

“Follow The Sun” Workflow In Global Software Development

Erran Carmel
American University

J. Alberto Espinosa
American University

Yael Dubinsky
IBM Haifa Research Lab, Haifa, Israel

To appear in Journal of Management Information Systems 2010.

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. While the potential impact on “time-to-market” can be profound, at least conceptually, FTS has enjoyed very 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 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 hand-off 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 article is that understanding calendar efficiency, hand-off efficiency, within-site coordination and cross-site coordination is necessary to evaluation – if FTS is to be successful in reducing software development duration.

Keywords: *duration reduction; calendar efficiency; time to market; handoff efficiency; global coordination; 24-hour development; round the clock development.*

Introduction

Follow The Sun (FTS from here on) is a rather intuitive idea: hand-off work at the end of every day from one site to the next site many time zones away (for example, USA to India), so that the work can be advanced while one's team rests for the night. The potential impact 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 hand-offs is very difficult..." [7]. This difficulty has not changed noticeably in the ensuing years despite improved technologies and methodologies [3]. In this article 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 [7] 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 hand-off 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: *can FTS be more effective in improving development speed – and if so, in which ways?*

Our research question can be further decomposed into more specific unresolved FTS issues, which we begin to address in this article: (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 co-located and conventional global work configurations.

Our goal in this article is to begin to address these issues and provide a conceptual framework to guide further studies of FTS. In the first section we provide a theoretical foundation for the study of FTS. We formally define FTS and then elaborate on related issues, particularly development speed – the primary driver for FTS. Throughout our discussion we formulate propositions. Next, we examine the structure, methodology and architecture of FTS work and then use a field study to compare FTS development speed against a co-located team. Finally, we present a conceptual analytical discussion and introduce three key variables of FTS analysis.

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 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.

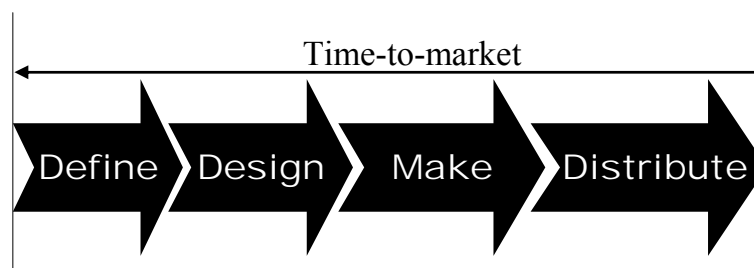


Figure 1. Timeline of time-to-market, measured from inception to use/sale

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 speed-up (e.g., work faster, skip steps, set aggressive deadlines). All these reactive efforts have real costs due to burnout and fatigue [33]. Adding personnel for speed-up is of little interest in software because of the wisdom gained long ago from the seminal Brooks' Law [4] – "adding manpower to a late software project makes it later." 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 development team [33, 37, 40].

The first well-documented global software team specifically set up to take advantage of FTS was at IBM in the mid-1990s [7]. This team was set up from inception to employ FTS, spread out across 5 sites around the globe. However, FTS was unsuccessful. It was uncommon to move the software artifacts daily as had been hoped. Finally, the decision was made that the effort of frequent daily hand-offs (tight coupling) was to be abandoned and collaboration between the sites was reduced to the loose coupling that is common in the vast majority of today's global collaborations.

The first researchers to examine FTS were Hawryszkiewicz, Gorton, and colleagues. They conducted a series of small controlled experiments in the mid 1990s [19] but did not continue their line of inquiry beyond this. Cameron [6] claimed some limited FTS success at the global American firm EDS (now HP), 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 last decade some have claimed successful FTS practices but, on closer inspection, while these projects were indeed dispersed, they did *not* practice the daily hand-offs of FTS [e.g., 43]. We note that this is consistent with the authors' experience in industry: the FTS term is used loosely and upon closer inspection there is no – or very little – FTS. For example, contrary to myth, Indian offshore firms do little FTS [8].

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 [28, 39, 41, 42]. We will return to these models later in this article.

As illustrated in Figure 2a and 2b, globally distributed configurations involve decomposing tasks and allocating them to multiple sites in a way that minimizes dependencies across sites [7, 22]: parallel work¹ or development phase (we note that there are other considerations besides minimizing dependencies, such as location of expertise). For our own shorthand notation we will 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 hand-offs from site to site, whereas the *opposite* is true for conventional global configurations in which an attempt is made to reduce interdependencies and hand-offs as much as possible.

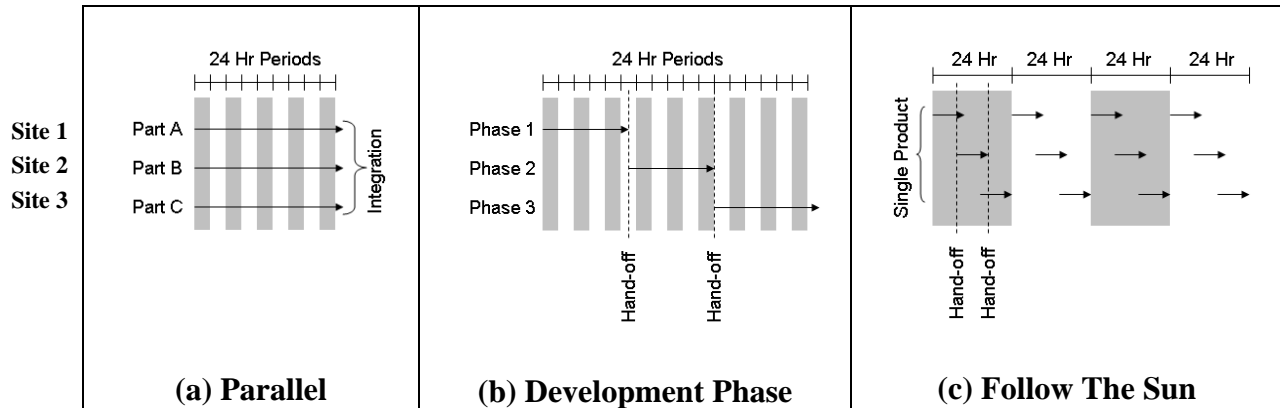


Figure 2. FTS compared to other globally distributed configurations. We refer to parallel- and phase-based as “conventional global configurations.”

Defining and Disambiguating “Follow the Sun”

Based on the foregoing discussion, in this section we propose a formal definition of FTS. A definition is critical since 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 development 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.

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

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. *Hand-offs are conducted daily at the end of each shift.*² This criterion differentiates FTS from conventional global configurations which minimize dependencies and hand-offs 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 work day 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 [20, 21] describes other knowledge-based applications that claim to do FTS—at General Motors and at Office Tiger. Second, the definition is consistent with broader definitions of global collaborative software development across global production sites [26]. 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 6 hours per site of intensive software development per day (“task time”), it is theoretically possible to manage FTS with even 4 sites spread out across time zones of the globe and perhaps even more. Fourth, our definition allows for work time overlap time between sites, if desired, since 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 work days involve parallel work (and there is no FTS hand-off), then these cases could be labeled *mixed* FTS-parallel.

Additionally, we state four key *assumptions* necessary for our definition to be robust: (1) each production site works during its day as a “sub-team” and it needs within-site coordination; (2) a sub-team can consist of one or more members; (3) the hand-off 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, which are not FTS.

Global knowledge work. Global knowledge work is a general label for geographically dispersed knowledge workers working collaboratively across global 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, while 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.

24-hour business processes. Such work arrangements are quite familiar in modern call centers since 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 helpdesks, for example, are set up to provide continuous service coverage around the clock. 24-

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

hour business processes are not the same as FTS because they fail criteria #1 and #4 of the FTS definition above.

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, which 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 itself along with its meta-data.

Co-located multi-shifts. A reasonable alternative to FTS is to choose one location where labor is cheap and run several 8-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 hand-off 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 co-located 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 under-studied in the disciplines of information systems (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 [22, 23] have found that multi-site software teams take longer than co-located teams.

In our own research stream, beginning in 2003, we studied the effects of time separation on speed. Cummings et al [11] 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 hand-off, the effect on delay is negligible. Espinosa et al [15] experimented with time zone variations in a computer lab and found that small increases in time zones (compared to co-located) reduced speed, but as more time zone separation was added, speed increased, suggesting that there are speed advantages to working across time zones. While 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 percent 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 work week utilizes 23.8% of the calendar workweek (40/168). Therefore, the calendar efficiency is only 23.8% 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 compression or improvement factor [6] 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 Column 4, beginning with a typical one-site team (the Baseline), which uses only a dismal 17.9% of the overall calendar time after taking into account non-task 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%. A very heavy Overload mode of 20 hours of weekly overtime raises calendar efficiency to 29.8%, but is not sustainable over a long time period because of employee burn-out.

Shifts/ sites	Descriptor	Calendar efficiency net (task time only)		Calendar efficiency (task + non-task time)		Maximum calendar time per week
		Hours	Percent	Hours	Percent	
Col 1	Col 2	Col 3	Col 4	Col 5	Col 6	Col 7
1	Baseline	30	17.9%	40	23.8%	168 hours
1	Overload (light) = + 10h/wk overtime	40	23.8%	50	29.8%	168 hours
1	Overload (heavy) = + 20h/wk 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 (+ overtime 10h/wk)	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

- **Days per week** = 7; Work days per week are only 5.
- **Hours per day** = 8 hours per shift except for the Overload modes.
- **Overload time** assumes that all additional hours are devoted to task (rather than non-task) activities.
- **Task activities** include all software development work including all meetings and all coordination time.
- **Non-task activities** = 25% of workday. These are activities such as staff recruiting, writing a memo about the previous project, filling out time sheets, or fixing the copier. These are not coordination activities, which we discuss later in this article. We use 25% 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% of the workday [10]
- **Coordination time losses or gains** are ignored in this table and will be introduced in the next section.
- **Time Off.** Company holidays at each site are ignored here. Generally, they represent about 6% of annual work days across nations. Individual absenteeism, as well as individual vacations, is assumed to have negligible effect.
- **Multi-tasking** is ignored.
- **Clarification for FTS with 4 sites.** At this level, 24 hours, or 100% task time is utilized per work day, since non-task activities can be conducted separately and do not detract from calendar usage (i.e. 6 hours of task time and 2 hours of non-task time per site).

Table 1: Calendar efficiency in different work modes

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%. 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:

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

Structural Considerations

In this section we motivate the following issues around FTS: phase specificity, choice of FTS development method, and FTS product architecture. We then illustrate some of these concepts with some 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 [18]. Testing is a good fit because the hand-off is structured, granular and – with trained staff – will usually 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 R&D in India and Silicon Valley, claimed that it performed rapid prototyping using FTS [5]. 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.

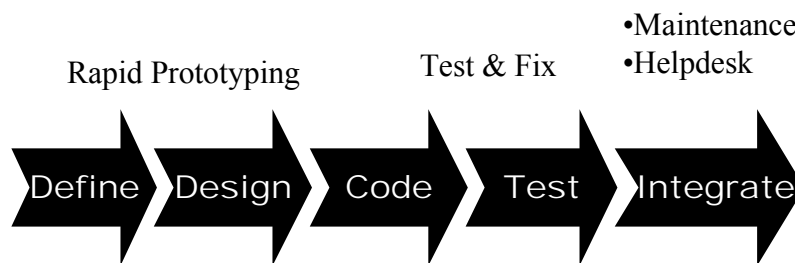


Figure 3. Phase-specific activities that fit FTS displayed above a generic waterfall SDLC.

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:

Proposition 2: *Relative to work that spans multiple SDLC phases, the work within a particular SDLC phase is more suitable for FTS development because its specificity allows for more structured and granular hand-offs.*

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 a FTS software development methodology that spans the entire development process and supports the special needs of daily hand-offs. IBM's classic FTS team of the 1990s constructed a unique organization structure and process [7]. Similarly, Cameron at EDS crafted a special methodological adaptation for FTS [6].

This leads to considering the advantages and disadvantages of linear-sequential approaches (e.g. waterfall, incremental) versus iterative models (e.g. Unified Process, Agile UP, Agile [2, 22, 25, 44]) and how they apply to FTS. The limitations in phase specificity, which we just 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 hand-offs 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 article – for the following reasons. First, it has the enabling property of using short time-boxed iterations of 2 to 4 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 hand-offs. 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 work day is common in Agile teams. It ensures high quality hand-offs, thus it 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, Gupta and colleagues have attempted to conceptually marry FTS with elements of Agile: [12, 20].³ All of which leads to the following:

Proposition 3: *Compared to conventional global configurations, FTS is more suitable 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 that dependencies (coupling) between

³ Denny et al.'s 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.”

development groups be minimized ([34]; also see Figure 2). Once decomposed, the work may then be assigned to different sites based in part on where expertise resides [7, 35], regardless of locations and time zones. This type of decomposition and partitioning is unsuitable for FTS development because there would be very 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 [23]. Top priority MRs involve the development of new features or modifications that have severe impact on client service, either in the form of new critical functionality or repair of critical client services [16]. 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. Our 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, modification requests, 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 [24]. Here, we not only observe the phenomenon, but we 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, in this case, a method/process.

Our exploratory study is described in more detail in a prior article [9] 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 and colleagues have also progressed in a somewhat similar direction in their FTS study in a year-long comparative field study at IBM [20]).

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 20-30. Most of the students were working part time in

software engineering roles in sophisticated firms during. The study task required 400 person-hours per team.

The control teams could interact in any way they wished, including face-to-face and synchronous. 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 hand-offs were allowed between the sub-teams at fixed intervals of time.⁴ Figure 4 depicts the actual staggered activity of the two sub-teams of an FTS team.

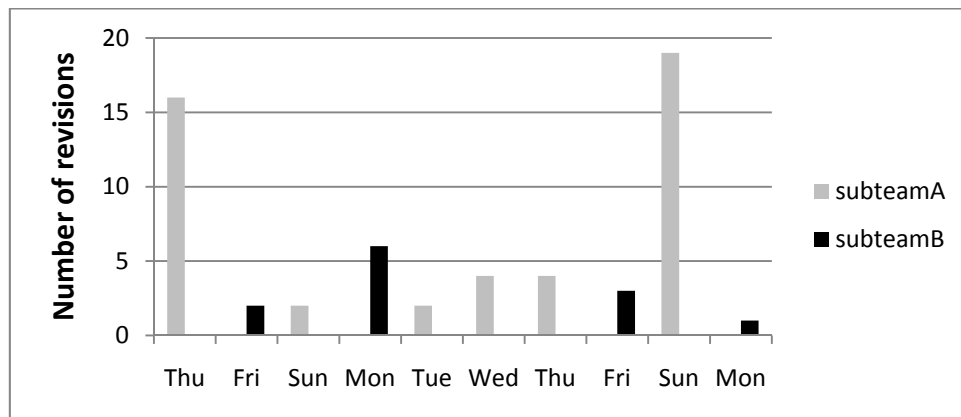


Figure 4: A slice of several days of activity from the FTS team in one of the Agile iterations of our field study (note: number of revisions refers to number of historical versions of source code files).

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 we 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, while both teams in both projects met the project requirements with respect to functionality and level of quality. More importantly, using our proxies for measure of duration reduction achieved by the FTS teams, the first team achieved reduction of approximately 10% and the second-team of approximately 50% as compared to 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 [23] in which distributed teams exhibited longer duration relative to co-located teams. We suggest that our limited, qualified field study success happened because of the Agile implementation, attributed to 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% more documentation lines

⁴ In the first study there were two hand-offs per day; in the second study there was one hand-off per day.

as well as about 25% more messages. In summary, as would be expected with FTS, since 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 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 actually increased, thus reducing task duration. In other words, FTS teams appears to behave differently – they tend to be more disciplined and time sensitive and when time is managed correctly this impacts outcomes positively. In fact, the Agile movement of recent decades advocated and showed 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 hand-offs 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 hand-off process) may be an effective method for a successful FTS practice. The next section further analyzes the importance of efficient hand-offs.

Analytical Discussion

The key to learning whether or not FTS can reduce duration lies in understanding how coordination cost and individual productivity varies when comparing co-located, FTS, and conventional global configurations. In this section we introduce the key FTS variables we use to analyze these issues: within-site coordination time, cross-site coordination time, and individual 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 [30]. 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' 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 they now 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, while coordination is necessary, the time spent coordinating is time diverted away from individual task work. Therefore, we contend that FTS will be beneficial to 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. parallel work within each site) and cross-site coordination (i.e. hand-off 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 hand-off activities from site to site. Due to the difficulty of coordinating and resolving task issues [29] 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 [36]. 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 re-work (i.e., a type of vulnerability cost [13]).

Figure 5: Speed-up as a function of hand-off efficiency for three FTS configurations.

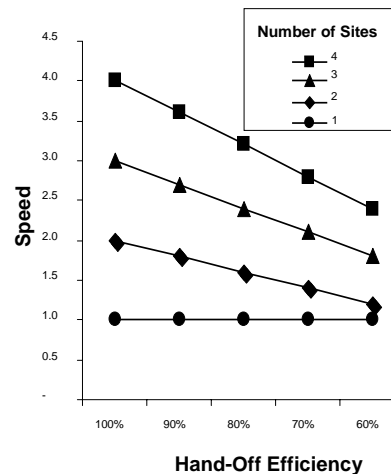


Figure 5 illustrates how hand-off efficiency affects duration. We assume for the moment that the labor resources at each site are fixed and equal and that only cross-site coordination is needed for hand-off. For example, a hand-off efficiency of 90% means that 90% of a developer’s

day is productive and that a total of 10% (on both the sending and receiving side) is devoted to hand-off activities. In the 1-site configuration of Figure 5 (the baseline) there is no hand-off, therefore duration is unaffected by hand-off efficiency. If we added one more site with an FTS arrangement then we would have doubled the speed, but only if the hand-off efficiency was 100%. As the hand-off efficiency declines, the gains in speed by adding further sites also decline because we are introducing cross-site coordination costs. Hence the two propositions:

Proposition 7: *As hand-off efficiency increases, so does FTS development speed.*

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

We make a few more comments about assumptions and factors that influence hand-off efficiency:

- *Hand-off failures.* A hand-off 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 [17].
- *Variable hand-off 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 hand-off 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 amount 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 sub-team so that, per Brooks' 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 sub-teams of size n/s . Therefore, the number of possible within-site dependency links within one sub-team 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:

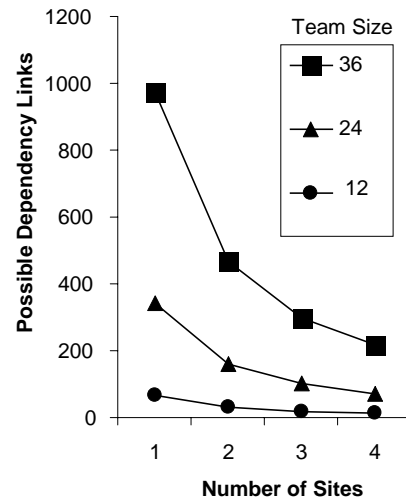
Proposition 9: *The potential for speed gains due to reduced within-site coordination increases exponentially for larger teams if the teams are subdivided into smaller sub-teams across multiple FTS sites.*

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 hand-off 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 best illustrated examining 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.

Figure 6 – Number of Possible Within-site Dependency Links by Site and Team Size



Personal productivity. Up until this point we have assumed that each individual’s productivity is fixed regardless of location and configuration, once non-task time and coordination times are netted out. Other models [27, 28, 42] have also made similar assumptions while in [39] productivity is a group variable that is impacted by coordination. However, based on our exploratory field study, we argue that personal productivity may actually go up in a FTS setup because of “time-boxing” or “daily deadlines.” Time-boxing impacts 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 found to have positive effects on performance [32]. The task (or work unit) is much smaller and easier to scope out. The individual is more focused in his/her work and gets distracted less often. Personal time-boxing curbs perfectionist tendencies, procrastination, and does not allow individuals to over commit to a task. Time-boxing of iterations was advocated in the older notions of *Rapid Application Development* by James Martin [31] and has been one of the foundations of Agile and UP [25]. Agile approaches also set a time-box for daily completion in that all work has to be test-ready. Hence the next proposition:

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 (co-located) configuration to a FTS configuration can be represented as:

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

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 $\Delta CROSS$ will certainly be positive – i.e., duration increases due to cross-site coordination).

- $\Delta WITHIN$ 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 $\Delta WITHIN$ is likely to be negative – i.e., duration decreases due to lower within-site coordination needs as sub-teams become smaller).
- $\Delta PERSONAL$ is the difference in duration resulting from changes in personal productivity alone (recall from the discussion around proposition 11 that that $\Delta PERSONAL$ is likely to be negative – i.e., duration decreases due to time-boxing).

It follows from Eq. 1 that task duration will be reduced when:

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

Stated in words, Eq.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 hand-off 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 would first think; there are quite 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 [12, 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 to other FTS models in the literature

As noted before, four recent studies conducted FTS modeling or simulation [28, 39, 41, 42]. All four studies deal to some extent with the issue of speed, while 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 & Jain [28] set out to understand how to optimally allocate FTS tasks to individual nodes using the methodology of critical path and network optimization; Sooraj P. & Mohapatra [41] set out to understand how to optimally allocate geographical sites in FTS using the methodology of a general sequential mode model; Taweel & Brereton [42] explored FTS sensitivity to overhead and hand-offs 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 article 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 have provided 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 hand-off efficiency in helping understand how to analyze, design and implement successful FTS practices that can reduce task duration. We have developed several testable propositions (summarized in Table 2) surrounding key FTS concepts, including: calendar efficiency; development method; product architecture; hand-off efficiency; and three key variables – within-site coordination, cross-site coordination and personal productivity. It follows from our discussion that hand-off 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, and not on product quality. Finally, we note that while 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.

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

Proposition 2: Relative to work that spans multiple SDLC phases, the work within a particular SDLC phase is more suitable for FTS development because its specificity allows for more structured and granular hand-offs.

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

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

Proposition 7: As hand-off efficiency increases, so does FTS development speed.

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

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 sub-teams across multiple FTS sites.

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.

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.

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 hand-off coordination.

Table 2: Propositions for the study of FTS

References

1. Andres, H. P. & Zmud, R. W., A contingency approach to software project coordination. *Journal of Management Information Systems*, 18, 3, (2002), 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.
4. Brooks, F.P. *The Mythical Man-Month: Readings In Software Engineering*. Reading, MA: Addison-Wesley, 1975.
5. *BusinessWeek*, The rise of India. December 8, 2003.
6. Cameron, A., A novel approach to distributed concurrent software development using a "Follow-the-Sun" technique. *Unpublished EDS working paper*, 2004.

7. Carmel, E. *Global software teams: collaborating across borders and time zones*, Upper Saddle River, NJ : Prentice Hall-PTR, 1999.
8. 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.
9. Carmel, E., Dubinsky, Y. and Espinosa, A. Follow the sun software development: new perspectives, conceptual foundation, and exploratory field study. *42nd Hawaii International Conference on Systems Sciences*, IEEE, 2009, 1-9.
10. Churchville, D. *Agile Thinking: Leading Successful Software Projects and Teams*. San Francisco: Lulu.com, 2008.
11. 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.
12. Denny, N.; Crk, I.; and Nadella, R.S. Agile software processes for the 24-hour knowledge factory environment. In A. Gupta (ed.), *Outsourcing and Offshoring of Professional Services: Business Optimization in a Global Economy*. Hershey, Pennsylvania: IGI Global, 2008, 287-289.
13. Espinosa, J.A. and Carmel, E. Modeling coordination costs due to time separation in global software teams, *International Workshop on Global Software Development*, part of the *International Conference on Software Engineering*, Portland, Oregon, USA, IEEE, 2003, 64-68.
14. Espinosa, J. A., Cummings, J. N., Wilson, J. M., & Pearce, B. M., Team boundary issues across multiple global firms. *Journal of Management Information Systems*, 19, 4, (2003), 157-190.
15. Espinosa, J.A., Nan, N., and Carmel E. Do gradations of time zone separation make a difference in performance? a first laboratory study, *International Conference on Global Software Engineering*, Munich, Germany, IEEE, 2007, 12-20.
16. 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.
17. Espinosa, J. A., Slaughter, S. A., Kraut, R. E., & Herbsleb, J. D., Team knowledge and coordination in geographically distributed software development. *Journal of Management Information Systems*, 24, 1, (2007), 135-169.
18. Godinez, V. Sunshine 24/7: As EDS' work stops in one time zone, it picks up in another. *Dallas Morning News*. January 2, 2007.
19. Gorton, I. Hawryszkiewicz, L. Fung, S. Enabling software shift work with groupware: a case study, *Hawaii International Conference on Systems Sciences*, IEEE, 1996, 72-81.
20. Gupta, A. The 24-Hour Knowledge Factory: can it replace the graveyard shift? *Computer*, 42, 1 (January, 2009), 66-73.
21. Gupta, A. Deriving mutual benefits from offshore outsourcing: The 24-Hour Knowledge Factory scenario, *Communications of the ACM*, 52, 6 (June, 2009), 122-126.
22. Hazzan, O. and Dubinsky, Y., *Agile Software Engineering*, London: Springer-Verlag, 2008.
23. 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.
24. Hevner, A., March, S. T., Park, J., and Ram, S. *Design Science Research in Information Systems*, MIS Quarterly, 28, 1 (March 2004), 75-105.
25. Highsmith, J. *Agile Software Development Ecosystems*. Boston: Addison Wesley, 2002.
26. Hildenbrand, T. , F. Rothlauf, M. Geisser, A. Heinzl, T. Kude, A. Approaches to collaborative software development, *Workshop on Engineering Complex Distributed Systems (ECDS)*. Barcelona: Spain, March 4-7, 2008, 523-528.
27. Jalote, P, Palit, A., Kurien, P., Peethamber, VT. Timeboxing: a process model for iterative software development. *The Journal of Systems & Software*. 70, 1-2, (February, 2004), 117-127.
28. Jalote, P. and G. Jain, Assigning tasks in a 24-h software development model, *Journal of Systems and Software*. 79, 7 (2006), 904-911.
29. Kankanhalli, A., Tan, B. C. Y., & Wei, K.-K., Conflict and performance in global virtual teams. *Journal of Management Information Systems*, 23, 3, (2007), 237-274.
30. Malone, T. and Crowston, K. The interdisciplinary study of coordination, *ACM Computing Surveys*. 26, 1 (1994) , 87-119.
31. Martin, J. *Rapid Application Development*, Indianapolis, IN: Macmillan, 1991.

32. Massey, A. P., Montoya-Weiss, M. M., & Hung, Y.-T., Because time matters: Temporal coordination in global virtual project teams. *Journal of Management Information Systems*, 19, 4, (2003), 129-156.
33. 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.
34. Parnas, D. On the criteria to be used in decomposing a system into modules, *Communications of the ACM*, 15, 12 (1972), 1053–1058.
35. Pressman, R. *Software Engineering: A Practitioner's Approach*. New York: McGraw Hill, 2007.
36. Ren, Y., Kiesler, S., & 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, (2008), 105-130.
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 S.L. Miller-Frost, 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. *Proceedings of Agile Conference*, Denver. Los Alamitos, CA: IEEE Press, 2005, pp. 218- 224.

Acknowledgements

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

Authors

Erran Carmel. 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* is a pioneering book on the topic. His second book *Offshoring Information Technology* came out in 2005 and 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. He is a tenured full Professor at the Information Technology department, Kogod School of Business at American University and has been awarded the International Business Research Professor. In 2008-2009, while writing this paper, he was the Orkand Chaired Professor at the University of Maryland, University College. He has been a Visiting Professor at the University of Haifa (Israel) and at University College Dublin (Ireland).

J. Alberto Espinosa. Professor Espinosa is an Associate Professor of Information Technology at the Kogod School of Business, 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 spatial and temporal distance. His current research areas include coordination of technical work across time zones and coordination in large-scale technical collaboration tasks like 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 at 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. Yael received her B.Sc. and M.Sc. in computer science and PhD in science and technology education from the Technion, Israel. Her research interests involve aspects of software engineering and information systems. Yael has significant experience guiding agile implementation processes in both industry and academia. She has presented her research at ICSE, Agile, and XP conferences. Her book 'Agile Software Engineering', co-authored with Orit Hazzan, was published by Springer in 2008.