has been extended since then in a second phase, so here we only summarize the key parameters and observations from the two phases. We use these preliminary observations to help develop the conceptual aspects of FTS. (We note that Gupta [20] has also progressed

in a somewhat similar direction in his FTS study in a yearlong comparative field study at IBM).

As we noted, evaluating and testing FTS requires making implementation choices regarding configuration and methodology. One of these choices was to use the Agile methodology. Our study compared teams engaged in Agile development, divided into FTS teams and control teams.

The participants were experienced computer science and electrical engineering students at an elite university, all between ages 20 and 30. Most of the students were working part-time in software engineering roles in sophisticated firms. The study task required approximately 400 person-hours per team.

The control teams could interact in any way they wished, including face-to-face and synchronously. On the other hand, in order to simulate time zone differences, the FTS teams had strict rules imposed on their interaction such that only asynchronous handoffs were allowed between the subteams at fixed intervals of time.[4] Figure 4 depicts the actual staggered activity of the two subteams of an FTS team.

Our comparative field studies generated a number of data streams. First, we measured duration and were also able to collect other data generated by the version control system, such as the number of check-in operations and the number of revisions per file. Further, we examined the electronic work logs and analyzed students' verbal reflections during the semester.

In our comparison studies, we found that FTS teams performed roughly equally or better than the control teams, and teams in both projects met the project requirements with respect to functionality and level of quality. More important, using our proxies for measure of duration reduction achieved by the FTS teams, the first team achieved reduction of approximately 10 percent and the second team of approximately 50 percent compared with their respective control teams. These results show some promise for FTS with short Agile iterations.

This preliminary positive finding about speed contrasts with previous results [22] in which distributed teams exhibited longer duration relative to collocated teams. We suggest that our limited, qualified field study success happened because of the Agile implementation, because of the tight iterations and deadlines involved.

Our analysis of the development log documentation was also revealing. In the first comparison, when asked to reflect on the process, the FTS participants described the special way they handed off the work at the end of their working hours. When we checked the electronic forum, we found that the level of development log documentation was markedly better for the FTS team. In the second comparison, the FTS team wrote about 140 percent more documentation lines as well as about 25 percent more messages. In summary, as would be expected with FTS, because all other channels of (synchronous) communication were forbidden, the high volume of documentation (e.g., commit logs) increased. In addition, we also observed better documentation quality.

Finally, we received verbal and written comments from the study participants. Our perception from participant feedback was that the cross-site coordination costs were not as serious as we had anticipated. Perhaps the most interesting finding was that the time pressure imposed on the FTS teams by the study design compelled participants

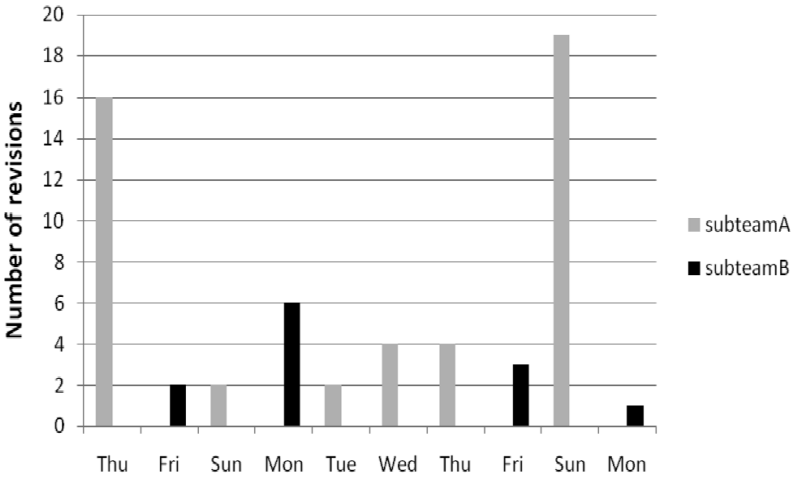


Figure 4. A Slice of Several Days of Activity from the FTS Team in One of the Agile Iterations of Our Field Study
Note: Number of revisions refers to number of historical versions of source code files.

to work more productively. In other words, what the FTS participants were telling us was that they were producing more per hour individually. This is related to the notion of *time-boxing,* which will be further elaborated in the next section. We had initially assumed that individuals have equal productivity regardless of their configuration, but we found indications that the individual productivity of FTS participants increased, thus reducing task duration. In other words, FTS teams appear to behave differently—they tend to be more disciplined and time sensitive, and when time is managed correctly, this affects outcomes positively. In fact, the Agile movement of recent decades advocates and shows findings that support ours [27]. We build on these findings in the next section, where we compare coordination and productivity in FTS with traditional approaches.

In summary, our field study shows that effective handoffs are necessary for FTS to be feasible. Furthermore, our study suggests that a work procedure that includes developing small code components (code and test) with continuous check-in every day (as the handoffs process) may be an effective method for a successful FTS practice. The next section further analyzes the importance of efficient handoffs.

Analytical Discussion

THE KEY TO LEARNING WHETHER OR NOT FTS CAN REDUCE DURATION lies in understanding how coordination cost and individual productivity vary among collocated, FTS, and conventional global configurations. In this section, we introduce the key FTS variables we use to analyze these issues: cross-site coordination time, within-site coordination time, and personal productivity. We then combine these variables into an overarching equation of FTS duration payoff to develop further propositions.

Coordination Costs

Coordination costs are at the crux of why FTS is difficult. Coordination is, by definition, the work necessary to manage dependencies among the task activities carried out by multiple developers [29]. For example, if there is a particular software job that requires x lines of code and each developer can individually produce an average of y lines of fully debugged code per hour, then a single developer could complete the job in x/y hours. Following Brooks's logic [4], two developers would take longer than $x/2y$ because the two developers not only need to carry out their individual software development assignments but also need to devote some time to coordinate and integrate their work. A key FTS question is, therefore, whether these two developers can finish the software production task faster by working in parallel and then integrating the code at the end, or by working sequentially in shifts and handing off the work to each other at the end of each shift.

In other words, although coordination is necessary, the time spent coordinating is time diverted away from individual task work. Therefore, we contend that FTS will be beneficial to reducing task duration when the total coordination costs associated with the project are minimized. For example, if the coordination costs of integrating the work of two developers working in parallel mode are identical to the FTS coordination costs of two developers working sequentially, then the complete software job will take the same time to complete in either mode. Therefore, we will show that when multiple developers are involved, then FTS will reduce duration when the combination of within-site coordination costs (i.e., costs of coordinating parallel work within each site) and cross-site coordination costs (i.e., handoff coordination costs) are minimized. Next, we expand on these basic ideas.

A Simple Model of FTS Coordination and Duration

Here we offer a basic model that can help us gain insights into how key elements of work affect duration in FTS. In our model, we decompose duration into three main components: (1) cross-site coordination time, (2) within-site coordination time, and (3) personal productivity.

Cross-Site Coordination Time

These are the costs associated with handoff activities from site to site. Due to the difficulty of coordinating and resolving task issues [28] across sites/shifts, the cross-site coordination cost will most likely be positive and nontrivial. For example, the lack of task awareness beyond a person's immediate workspace has been found to impede effective responses to unexpected events [35]. Cross-site coordination is closely related to the concept of *handoff efficiency,* which we use further below. FTS is difficult and uncommon because the production teams are sequentially handing off work in progress (unfinished objects). The production objects require daily "packaging" so that the task is understood by the next production site. There are times when the next

production site needs more information and cannot proceed fully without clarification. When clarification is required, an entire day may elapse because the previous site has already gone home. Sometimes misunderstandings also lead to rework (i.e., a type of vulnerability cost [12]).

Figure 5 illustrates how handoff efficiency affects duration. We assume for the moment that the labor resources at each site are fixed and equal and that only cross-site coordination is needed for handoff. For example, a handoff efficiency of 90 percent means that 90 percent of a developer's day is productive and that a total of 10 percent (on both the sending and receiving side) is devoted to handoff activities. In the one-site configuration of Figure 5 (the baseline) there is no handoff, therefore duration is unaffected by handoff efficiency. If we added one more site with an FTS arrangement, then we would have doubled the speed, but only if the handoff efficiency was 100 percent. As the handoff efficiency declines, the gains in speed by adding further sites also decline because we are introducing cross-site coordination costs. Hence, our next two propositions are:

Proposition 7: As handoff efficiency increases, so does FTS development speed.

Proposition 8: Increasing the number of FTS sites that have low handoff efficiency leads to decreasing marginal improvements in duration speed.

We make a few more comments about assumptions and factors that influence handoff efficiency:

- *Handoff failures.* A handoff failure occurs when the receiving site fails to understand the work in progress sent by the previous site, which is more likely when team members have less shared knowledge [16].
- *Variable handoff time.* With each added site (e.g., going from three to four sites), there is cumulatively an increasing amount of information that needs to be conveyed and received.
- *Unequal handoff effort for sender and receiver.* It takes time t_a to prepare and process a day's work for the sender, and it takes t_b for the receiver to process and comprehend this work. It is unlikely that $t_a = t_b$.

Within-Site Coordination Time

From this point we relax the simplifying assumption about labor resources that we used in the prior subsection and in Figure 5 and assume a scenario in which there is a fixed number of people across sites, regardless of the number of sites. Thus, by splitting the human resources into two or more sites, the overall impact is to reduce the number of members in each site's subteam so that, per Brooks's law, the coordination time needed *within* each site decreases because of reduced task dependency links requiring communication [1].

Recall that the number of possible coordination links among n members in a team is $(n/2)(n - 1)$. By splitting a team into s sites, the number of links inside each site is reduced. That is, a team of n developers distributed across s sites will result in s

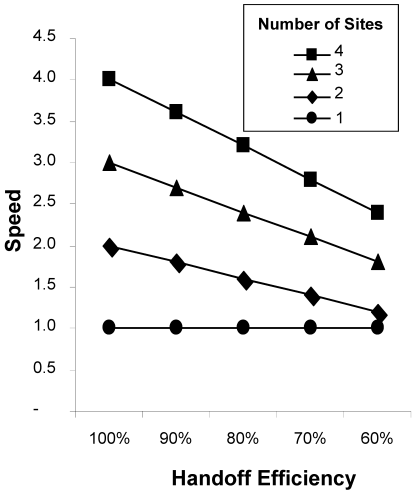


Figure 5. Speedup as a Function of Handoff Efficiency for Three FTS Configurations

subteams of size n/s. Therefore, the number of possible within-site dependency links within one subteam will be $(n/2s)(n/s - 1)$ and the total number of links for all s sites will be $s(n/2s)(n/s - 1)$. The difference in the total number of links between 1 site and s sites is exponential. Hence,

> *Proposition 9: The potential for speed gains due to reduced within-site coordination increases exponentially for larger teams if the teams are subdivided into smaller subteams across multiple FTS sites.*

Naturally, the number of possible FTS sites, s, is limited by the task time per day. For example, if the daily task time is six hours, then the maximum number of sites is four (net of handoff coordination time). Hence,

> *Proposition 10: The potential for speed gains due to reduced within-site coordination needs in FTS increases exponentially as a team is distributed across more FTS sites, but these gains are limited by the daily task time.*

Propositions 9 and 10 are illustrated by Figure 6 (where the number of links are calculated with the formula above). As the diagram shows, the number of possible within-site dependency links drops sharply as the team is split into more sites, and this drop is more pronounced for larger teams.

Personal Productivity

Until this point we have assumed that each individual's productivity is fixed regardless of location and configuration, once nontask time and coordination times are netted out. Other models [26, 27, 42] have made similar assumptions, whereas in Setamanit et al. [39], productivity is a group variable that is affected by coordination. However,

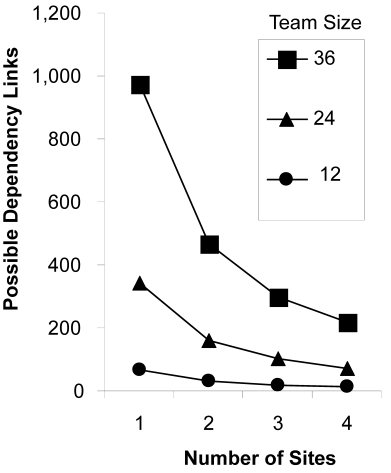


Figure 6. Number of Possible Within-Site Dependency Links by Number of Sites and Team Size

drawing on our exploratory field study, we argue that personal productivity may actually increase in an FTS setup because of "time-boxing" or "daily deadlines." Time-boxing affects individual behavior—of the individual programmer—by setting very strict deadlines in each iteration. The effect of such temporal coordination on individual interaction behavior has been found to have positive effects on performance [31]. The task (or work unit) is much smaller and easier to scope out. The individual is more focused in his or her work and gets distracted less often. Personal time-boxing curbs perfectionist tendencies, decreases procrastination, and does not allow individuals to overcommit to a task. Time-boxing of iterations was advocated in the older notions of *rapid application development* by Martin [30] and has been one of the foundations of both Agile and Unified Process methods [24]. Agile approaches also set a time-box for daily completion in that all work has to be test ready. Hence, our next proposition is:

> *Proposition 11: Time-boxing, a by-product of any FTS configuration, spurs individual productivity (relative to other configurations) due to added rigor, sense of deadline, and goal orientation.*

Next, we combine these three key variables and derive a basic equation that captures FTS duration. Note that all three variables represent a summation (Σ) of all links and all individuals, all variables are measured in units of time (e.g., days), and there is an equal number of individuals in each configuration. Thus, the difference in duration when going from a single-site (collocated) configuration to an FTS configuration can be represented as

$$\Delta DURATION = \Delta CROSS + \Delta WITHIN + \Delta PERSONAL, \quad (1)$$

where $\Delta CROSS$ is the difference in duration due to cross-site coordination between single-site and FTS configuration (recall from the discussion around Propositions 7

and 8 that Δ*CROSS* will certainly be positive—i.e., duration increases due to cross-site coordination); Δ*WITHIN* is the difference in duration due to within-site coordination between single-site and FTS configuration (recall from the discussion around Propositions 9 and 10 that Δ*WITHIN* is likely to be negative—i.e., duration decreases due to lower within-site coordination needs as subteams become smaller); and Δ*PERSONAL* is the difference in duration resulting from changes in personal productivity alone (recall from the discussion around Proposition 11 that Δ*PERSONAL* is likely to be negative—i.e., duration decreases due to time-boxing).

It follows from Equation (1) that task duration will be reduced when:

$$\Delta CROSS + \Delta WITHIN + \Delta PERSONAL < 0. \quad (2)$$

Stated in words, Equation (2) indicates that FTS task duration reduction is accomplished when the decrease in duration due to lower within-site coordination and increased personal productivity is larger than the increase in duration due to higher cross-site coordination.

This is our central FTS duration equation. It captures the essence of our research challenge: to parse these three variables and explore how their negative effects can be mitigated while accentuating the positive effects. We summarize the FTS duration payoff equation with this final proposition:

> *Proposition 12: FTS is beneficial for software development speed when the reduction in duration due to reduced within-site coordination, plus increased individual productivity due to time-boxing, is greater than the duration increase due to increased cross-site handoff coordination.*

As with any modeling, we have made simplifying assumptions. Here we note two more assumptions not mentioned above, which could be relaxed, thus making the model more complex. Notice, however, that as we relax the two assumptions, FTS becomes *less* attractive:

1. *Personnel are interchangeable.* This is an assumption common to most other models. This assumption is not as far-fetched as one might first think; there are a number of software development organizations that have set up mirror sites. A mirror site means that groups with similar distribution of programming skills are created in time-dispersed locations (similar to assumptions in other studies [11, 39]). Another similar assumption is made in the concept of "extreme (paired) programming" (XP).
2. *No absenteeism.* If one of the nodes is operating at less than full capacity because of sick days or vacation, then work may be delayed, thus affecting FTS task duration.

Comparison with Other FTS Models in the Literature

As noted before, four recent studies conducted FTS modeling or simulation [26, 39, 41, 42]. All four studies deal to some extent with the issue of speed, but only two of

them [39, 42] inquire whether FTS is beneficial, as we have done in the present study. Interestingly, the four studies are complementary in a number of respects: they use different methodologies, assumptions (especially regarding coordination), and objectives. Jalote and Jain [26] set out to understand how to optimally allocate FTS tasks to individual nodes using the methodology of critical path and network optimization; Sooraj and Mohapatra [41] set out to understand how to optimally allocate geographical sites in FTS using the methodology of a general sequential mode model; Taweel and Brereton [42] explored FTS sensitivity to overhead and handoffs using a mathematical model; and finally, Setamanit et al. [39] evaluated several global software development configurations, including FTS, using discrete event simulation.

Summary and Conclusions

"THE FTS CHALLENGE" IS TO MOVE BEYOND the idealized appeal of FTS toward verifiable and consistent execution. This paper is the first to comprehensively provide the conceptual foundation for the study of FTS theory and practice. A first step is for research to apply a consistent formal definition of FTS, as we have done. We acknowledge that it is possible that even after concerted efforts, the FTS challenge may not be achievable and we may need to conclude that there are no achievable benefits to FTS, in which case the label may well be "the FTS myth." Thus, FTS achievability is an empirical question for which a necessary first step is a consistent definition.

The conceptual framework we provide is important because it helps analyze the specific conditions under which FTS can be beneficial in reducing product development duration. We have highlighted the importance of understanding the concepts of calendar efficiency and handoff efficiency in helping understand how to analyze, design, and implement successful FTS practices that can reduce task duration. We have developed several testable propositions surrounding key FTS concepts, including calendar efficiency, development method, product architecture, handoff efficiency, and three key variables—within-site coordination, cross-site coordination, and personal productivity. It follows from our discussion that handoff efficiency is paramount to FTS success in reducing software development duration.

A limitation of our overarching analysis is that we have only investigated the effect of FTS on speed, not on product quality. Finally, we note that, whereas our study of FTS refers primarily to software work, our concepts and propositions are generally applicable to most knowledge work that is digitized and distributed globally.

Acknowledgment: J. Mark Johnston, a former student, helped on early versions of this paper.

NOTES

1. In the case of parallel work, it is important to emphasize that coordination costs are usually quite high during the integration phase.

2. Here we use the term *shift,* which, when it happens across time zones, involves a different "site."

3. Denny et al.'s [11] Agile-FTS conceptual model introduces the notion of "composite persona" as a potential collaboration model to deal with the iterations. Composite persona (CP) is "a highly cohesive micro-team that, like a corporation, has simultaneous properties of both individual and collective natures. . . . With respect to CPs, each site is a mirror of the other, having exactly the same CPs as each other site" [11, p. 92].

4. In the first study there were two handoffs per day; in the second study there was one handoff per day.

References

1. Andres, H.P., and Zmud, R.W. A contingency approach to software project coordination. *Journal of Management Information Systems, 18,* 3 (Winter 2001–2), 41–70.

2. Beck, K. *Extreme Programming Explained.* Reading, MA: Addison-Wesley, 2000.

3. Betts, M. 24/7 global application development? Sounds good, doesn't work. *Computerworld* (September 16, 2005) (available at http://blogs.computerworld.com/node/1005).

4. Brooks, F.P. *The Mythical Man-Month: Essays on Software Engineering,* anniversary ed. Reading, MA: Addison-Wesley, 1995.

5. Cameron, A. A novel approach to distributed concurrent software development using a "follow-the-sun" technique. EDS Working Paper, Plano, TX, 2004.

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

7. Carmel, E. Building your information systems from the other side of the world: How Infosys manages time differences. *MIS Quarterly Executive, 5,* 1 (2006), 43–53.

8. Carmel, E.; Dubinsky, Y.; and Espinosa, A. Follow the sun software development: New perspectives, conceptual foundation, and exploratory field study. In R.H. Sprague (ed.), *Proceedings of the 42nd Annual Hawaii International Conference on System Sciences.* Los Alamitos, CA: IEEE Computer Society Press, 2009.

9. Churchville, D. *Agile Thinking: Leading Successful Software Projects and Teams.* San Francisco: Lulu, 2008.

10. Cummings, J.; Espinosa, J.A.; and Pickering, C. Crossing spatial and temporal boundaries in globally distributed projects: A relational model of coordination delay. *Information Systems Research, 20,* 3 (2009), 420–439.

11. Denny, N.; Mani, S.; Sheshu, R.; Swaminathan, M.; and Samdal, J. Hybrid offshoring: Composite personae and evolving collaboration technologies. *Information Resources Management Journal, 21,* 1 (January–March 2008), 89–104.

12. Espinosa, J.A., and Carmel, E. Modeling coordination costs due to time separation in global software teams. In *International Workshop on Global Software Development.* Los Alamitos, CA: IEEE Computer Society Press, 2003, pp. 64–68.

13. Espinosa, J.A.; Nan, N.; and Carmel E. Do gradations of time zone separation make a difference in performance? A first laboratory study. In *International Conference on Global Software Engineering.* Los Alamitos, CA: IEEE Computer Society Press, 2007, pp. 12–20.

14. Espinosa, J.A.; Cummings, J.N.; Wilson, J.M.; and Pearce, B.M. Team boundary issues across multiple global firms. *Journal of Management Information Systems, 19,* 4 (Spring 2003), 157–190.

15. Espinosa, J.A.; Slaughter, S.A.; Kraut, R.E.; and Herbsleb, J.D. Familiarity, complexity and team performance in geographically distributed software development. *Organization Science, 18,* 4 (July–August 2007), 613–630.

16. Espinosa, J.A.; Slaughter, S.A.; Kraut, R.E.; and Herbsleb, J.D. Team knowledge and coordination in geographically distributed software development. *Journal of Management Information Systems, 24,* 1 (Summer 2007), 135–169.

17. Godinez, V. Sunshine 24/7: As EDS' work stops in one time zone, it picks up in another. *Dallas Morning News,* January 2, 2007 (available at www.dallasnews.com/sharedcontent/dws/bus/stories/010207dnbuseds.3166f5b.html).

18. Gorton, I.; Hawryszkiewycz, L.; and Fung, S. Enabling software shift work with groupware: A case study. In R.H. Sprague (ed.), *Proceedings of the 29th Annual Hawaii International Conference on System Sciences.* Los Alamitos, CA: IEEE Computer Society Press, 1996, pp. 72–81.

19. Gupta, A. Deriving mutual benefits from offshore outsourcing: The 24-hour knowledge factory scenario. *Communications of the ACM, 52,* 6 (June 2009), 122–126.

20. Gupta, A. The 24-hour knowledge factory: Can it replace the graveyard shift? *Computer, 42,* 1 (January 2009), 66–73.

21. Hazzan, O., and Dubinsky, Y. *Agile Software Engineering.* London: Springer-Verlag, 2008.

22. Herbsleb, J.D., and Mockus, A. An empirical study of speed and communication in globally distributed software development. *IEEE Transactions on Software Engineering, 29,* 6 (June 2003), 481–494.

23. Hevner, A.; March, S.T.; Park, J.; and Ram, S. Design science research in information systems. *MIS Quarterly, 28,* 1 (March 2004), 75–105.

24. Highsmith, J. *Agile Software Development Ecosystems.* Boston: Addison-Wesley, 2002.

25. Hildenbrand, T.; Rothlauf, F.; Geisser, M.; Heinzl, A.; and Kude, T. Approaches to collaborative software development. In *Second Annual Workshop on Engineering Complex Distributed Systems (ECDS).* Los Alamitos, CA: IEEE Computer Society Press, 2008, pp. 523–528.

26. Jalote, P., and Jain, G. Assigning tasks in a 24-hour software development model. *Journal of Systems and Software, 79,* 7 (2006), 904–911.

27. Jalote, P.; Palit, A.; Kurien, P.; and Peethamber, V.T. Timeboxing: A process model for iterative software development. *Journal of Systems and Software, 70,* 1–2 (February 2004), 117–127.

28. Kankanhalli, A.; Tan, B.C.Y.; and Wei, K.-K. Conflict and performance in global virtual teams. *Journal of Management Information Systems, 23,* 3 (Winter 2006–7), 237–274.

29. Malone, T., and Crowston, K. The interdisciplinary study of coordination. *ACM Computing Surveys, 26,* 1 (1994), 87–119.

30. Martin, J. *Rapid Application Development.* Indianapolis: Macmillan, 1991.

31. Massey, A.P.; Montoya-Weiss, M.M.; and Hung, Y.-T. Because time matters: Temporal coordination in global virtual project teams. *Journal of Management Information Systems, 19,* 4 (Spring 2003), 129–156.

32. Millson, M.R.; Raj, S.P.; and Wilemon, D.A. Survey of major approaches for accelerating new product development. *Journal of Product Innovation Management, 9,* 1 (March 1992), 53–69.

33. Parnas, D. On the criteria to be used in decomposing a system into modules. *Communications of the ACM, 15,* 12 (1972), 1053–1058.

34. Pressman, R. *Software Engineering: A Practitioner's Approach.* New York: McGraw-Hill, 2007.

35. Ren, Y.; Kiesler, S.; and Fussell, S.R. Multiple group coordination in complex and dynamic task environments: Interruptions, coping mechanisms, and technology recommendations. *Journal of Management Information Systems, 25,* 1 (Summer 2008), 105–130.

36. The rise of India. *BusinessWeek* (December 8, 2003) (available at www.businessweek.com/magazine/content/03_49/b3861001_mz001.htm).

37. Rosenau, M.D., Jr. Schedule emphasis of new product development personnel. *Journal of Product Innovation Management, 6,* 4 (December 1989), 282–288.

38. Saunders, C.S.; Van Slyke, C.; and Vogel, D. My time or yours? Managing time visions in global virtual teams. *Academy of Management Executive, 18,* 1 (2004), 19–31.

39. Setamanit, S.; Wakeland, W.W.; and Raffo, D. Using simulation to evaluate global software development task allocation strategies. *Software Process: Improvement and Practice, 12,* 5 (September–October 2007), 491–503.

40. Smith, P.G., and Reinersten, D.G. *Developing Products in Half the Time.* New York: Van Nostrand Reinhold, 1991.

41. Sooraj, P., and Mohapatra, P.K.J. Modeling the 24-hour software development process. *Strategic Outsourcing: An International Journal, 1,* 2 (2008), 122–141.

42. Taweel, A., and Brereton, P. Modeling software development across time zones. *Information and Software Technology, 48,* 1 (January 2006), 1–11.

43. Treinen, J.J., and Miller-Frost, S.L. Following the sun: Case studies in global software development. *IBM Systems Journal, 45,* 4 (October 2006), 773–783.

44. Yap, M. Follow the sun: Distributed extreme programming development. In *Proceedings of Agile Conference.* Los Alamitos, CA: IEEE Computer Society Press, 2005, pp. 218–224.