ANALELE ŞTIINŢIFICE ALE UNIVERSITĂŢII „ALEXANDRU IOAN CUZA" DIN IAŞI
Tomul LII/LIII Ştiinţe Economice 2005/2006

ISSUES OF COMMUNICATION AND COOPERATION IN OFFSHORE SOFTWARE DEVELOPMENT

FLORIN DUMITRIU*
GABRIELA MEŞNIŢĂ*

Aspecte ale comunicării şi cooperării în dezvoltarea software-ului în regim offshore

Abstract

Offshore software development (OSD) is an important trend in information technology outsourcing field. Within this trend software development is increasingly going global, with development occurring in multiple sites located in different geographic regions. On the other hand, software development is a collaborative endeavor, in which communication and cooperation play an important role. Effective communication and coordination from multiple sites is extremely important for OSD.

Some studies reveal that cross-site communication and coordination issues can cause a substantial loss of development speed. This paper examines and analysis the issues of cross-site communication and cooperation in OSD, and how can these issues be addressed. There are various approaches to manage communication and coordination from distance. These include the introduction of communication technologies, explicit control mechanisms, and models for reducing the need for cross-site coordination.

Key words: IT offshore outsourcing, software development, collaboration, virtual environment, CMMI, distributed team

1 Introduction

Information Technology (IT) outsourcing continues to be a booming business. System outsourcing is a form of contracting (or subcontracting) with an external organization for the development of complete or partial system components, the purchase of packaged or customized package system components, or activities to support the system development life cycle. Depending on vendor's location, there are two types of outsourcing: **domestic outsourcing**, when the customer and the vendor are located in the same country, and **offshore outsourcing**, also called **international outsourcing**, that allows a firm to look outside for foreign service providers.

Offshore software development (OSD) is one of most prominent trends in the IT industry, fueled by the globalization of IT and the improvement of telecommunication facilities. The most cited reasons for offshoring include [Kontsevoi, 1999], [Yalaho, 2002,

* Associate Professor, Phd, Business Information Systems Department, Faculty of Economic and Business Administration, „Alexandru Ioan Cuza" University of Iasi, e-mail: fdumi@uaic.ro
* Associate Professor, Phd, Business Information Systems Department, Faculty of Economic and Business Administration, „Alexandru Ioan Cuza" University of Iasi, e-mail: gabim@uaic.ro

30-36), [Goolsby, 2002], [Shami, 2004]: overcoming the shortage of IT professionals, the low cost opportunities of foreign software houses, global competition imposes businesses to be proactive and to take advantage of mergers, acquisitions and alliances, the advantages of proximity to the market (international market), improve time-to-market by using time zone differences in "round-the-clock" development, improved information and communication technologies. OSD also involves some risks such as scope creep, timely completion of the project, increased coordination costs, poor software quality, work flow communication [Narayanaswamy, 2005].

Within this trend software development is increasingly going global, with development occurring in multiple sites located in different geographic regions. Global software development is shaped and challenged by at least the following boundaries [Yalaho, 2005]: temporal (various time zones), geographic (multiple global locations), social (many participants engaged in joint development work), cultural (various nationalities and organizational cultures), historical (different product development practices have emerged over time in the contractor and subcontractor companies and may be difficult to align to enable successful collaboration), technical, and political (different interests of the contractors and subcontractor(s)).

All these boundaries have an important impact on the communication and collaboration issues in OSD. Given the geographical distance introduced by offshore outsourcing, this paper offers an analysis of the communication and collaboration issues in cross-site software development and the way to address these issues.

2 The role of communication and coordination in software development

Information system development is a complex, intensive, and dynamic activity that requires close cooperation and coordination among diverse stakeholders. Systems development is a collaborative effort among managers, users, and system developers. It is a process which acquires expertise from multiple experts. Domain specific knowledge about an application and its environment, as well as technical knowledge about systems development, has to be incorporated into a cohesive framework to make the target system operable. Many people need to be involved in the systems development process because no one person can possess all the knowledge required.

Prior empirical studies of software development have found that software developers spend a considerable amount of time in communication and coordination activities [Shami, 2004]. These studies highlight the importance of communication and coordination in software development and, as a result, new innovative methodologies, such as collaborative programming, have emerged. Traditional processes are often implemented in a rigid and change-resistant manner. These methods worked well in the past but may not always be the most appropriate for today's business climate and organizational structures [Domino, 2003].

Agile methods stress the collaborative way of software development. Agile methods are a set of development methods, derived from good practices and organized in an innovative process structure. Pair programming is one such agile method. The essential elements that pair programming embraces are people and collaboration. The pair programming method involves two developers, working together in intense collaboration, producing one artifact. Pair programming relies heavily on the interpersonal interactions of those who work together, all of whom may not possess the same mind-set or be particularly compatible.

It has been long recognized that breakdowns in communication and coordination efforts constitute a major problem in collaborative software development. The complexity of the dialogue among people who involved has always been identified as one of the major factors which cause the failure of information systems projects [Chen, 1991]. One of the reasons is the large number of interdependencies among activities in the software development process, among different software artifacts, and finally, within different parts of the same artifact. For a complex and large software project, the participation of users and managers is critical to a project's success.

As interactions among individual members are very time - consuming and sometimes unfeasible, structured methods and computer systems to support project meetings become essential for the full participation of project members. Also, to overcome this problem, the field of software engineering has developed tools, approaches, and principles to deal with interdependencies. Configuration management and issue-tracking systems are examples of such tools, while the adoption of software development processes [de Souza, 2004]

3 Collaboration issues in distributed teams

As software development becomes more globally distributed, there are great rewards to be reaped for overcoming barriers to effective communication and coordination across multiple sites. Some studies of distributed software development teams suggests that distance can be a formidable barrier to global software development.

Herbsleb et al. found that cross-site communication and coordination issues cause a substantial loss of development speed [cited in Shami, 2004]. Distributed items appeared to take about two and a half times longer to complete than similar items where all the work was collocated. Difficulties arising out of working across distance also affect the inter-personal relationships among distributed workers. Software engineers involved in multi-site software development have to deal with the frustration of communicating with remote workers in different time zones, difficulties of language and culture, and lack of trust that restrict communication.

One of the most cited difficulties in distributed software development teams is *group awareness*. Group awareness is the understanding of who is working with you, what they are doing, and how your own actions interact with theirs. Awareness of others provides information that is critical for smooth and effective collaboration [Gutwin, 2004]. Software developers often have no straight forward way of finding out who was responsible for a particular component on a remote site without resorting to workarounds [Shami, 2004]. Group awareness is useful for coordinating actions, managing coupling, discussing tasks, anticipating others' actions, and finding help.

Three mechanisms help people to maintain awareness in collocated situations [Gutwin, 2004]: **explicit communication**, where people tell each other about their activities; **consequential communication**, in which watching another person work provides information as to their activities and plans; and **feedthrough**, where observation of changes to project artifacts indicates who has been doing what. Although group awareness is taken for granted in face-to-face work, it is difficult to maintain in distributed settings.

Of the three mechanisms stated above, awareness through explicit communication, through newsgroups, email, text chat, and instant messaging is the most flexible. However, since intentional communication of awareness information also requires the most additional effort, many awareness systems attempt to support the implicit mechanisms as well as communication. General approaches include providing visible embodiments of participants and visual representations of actions that allow people to watch each other work, and overview visualizations that show authorship and changes to the project artifacts. Such a tool is Jazz, an IBM Research project that embeds collaborative capabilities into an application development environment [Hupfer].

In such situations, cross-site social networks become extremely important. In the absence of such networks, the issue gets escalated to a manager. The value derived from a person's social networks is often referred to as *social capital*. If we take two companies whose employees each have exactly the same profile of skills and knowledge, they would have identical intellectual capital. However, their performance might still differ radically from each other due to different levels of social capital [Shami, 2004].

Various approaches have been suggested to manage communication and coordination across distance. These include *the introduction of communication technologies*, *explicit control mechanisms*, and *models for reducing the need for cross-site coordination*.

With the aim of improving both formal and informal communication among distributed workers, communication technologies such as video, audio, chat, instant messaging and text have been introduced in the workplace. These technologies have had limited success and face challenges ranging from poor design to adoption. However, these technologies play an important role in creating a collaborative virtual environment.

Collaborative virtual environments are online digital places and spaces where the members of a team can be in touch, work together, even when they are geographically separated. In order to support collaborative and cooperative activities, it is important that virtual environments offer the means to access appropriate information as well as communication tools. Until recently, most collaborative virtual environments have been used as meeting places where group activities are the central task. In such environments, software should aim to support a number of key features [Snowdon, 2001, 9-11]:

- *Shared context.* It can mean shared knowledge of each other's current activities, shared knowledge of other's past activities, shared artifacts and shared environment. Together, these lead to shared understandings.
- *Awareness of others*. There can be found two types of awareness: intentional and tacit. The intentional awareness is an understanding of the activities of others, which provides a context for the activity of another team member. Tacit or background awareness involves consideration of peripheral as well as focused attention and more accurately characterizes what occurs when team members are engaged in parallel but independent ongoing activities. In contrast to intentional awareness, such tasks often require moment-to-moment peripheral coordination.
- *Negotiation and communication*. Conversations are crucial for negotiation and communication about collaborative activities. Collaborative work requires the negotiation not only of task-related content, but also of task structure in terms of roles and activities and task/sub-task allocations.
- *Flexible and multiple viewpoints*. Tasks often require use of multiple representations, each tailored to a different points of view and different sub-tasks. In certain cases, different individuals may require tailored representations to provide information specific to their tasks.

Another approach to deal with the challenges of coordination across distance involves adopting explicit control mechanisms and paying attention to software processes. The adoption of practices of the Capability Maturity Model Integrated may overcome this problem. These practices typically involve explicit and visible mechanisms for coordination such as planning, defining and following a process, carefully managing requirements and design specifications, measuring process characteristics, regular status meetings to track progress, implementing a work flow system and so on. Communication and coordination mechanisms in offshore development reduce project uncertainty and improve performance [Gopal, 2002].

CMMI models are tools that help organizations improve their ability to develop and maintain quality products and services. CMMI models are an integration of best practices from four discipline: project management, systems engineering, software engineering and integrated product and process development [SEI, 2002]. CMMI consists of best practices that address product development and maintenance. It addresses practices that cover the product's life cycle from conception through delivery and maintenance. There is an emphasis on both systems engineering and software engineering and the integration necessary to build and maintain the total product.

When selecting an approach that is best suited for an organization, a first decision is to choose a representation of CMMI, either staged or continuous. The staged representation defines a set of process areas that represent five levels of maturity – and organizations at each level generally exhibit specific behaviors. The continuous representation encourages an organization to select the order of improvement that best meets the business objectives and mitigates the organizations primary areas of risk, rather than focusing on levels of maturity. Organizations measure process improvement using five capability levels. Each capability level corresponds to a level of institutionalization of the specific practices within the process area.

The final approach to managing distance has been to minimize cross-site communication and coordination needs. Conway was one of the first people to recognize the relationship between product architecture and communication structure [cited in de Souza, 2004]. He said that the structure of the product will mirror the structure of the organization that designed it. The notion of modularity, a cornerstone of software engineering, allows modules or 'work items' to be split in such a way that design decisions about each component can be made in isolation from other components.

The principle of information hiding also helps in managing dependencies. According to this principle, software modules should be both open (for extension and adaptation) and closed (to avoid modifications that affect clients). Information hiding aims to decrease the dependency (or coupling) between two modules so that changes to one do not impact the other. This principle is instantiated as several different mechanisms in programming languages that provide flexibility and protection from changes, including data encapsulation, interfaces, and polymorphism.

In particular, separating interface specifications from their implementation is a growing trend in software design. Furthermore, interface specifications are believed to be helpful in the coordination of developers working with different components.

Such division of labor reduces the requirement for coordination and communication across different parts of the organization involved in the design of different components. An implication of this would be to collocate the people involved with the development of the same component, since the tasks related with designing that component are tightly coupled and would require more communication. Tasks associated with different components are loosely coupled, and would require less communication.

Another way to minimize cross site communication and coordination needs is to outsource those tasks which require low user contact and low-skills [Yalaho, 2005]. For example, coding and testing tasks are relatively less skill-intensive but more labor-intensive. Coding does not rely on creativity, organizational understanding, or consultation with end users. Outsourcing parts of this process involves the partitioning of responsibilities and the definition of clear although necessarily overlapping knowledge boundaries between the customer and the provider.

4 Conclusions

In this paper we highlighted the issues and the approaches for addressing communication and cooperation in offshore software development. Some studies of distributed software development revealed that distance can be a formidable barrier to global software development. Cross-site communication and coordination issues cause a substantial loss of development speed due to difficulties that affect the inter-personal relationships among distributed workers and group awareness.

We presented and analyzed some approaches which address these issues, such as: the creation of cross-site social networks, the introduction of communication technologies, explicit control mechanisms, and models for reducing the need for cross-site coordination. The use of communication technology may improve both formal and informal communication among distributed workers, although there is still much room for research on the design of such technologies. The integration of CASE and computer supported collaborative work tools could be a response to these needs.

Disciplined software processes in an organization have a significant impact on coordination mechanisms. The adoption of CMMI model may help organization to improve their ability to develop and maintain quality products and services by using their explicit and visible mechanisms for coordination. The ways to minimize cross-site communication and coordination needs are: the application of modularity and information hiding principles in software architecture design and outsourcing those tasks which require low user interaction and low-skills.

References

Chen, M., Nunamaker, J.F., The architecture and design of a collaborative environment for systems definition, *ACM SIGMIS Database*, Volume 22, Winter/Spring 1991, ACM Press, New York, USA, 1991.

Domino, M.A., et al., Conflict in Collaborative Software Development, *Proceedings of the 2003 SIGMIS conference on Computer personnel research: Freedom in Philadelphia--leveraging differences and diversity in the IT workforce*, Philadelphia, Pennsylvania, ACM Press New York, NY, USA, 2003

Goolsby, K., *Offshore is not Offhand. Recommendations for Effective Offshore Outsourcing*, 2002, at http://www.c-trade.org/files/clusters/ICT/Effective%20 Offshore%20Outsourcing%20White%20Paper.pdf, accessed on 12.03.2004.

Gopal, A., Mukhopadhyay, T., Krishnan, M.S., The Role of Software Processes and Communication in Offshore Software Development, *Communication of the ACM*, ACM Press, New York, Volume 45, April 2002.

Gutwin, C., Penner, R., Schneider, K., Group awareness in distributed software development, *Proceedings of the 2004 ACM conference on Computer supported cooperative work*, Chicago, Illinois, USA, ACM Press, New York, USA, 2004.

Hupfer, S., et al., Reinventing Team Spaces for Collaborative Development Environment,at http://domino.watson.ibm.com/library/cyberdig.nsf/papers/EF2DFD427B2 7AC2185256F890071AB57/$File/rc23484.pdf, accessed on 18.03.2006.

Kontsevoi, B., *Outsourcing Offshore: a Forced Trend?*, 1999, at http://www.webspacestation.com, acessed on 17.04.2005.

Narayanaswamy, R., Henry, R.M., Effects of culture on control mechanisms in offshore outsourced IT projects, *Proceedings of the 2005 ACM SIGMIS CPR conference on Computer personnel research*, ACM Press, New York, USA, 2005.

Shami, N.S., Bos, N., Wright, Z., Hoch, S., Kuan, K.Y., Olson, J., Olson, G., An Experimental Simulation of Multi-site Software Development, *Proceedings of 2004 Conference of the Centre for Advanced Studies on Collaborative Research*, IBM Press, Markham, Ontario, Canada, 2004.
Snowdon, D., Churchil, E.F., Munro, A.J., Collaborative Virtual Environments: Digital Spaces and Places for CSCW: An Introduction, *Collaborative Virtual Environments*, Springer-Verlag, London, 2001.
Software Engineering Institute (SEI), *CMMI^{SM} for Software Engineering (CMMI-SW, V1.1). Staged Representation*, Software Engineering Institute, Carnegie Mellon University, 2002, at http://www.sei.cmu.edu/cmmi/models/v1.1se-sw-staged.doc, accessed on 24.02.2006.
de Souza, C.R.B., et al., How a good software practice thwarts collaboration: the multiple roles of APIs in software development, *Proceedings of the 12th ACM SIGSOFT twelfth international symposium on Foundations of software engineering***,** Newport Beach, CA, USA, ACM Press, New York, USA, 2004.
Yalaho, A., Wu, C., *IT-Supported International Outsourcing of Software Production*, 2002, at http://www.cs.jyu.fi/sb/Publications/WuYalaho_Masters Thesis_06092002.pdf, accessed on 16.07.2004.
Yalaho, A., Nahar, N., Kakola, T., Wu, C., A conceptual process framework for IT-supported international outsourcing of software production, *Proceedings of the IEEE EEE05 international workshop on Business services networks*, Hong Kong, IEEE Press, Piscataway, New Jersey, USA, 2005.