# Follow The Sun Software Development: New Perspectives, Conceptual Foundation, and Exploratory Field Study

Erran Carmel
Univ. of Maryland, Univ. Coll.
ecarmel@umuc.edu

Yael Dubinsky
IBM Haifa Research Lab
dubinsky@il.ibm.com

Alberto Espinosa
American Univ.
alberto@american.edu

## Abstract

*Follow The Sun (FTS) is a special case of global software development. FTS means that software work is handed off every day from one development site to the next -- many time zones away. The main benefit is reduction in development duration. Surprisingly, unlike the broader trend of offshore outsourcing, FTS is practiced rarely and misunderstood often.*

*In this article we present a foundation for understanding FTS including a definition, a description of its place in the life cycle, and choice of methodologies. We also present the outcomes of a first quasi-experiment designed to test FTS and measure the speed of software work. This quasi-experiment is part of our comprehensive research to explore FTS and its implications.*

## 1. Introduction

Follow The Sun (FTS from hereon; also called: 24-hour development and round-the-clock development) is a rather simple idea: Hand-off work every day from one site to the next as the world turns (USA to India, for example). Thus, reduce the development duration by 50% if there are two sites and by 67% if there are three sites.

FTS is a *subset* of global software development and shares with global software development the many issues and challenges of coordination, culture, and communication. However, FTS, is uniquely focused on *speed* in that the project is designed to reduce cycle-time (also known as time-to-market reduction, or duration reduction). Unfortunately, there have been few (if any) successful cases of FTS due to coordination difficulties. Meanwhile, our Management Information Systems (MIS) literature has devoted some attention to investigating the time domain but has largely focused on perceptions of time [25] rather than approaches to increasing speed.

### 1.1 Why is Time-to-market interesting?

FTS team structures are configured in order to achieve time-to-market *reduction*.[1] Time-to-market is the length of time it takes from a product conception until the product is available for use or sale [26]. Figure 1 depicts this definition. Time-to-market is most important in industries where products become outmoded quickly, such as, these days, mobile telephone handsets. Therefore time-to-market is critical for handset software. Time-to-market is also important for strategic information systems such as competitive e-commerce systems or innovative Supply Chain Management systems.

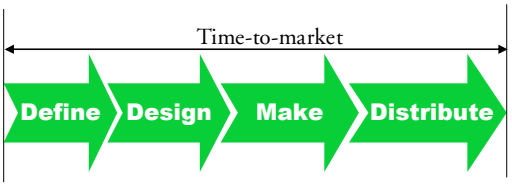


Figure 1. Timeline of time-to-market, measured from inception to use/sale

A desire for rapid development -- a sense of urgency -- is shared by most firms and projects in a competitive marketplace, but such a desire usually tends to be a reactive behavior rather than a proactive one. We see FTS as a proactive way to achieve time-to-market, hence we emphasize what FTS is not. In software work, reactive techniques include: speed-up, setting deadlines, and adding personnel. Speeding up, or "stepping on the gas," means doing what needs to be done -- only faster. This is generally not recommended because of fatigue [22]. Adding personnel is of little interest in software because of the wisdom gained from the seminal Brooks' Law [3]

[1] Time-to-market reduction needs to be seen within the larger context of other competitive factors and project goals: One needs to make possible trade-offs between features, project cost, and quality [24].

978-0-7695-3450-3/09 $25.00 © 2009 Crown Copyright

("adding manpower to a late software project makes it later"). In contrast, FTS is set up proactively with a high awareness of time-to-market [5] by rethinking development processes. In summary, time-to-market reduction achieved by using FTS requires a deliberate design around the objective of speed.

Time-to-market is an important area of inquiry because it is relatively understudied in the disciplines of MIS and software engineering (an exception is found in [16] where multi-site software teams took longer than in co-located teams).

### 1.2. Follow the sun literature and history

The first global software team specifically set-up to take advantage of FTS was at IBM in the mid-1990s and is documented extensively in [7]. This team was set up from inception to work in FTS. It was spread out in 5 sites across the globe. However, FTS was difficult to achieve. It was uncommon to move the software artifacts daily as had been hoped. Since FTS was not working the managers gave up on that objective rather quickly while continuing global software development.

Others have claimed implementing FTS but did not succeed in implementing it [31][30]. Cameron [4] has claimed some limited success at FTS at the global American firm EDS (now HP). Hawryszkiewycz & Gorton were the first to examine FTS in a series of small controlled experiments [13] though they did not continue their line of inquiry. In recent years others have written about the promise and failures of FTS [14][15][8][2][13]. Contrary to myth, Indian offshore firms do little, if any, FTS [6].

With rather limited progress in theory or practice in the last decade, recent FTS literature has moved in another interesting trajectory: mathematical modeling of FTS [11][20][27][28][29]. Each one of these papers uses somewhat different approaches and assumptions. All of them focus, to some extent, on the issue of speed, while making critical assumptions that deal with the hand-over/ coordination issues.

## 2. Follow the sun: definition and practice

Central to FTS is the hand-off of work from one site to the next. Most global teams do the utmost to do the exact opposite – to reduce the number of hand-offs as much as possible. See Figure 2 for typical configurations of global teams. Note that the other three approaches, other than FTS, attempt to minimize hand-offs.

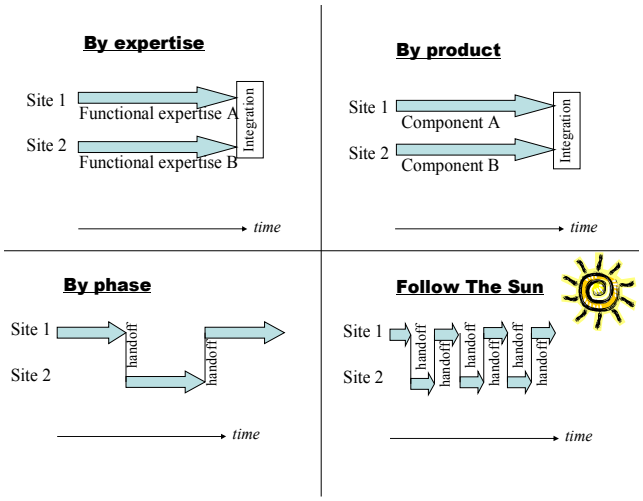


Figure 2. FTS compared to other globally distributed configurations.

FTS is the least common of these global configurations. Partially because of its rarity, we have noticed confusion about FTS in industry. The two most common misconceptions are that FTS is:

a. Global knowledge work. Knowledge workers are working around the world. However in most cases these knowledge workers have little dependency and do not pass on work in order to reduce duration. *Therefore this is not FTS.*
b. 24h work/ shift work (3 shifts a day). A call center scheduler routes calls to workers as the earth revolves. However in most cases these knowledge workers have little dependency and do not pass on work in order to reduce duration. *Therefore this is not FTS.*

### 2.1 FTS definition

In light of the above, we propose a formal definition of FTS. We claim that software development FTS means satisfying all 4 of these criteria:

1. Production sites are far apart in time zones.
2. One of the main objectives is duration reduction/ time-to-market reduction.
3. At every point of time there is only one site that owns the product. (We must specify this criterion in order to make the dependency unambiguous).
4. Hand-offs are conducted daily, where a hand-off is defined as a check-in of a work unit that the next site is dependent upon in order to continue. (We specify this in order to make the dependency unambiguous).

Therefore, FTS is defined as:

> *A type of global knowledge workflow, designed in order to reduce project duration, in which the knowledge product is owned and evolved by a production site and is handed-off daily to the next production site that is many time-zones apart to continue that work.*

We also state one key assumption and several other points. The key assumption is that each production site works during their day because most people around the world tend to work during the day and sleep at night. Related to this is an assumption that each production site works as a team and coordinates inside the team.

Some other foundational points and assumptions:

1. In software work there is one common repository. This allows all sites to "commit" the code/objects at the end of the workday.
2. Exception condition: the unit of work can be sometimes = zero, in cases of holidays etc. While work is usually passed on daily, this cannot happen for every team every day.
3. A team can consist of one or more members.
4. Since Time To Market is a major goal then it is measured using objective units and possibly benchmarked.

Our definition is consistent with other, broader definitions of global software development. For example, it assumes full *cooperation* between production sites, as defined by [19].

Our careful definition allows us to expand our thinking of FTS. We can all envision FTS with 2 or 3 sites. Assuming 6 hours of intensive software development per day, it is possible to do FTS with even 4 sites spread out across time zones of the globe. It may even be beneficial. We assume that each team will work intensively for 6 hours and during the beginning and end of day will perform other admin/work tasks.

## 2.2. Why is FTS so difficult?

Globally distributed development adds difficulties to productive work. The frenetic pace of development will tend to pull apart teams by Carmel's [7] five centrifugal forces of global software teams: loss of communication richness, coordination breakdown, loss of 'teamness', cultural differences, and geographic dispersion.

FTS is even more difficult and quite uncommon because the production teams are *handing off* work-in-progress (unfinished objects). The production objects require daily "packaging" so that the task is understood by the next production site without synchronous interaction.

Naturally there are times when the next production site needs more information. When clarification is required then an entire day may be wasted because the previous site has already gone home. Sometimes a misunderstanding causes the need for re-work. This is also wasted time, or more formally known as vulnerability costs.

## 2.3. What methodology is best for FTS?

Those who have examined FTS closely have recognized the importance of selecting and adapting a methodology for the special needs of daily hand-offs. FTS requires very careful considerations of methodology and process. Cameron at EDS crafted a special methodological adaptation [4]. Similarly, IBM's classic FTS team of the 1990s constructed a unique organization structure and process [7].

Some practitioners have told us that waterfall-like approaches seem best for FTS. Others have proposed other process models.

There are many special conditions that seem even more acute for successful FTS. Clearly, software engineers will need to be trained in cooperation and collaboration skills. Managers cannot micro-manage or they will lose the 24-hour capability. Reward systems balancing individual recognition and team recognition issues would have to be worked out.

We considered several development methods including iterative methods (e.g., RUP) and Agile methods [1][17]. We selected the Agile approach as promising for FTS—and we explain why below. Later, we describe our first FTS exploratory experiment using an Agile approach.

**2.3.1 FTS with waterfall-like approaches.** Let's begin by examining the generic software development lifecycle (SDLC) phases. While some phases can work easily for FTS, at least over short periods (see Figure 3), others do not.

Testing has worked well in FTS for many companies. One team searches for bugs, documents these in the bug database, which is then accessed and worked on by the software team at another site. Testing works well with FTS because the hand-off is structured, granular-- and with trained staff-- will usually not suffer from miscommunications. Prototyping is another specific phase that suits FTS.

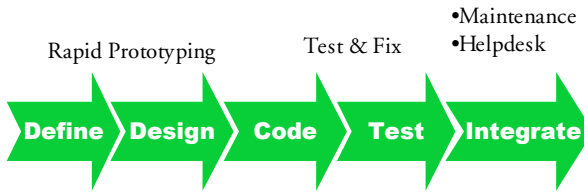


Figure 3. Generic waterfall SDLC. Notation above the arrows points to specific phases that are a good fit for FTS

The peculiarities in each phase means that the theoretical advantages of FTS (reducing time-to-market by 50%), tend to be in isolated phases. If one manages to perform FTS in one specific phase, it does not mean that those techniques are transferable to the other phases. Thus, if only testing used FTS successfully then overall project duration is reduced by only 12.5% rather than the ideal 50% (see Figure 4).

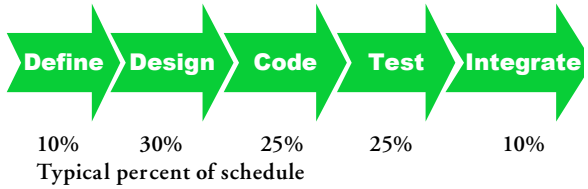


Figure 4. Typical percent of project schedule duration in each SDLC stage

**2.3.2 FTS with the Agile approach.** There has been some research on synthesis of distributed global teams and the Agile approach [e.g., 18]. We focus on FTS with Agile. In the case of FTS, examining the SDLC when using the Agile approach shows that in every agile iteration all software development activities (define, design, code, test, integration) are performed for a smaller scope which is usually feature-based (conceptually described in [1][23]). The iteration length varies from two weeks to four weeks depending on the specific agile method that is used. We argue that in FTS agile environment, testing of small portions along the way should lead to a duration reduction of at least 12.5% as in waterfall-like approaches (see Section 2.3.1), and possibly more if we take into account the accumulative effect of testing on the development process and software quality (stability, maintainability, etc.).

We further argue that the Agile approach has some characteristics that assist in structuring FTS settings:

- **Support daily hand-offs.** Continuously integrating 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* (all test pass) at the end of the work day is common in Agile teams and nicely fits with FTS requirements. Note that we mainly focus on the technical side of the code integration versus practices like continuous coordination [23] that involve other aspects. We further note that also other approaches are iterative and can handle daily hand-offs, e.g., RUP and specifically Agile RUP which can fit in this case.
- **Deal with communication.** The Agile approach places emphasis on communication espousing face-to-face. We advocate this kind of communication as intra-site communication. However, in FTS settings we aim at reducing the inter-site communication between production sites to the bare minimum, relying more on tools and mechanistic processes to support the technical, managerial, and social aspects of development.
- **Elicit cooperation / collaboration.** Collaboration is vital in software development processes [19]. The Agile approach emphasizes collaboration among all people involved in the development -- and especially the customer. In FTS settings, the collaboration significant in the way team members keep the process rules, such as protocols to allow the next site to be able to continue the work with minimum inter-site synchronous communication.

## 3. Long-term research strategy for FTS

We plan to investigate FTS in different settings with the goal of improving measures -- on one hand -- and deriving actionable practical lessons on the other hand. In the first quasi-experiment described in Section 4 we have taken one step in setting this foundation.

What needs to be done? Given the paucity of knowledge in both theory and practice, we propose five action points:

- Controlled lab projects. Study FTS under controlled conditions. See for example some similar research in [12].
- Quasi-experiments. In spite of limitations, quasi-experiments allow for learning and refinement of research techniques (such as controlled experiments) and practical FTS techniques. We see the quasi-experiment as a first step in systematically studying FTS using multiple research methods. We describe our first quasi-experiment further below.
- Expand on a meta-study of FTS. Highlight true successes from industry. With the caveat that there are very few of these, we do see promise. The problem in such a study is that an industry FTS project is quite unlikely to have benchmark data:

what would be duration in "normal" development conditions versus FTS?

- Develop research frameworks for time separation and create mathematical and simulation models to better understand how to optimally architect FTS and how to evaluate its costs and benefits.

## 4. First quasi-experiment

In this introductory article we describe our first step in the broader long-term research strategy. That is, a first quasi-experiment (also called a *field experiment*). A quasi-experiment is a *research design having some -- but not all -- of the characteristics of a laboratory experiment*. Our experiment is focused on measuring speed in software work conducted by distributed teams working according to the Agile software development approach. We describe our design in a manner consistent with [21].

**Experiment goals:** investigate FTS in an Agile environment. Specifically, measure duration in order to test the central premise of time-to-market reduction.

**Experiment participants and setting:** Subjects 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 their studies.

The experiment was conducted as part of practicum course in software engineering. This was a semester-long course of fourteen weeks. The project was the principal component of the course grading, so students were motivated to do well. During the entire process of development there was an active electronic website for course and group communication. The following Agile practices were integral to the project and student learning: time boxing, testing, continuous integration, and constant customer availability for requirements clarifications.

The general approach of a semester project was not new [9]. It had been guided by one of the authors every year since 2002. The only new dimension, this time, was to impose a FTS approach on one of the student groups.

**Experimental task:** The groups were required to build a simulator of processes and threads scheduler in the Unix operating system. There was no programming language that was enforced. There was a requirement to use a CVS[2] server for the integration build. Both groups developed a software project on the same subject and same functionalities.

**Experimental Design:** There were two project teams which were measured: one FTS team of 8 students and one control (CO) team of 7 students. Thus, we had two data points.

**Special Rules for the FTS team:** While the CO team worked in a typical student manner without any time constraint, the FTS teams were split into two subteams and had very strict communication rules in order to simulate the time differences of FTS.

The FTS team was divided into two subteams of 3 and 5 students to simulate working in two non-overlapped time zones. The team of 3 students (we shall call it SubTeamA) worked between 21:30 and 11:00 while the team of 5 (SubTeamB) worked between 11:30 and 21:00. The 30-minute gap permits overtime of 30 minutes in each team. Most students also had demanding part-time jobs, so the odd hours were not a hardship.

The FTS team was required to perform the task using the same effort (in man-hours) in 2.5 weeks instead of the 5 weeks that the control team had.

The academic supervisor operated in the same time zone as SubTeamA for the entire semester. Weekly meetings were planned to be face-to-face with SubTeamA and over phone with SubTeamB.

Direct communication was prohibited between SubTeamA and SubTeamB: no voice, no IM, no SMS.

Every access to the CVS integration server was only allowed in prescribed working hours, e.g. a SubTeamA student could only access the CVS in the 13.5 hour period of the day allotted to SubTeamA.

**Measuring duration in FTS:** Our main objective was to measure differences in project duration. Given the constraints of a class project we had to find a reasonable way to do this and we did this by artificially imposing a deadline at exactly the halfway point for the FTS team. Figure 5 illustrates it. The logic is that 2 FTS teams would theoretically reduce duration by half = 50%.

Note that both teams, CO and FTS, had the same amount of development hours (effort) and the same functionalities to develop.

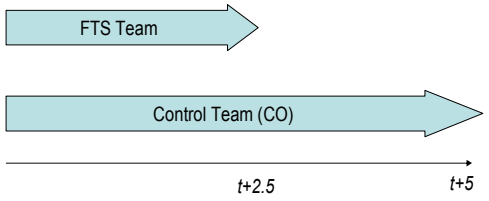


Figure 5. Planned duration for measured iteration

[2] See http://sourceforge.net/cvs/?group_id=3630.

**Other measures and data:** The experimental data were generated by the CVS server, e.g. number of check-in operations and number of revisions per file. Further, we examined the electronic forum for work logs compiled by students and we analyzed students' reflections during the semester.

## 4.1 Experimental outcomes

Our primary goal was to examine development duration. We conclude that there was an approximate 10% reduction in development duration -- rather than the theoretical 50% of FTS. That is, instead of completing the development work at *t+2.5* weeks, it was completed at *t+4.5* weeks.

The FTS team had no incentive to stop at the 2.5 week FTS milestone. So, with most functionality done by the 2.5 week milestone, they kept on tweaking and improving their software for much of the "extra" 2.5 weeks. While this was a weakness in the experimental design, it points to the relative success of the team in duration reduction.

Examining the last half week before the presentation, The CO team performed 385 check-in operations with average of revision level of 8.8, while FTS team performed 336 check-in operations (13% less) with average of revision level of 18 (100% more). Meaning the FTS group changed less files with much higher refinements level. In other words, the FTS team was tweaking, while the CO team was performing coarser, more fundamental tasks.

It is very important to emphasize that both teams (CO and FTS) met the project's requirements with respect to the functionality and level of quality.

We note some additional data in the remainder of this subsection.

**Working hours.** We checked the data from the CVS server to see if the artificial time-zone rules were obeyed. They were. Figures 6 and 7 present file check-in operations for CO and FTS which spanned from 24.12.07 to 28.1.08.

Examining the graphs we found the following:

- CO generally worked while the sun was up, mostly from about 9:00 and until about 22:00. The 7 students performed 754 check-in operations (average of 108 operations per student).
- FTS kept the time zone rules. We can see clearly that both time zones are occupied. The lines across 09:00 and 21:00 mark the artificial time zones. Note that SubteamA worked between 21:30 and 11:00 while the SubteamB worked between 11:30 and 21:00. Therefore there were more operations between 11:30 and 21:00 since that subteam had more students. In total FTS had 8 students and they performed 971 check-in operations (average of 121 operations per student).

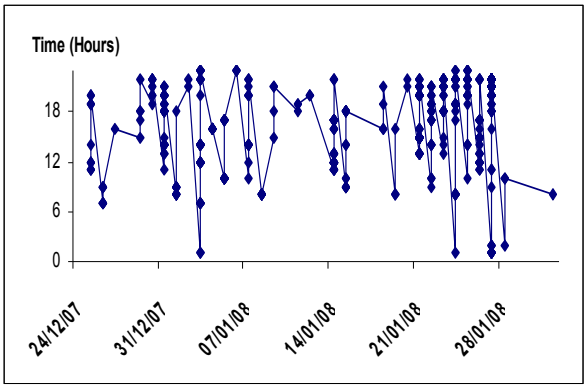


Figure 6. File check-in operation in for Team CO

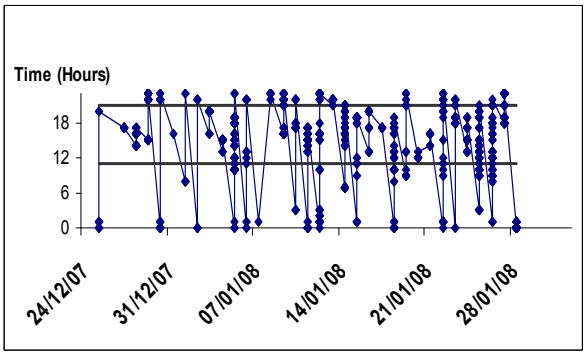


Figure 7. File check-in operation in Team FTS

**Revision level.** Figure 8 shows revision information during the critical last week. The revision level indicates the level of refinement, where a high number is a very fine revision. A low number is an early revision.

As can be observed, the levels of revisions in FTS are higher, meaning time was invested to revisit further issues and make advanced refinements. All together CO had 515 check-in operations in a lower level of revision whereas FTS had 400 check-in operations (28% less).

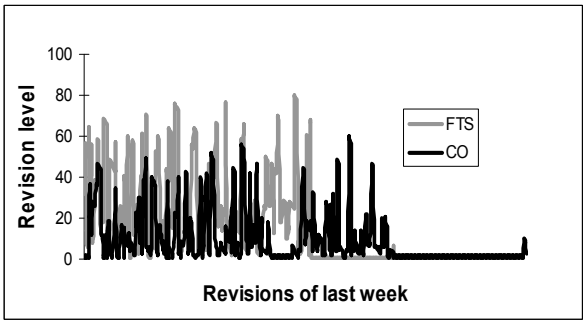


Figure 8: Revision level of last development week

**Role scheme.** Each of the students had a personal role in addition to his development work [10]. Typical roles were designer, tester, and tracker. We began the semester with a specific role scheme that we had used in previous years. However, the FTS students asked at *t+2* weeks that they need another role – a liaison person. Specifically, the students asked to add the liaison role in the two subteams such that the two liaisons will be in charge of the communication and coordination between the two subteams.

The liaison role is well-known in distributed environments [7]. In this case, it shows us that the students keep the communication rules since they encountered problems that occur in such environments.

**Development log documentation.** When asked to reflect on the process, the FTS students described the special way they hand 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 FTS. This is to be expected since each subteam needed to inform the other subteam about their main changes. Metaphorically, we like to call this the *passing of the baton* – just as one does in a relay race. A sample dialogue appears in Appendix A.

### 4.2 Limitations of first quasi-experiment

As we noted before, quasi-experiments are conducted as stepping stones for learning and refinement and naturally have many limitations to generalization. The main limitations were, first, that we had a very small sample. Second, our measures were imperfect. We forced a duration on the FTS team and had to derive a number for the duration reduction.

We list additional secondary limitations: students self-selected to the experimental condition. Students could have "cheated" (although we monitored them and are rather certain that they did not).

In a real world situation, there are about 8 iterations in an Agile release before producing a deliverable. Since the experiment was in an academic setting we had to ask for a deliverable after only 1 measurable iteration. This compression of the project life-cycle led to the situation where the main time span we measured, of the students' iteration, included additional tasks which are not usually done in a typical "real" iteration.

We note refinements for future quasi-experiments. First, since our students were perfectionists and continued working even past the deadline we shall enforce the deadline, perhaps through incentives. Alternatively, we have also designed a staggered timeline for the FTS team that reduces the programmers' daily work window. In either case, we have learned from the first experiment and our future measures will be stronger.

Second, we will introduce some of our own process learning into the next FTS teams by imposing some process improvements. We could enforce the liaison role in each subteam and the development log documentation as coordination and synchronization tools.

## 5. Conclusions

Our goal was to explore FTS and its implications on speed in software development environments. We presented the details of the FTS concept and the outcomes of a first quasi-experiment designed to test FTS and measure the speed of software work.

We point to two important conclusions that emerged and need further investigation.

The first conclusion relates to our finding that both co-located and FTS teams completed their work in the same time. This is in contrast to previous results like in [16] in which the software work of distributed teams takes longer than in co-located teams. We suggest that this happened in our case because of Agile implementation-- specifically the tight iteration.

The second conclusion deals with the continuous integration practice. We found out that implementing a work procedure that includes developing small pieces of code and test and continuous check-in every day into one common repository may be significant to the professional FTS hand-offs.

## 6. References

[1] Beck, K. Extreme Programming Explained. Addison-Wesley, 2000.

[2] M. Betts, "24/7 global application development? Sounds good, doesn't work", Computerworld. September 16th, 2005.

[3] Brooks, F.P. The mythical man-month: readings in software engineering. Reading, MA: Addison-Wesley, 1975.

[4] A. Cameron, "A Novel Approach to Distributed Concurrent Software Development using a "Follow-the-Sun" Technique."", Unpublished EDS working paper, 2004.

[5] E. Carmel, "Cycle-Time In Packaged Software Firms", Journal of Product Innovation Management, 12(2), March, 1995.

[6] E. Carmel, "Building your Information Systems From the Other Side of the World: How Infosys manages time differences", MIS Quarterly Executive, 5(1), 2006.

[7] Carmel, E. Global Software Teams: collaborating across borders and time zones, 1999. Published by Prentice Hall-PTR.

[8] Denny, Nathan, Igor Crk, and Ravi Sheshu Nadella, *Agile Software Processes for the 24-Hour Knowledge Factory Environment, in Gupta, A.* :Outsourcing and Offshoring of Professional Services: Business Optimization in a Global Economy, Publisher: IGI Global, Hershey Pennsylvania, 2008.

[9] Y. Dubinsky, and O. Hazzan, " The construction process of a framework for teaching software development methods", Computer Science Education, 15:4, 2005, pp. 275–296.

[10] Y. Dubinsky, and O. Hazzan, "Using a Role Scheme to Derive Software Project Metrics", Journal of Systems Architecture, 52, 2006, pp. 693–699.

[11] J.A. Espinosa, and E. Carmel, "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, May, 2003.

[12] J.A. Espinosa, N. Nan, and E. Carmel "Do Gradations of Time Zone Separation Make a Difference in Performance? A First Laboratory Study", International Conference on Global Software Engineering, Munich, Germany, August 27-30, 2007.

[13] I. Gorton, I. Hawryszkiewycz, L. Fung: "Enabling Software Shift Work with Groupware: A Case Study", HICSS (3) 1996: 72-81.

[14] A. Gupta, S. Seshasai, and R. Aron, "Research Commentary: Toward the 24-Hour Knowledge Factory - A Prognosis of Practice and a Call for Concerted Research" (November 19, 2006). Eller College of Management Working Paper No. 1038-06 Available at SSRN: http://ssrn.com/abstract=946012

[15] A. Gupta, Deriving Mutual Benefits from Offshore Outsourcing: The 24-Hour Knowledge Factory Scenario" To appear in Communications of the ACM, 2008.

[16] J. Herbsleb, J., et al. "An Empirical Study of Global Software Development: Distance and Speed", in 23rd. International Conference on Software Engineering (ICSE). 2001. Toronto, Canada: IEEE Computer Society Press.

[17] Highsmith, J. Agile Software Development Ecosystems. Addison Wesley, 2002.

[18] T. Hildenbrand, M. Geisser, T. Kude, D. Bruch, T. Acker, "Agile Methodologies for Distributed Collaborative Development of Enterprise Applications", In workshop on Engineering Complex Distributed Systems (ECDS), Complex, Intelligent, and Software Intensive Systems (CISIS), 2008, pp. 540-545.

[19] T. Hildenbrand, F. Rothlauf, M. Geisser, A. Heinzl, T. Kude, "Approaches to Collaborative Software Development", In workshop on Engineering Complex Distributed Systems (ECDS), Complex, Intelligent, and Software Intensive Systems (CISIS), 2008, pp. 523-528.

[20] P. Jalote, G. Jain, "Assigning tasks in a 24-h software development model", Journal of Systems and Software 79(7): 904-911 (2006).

[21] B.A. Kitchenham, S.L. Pfleeger, L.M. Pickard, P.W. Jones, D.C. Hoaglin, K. El-Emam, J. Rosenberg, "Preliminary Guidelines for Empirical Research in Software Engineering", IEEE Transaction on Software Engineering, 2004.

[22] M.R., Millson, S.P. Raj, and D. A Wilemon, "Survey of Major Approaches for Accelerating New Product Development", Journal of Product Innovation Management, 9(1): 53-69 (March, 1992).

[23] D. Redmiles, A. van der Hoek, B. Al-Ani, T. Hildenbrand, S. Quirk, A. Sarma, S. Silveira, R. Filho, C. de Souza, E. Trainer, "Continuous Coordination: A New Paradigm to Support Globally Distributed Software Development Projects". In Wirtschafts Informatik, (49) , 2007, pp. S28-S38.

[24] M.D. Jr. Rosenau, "Schedule emphasis of new product development personnel", Journal of Product Innovation Management, 6(4): 282-8 (December, 1989).

[25] C.S. Saunders, C. Van Slyke, and D. Vogel, "My Time or Yours? Managing Time Visions in Global Virtual Teams," Academy of Management Executive, 2004, Vol. 18, No. 1, 19-31.

[26] Smith, P.G. and Reinersten, D.G. Developing products in half the time. New York: Van Nostrand Reinhold, 1991.

[27] Setamanit, Siri-on, Wayne W. Wakeland, David Raffo. Using simulation to evaluate global software development task allocation strategies. Software Process: Improvement and Practice, Vol. 12, Num 5, September/October 2007, 491-503.

[28] Sooraj. P. and Mohapatra, Modeling the 24-hour software development process, working paper Indian Institute of Technology, Kharagpur, (2008).

[29] A. Taweel and P. Brereton, "Modelling Software Development across Time Zones", Information and Software Technology, Volume 48, Issue 1 , January, 2006.

[30] J.J. Treinen, S.L. Miller-Frost, "Following the Sun: Case Studies in Global Software Development", IBM Systems Journal, 45(4), October 2006.

[31] M. Yap, "Follow the sun: distributed extreme programming development", Proceedings of Agile Conference, 2005.

## Appendix A: Quasi-experiment commit log messages

We illustrate passing the baton using messages taken from the electronic forum of the FTS students. The following are commit log messages that the students used in order to share information and coordinate among subteams. The first message is sent by a student of SubTeamA followed by a message of a student in SubTeamB, and then two messages of SubTeamA.

- From a student in SubTeamA:
  Date: Mon, **14 Jan 2008 00:26:00**
  Subject: Commit log - 14/01/2008
  RunDB:
  - Add suspendThread
  - Add resumeThread
  Logic Package:
  - remove unused peek method from all algorithms.
  Added tests to algorithms.
- From a student in SubTeamB
  Date: Mon, **14 Jan 2008 21:27:36**
  Subject: commit log 14.1.07
  UpdateRunInfo - fixed bugs and put updateRunQueueList in comment (causing exceptions)
- From a student in SubTeamA:
  Date: Mon, **14 Jan 2008 23:37:11**
  Subject: commit log 14/01/08
  The "Done" button in addProcessorShell is fixed accordingly "Meeting points"
  There is some problem in addProcessShell with "Process Memmory" text box.
- From a student in SubTeamA:
  Date: Tue, **15 Jan 2008 03:05:36**
  Subject: Re: commit log 14/01/08
  Hi all, today - unfortunately - short day. Tomorrow I'll do better.
  Log:
  SimulatorGUI:
  - Repaired bug in editIODevice - now, when updating, linked process get updated too.
  …
  Process memory works fine. It has been integrated together with the available free memory in the system. No memory - no data entry.
  In the above commits - the two items in the paper written today have been solved.
  Good night.