Matching expectations: When culture wreaks havoc with global software development

Yvonne Hsieh,MaSc student
Philippe Kruchten, Professor, P. Eng
Eve MacGregor, MaSc student, EIT
Electrical and Computer Engineering
University of British Columbia
2332 Main Mall
Vancouver, BC, Canada
V6T 1Z4

Abstract

In this research, we examine how intercultural factors affect—positively or negatively—the outcomes of software development practices. In the past decade, the North American and Western European IT industries have observed a rapid increase in the number of companies either outsourcing software projects for development abroad or starting their own development centers in remote locations. In spite of great promises and anticipation, many global software development projects fail. After failures, one party is quick to blame the other's perceived lack of diligence, commitment, or ability; or to blame technology. But we observed that projects often fail because of subtle intercultural issues that impact the effectiveness of coordination in the distributed team. To explore this matter, we examine the concept of culture and the potential impact of intercultural dynamics on global software development projects.

There has been little analytical research done in this area and the effect of intercultural factors has, thus far, been assessed based on anecdotal accounts by project managers. Our research combines the grounded theory and case study research, starting with a collection of critical incidents in global projects. We present a descriptive

conceptual framework, for coordination between individuals and teams, that has emerged from our data and use it to analyze and explain some of our findings.

1. Introduction

It is far easier to globalize, "virtualize," outsource, or offshore activities that involve only bits (of information) as opposed to atoms (of materials). Bits move quickly and inexpensively, whereas atoms, in the forms of foods, manufactured goods, or raw materials, need to be shipped slowly and at great expense. The development of software systems has therefore emerged as a prime candidate for becoming a globally distributed activity.

The growth of global software development is propelled by a number of enablers and promised benefits (e.g., reduced labour costs, large skill pools). Creating software in the distributed environment is a challenging endeavor that is confronted by spatial, temporal, organizational, technological, and cultural nuances and complexities (Herbsleb & Moitra, 2001). Researchers and practitioners have recognized the need to more systematically study these issues and their impacts on software development (Damian & Moitra, 2006; Herbsleb & Moitra, 2001). In our study, we focus on the issue of culture and its role in globally distributed software development work. This paper looks at the effects of intercultural factors, presents findings from a study of large software projects, and proposes the AxE model, a conceptual framework for analyzing coordination problems in software development. Section 2 presents the motivation and context for our study, Section 3 outlines examples of culturally-based coordination mishaps, and Sections 4 and 5 discuss in greater detail the central research problem and our research strategy. In Section 6 we discuss the specifics and applicability of the AxE model, and

Section 7 uses the model to interpret the mishaps we have observed. Section 9 discusses potential research directions, and gives our conclusion.

2. Motivation

Software engineering (i.e., the engineering of software-intensive systems) is a highly knowledge-intensive activity. Unlike other engineering disciplines, this "soft" nature makes it hard to modularize a software system into clear-cut components that can be assigned to and developed by different individuals and then later assembled. Rather, there is consistently a high level of interdependence among the many activities and work products involved in software development. Software engineering is also a young discipline with few established practices and in which methods, notations, processes and tools are rapidly evolving (Kruchten, 2004). This "technological churn" (Booch, 2002) further slows down the emergence of a common, well-understood way to proceed.

Originally, software was developed by collocated teams. Based on these collocated projects, a body of knowledge on how to develop software has been assembled and is slowly being codified, as evidenced by the Software Engineering Body of Knowledge (Bourque & Dupuis, 2001). Despite the emergence of these development methods and processes, failture rates of software projects remain high. The *Chaos reports* (1995, 2001) state that only a third of software projects are successful and half are seriously challenged (e.g., they are over budget, produce systems with poor functionality or mediocre quality). Compared to other engineering disciplines or economic activities, software development is highly risky and unpredictable.

The 1990's saw the emergence of global software development (Carmel, 1999; Herbsleb & Moitra, 2001; Karolak, 1998; Kraut & Streeter, 1995; Sahay *et al.*, 2003;

Sathyanarayan, 2003; Yourdon, 2004). In this distributed environment, software organizations initially tried to use the same recipes and processes that they had used internally for collocated development. But as Borchers (2003) reports, in global projects, "methods that were thought to be 'best practices' turned out to be ineffective or very difficult to implement."

In the distributed environment, companies run into spatial and temporal gaps: physical distances and time-zone differences make any form of direct communication much more difficult. Anything that is not the routine execution of predefined tasks, such as getting quick information and resources, brainstorming new ideas, or reaching consensus on a design, proves to be problematic (Carmel & Agarwal, 2001). Collaborative tools attempted to fill these gaps, but proved inadequate (Massey *et al.*, 2003; Sarker & Sahay, 2004). Adding to the challenges were the widely held and underlying assumptions that "a world-wide computer-literate culture, the internet, a programmer (hacker) culture largely dominate the dynamics of these global teams. As a result of this 'net culture', programmers behave the same in San Jose, Boston, Budapest or Bangalore" (Larry Constantine keynote, at the TOOLS pacific conference in Sydney, February 2002), or that "because [the software personnel] require little human interaction to be successful, the individual does not need extensive intercultural skills" (Beaman, 2004).

To summarize, software engineering differs from other globally distributed economic activities in the following respects:

- It is entirely an intellectual activity: all bits, no atoms (Kruchten, 2002).

- It does not lend itself to simple work breakdown structures, or "nearly decomposable subsystems" (Simon, 1996).
- It requires a large amount of interpersonal communication.
- It does not have any widely accepted processes.

While the intellectual aspect makes software development an ideal candidate for distributed work, the other characteristics make it particularly vulnerable to challenges introduced by geographical, temporal, organizational, and cultural boundaries. In particular, what impact could culture have on global software development? Collaboration in distributed and culturally diverse teams creates new dynamics: "In the new paradigm, there is no obviously dominant culture and there may be little or no incentive to create a negotiated culture" (Hsieh *et al.*, 2005). Our research investigates what and how software development processes and practices are affected in this culturally-diverse setting, and also what coordination problems are likely to result. We start by looking at some examples.

3. What happens in practice

Evidence suggests that mishaps in global software development are sometimes culturally related (Olson & Olson, 2004), and most project managers have one or two stories they love to tell. To illustrate our point, here are some incidents we have collected.

3.1 How much is enough?

A manager at company C requests that a developer at company D investigate a series of options and make a recommendation regarding a pending small design decision. Considering the importance of this decision, the manager expects the investigation to be

completed by a morning's work and reported in an informal memo. A week later the manager has still not heard from the remote developer and inquires whether or not his request is being worked on. He wonders if the developer has forgotten his request. Two weeks after the initial request he receives a 5-page report on the various options. The manager does not have time to read such an in-depth report and ultimately does not rate the recommendations as useful. From that point on, the manager only channels his requests to a few developers whom he trusts to understand his requests.

3.3 What are plans for?

Company E negotiates a subcontract for a multimillion dollar project with Company F. As part of the contract, Company F is required to produce a process document that details the software development process to be used in the project. The document is essentially a road map for the project, defining key documentation and review processes as well as project progress. Company F produces a process document that satisfies the contract requirements and the project begins. Two years later, close to the scheduled delivery date, Company E realizes that the project has not progressed as stated and sends engineers to investigate. During the course of the investigation the engineers ask where the process document is. The engineers at Company F look around and find the document on a shelf, covered with a layer of dust. When Company E asks about this, the response is "you didn't actually expect us to follow that plan, did you?"

4. Problem statement, hypothesis

The problems described in the previous section, along with many others commonly found in global software projects, can be considered as *mismatches in expectations* between

remote sites. These mismatches create breakdowns in project coordination and may result from a variety of reasons including communication delays and noise; differing language skills; and disconnects in knowledge. While the mismatches can be found in collocated projects, they are exacerbated by the various boundaries inherent in the distributed environment. Furthermore, developers from difference cultures collaborate in the distributed setting, the mismatches are often results of intercultural differences. Our research examines these expectation mismatches in the context of culture and hypothesizes that, in many cases, mismatches result from intercultural differences in factors such as values, behavioural norms, social systems, and communication approaches.

4.1 Definition

Before we go further in explaining our hypothesis, we should take a closer look at what the term *culture* actually means. For the purpose of this study, it is appropriate to use Spencer-Oatey's definition of culture as "a fuzzy set of attitudes, beliefs, behavioural norms, and basic assumptions and values that are shared by a group of people, and that influence each member's behaviour and his/her interpretations of the 'meaning' of other people's behaviour" (Spencer-Oatey, 2000). One of the implications of this definition has to do with the function of culture. More specifically, besides being an influencing factor on people's behaviours, culture also affects how people *interpret* others' behaviour. This interpretive function of culture is especially important in intercultural situations as the behavioural norms of one culture may be misunderstood or regarded as incongruous by other cultural groups. In the global project, such mismatches in interpretation may result

in misunderstanding, miscommunication, conflicts, mistrust, and even underutilization of talents, ultimately degrading the team performance.

4.2 Problem

Our research is based on the premise that there are and will continue to be intercultural factors that affect both collocated and distributed software development efforts. This idea is supported by Gibbs (2002) who points out that there is a growing body of literature that indicates that cultural differences will remain a significant factor within, and may even be magnified in response to, organizational culture. It should not be assumed that these factors are all negative. In fact, at least two project managers have reported that a good mix of cultural approaches was beneficial to their projects (Borchers, 2003; Kruchten, 2004a).

Despite the fact that software engineering is a highly analytical process, culture and intercultural factors are playing an often unrecognized role in projects. While there will be intercultural factors in teams that are collocated, the potential impact of intercultural factors in distributed teams is much greater. “The largest factor in global teams that we have seen is that they come from different cultures” (Olson & Olson, 2003).

4.2.1 Defining “team”

Our traditional notion of a “team” includes people who work together on a common task in a collocated environment. If this is so, what does “team” mean in distributed context?

In a distributed context teams are variously and interchangeably referred to as being *virtual* and/or *global*. Gibbs (2002) defines global teams as

> "…crossing boundaries - cultural, geographical, temporal, and organizational. They are both virtual and multicultural. They also rely predominantly on the use of electronic communication rather than face-to-face interaction, due to their geographical dispersion. Team members are at least somewhat interdependent, to the extent to which they work collaboratively to complete common tasks. Finally, the team's strategy and work itself is global in scope."

Although we will draw on existing cultural theories and models to help explain data collected in our study, we recognize that the fluidity of culture makes it nearly impossible for us to pinpoint any event to a particular cultural layer or group. Our focus is on examining the differences in how software processes and practices, are regarded by developers at different sites, and explaining these differences in the context of cultural predispositions, be they team, organizational, or national.

5. Research method

5.1 Research question

At a high level we are interested in the question:

> *How is culture affecting particular software development practices and related artifacts?*

To answer this question, we have taken a grounded approach and use a two-phase research strategy. In the first phase, we look for specific areas of interest through

interviews with software professionals, while in the second phase, we conduct more rigorous and detailed analysis to further refine our hypothesis.

5.2 Data collection and areas of interest

In the first phase of this research we interview project managers and other personnel who have been, or are currently, involved global software development projects. Our aim is to see what emerges in terms of (potentially) culturally-based mishaps and successes. Our current case studies are with companies involved in projects spread between Canada, USA, Russia, Italy, India, Nepal and China. Thus far we have conducted 21 semi-structured interviews with a total of 11 participants from these organizations.

One area of interest that has emerged has to do with the *shared understanding of expectation* between remote development sites (e.g., what constitutes proper documentation or process? How are review carried out? How are priorities rated?) We investigate how these expectations are satisfied or failed, and what influence intercultural factors may have on these interactions. Based on the data collected, we have also developed a small descriptive framework that will be used to describe expectations (both met and unmet) in coordination (see Section 6).

6. The AxE coordination model

Collaboration implies (1) breaking down work into smaller "chunks;" (2) allocating the chunks to various groups or individuals, based on skills, competence, responsibility, availability, and/or other criteria; and (3) aligning all the chunks between the various parties to integrate and produce the final software system. Depending on the complexity

of the work, work processes of varying granularities may be involved in each of three aspects.

Coordination has been studied from various perspectives (Crowston, 1997; Malone & Crowston, 1994) and in software development by Kraut and Streeter (1995). Most of these theories of coordination and the related tools and techniques (e.g., step-by-step manuals, procedures) seem rooted in the Input-Process-Output (I-P-O) model. "Ideally, architecture, plans, and processes—coordination mechanisms—would be sufficient to establish effective coordination among teams" (Herbsleb & Grinter, 1999). However, what is striking in the mishaps that we have collected is that, in globally distributed projects, often the inputs, processes, and outputs are apparently well-defined, but still all kinds of failures occur.

We have developed a small framework, called AxE, for Anticipation-eXecution-Expectation, to analyze, explain, and reason about the failures that we have encountered. The model differs from other models of coordination in that it takes into account the transient and social *micro*-attributes associated with coordination inputs, outputs, and processes. The model consists of 3 elements:

1. the concept of discrepancies between expected and actual inputs and outputs ("the deltas")
2. a flow over time, in three steps ("the waltz")
3. the concept of a "trust capital"

6.1 Handing over information

Assuming that during the execution of the software development process, some artifact U (e.g., a document, some code) has to be handed over from a party S to another party D, there are 3 attributes of U that may affect the effectiveness of coordination in the process:

- C: a definition of the content of U. Content refers to not only the material but also its representation. For example, the components of a use-case may include the name, description, pre- and post-conditions, involved actors, main flow of events, and alternate flows of events; the definition would also specify (or imply) how the components should be represented.
- Q: the level of quality of U at the time of the handover (e.g., is U a sketch or very complete? Has it been reviewed? How much detail is included?)
- T: the time at which the handover occurs.

For the triplet of attributes U [C, Q, T], three instances are defined (Figure 1):

- *Anticipated*, by the source S, U^A:

  The anticipated U is what S initially thinks he will deliver to D: an anticipated content and an anticipated level of quality at an anticipated time of delivery.

- *Expected*, by the destination D, U^E:

  The expected U is what D initially assumes she will receive from S: an expected content and an expected level of quality at an expected time of receipt.

- *Executed*, what is actually handed over by S to D, U^X:

  The executed U is the actual artifact that S hands over to D: a certain content of a certain level of quality delivered at a certain time.

6.2 Discrepancies (deltas)

Often the cause of breakdowns in coordination is the differences between the 3 instances of U:

- ΔEx is the difference between what D expected and what she received (i.e., U^E - U^X). The discrepancy may be on any of the 3 attributes: different contents, different levels of quality (e.g., completeness, accuracy), and/or different times (e.g., late, most often).
- ΔAx is the difference between what S anticipated he would deliver and what he actually delivered (i.e., U^A - U^X). This discrepancy is less obvious: why would someone deliver something other than what he anticipates? But this is often the case in software projects as time pressure or work overload compromise quality and/or timeliness.
- ΔAE is the difference in perception between S and D regarding the 3 attributes (i.e., U^A – U^E). One would think that, in a well-oiled and disciplined organization that rigorously follows a defined process, such discrepancy would be minimized. In software projects, nevertheless, ΔAE is common, due to the nature of software development. Many software engineering artifacts are intangible—information, intents, plans, partial definitions, risks, estimates, among others—leaving a great deal to interpretation (and therefore discrepancies in perception). S and D may very well adhere to the same process but interpret the details of the process differently.

It should be noted that ΔAE is not necessarily the sum of ΔEx and ΔAx. In some cases ΔEx and ΔAx are greater than ΔAE: the expected and anticipated values are close, but the

execution falls far from both. Also, A wide ΔEx may not necessarily be the fault of the source S; the cause may be unreasonable expectations from the destination D.

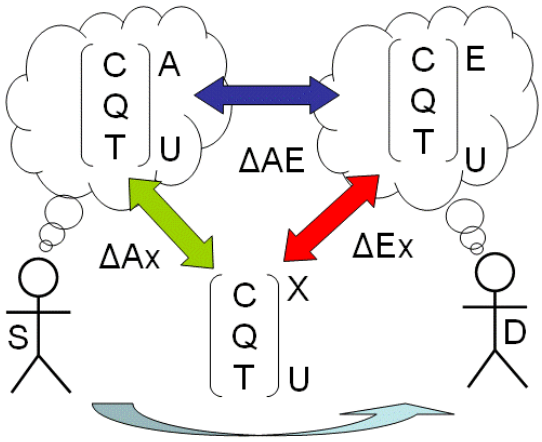

Figure 1: Overview of the AxE discrepancies (the 3 deltas)

6.3 Stages for coordination (the waltz)

The handover of U from S to D proceeds in 3 stages:

- *Planning*, where anticipated U^A and expected U^E are formed;
- *Execution*, where the actual artifact U^X is handed over;
- *Feedback*, where ΔEx and ΔAx are actually perceived, evaluated, and (maybe) communicated.

6.3.1 Communication channels

These 3-stage handover implies some form of communication between S and D. This communication can take multiple forms (as shown in Figure 2):

- *Direct:* S and D speak to each other face-to-face.
- *Mediated*: S and D use some medium (e.g., email, instant messaging).

- *Indirect*: S and D access some common repository, document(s) (e.g., plans), and/or process(es).
- *By proxy*: S and D communicate via third parties (e.g., management); S and D may not be aware of their mutual existence.
- *Implicit*: S and D use the context to implicitly deduce the anticipated and/or expected values.

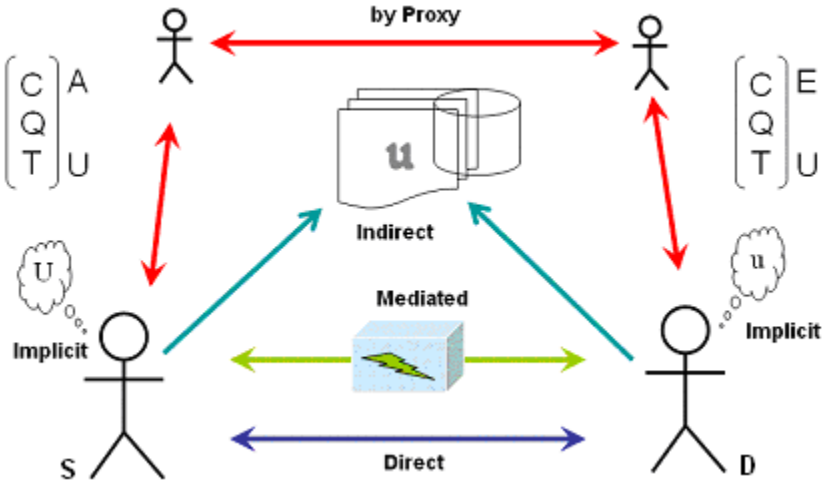

Figure 2: Modes of communication during coordination

6.3.2 Planning

How do S and D form U^A and U^E, respectively? The time component is usually the simplest: S and D are both aware of some schedule or plan. The content and the quality are more subjective to interpretation. There may be guidelines, but the specifics of the content and quality are affected by factors including (1) previous interactions between S and D; (2) the extent of a common understanding between S and D (regarding the artifact, overall goal, process, and environment); and (3) the level of trust between them.

6.3.3 Execution

Execution is the least problematic part of coordination. S creates or refines U, and makes it available to D using the most appropriate (and available) communication mechanism (shown in Figure 2). In doing so, S uses what he considers the "right" process but is still subject to external conditions that may affect his ability to actually produce what he anticipates.

6.3.4 Feedback

Once U^X is delivered, the discrepancies ΔEx and ΔAx are perceived by D and S, respectively. The question of feedback concerns whether D makes S aware of ΔEx, when, and how. Feedback mechanisms vary across projects, activities, and individuals. Feedback can be immediate (e.g., "you are late,") or communication can proceed indirectly (e.g., through management meetings). In the global project, feedback may be misinterpreted, ignored, or exaggerated, often for cultural reasons: saving face or catering to an asymmetrical relationship. Often in the distributed setting, the source is never made aware of ΔEx, or only indirectly and much later, through a growing level of mistrust. To deal with this, some software organizations have instituted systematic feedback mechanisms (e.g., postmortem meetings, retrospectives).

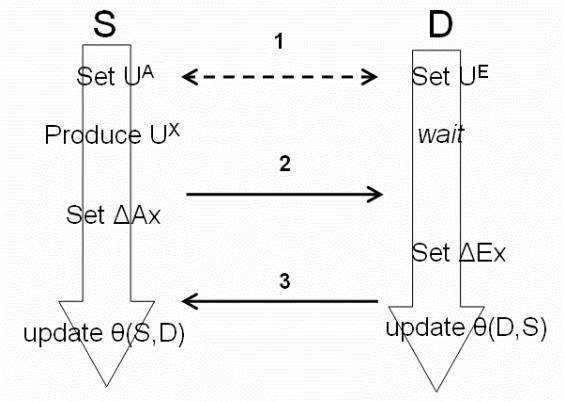

Figure 3: The 3 stages of coordination (the waltz)

6.4 Trust and mistrust

When coordination between S and D occurs for the first time, they start with some level of trust in each other.

This trust capital **θ** plays an important role in the forming of both U^A and U^E in the planning stage. And the trust capital is revised after execution. A large discrepancy between what is expected and what is delivered (ΔEx) will decrease the trust capital, especially if there is no efficient feedback mechanism. On the contrary, a small ΔEx will increase the trust capital that D holds for S, θ(D,S). Repeatedly large ΔEx will develop the distrust to such an intolerable level for D that D will no longer collaborate with S (i.e., "I'd rather do it myself").

6.5 Scaling AxE up

We have described the AxE coordination model from the perspective of one individual handing an artifact to another. But the model can be generalized to interaction between large groups of individuals and teams. And it generalizes to more complex or composite artifacts. Though not explicitly stated, the model applies to coordination in both distributed and collocated teams. In the distributed team, time differences, physical distances, and other cultural and organizational factors affect the forming of UA and UE, the communication modes, and the availability and immediacy of feedback. Close attention needs to be paid to these factors when using the AxE model.

7. Data analysis

Using the AxE model, we will revisit the examples presented in Section 3.

7.1 “How much is enough” revisited

In this case, the coordination centers on an investigation conducted by developer D that is to be reported to manager C. The request to conduct an investigation and to submit a written report (C) is made explicitly by C to D. The quality (Q) and delivery time (T) associated with the investigation, however, is not explicitly stated. Manager C assumes that developer D would be able to deduce the Q and T necessary, based on past experience, current workload, and the priority of the request, among other factors. The coordination ends with a $U^A[Q,T]$ that is much greater than $U^E[Q,T]$, leading to a large ΔAE and reduced $\theta(D,S)$. Manager C considers the time and effort spent on the investigation and the report wasted and should have been spent on more worthwhile work. Manager C starts delegating this type of work to others (or whom he has a higher $\theta(D,S)$ i.e., a higher trust capital).

7.2 “What are plans for” revisited

The mismatch in this case has to do with the question of whether or not plans are to be followed. The idea that the plan will not be followed is foreign to Company E, while the idea that Company E actually expects Company F to follow the plan comes as a surprise to D.

Examining the issue from a cultural perspective, we find differences between the two companies on many levels. The organizational culture of company E is one that is highly process oriented. In other words, plans are of great importance in Company E and everyone in the organization is expected to be committed to following plans. Company F, however, places little value on the process document and just considers it a hoop to jump through

9. Status, future work, and conclusion

No, programmers do not behave the same in San Jose, Boston, Budapest or Bangalore. Beyond the issues of language, time zones and collaborative tools, there is a range of incidents, mishaps, and failures that are mostly attributable to differences in culture, in the fundamental value systems that are part of the society's education, politics, organization, and even religion.

We are in the process of conducting a systematic analysis of this body of data. From our study, we hope to be able to derive some guidelines for global software project managers to mitigate risks associated with cultural issues, particularly in relation to coordination of activities across a big cultural divide.

Bibliography

Beaman, K. V. (2004). Myths, mystiques, and mistakes in overseas assignments: The role of global mindset in international work. *IHRIM Journal*, 40-53.

Borchers, G. (2003). *The software engineering impacts of cultural factors on multi-cultural software development teams.* Paper presented at the 25th International Conference on Software Engineering (ICSE'03), Portland, OR.

Bourque, P., & Dupuis, R. (Eds.). (2001). *Guide to the software engineering body of knowledge*. Los Alamitos, CA: IEEE CS Press.

Carmel, E. (1999). *Global software teams: Collaborating across borders and time zones*. Upper Saddle River, NJ: Prentice-Hall.

Carmel, E., & Agarwal, R. (2001). Tactical approaches for alleviating distance in global software development. *IEEE Software, 18*(2), 22-29.

Crowston, K. (1997). A coordination theory approach to organizational process design. *Organization Science, 8*(2), 157-175.

Damian, D. E., & Moitra, D. (2006). Global software development: How far have we come? *IEEE Software, 23,* 17-19.

Gibbs, J. L. (2002). *Loose coupling in global teams: Tracing the contours of cultural complexity.* Unpublished Dissertation, University of Southern California, Los Angeles.

Herbsleb, J. D., & Grinter, R. E. (1999). Architectures, coordination and distance: Conway's law and beyond. *IEEE Software, 16*(5), 63-70.

Herbsleb, J. D., & Moitra, D. (2001). Global software development. *IEEE Software, 18*(2), 16-20.

Hsieh, Y., MacGregor, E. L., & Kruchten, P. (2005, May 1-4). *Intercultural factors in global software development.* Paper presented at the 18th Annual Canadian Conference on Electrical and Computer Engineering (CCECE'05), Saskatoon, SK.

Karolak, D. (1998). *Global software development: Managing virtual teams and environments*. Los Alamitos, CA: IEEE Computer Society Press.

Kraut, R. E., & Streeter, L. A. (1995). Coordination in software development. *Communications of the ACM, 38*(3), 69-81.

Kruchten, P. (2002, March 15-16). *The nature of software--what's so special about software engineering?* Paper presented at the International Conference on the Sciences of Design -- The Scientific Challenge for the 21st Century -- In honour of Herbert Simon, Lyon, France.

Kruchten, P. (2004, April 14-16). *Putting the "engineering" into "software engineering".* Paper presented at the Australian Software Engineering Conference (ASWEC 2004), Melbourne, Australia.

Malone, T. W., & Crowston, K. (1994). The interdisciplinary study of coordination. *ACM Computing Surveys, 26*(1), 87-119.

Massey, A. P., Montoya-Weiss, M. M., & Hung, Y. T. (2003). Because time matters: Temporal coordination in global virtual project teams. *Journal of Management Information Systems, 19*(4), 129-155.

Olson, J. S., & Olson, G. M. (2004). Culture surprises in remote software development teams. *ACM Queue, 1*(9), 52-59.

Sahay, S., Nicholson, B., & Krishna, S. (2003). *Global it outsourcing: Software development across borders*. Cambridge, UK; New York: Cambridge University Press.

Sarker, S., & Sahay, S. (2004). Implications of space and time for distributed work: An interpretive study of us-norwegian systems development teams. *European Journal of Information Systems, 13*(1), 3-20.

Sathyanarayan, M. M. (2003). *Offshore development--proven strategies and tactics for success* (1 ed.). Cupertino, CA: GlobalDev Publ.

Simon, H. (1996). *The sciences of the artificial* (3 ed.). Cambridge, Mass.: The MIT Press.

Spencer-Oatey, H. (2000). *Culturally speaking: Managing rapport through talk across cultures*. New York: Cassel.

Standish Group. (1995). *The chaos report*. West Yarmouth, MA: The Standish Group.

Standish Group. (2001). *Extreme chaos*. West Yarmouth, MA: The Standish Group.

Yourdon, E. (2004). *Outsource: Competing in the global productivity race*. Upper Saddle River, N.J.: Prentice-Hall.