Global Software Development: Challenges and Perspectives

Cleidson R. B. De Souza
Information and Computer Science,
University of California, Irvine
Irvine, CA, USA
e-mail: cdesouza@ics.uci.edu
Also at:
Computer Science Department,
Federal University of Pará
Belém, PA, Brazil

Abstract

With the rapid globalization of companies, also the globalization of software development has become a reality. Many software projects are now distributed in diverse sites across the globe. The distance between these sites creates several problems that did not exist for the co-located teams. Problems in the coordination of the activities, as well as in the communication between team members emerge. This paper presents the most important problems in global software development, as well as the techniques and tools designed to solve these problems.

***Key Words**: Global Software Development, Distributed Software Development, Cooperative Software Development.*

1 Introduction

Nowadays, globalization is a world that has been heavily explored. Basically, it means that the economical, cultural and social boundaries of the countries are disappearing. For example, in Brazil you can buy the same products that you can find in the United States (USA). The technology innovations are ubiquitous. The cellular phone roaming feature allows mobile phone users to travel and keep in touch with friends and coworkers. Cultural barriers are also becoming subtler. In the US, for example, you can watch the same movies and television series that are being presented in Brazil.

Software is no exception to this rule. Microsoft, for example, derived 55% of its sales from outside the US[Kar, 1999]. Due to different government regulations, several companies, especially in the telecommunications domain, need to have development sites in different countries, in order to get a position in the local market [HMFG, 2000].

Therefore, these companies need to adapt their processes, tools, and organizational culture to overcome the distance between the sites. When the groups are dispersed, the sense of working in a team decreases due to the lack of interaction between the members of different sites. Also, there is a lack of trust because the members usually do not have knowledge about overseas' culture. Relatively simple activities like discussing requirements in meetings can not be performed. All these problems must be understood and properly solved for global software teams succeed.

This paper presents the main problems (the challenges) faced by developers during global (or distributed) software development, as well as some solutions that have been proposed and applied (the perspectives). It is important to note that some solutions are more suitable to be classified as management techniques, such as providing training through workshops about the other's culture[Car, 2001]. The emphasis of this paper lies on techniques related with software engineering concepts, therefore management techniques will be briefly covered.

The rest of this paper is organized as follows. A better understanding about global software development, their motivations and advantages are necessary to understand this paper. Therefore, section 2 presents these concepts. Then, the next section presents the problems that make global software development more complex than the traditional co-located software development. After that, section 4 presents solutions to the problems discussed. Finally, some conclusions are described in section 5.

2 Global Software Development: Definition and Motivations

In this section, the concept of global software development is described as well as the motivations for this type of work.

2.1 Definition

Global software development, also called distributed, means that the software development is scattered along several sites that could be located in different countries and even continents. Accordingly, a global software team is defined as separated by a national boundary while actively collaborating on a common software/system project[Car, 2001].

Several companies such as Lucent[HG,1999a] [HG, 1999b][GHP, 1999][HMTG, 2000], IBM, Motorola[BCK+,2001], among others are adopting this strategy. In fact, according to [CA, 2001] 203 of the US Fortune 500 companies are engaged in offshore outsourcing. Furthermore, more than 50 nations are currently participating in collaborative software development internationally. Just to mention one example, in India there are 800 firms competing for work globally.

2.2 Motivations

There are several reasons for globally developing software. Most of them are economical, although there are some practical and political. For example, global software development can occur because of business arrangements like[Kar, 1999][HMFD, 2000]:

- mergers and acquisitions to adjust and complement product lines often lead to new sites becoming part of the company;
- to participate in some markets (specially telecommunications),government regulations request the location of some local development operations; and
- it can make sense for market reasons to locate parts of the corporation where the market for a particular technology exists.

Furthermore, the competition for highly technical staff is driving companies to hire them wherever in the world the talent can be found. This is another problem since there is a constant need for more developers, especially in some countries[Fox, 1999]. As these developers are not mobile, i.e., they can not just move to other countries, the development task has become distributed. Another advantage of this approach is that these resources are available at lower cost[Kar, 1999]. Therefore, companies are finding that it is economically attractive to outsource or codevelop overseas.

Finally, most corporations, especially those in the software business, hope that geographic distribution could lead to *round-the-clock* development, which offers the promise of reducing development cycles by increasing the amount of time in the day that software is being developed. The idea is to have developers working in the code as much hours as possible during a day, since these developers work in different time zones. For instance, when a developers stops to work in the code in California, US another one starts to do it in Bangalore, India. This concept has been called "*follow-the sun*"[Carmel, 1999] and, theoretically, can reduce the software development time in 50%.

3 Problems

This section summarizes the main problems reported in the literature about global software development.

Before describing these problems in detail it is necessary to understand why global software development is different from the traditional co-located development. The main difference is the *distance* between the groups of developers. In fact, the definition presented in the previous section for global software development teams is about distance.

This distance imposes several constraints in the tasks carried during the software development. It exacerbates coordination and control problems directly or indirectly through its negative effects on communication[CA,2001]. In fact, there is considerable evidence in literature[PSV, 1994] [HKO+, 1995] which indicates that informal (unplanned) communication is essential in successful software development activities. Developers also rely on this communication to coordinate your activities by handling exceptions, correcting mistakes and managing the effects of all these changes[[HG, 1999b]. Furthermore, the lack of informal communication also decreases the trust among teams.

The communication across teams is also difficult because of the difference between time zones and technological constraints.

Finally, the physical distance between the members influence the degree of cooperation among them[KS, 1995]: the rate at which scientists collaborate spontaneously with one another is also a function of distance between offices. Also, the communication among engineers decrease with the distance[HMFG, 2000].

Distributed development of software, compared with non-distributed, introduces *delays* in the process[HMTG,2000]. A delay is an additional time needed to resolve an issue. So for example, if a part of the design or code need to be changed, or if someone needs a better understanding of how some part of the product works, people at more than one site may need to be involved in information exchange, negotiation, and so on, in order to find a solution[HMFG,2001]. In fact, these authors presents qualitative data showing that these delays can slow the software development process considerably.

In short, the most important class of problems is related to the communication, mostly informal, among developers that is more difficult due to the distance between the teams. However, other problems also arise such as lack of trust, difference between time zones, cultural problems and identification and selection of expertise. Each one of these problems will be discussed in the following subsections.

3.1 Communication Problems

This is the most important class of problems, since it is directly affected by the distance between the teams. In this case, there are two types of problems: Language issues and Lack of Informal Communication.

Language Issues

One of the most obvious problems in a global software development activity is the language barrier, i.e., since the groups can be dispersed around the globe, there is no guarantee that all groups will speak the same language.

Even if a specific language is adopted by the entire group problems may occur. For example, [HGVF,2001] report a case study where German and British had misunderstandings due to the use of one specific word: "should". A German manager said that the British members "should" do something while his idea was that they should "consider" the new task. However, the British dropped what they were doing and completed the new task. The difficulty of precise translation of words that often have somewhat different connotations caused a significant misunderstanding [HGVF, 2001].

Another example of communication problem is presented by Anthes[Ant, 2000]. He describes a situation where a developer had to travel from Swindon to Germany because the test specification said to type a blank (hit the space key) when the tester got a certain point. However, the tester was actually typing the word "blank".

One may think that this kind of misunderstanding is a problem only when the groups do not communicate using the same first language, i.e., one of the sites is obliged to use another language different from their first one. For example, British and Americans would not have this problem because both speak English. However, [HGVF, 2001] reports that, even in this case, problems happen: in American English the word "quite" has a positive meaning, while in British it also can have a negative one meaning that something could have been better. Situations like that caused confusion sometimes.

Lack of Informal Communication

The second problem of related with communication, and perhaps, the most important, is the lack of informal communication. This problem has been reported in studies by [HG, 199a], [HG, 199b], [Car, 1999] [Car, 2001], and [CA, 2001].

The idea is quite simple: since the teams are not physically together, they can not meet each other around the water cooler, in the hallway or in other public areas. Therefore, the extent of informal communication is reduced. By informal communication, we mean, personal, peer-oriented and interactive communication in contrast to formal communication through writing, structured meetings and other relatively non-interactive and impersonal communication channels[Kraut, 1990].

According to studies in organizational theory and CSCW (which call these events as opportunistic interactions), this type of communication is very important to coordinate teams in uncertain tasks such as software development. By uncertainty, we mean the unpredictability of both the software and the tasks that software developers perform[KS, 1995]. There are several reasons that make software development an uncertain task. For instance, fluctuating and conflicting requirements[CKI, 1988]. The requirements will appear to fluctuate when the development team lacks application knowledge and performs an incomplete analysis of the requirements.

In other words, software developers also rely on informal, ad-hoc communication to fill in details, handle exceptions, correct mistakes and bad predictions, and manage the effects of all these changes[[HG, 1999b]. As this type of communication is absent in global software development the coordination of these tasks is much more difficult.

Lack of Context

One of the main problems during the communication between distributed sites is the lack of context. It means that, sometimes the receiver of a message does not understand the context well enough to determine the question's importance. Therefore, the receiver assumed that the message is not important taking a long time to answer it. Of course, this is one of the reasons for the delays in global software development. The context, in this case, means the artifacts related with the communication like diagrams, code, test cases, requirements and so on.

This is a well-know problem in the design rationale literature[Lee, 1997]. For example, Fischer *et al.* [FLM, 1991] identified the need for integration of decision-making (a type of communication) and artifacts in his experiments where ‘‘... designers were often unable to judge the relative merits of issues because they could not see their influence on construction...”. The solution to this problem adopted by DR researchers was to integrate the artifact with the discussion about it. This point is discussed in section 4.3.

3.2 Coordination Problems

Coordination is the act of integrating each task with each organizational unit (or team member), so the unit contributes to the overall objective. In software development, it means that different people working on a common project agree to a common definition of what they are building, share information, and mesh activities[KS, 1995].

Coordination often requires intense communication, i.e., the exchange of complete and unambiguous information helps the members to reach a common understanding[CA, 2001]. In fact, empirical studies performed by [PSV, 1994] and [KS, 1995] show that informal communication is essential to successful software development, since it helps the members to coordinate their activities. We already know that in distributed settings, due to the distance between sites, the informal communication decreases, then it is clear that coordination is much more difficult in global software development.

The critical role of communication in successful coordination leads us to conclude that communication is the main challenge in global software development. The discussion about communication in this context was presented in the previous section (3.1).

3.3 Time zones

[CA, 2001] defines temporal distance as the difference between the time zones in two different sites. If the temporal distance is great, typically asynchronous technology is used to support communication. In this case, the advantages of “follow-the-sun” type of work discussed in Section 2 can be achieved [Carmel, 2001]. On the other hand, asynchronous communication is more “poor” than synchronous: it does not convey information such as the speed and tone of voice, facial information, body language, pauses, etc[Jar, 1998]. Therefore synchronous communication is more effective and helps to solve conflicts faster. A small issue can take days of back-and-forth discussion over e-mail (asynchronous) to resolve, but a briefly conversation (synchronous) can quickly clarify the problem [Car, 2001].

However, synchronous communication can be costly for professionals who complains about the need to compromise personal life to speak to colleagues far away many time zones removed[Carmel, 2001].

In other words, there is a direct dependency among the temporal distance between the sites, the technology used for communication and the advantages that can be gained with global software development.

3.4 Culture

Another important challenge in global software development is the cultural differences between team members. [CA, 2001] describes two basic types of culture differences: nationality, the most obvious and the organizational culture.

National culture encompasses an ethnic group’s norm, values and spoken languages, often delineated by political boundaries of the nation state. For example, a typically American attitude is to reschedule his vacation if a project is running a week late, however this would not be done in Europeans countries[Rot, 1998].

Organizational culture encompasses the unit’s norms and values, it includes the culture of system development, such as the use of methodologies and project management practices. In this case, differences between teams can lead to lack of trust (see sections 3.5) and makes coordination more difficult.

Differences in national cultures manifest themselves in different ways like hierarchy issues and the communication style, while differences in organizational cultures are materialized in problems with the orientation towards the software process. Each one of these three problems is discussed below.

Hierarchy

[HGV, 2000] identified that German developers have a greater tendency to take a formal approach toward hierarchy, and tend to be more careful in following hierarchy-related protocols in contrast to British. Later, Herbsleb[Her, 2001] presents his experience with Indians and Americans. The purpose was to convince two groups of developers to adopt a Instant Messaging system. While for Americans the best approach was to talk directly to the employees, for Indians the approach recommended and used was to talk to their managers.

In short, the different orientation towards hierarchy in each country suggests different approaches. These differences must be understood by project managers and team leaders.

Communication Style

One example of this issue is the difference between German and British reported by [Herbsleb, 1999, IEEE]. Germans use a more direct style of communication: they call someone and immediately say that there is a problem in the other's code. In contrast, British tend to expect a more "polite" approach in which the error is "suggested" instead of being directly pointed.

This difference influences the team ability to communicate effectively creating misunderstandings, miscommunication, and lack of trust, which can decrease the productivity.

[Rot, 1998] presents another example where the terminology adopted among organizations was different. In this case, the word "fixed" have two different meanings for Americans and Europeans bringing confusion among team members. For one, it means that the problem was already solved, while for the other it means that the problem already was identified and "scheduled" for being solved. She also describes other words that caused confusion specific to the software development activity.

Orientation towards the process

In the German site studied by [HGV, 2000] the employees have a high regard for development processes. They had a defined process that they tended to follow diligently. On the other hand, the British site does not have a process and they were particularly willing to abandon when situation calls for quickly delivery.

This situation imposed tensions in both sites since "(...) British think that Germans would follow the process even when it was going to take too long and cause unnecessary delays. From the German perspective the British developers had little control of their process to begin with, and were far too ready to abandon the process and risk compromising technical quality and reliability."[HGV, 2000].

[Car, 2001] suggests the adoption of a common development process in order to solve this problem. In his work, he calls it as "Centripetal Force 5: Software Development Methodology". The adoption of a unified process can avoid several problems in a global development. A process defines how a software should be produced, specifying for example, the products developed along the process, the milestones, etc. If these items are previously defined and accepted by all sites, it can help to decrease disagreements and misunderstandings since these items are frequently source of these problems between development units[Car, 2001].

However, the adoption of a single process must be carefully evaluated since the learning curve can impact the delivery of the system[BCK+,2001] as well as its costs. If this is not possible another solution is a blend of the processes of the sites, or at least, an agreement on high level process components such as stages and their respective entry and exit conditions.

3.5 Lack of Trust

Trust is a recurrent problem in virtual collocated teams. It is co-related with the poor communication that happens in these teams due to the distance and the infrastructure used.

Herbsleb and Grinter[HG, 1999a] observed that groups in different sites during the beginning of the process of the cooperation had little trust in each other. This reflected in the absence of willingness to communicate openly and in misunderstandings about the behavior of each other's. For example, if someone say "we can not make these changes", it was often interpreted as "we do not want to make these changes" whether it could benefit the overall project or not. It is clear that this kind of behavior can decrease the productivity of the team.

The lack of informal communication also decreases the trust since these interactions are important to build interpersonal relationships among member of team, and therefore trust. The cultural difference among sites can also contribute to this problem, since similarity with others positively reinforces members' own identities and contributes to their willingness to trust[Jar, 1998].

3.6 Expertise Identification and Selection

McDonald and Ackerman[MA, 1998] define expertise as the embodiment of knowledge and skills within individuals. These authors also distinguish two steps in finding expertise. "Expertise identification is the problem of knowing what information or special skills other individuals have. (...) Expertise selection is appropriately choosing among people with required expertise. If there are multiple potential experts or people with requisite expertise, it is necessary to select one (or more) to ask." Both steps are harder in global software development. In fact, the expertise identification problem was reported by developers as one of their major concerns [HG, 1999b].

Expertise Identification

In this case, the problem is how to identify whom to contact to in order to solve a specific problem. In a non-distributed setting, informal communication channels can be used to find the answer for this question, as well as the personal network of each employee. However, as previous discussed, this information can not be used in GSD because the teams are dispersed around the globe. Since informal channels are not useful other approaches need to be used.

For example, one approach used by developers and reported by [HG, 1999b] was to contact the system architect or project manager at the other site because they had a broad knowledge of who was working on what. In this case, a formal channel (chain on command) was used. However, it can create a slow down that reflects on the time for solving the problem.

Expertise Selection

Now, since one (or more) people with the required expertise are identified the question is how to decide whom to ask? In their field study [MA, 1998] observed that developers use different rules-of-thumb to make their selection. One of them is based on the workload of the developers. When the information about the workload is not available, developers infer this information based on the word-of-mouth, closed office doors and co-worker's assessment of one another's workload[MA, 1998]. In a distributed setting this type of information is not available, therefore it makes the process of selection more difficult.

The other techniques identified by [MA,1998] can not be used to help in the process of expertise selection in a distributed setting because they are all strongly related with face-to-face communication. The problem of expertise selection is important and difficult, however it is out of the scope of this paper.

4 Solutions

There are just a few authors and approaches in the literature that address the problems in global software development. Some of these ideas are more related with management techniques that should be used in order to minimize the problems previously described. As this paper is more concerned about software engineering concepts, techniques and tools, these ideas about management will be briefly discussed, mainly in sections 4.1 and 4.2. To the interested reader, another interesting ideas are presented by [Rot, 1998]. The other sections present some tools identified in the literature.

4.1 The Six Centripetal Forces for Successful GSD [Car, 2001]

[Car, 2001] presents six solutions to the problems in global software development. He calls them as "Centripetal Forces". They are:

(i) Collaborative Technology;
(ii) Team building;
(iii) Leadership;
(iv) Product Architecture and Task Allocation;
(v) Software Development Methodology; and
(vi) Telecommunications Infrastructure

Although these forces are an important contribution, their ideas are not discussed here because they can not be directly mapped to tools. They are recommendations that should be used by managers in order to succeed in global software development projects. For example, the first centripetal force suggests that global software developers should use asynchronous and synchronous collaborative tools in order to increase the effectiveness of the cooperation. This paper is more concerned about the need to develop such tools and what tools have already been developed.

Another example is the third centripetal force that discusses five unique leadership qualities of a software manager that makes him able to handle the multicultural and dispersed aspects of GSD.

4.2 The Cultural(Contact) Liaison

The cultural liaison is a member of one team that spends a significant part of time in the other site. Herbsleb and Grinter[HG; 1999a, 1999b] verified in their interviews that a liaison is very useful. They detected that "people who spent a significant amount of time at the other site, became a contact person." For example, a contact liaison became responsible for helping developers to figure out whom to contact to in the other site (an *expertise concierge* according to [MA, 1998]).

Furthermore, after living in the other site, this liaison can also explain for his colleagues how things work at the other sites. He has experience with the other culture in the context itself, therefore less mystifying. So he can also facilitate the cultural and linguistic flow of information bridging cultures, mediating conflicts and resolving miscommunications.

Finally, the cultural liaison helps to establish trust between the sites. For example, [BCK+, 2001] describes an experience in global software development where the American engineers had a concern about the international teams, i.e., they are worried if those teams would be able to develop as needed. Then, some international engineers spend some time in the US discussing the requirements for the project, i.e., those engineers became contact liaisons. After that, the US team realized that "all of the non-US teams understood software engineering concepts. The interaction [with the liaisons] made the staff confortable with the non-US engineers' qualifications to build their network elements". In fact, according to the authors the cultural liaison turned to be a key factor on the success of the project.

The same findings were reported by [HG, 1999b] who describes a significant improvement in the relationships between the sites after a visit of some developers. The skeptical behavior about the other site was alleviated.

These are the reasons why [Car, 1999] and [CA, 2001] recommend the adoption of this role and suggests that he should be traveling back and forth between the sites.

4.3 Solutions to the Communication Problem

The ability to communicate can be improved using different technologies such as phone calls, conference calls, electronic mail, videoconference, etc. However, these tools and their use must be adapted to face the problems in GSD. For example, in order to minimize personal problems related with the need to communicate with partners in the other side of the globe, Motorola adopted conference calls from their houses[BCK+, 2001]. Otherwise, these technologies can have the opposite effect leading team members to avoid or circumvent their communications need.

Meanwhile, theories such as media richness and social presence theories suggest that computer-based communication media may eliminate the type of communication cues that individuals use to convey trust, warmth, attentiveness, and other interpersonal affections[Jar, 1998]. If these theories are correct, the communication problems using such technologies can be alleviated but can not be extinguished while better communication tools are not developed.

Instant Messaging

Instant Messaging (IM) is a communication technology designed to stimulate informal communication (opportunistic interactions) among workers at different sites. This technology has spread very rapidly and it is beginning to infiltrate in the work place[NWB, 2000]. Basically, it is a near-synchronous computer-based one-to-one communication technology. In this case, users do not go into "rooms" to converse with whomever is there; instead there is a single individual with whom they communicate (although they may have several concurrent conversations with different individuals in progress at a given time). There are several different systems such as AOL's Instant Messenger, ICQ and Yahoo Messenger.

Usually, IM systems provide awareness information about the presence of others. For example, using ICQ you can create a "Contact List" with the names of your friends. Whenever they are logged in the ICQ server, you are notified about it with, for example, a sound signal. Therefore, you can easily start a conversation with them by double clicking in the user name in the contact list.

Privacy is also an issue in such systems, therefore you can also control the published information about you. For example, in ICQ there is an "invisible" mode where you are connected and you have information about the others, but the others are not allowed to access your information. For them, you are not connected in the ICQ server.

In order to facilitate the informal communication among distributed software developers the Bell Labs Collaboratory Group developed an IM system called Rear View Mirror[Her, 2000] [Her, 2001]. This system allows a user to create contact lists receiving information about the presence of the users in this list. A user can also create an access control list, which is the list of users that are allowed to receive information about him. If a user is not in this list, he does not receive nay information about the other. This feature address the privacy issue previously described.

The system is also integrated with a chat system. Therefore, a user can initiate a conversation by right clicking on the other user's icon, assuming that the co-worker is present. In this case, an invitation is sent to the co-worker who can accept or deny it. Unfortunately, there are no evidences in the literature about the effectiveness of IM systems.

The Lack of Context

Researchers in the design rationale field identified that this problem is minimized using artifact integration, i.e., associating the artifact with the communication about it. Artifact integration is very important because the communication about problem solving is densely populated with references to the artifact that has the problem.

In fact, this was the approach used by *ConnectIcon*[Her,2001], a tool from Lucent Technologies. The goal of this tool is to short the time needed by developers in two different sites to communicate and solve a problem. It supports electronic mail and chat and through it developers can send other information important to the communication, such as artifacts (test cases, source code, etc), contact information, etc. This tool is also integrated with the other tools produced by Lucent.

4.4 Expertise

Expertise Identification

One approach for solving the problem of expertise identification is proposed by [Mockus,2001]. The idea is based on the mechanism called by [MA,1998] Historical Artifacts, who identified in their field study in a software development company. Basically, a change history of the important artifacts is stored with the information of who make the change and when the change was made. Based on the observation of the logs, one can identify people who have most experience in changing the artifact. The developers can also identify the person who most likely has the “freshest” memory of the artifact[MA, 1998].

The Expertise Browser [Mockus, 2001] was built on Java as an applet and retrieves the relevant information form the change management system used by Lucent Technologies. It presents those people that changed a selected code unit, where the height of the line for each person is proportional to the corresponding person’s experience with the code. Persons who made more change have their lines larger and presented before the others. The tools also presents people’s contact information (e-mail, phone numbers, etc) to help the immediate communication between developers.

Expertise Selection

Expertise Browser seems to solve the problem of expertise identification since it points out developers that seem to have expertise about parts of the code. However, it does not solve the problem of expertise selection. It does not provide any information that can help developers to decide whom to ask. As previously described, this information is important because it can improve the resolution of the issue. If a developer asks for an expert who is in vacation or is a very high workload, he can waits for a long time before realize that he needs to ask to another expert.

5 Conclusions

Software development performed by traditional co-located teams is a task innately difficult. It is difficult because of the need of coordination and communication among developers. Also because of the complex nature of the activities executed. When performed in a distributed setting, this task becomes more challenging. In a global software development, several problems arise due to the distance among the developers. For example, the communication is much more difficult because of language barriers, informal communication among the team members can not happen and trust is much more difficult to achieve.

This paper presented these problems as well as some solutions that address them. The most important problem reported in the literature is the lack of informal communication. Developers in co-located sites heavily use this type of communication to coordinate their activities; handling exceptions, solving problems and managing changes. This result suggests that a better understanding of the process by it happens must be explored in order to be properly supported in global software development teams.

6 References

[Ant, 2000] Anthes, G. H. Software Development goes Global. *Computerworld magazine*, June 26, 2000.http://www.computerworld.com .

[BCK+, 2001] Battin, R., Crocker, R., Kreidler, J. and Subramanian, K. *Leveraging Resources in Global Software Development*, IEEE Software, vol. 18, Issue 2, March/April 2001, Special Issue in Global Software Development. To be published.

[Car, 1999] Carmel, E. The explosion of global software teams. *Computerworld magazine*, Dec 08, 1997. http://www.computerworld.com .

[Car, 2001] Carmel, *E. Global Software Teams: a framework for managerial problems and solutions.* To appear as chapter for the book: Global Information Technology And Electronic Commerce: Issues for the New Millenium. Edited by P. Palvia, S. Palvia and E. Roche. To be published by Ivy League Publishing.

[CA, 2001] Carmel, E. and Agarwal, R. Tactical Approaches for Alleviating Distance in Global Software Development. *IEEE Software*, vol. 18, Issue 2, March/April 2001, Special Issue in Global Software Development. To be published.

[CKI88] Curtis, B., Krasner, H. and Iscoe, N. "A Field Study of the Software Design Process for Large Systems". *Communications of the ACM* vol. 31, n. 11, pp. 1268-1287, November 1988.

[FLM, 1991] Fischer, G., Lemke, A.C., and McCall, R. Making Argumentation Serve Design, *Human-Computer Interaction*, vol. 6, pp. 393-419, 1991.

[Fox, 1999] Fox, R. News Track, *Communications of the ACM*, vol. 42, N. 6, pp. 9-10, June, 1999

[Gar, 1999] Garner, R. Round-the-World Teamwork. *Computerworld magazine*, May 24, 1999. http://www.computerworld.com .

[GHP, 1999] Grinter, R, Herbsleb, J. and Perry, D. E. *The Geography of Coordination: Dealing with Distance and R&D Work*, In Proceedings of ACM GROUP Conference, 1999.

[Her, 2000] Herbsleb, J. Personal Communication. October, 2000.

[Her, 2001] Herbsleb, J. From Instant Messaging to Services for Converged Networks. Invited Talk, University of California, Irvine, March, 2001.

[HG, 1999a] Herbsleb, J. and Grinter, R. *Splitting the Organization and Integrating the Code: Conway's Law Revisited*. In Proceeding of International Conference on Software Engineering, 1999.

[HG, 1999b] Herbsleb, J. and Grinter, R. Architectures, Coordination, and Distance: Conway's Law and Beyond, *IEEE Software*, September/October, 1999.

[HGVF,2001] Herbles,J., Grinter, R.E.,Votta Jr., L., Finholt. T. Geographically Distributed Software Development. *Draft for review*, 2001.

[HKO+, 1995] Herbsleb, J Klein, H., Olson, G. M., Brunner, H., Olson, J. S., and Harding, J. Object-oriented analysis and design in software project teams. *Human-Computer Interaction*, vol 10, 249-292

[HMFG, 2000] Herbsleb, J., Mockus, A. Finholt, T. A. and Grinter, R. *Distance, Dependencies, and Delay in a Global Collaboration*, In Proceedings of ACM Conference in Computer Supported Cooperative Work, 2000.

[HMFG, 2001] Herbsleb, J., Mockus, A. Finholt, T. A. and Grinter, R. *An Empirical Study of Global Software Development: Distance and Speed*, In Proceeding of International Conference on Software Engineering, 2001. To be published.

[Jar, 1998] Jarvenpaa, S. L. Communication and Trust in Global Virtual Teams. *Journal of Computer-mediated communication*, Vol 3. N. 4, 1998. http://jcmc.huji.ac.il

[Kar, 1999] Karolak, D. W. *Global Software Development*. Chapter 1, "What's Driving Global Development?", IEEE Press, 1999.

[KS, 1995] Kraut, R. E. and Streeter, L.A., Coordination in Software Development, *Communications of the ACM*, vol. 38. N. 3, pp. 69-81, 1995.

[Lee, 1997]. Lee, J. Design Rationale Systems: Understanding the Issues, *IEEE Expert*, pp. 78-85, may/june 1997.

[MA, 1998] McDonald, D, and Ackerman, M. A., *Just Talk to Me: A Field Study of Expertise Location*, In proceeding of the 1998 ACM Conference on Computer Supported Cooperative Work, Seattle, WA, November, 14-18, 1998.

[Mockus,2001] Mockus, A. *Personal Communication*, March, 2001.

[NWB, 2000] Nardi, B.A., Whittaker, S. and Bradner, E. *Interaction and Outeraction: Instant Messaging in Action*, In Proceedings of ACM Computer Supported Conference in Group Work, pp. 79-88, 2000.

[PSV, 1994] Perry, D.E., Staudenmayer, N.A. and Votta, L.G. People, Organizations, and Process Improvements. *IEEE Software*, vol. 11, n. 4, pp. 36-45, 1994.

[Rot, 1998] Rothman, J. Managing Global Teams. *Software Development Magazine*, available at http://www.sdmagazine.com. August, 1998.

[Ste, 1998] Steen, M. *Thinking globally*. Nov. 02, 1998. http://www.infoworld.com/