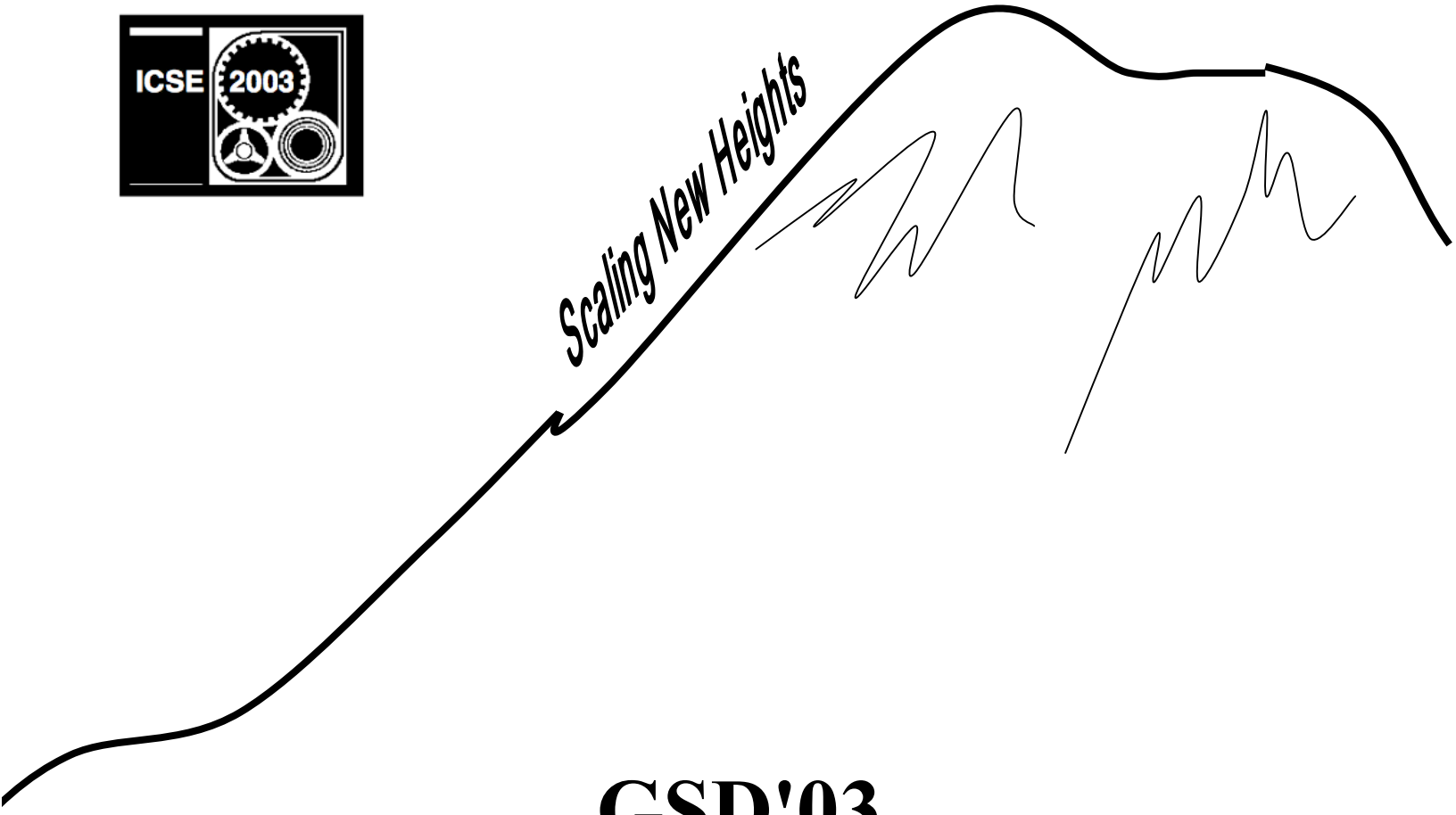


GSD'03
The International Workshop on Global Software Development

ICSE'03
International Conference on Software Engineering
Portland, Oregon
May 9, 2003

TABLE OF CONTENTS

WORKSHOP INTRODUCTION

KEYNOTE ADDRESS

SESSION 1: TOOL SUPPORT

SESSION 2: EMPIRICAL STUDIES & EXPERIENCE REPORTS (1)

SESSION 3: EMPIRICAL STUDIES & EXPERIENCE REPORTS (2)

Addressing the Challenges of Software Industry Globalization: The Workshop on Global Software Development

Daniela Damian
University of Victoria, BC, Canada
danielad@cs.uvic.ca

Filippo Lanubile
University of Bari, Italy
lanubile@di.uniba.it

Heather L. Oppenheimer
Lucent Technologies, NJ, USA
hoppenheimer@lucent.com

Abstract

The goal of this workshop is to provide an opportunity for researchers and industry practitioners to explore both the state-of-the art and the state-of-the-practice in global software development (GSD).

Increased globalization of software development creates software engineering challenges due to the impact of temporal, geographical and cultural differences, and requires development of techniques and technologies to address these issues. The workshop will foster interaction between practitioners and researchers and help grow a community of interest in this area. Practitioners experiencing challenges in GSD will share their concerns and successful solutions and learn from research about current investigations. Researchers addressing GSD will gain a better understanding of the key issues facing practitioners and share their work in progress with others in the field.

1. Workshop description

This workshop is a continuation of the last five ICSE workshops on the same topic (1998-2002; [1-5]). Last year [5], after changing the title from the technology-focused "Software Engineering over the Internet" to the more general "Global Software Development", we observed increased participation by practitioners and more fruitful discussions between industry and academia. The report summarizing the workshop [5] discusses the challenges of engineering software in geographically distributed settings and indicates that further research needs to equally address issues of technical and social nature in global software development.

Global software development has been and continues to be a phenomenon fueled by factors such as access to a large and specialized labor pool, reduction in development costs, global presence and proximity to the customers. While we are witnessing reports of successful global teams, research reveals that distance contributes to heightened complexity in organizational processes. Primarily, processes of communication, coordination and control are affected by distance, with direct consequences on how software is defined, constructed, tested and delivered to customers, as well as how its development is managed. Further, cultural issues are possibly a most confusing and interesting feature of global teams. Members with diverse attitudes towards hierarchy, time management and risk avoidance come to work together in cross-functional teams.

These are only some of the factors that bring challenges to managing software projects developed in geographically distributed structures. Understanding the intricacies of this complex phenomenon is critical in framing research directions that aim at improving global software development practice. There is a need for tools and techniques that not only improve development processes but also address organizational and social issues in global software development. The previous workshops represented one more step in identifying and understanding issues in the complex phenomenon of global software development. In particular, the empirical evidence and discussions during the workshop last year indicate that technology is only a small part of enabling effective global teams; there is a strong need to address the study and practice of global software development from a multidisciplinary perspective, in which issues of social nature are as important as those of technical nature.

In this workshop we intend to continue fostering fruitful interactions between industry practitioners and researchers and help grow a community of interest in this area. Industry practitioners experiencing challenges in GSD will be encouraged to share their own solutions and learn from research about current investigations in this area. Researchers addressing GSD will have the opportunity to gain a better understanding of the key issues facing industry practitioners and share their work in progress with others in the field.

2. Workshop themes

The workshop solicits papers on topics that include, but are not limited to:

- Empirical evaluations of effectiveness of global software projects
- Technologies & tools for distributed development environments
- Software engineering methodologies & processes for GSD
- Communication, collaboration, and knowledge management in distributed organizations

3. Workshop format

This workshop will focus on identifying issues of Global Software Development, sharing solutions, and brainstorming new approaches to those issues. The workshop is open to all participants interested in the topic; position papers or technical papers, though encouraged, are not required for attendance.

The "issues" part of the workshop will be dedicated to identifying, classifying, and categorizing Global Software Development issues that have been raised in previous literature, described in the position papers, and gathered during GSD 2002. To help focus the afternoon discussion, we will create some GSD scenarios that typify key issues. Throughout the workshop, participants can propose additional issues to be discussed at the end of the day.

The "solutions" part of the workshop will study existing techniques and methods for Global Software development. Selected authors of accepted technical papers will deliver very short presentations on their technological solutions or methodological approaches. Both existing and proposed technology will be assessed from a technical consistency perspective and evaluated for industrial applicability and feasibility. We will encourage each technical presenter to discuss how his or her ideas address or relate to the problems illustrated in the scenarios. The presentations will include a small amount of time for audience discussion of each set of presentations, hopefully allowing the group both to better understand the ideas and to relate them to other presentations and to the workshop themes.

The final part of the workshop will be a plenary discussion aimed at finding synergies between solutions, where crossover work can lead to advances that might otherwise go unexplored, and identifying opportunities for further work. At the end of the day, we will integrate and present the results of the discussion. The workshop will lead to a list of issues discussed, solutions proposed, conclusions reached, disagreements identified, and topics to be researched further.

All accepted papers will be published in both the workshop proceedings and on the workshop website at: http://gsd2003.cs.uvic.ca

4. About the organizers

Daniela Damian is an Assistant Professor at the University of Victoria, BC, Canada. She is a Killam Scholar in Canada and the recipient of the Best Paper Award at the International Conference on Requirements Engineering 2000. Daniela was the primary contact and co-organizer of the Workshop on Global Software Development (2002) and acted as a Member of Program Committee of the Australian Workshop on Requirements Engineering, 2000, 2001 and 2002, as well as the Time-Constrained Requirements Engineering Workshop at the Int'l Conf. on Requirements Engineering '02.

Filippo Lanubile is an Associate Professor at the University of Bari, Italy. While at University of Maryland (1995-1997) he was a recipient of the NASA Group Achievement Award (1996). On 2002, he was a program committee member for the Workshop on Global Software Development and the Workshop on Cooperative Support for Distributed Software Engineering Processes. In the last years he was also a member of the program committees of METRICS and ICSM. In 1997, he organized the Int. Workshop on Empirical Studies of Software Maintenance.

Heather Oppenheimer is a Distinguished Member of Technical Staff at Bell Laboratories, USA with 17 years of industry experience in software engineering. She has managed groups responsible for system test, process & quality engineering, and internet & collaborative development environments as well as all other aspects of the development life cycle. She currently leads a corporate software best practices initiative, develops and instructs internal courses in software architecture for systems engineers and for managers of development teams and leads software architecture reviews and project management audits throughout the Lucent Technologies global organization. She was a program committee member for the Workshop on Global Software Development (2002).

References

[1]. First Workshop on "Software Engineering over the Internet", online proceedings: http://sern.ucalgary.ca/~maurer/ICSE98WS/ICSE98WS.html

[2]. Second Workshop on "Software Engineering over the Internet", online proceedings: http://sern.ucalgary.ca/~maurer/ICSE98WS/ICSE99WS.html

[3]. Third Workshop on "Software Engineering over the Internet", online proceedings: http://sern.ucalgary.ca/~maurer/ICSE98WS/ICSE2000WS.html

[4]. Fourth Workshop on "Software Engineering over the Internet", online proceedings: http://sern.ucalgary.ca/~maurer/ICSE98WS/ICSE2001WS.html

[5]. International Workshop on Global Software Development, online proceedings at: http://www.cis.ohio-state.edu/~nsridhar/ICSE02/GSD

KEYNOTE ADDRESS

Research Methods and Theory in Global Software Development

James Herbsleb
School of Computer Science
Institute for Software Research International

The talk will discuss the research methods that I have found to be useful in researching coordination issues in GSD, some of the challenges I've faced in addressing specific research questions, and ways of combining methods to address these problems. Finally, I will briefly present some current theoretical work that evolved from studies of GSD, that attempts to create empirically testable theory to characterize and make predictions about coordination of engineering decisions. I will argue that GSD provides a setting in which very general problems of coordination are naturally exposed to observation and experimentation.

Awareness meets requirements management: awareness needs in global software development

Daniela Damian, James Chisan, Polly Allen
Dept. of Computer Science
University of Victoria, BC
PO Box 3055, BC V8W 3P6, Canada
{danielad,chisan,allenp}@uvic.ca

Brian Corrie
New Media Innovation Centre
600-515 West Hastings Street
Vancouver, B.C., Canada, V6B 5K3
Brian.Corrie@newmic.com

Abstract

There has been growing attention to awareness issues in group collaborative processes. In this paper we address workspace awareness in requirements management processes in global software development endeavors.

When working on a software project, developers, system analysts, testers, and managers make use of information about current sets of requirements, design artifacts and relationships to customer requirements or test cases, as well as roles and responsibilities assigned to particular work artifacts. Co-located teams benefit from social mechanisms and processes that naturally facilitate the work practice and diminish the perceived need for explicit workspace awareness support. However, limited access to informal communication in geographically distributed teams make problems caused by reduced workspace awareness more acute.

We propose a set of features for awareness support in geographically distributed requirements management activities and outline our first step in researching such awareness issues. In finding usable and sensible solutions to this problem, we may be creating solutions long overdue in requirements engineering in general.

Introduction

In this paper we address the issue of awareness in requirements management processes of global software development (GSD) teams. We believe that awareness needs are inherent in all collaborative processes of software development but particularly critical in requirements management activities carried throughout the software development life-cycle. When working on a software project, developers, system analysts, testers, and managers make use of information about sets of outstanding requirements, design artifacts and relationships to customer requirements, test cases and traceability links to system requirements, and roles and responsibilities assigned to work on particular work artifacts. We will refer to the availability of this information as workspace awareness.

Co-located teams benefit from social mechanisms and processes that naturally facilitate the work practice and diminish the perceived need for explicit workspace awareness support. However, access to informal communication in geographically distributed teams is significantly limited. Therefore, problems caused by reduced workspace awareness are more acute.

After discussing the need to address awareness in requirements management in GSD, this paper proposes a preliminary list of features of awareness support in requirements management, and discusses our immediate practical approach to research such awareness support in global software projects.

The need to address awareness needs in requirements management in GSD

The issue of awareness in requirements management in software development requires more adequate research treatment. Requirements as expressions of interests, goals and needs in software development reside with the stakeholders, who are rarely all physically co-located. Requirements are thus naturally distributed and we believe that the need for workspace awareness in requirements engineering has always existed. Even in ideal circumstances, it is extremely unlikely that all stakeholders of a software project would share the same workspace. Consider, for example, a classic three-stakeholder software project consisting of customer, user and developer. If these stakeholders inhabited the same physical space, they would all have an intimate appreciation of the day-to-day dilemmas they each face. In reality these roles are separated organizationally, politically and especially, physically. Knowing the source or rationale of a particular requirement, who is working on implementing a particular requirement, or who has intimate knowledge about the rationale for a test scenario, is often not a trivial task. Although the requirements process might explicitly require the dissemination of such

information during the development process, this necessity is, unfortunately, rarely in place.

Research into the tasks and communication patterns that occur in traditional large software projects (Curtis, Krasner and Iscoe, 1988) finds that in co-located environments information about design problems is often propagated informally, through people meeting in the hallway, chatting over coffee, during design or code review meetings. Although companies establish formal processes for making and reviewing decisions about the design of large systems, these structures are often ineffective for communicating design problems that arise in sections of the organization that are not part of the formal process. Rather, informal personal contacts are frequently the most effective way to communicate messages across organizational boundaries.

Awareness in collaborative activities

Awareness simply refers to knowledge one has of the environment in which one finds oneself, essentially, *knowing what is going on* (Endsley, 1995). Awareness is thought to have two basic characteristics. First, awareness constitutes accurate knowledge about the state of a dynamic, changing environment. Second through normal interactions with the environment one is able to maintain one's awareness as a side effect of that interaction (Gutwin and Greenberg, 2001). This suggests awareness is naturally tacit, and that the means for acquiring this knowledge must be low-cost or even no cost for the recipient. Awareness serves to provide understanding of what is happening, who is affecting change, and where it is happening (Kobylinsky, Creighton, Dutoit and Bruegge, 2002). While there has been much attention paid to addressing general awareness in the physical environment, there is significant opportunity to support awareness of the workspace environment. In software development, this environment consists primarily of documents and code. Capitalizing on this opportunity requires comprehending what aspects of this environment may be of some value to stakeholders, how to capture those aspects and how to deliver awareness.

A number of awareness issues have been identified with respect to requirements engineering, in particular document change and contact identification (Herbsleb, Mockus, Finholt and Grinter, 2000). It has been found that prescribed apparatus to propagate information updates (typically formal documentation) is untimely, if not ineffective. Furthermore, coordination and conflict resolution depends on knowing whom to contact about what, a task commonly impeded by the unavailability of such information. Progress often suffers from 'organizational amnesia' where issues that have already been discussed and seemingly resolved are rehashed, reflecting limited collective memory (Catledge and Potts, 1996). Attempts have been made to address these issues with tools such as instant messaging, shared calendars, and web boards. It is our belief that reliance on informal communication still prevails in spite of these attempts.

GSD removes the informal interactions that normally promote awareness, so their effects need to be replicated by some kind of technology. While awareness about requirements information could be maintained in organizational and project documents, they have been shown to be very poor communication media (Curtis *et al*., 1988; Al-Rawas and Easterbrook, 1996). In particular, when requirements change, formal mechanisms such as documents do not react quickly enough and often news is propagated informally (Herbsleb *et al*, 2000)

A strong motivator for this research is an earlier focused investigation of requirements engineering challenges in globally distributed software development organizations (Damian and Zowghi, 2002). This work clearly emphasizes the need for mechanisms to support awareness of requirements and related artefacts in their management in GSD. In software projects where stakeholders worked on several continents, there was incomplete and unreliable information about incoming requests, or priorities and issues related to particular software requirements. Not only did decision makers have difficulties keeping up to date on the latest decisions made on outstanding or emergent issues, but they also had difficulties obtaining a snapshot of the current version of features being implemented and related information such as decisions about change request approvals. Similarly, software developers reported difficulties in contacting originators of requirements and related issues, for clarification or elaboration, and in determining the responsible stakeholders for a particular decision.

Although we are witnessing rapid advances in collaboration technologies that potentially provide support for software development in global settings, studies of global teams (e.g. (Damian and Zowghi, 2002), (Herbsleb *et al*, 2000)) identified that (1) collaborative technologies for global collaboration are used in an ad-hoc and inadequate manner and (2) we lack an understanding of appropriate technological capabilities necessary to overcome challenges due to geographical distribution.

The main barrier to using collaborative tools more effectively is the lack of understanding we have of the tasks being performed and the information that is pertinent to completing those tasks successfully. In requirements management it is necessary to understand not only the formal process in great depth, but also to understand the informal processes that are used to facilitate the formal process. Generic awareness tools that provide information on the physical environment, although useful, do not provide specific information that is relevant to the requirements management process. It is our belief that tools that provide awareness about the

artefacts involved in requirements management are required to deliver a successful distributed requirements management environment.

Our Research approach

In this paper we propose a set of features for awareness support in geographically distributed requirements management. We then discuss a research project in which we are attempting to study the features of such awareness support. We welcome feedback from discussions in the workshop. At the workshop we intend to report on our insights into refining these requirements from the experience of implementing a system to meet these requirements.

In discussing the features of an awareness system in requirements management, we consider the support it should provide to tasks team members perform during requirement management activities, particularly tasks which require requirements awareness information. Features are highlighted in bold font while data to be tracked by the system is underlined.

1. Allow project stakeholders to seek existing information

In general, this is the simplest though potentially most frequent task that project team members perform during requirements management: finding information about requirements and the status of their implementation. The awareness system should allow team members to query or browse information about:

- Requirements: This includes a description of the requirement itself, meta-information about the requirement (such as rationale, priority, stability level) as well as inter-dependencies with other requirements and software development artefacts (e.g. design, code or test). Another piece of information that often resides with the team members and is rarely well documented are the issues and decisions related to particular requirements. The ability to access this rich information about requirements contributes to alleviating problems of "organizational amnesia" (Catlege and Potts, 2000).
- People: This includes the roles and responsibilities for each team member related to the implementation of particular requirement(s), including personal contact information. This contact information can be as simple as a phone number or an e-mail address, but in geographically distributed settings it may be important to include at a minimum, some indication of the team member's office location and time zone. More elaborately, it may even include physical awareness cues, giving system users indications of where team members can be found and how best to contact them.
- Specific relationships between People and Requirements: This information would identify initiators, decision makers, analysts, estimators and those currently responsible for the realization of a requirement. These relationships can be stored by tracking the initiators of a requirement, issues (including who initiated the issue, the resolution status of the issue, and any decisions made because of the issue), meetings (including the date, time and location of the meeting, the stakeholders involved, the issues discussed and decisions made), and change requests (including who initiated the change request, the status of a decision regarding the change request, who was involved in that decision and the decision itself).

2. Support stakeholders in decision making

Besides seeking information about requirements, for purely informative purposes, project members need the information outlined above to make decisions. Decisions with respect to the implementation of requirements are needed throughout the project lifetime. Project managers may try to decide how to assign resources to a project, whether it is feasible to implement a feature with the given resources, or whether the project plan is proceeding smoothly. Team members may try to decide if their understanding of the current requirements baseline is correct, whether their implementation of a feature fits with the rationale behind a requirement, or who to contact for more information regarding a previously made decision. All decisions need to be recorded so that they can feed back into the system; in this way, all team members benefit from the group's experience.

The high-level task of decision making may involve any of the following subtasks:

- **Assigning responsibilities and managing a project**

  At the beginning of a project, people or teams are made responsible for the implementation of each requirement or feature. If the information about responsibilities is captured, it is possible to track the particular states of requirements with respect to their progress. This enables project members to easily become aware of who is working on a particular requirement and query the current state of requirements. Similarly, the project manager can easily identify assigned people responsible for the requirements from the assigned people and/or

dynamically delegate the implementation of particular requirements to available personnel.

- **Gathering and managing estimation data**
  Decision-makers often need to involve other stakeholders during estimation gathering for the purposes of negotiation and project management. This may be thought of as an iterative propagation of requests-for-information, clarifications and estimations between stakeholders. Tracking information about the interactions that occurred, the history of estimates as clarifications were exchanged, and the eventual estimates on which decisions were based aids in future decision-making, it also allows users to decide how the project is proceeding according to original, or amended, estimates.

- **Impact analysis**
  To support decision-making, users need to analyze the impacts of potential decisions. Again, awareness of the relationships between requirements, and between requirements and other artefacts of the software lifecycle, can help users understand the extent of changes being proposed. Further, the awareness of relationships between requirements and personnel responsible for the implementation of requirements is needed in the change notification process. Decisions made as a result of change requests need be made available to the affected stakeholders

3. Decision notification

For future decision-making activities, the decision must be documented. This documentation should include references to the information listed above under "Seeking existing information", as well as rationale for the final decision. An awareness tool must provide a means to decide which stakeholders are affected. To this end, the responsibilities of each team member must be tracked as mentioned above. Finally, the decision must be made available to all stakeholders to support coordination.

Design and evaluation of awareness support in requirements management of global projects

Our current endeavors in researching such awareness support for requirements management is a design and development effort for a tool that meets the criteria outlined above. A first step is being taken in a graduate course in the department of Computer Science at the University of Victoria, between January and April 2003. This research effort will serve as a case study for requirements engineering, and indeed software system engineering, in a geographically distributed setting. Our goal in this study is to discover if the requirements outlined above are complete or if other needs arise during the development of the tool. It is our hope that the workshop will present the authors with the opportunity to discuss the case study and our experience of designing and using awareness tools for requirements management in geographically distributed software development.

In this course, the authors themselves design and develop awareness tools in a geographically distributed software development environment. The course provides the opportunity for a case study in which the features outlined above are considered as a starting point in the development. The activities of distributed requirements management are experienced during this development effort.

Group face-to-face meetings will be held at the University of Victoria once every four weeks. Except for these meetings, one participant will be physically separated from the University, working in Vancouver and participating in formal group meetings via audio or videoconference. During these formal meetings, documents and applications will be shared between locations using distributed application sharing software, and interactions will be facilitated and recorded using an electronic whiteboard. Microsoft's NetMeeting software was chosen due to its ubiquitous availability on all Windows platforms. Requirements will be managed using the Rational RequistePro requirement management tool. The two other participants, both working on the University campus, will be co-located for formal meetings, but will attempt to avoid informal communication regarding the project outside of the computer-supported environment. Informal communication between all three participants will be limited to electronic mail, instant-message and chat programs, voice-over-IP or telephone conversations, or videoconference tools. The group uses the Groove collaborative software (http://www.groove.net) to provide both formal collaboration capability (shared documents) as well as informal communication (instant messaging, threaded discussion groups, and physical awareness).

The tools used for this project are a selection of widely available collaboration technologies that have been coalesced into a suite of tools directed at accomplishing the requirements management task. The tools being used for this project were selected based on several requirements:

- They provided the collaboration capabilities required by the project
- They provided adequate quality (audio, video, etc.) for the distributed collaboration
- They were easily available to all collaborators

Discussion

We expect that our experience in developing the awareness tools using this computer-supported collaboration infrastructure will be invaluable. Our goal is to take a first step toward the development of an awareness tool that is consistent with the requirements described in this paper. By engaging in this exercise within a geographically distributed environment we expect to experience, first-hand, many of the same awareness problems, that we wish to address. We hope that both the development and this experience will equip us with improved understanding of the problem and indications for future direction of research and exploration.

We are aware that this research project represents only a first step in studying the issue of awareness in requirements management for global software teams. We acknowledge a number of potential limitations of the study:

- Due to the size of the group, the study is not fully representative of a global software development environment.
- Due to the fact that the study is offered as part of a course, the project environment is controlled.
- Due to the fact that the project only simulates global distribution, issues of cultural, organizational, and environmental factors are minimal

Conclusion

Requirements management is gaining increased interest and appreciation as a difficult task in software development. Much of the information about project status and change management is typically propagated though informal communication in an organization. Perhaps these channels have always been inefficient; many system errors can still be traced to requirements misunderstandings, miscommunications or mismanagement. Only since the large-scale adoption of GSD, and the removal of these informal mechanisms, has the problem of lack of awareness been highlighted. In finding usable and sensible solutions to this problem, we may be creating solutions long overdue in requirements engineering in general.

References:

1. Al-Rawas, A. and Easterbrook, S. Communication problems in requirements engineering : a field study, Proc of the First Westminster Conference on Professional Awareness in Software Engineering, Londong, 1996, 47-60
2. Catledge, L. and Potts, C., "Collaboration During Conceptual Design," in Proc. 2nd Intl. Conf. on Requirements Engineering, (Colorado Springs, Colorado, USA), pp. 182--189, 1996
3. Curtis, B., Krasner, H. and Iscoe, N. A field study of the software design process for large systems, Communication of ACM, 31(11), 1988, 1268-1287
4. Damian, D. and Zowghi, D. The impact of stakeholders' geographical distribution on requirements engineering in a multi-site development organization, Proc. of the 10th IEEE Int'l Conference on Requirements Engineering (RE'02), Essen, Germany, 2002, 319-328
5. Endsley M.R. Toward a Theory of Situation Awareness in Dynamic Systems, Human Factors, 37(1), pp.65-84, 1995
6. Gutwin, C., and Greenberg, S. (In Press) The Importance of Awareness for Team Cognition in Distributed Collaboration. In E. Salas, S. M. Fiore and J. A. Cannon-Bowers (Editors) Team Cognition: Process and Performance at the Inter- and Intra-individual Level. APA Press. 2001
7. Herbsleb, J.D., Mockus, A., Finholt, T.A., & Grinter, R.E. Distance, Dependencies, and Delay in a Global Collaboration. Proceedings of CSCW 2000, Philadelphia, PA, Dec. 2-7, 2000.
8. Kobylinski, R., Creighton, O. Dutoit, A.H. and Bruegge, B. Building Awareness in Global Software Engineering Projects: Using Issues as Context, International Workshop on Global Software Development (co-located with ICSE '02), Orlando, FL, May 21, 2002.

A P2P Toolset for Distributed Requirements Elicitation

Filippo Lanubile
Dipartimento di Informatica, University of Bari
Via Orabona 4, 7026 Bari, Italy
lanubile@di.uniba.it

Abstract

Geographically distributed teams need nowadays models and tools to support a load of activities that usually take place through the direct interaction among people. Our research effort is aimed to understand how decentralized systems, based on a peer-to-peer architecture, can be exploited to support the key activities of global software development. As a first step, we have focused on requirements elicitation because it is among the most communication-rich processes of software development. This paper presents a toolset for distributed requirements elicitation, which is developed on the basis of a peer-to-peer infrastructure platform, called Groove.

1. Introduction

Globally distributed workgroups usually rely on centralized systems, mostly built on top of web-based development platforms. Two examples of centralized infrastructure are SourceCast [10] and SourceForge [11], two collaborative development platforms, that include web-based access to defect and issue tracking, version control and configuration management, mailing lists and discussion forums.

Our research effort is aimed to understand how decentralized systems, based on a peer-to-peer (P2P) architecture, can be exploited to support collaboration across time and space for global software development.

Collaborative P2P applications are increasingly becoming popular to exchange instant messages, share common information and applications, and jointly review and edit documents. Collaborative P2P systems exhibit the following advantages with respect to client-server counterparts: autonomy, intermittency, and immediacy.

- Autonomy. In a P2P system every peer is an equal participant while being a final authority over its local resources. In this way everyone can share information but at the same time can pose restrictions on confidential data through access rights management and data encryption. When enterprise data are distributed on many places and on different devices, P2P systems can provide an easier and cheaper alternative to enforcing a convergence into a centrally managed data repository.
- Intermittency. P2P systems are designed by giving for grant that any peer can disappear at any time because of network disconnections, either deliberate or accidental. P2P collaborative systems use resource replication and different synchronization mechanisms, based on proxies for sending/receiving messages in the network on behalf of the disconnected sender/receiver. In this way, users can work to shared content even when offline and automatically propagate changes at the first reconnection.
- Immediacy. P2P applications have shown themselves able to support direct exchanges between peers, as in the case of instant messaging. P2P collaboration systems, based on near real-time communication mechanisms and synchronous presence of the peers, can provide immediate responses by participants to enable effective person-to-person interaction.

Under these conditions, a P2P collaborative infrastructure can complement or even replace client-server platforms for the creation of ad-hoc or small software teams. Due to P2P own features, it is possible to quickly establish dynamic collaborative groups composed of people from different organizations accessing shared resources and interacting in a near real-time manner.

Among the many collaborative software engineering activities, the focus in this paper is directed at the software requirements activities, specifically requirements elicitation. Eliciting requirements engage different stakeholders, both from the customer and the developer sides, who need to intensively communicate and collaborate. As a key part of the requirements engineering process, requirements elicitation has a great impact on the later development activities; any omission and incompleteness may lead to important mismatches between customers needs and released product.

This paper introduces a distributed requirements elicitation toolset which was developed on the basis of a P2P infrastructure platform, called Groove. Section 2 presents the functionality of the collaborative toolset for eliciting requirements from geographically dispersed stakeholders. Section 3 describes the P2P platform which we used as an application framework. Section 4 concludes the paper and suggests opportunities for future joint research.

2. Requirements elicitation toolset

Requirements engineering is the most communication rich process of software development, and then the effectiveness of a requirements engineering process is greatly constrained by the geographical distance between stakeholders [3], as in the case of global software development.

Some simple P2P tools have been already introduced to support communication between distant stakeholders. For instance, Microsoft's NetMeeting has been used in requirements negotiation [2] for its instant messaging capabilities, to create a video and audio link between people, and to share desktop applications. But the features offered by generic tools provide a partial support to specific requirements engineering activities, and often there is no way to integrate them with commercially available requirements engineering tools, such as Rational RequisitePro.

For this reason, there is a need to develop a tool infrastructure to support globally distributed workgroups when developing requirements. Given the characteristics of autonomy, intermittency and immediacy, we believe that a P2P-based toolset is suited for the cooperation-intensive needs of requirements engineering activities.

As a first step, we focused on requirements elicitation which is regarded as the first activity in the requirements engineering process [4]. Eliciting requirements is an information gathering activity with the goal to identify system stakeholders, their needs and expectations, system objectives and boundaries. Elicitation techniques include questionnaires and surveys, interviews and workshops, documentation analysis and participant observation.

We developed a P2P toolset to be used for eliciting requirements in a distributed context. In the following we briefly describe each of the five tools comprising the integrated toolset.

2.1. Stakeholders tool

The first step in requirements elicitation is often the identification of who brings an interest in the software project, both on the customer and the supplier sides, and then will have an influence on requirements definition.

The *Stakeholders* tool acts as an archive of information regarding the stakeholders involved in a project. The tool can be thought as a shared contacts list augmented with the annotation of the role (even multiple roles) in the software project. Given the great variability for role definition, the tool was developed with a set of default roles from the Volere requirements specification template [9], although it can be customized according to different needs. The tool accounts for changes of roles during the project lifetime, and for multiple roles of a single stakeholder. Provided that a stakeholder has been included, he or she can communicate with the other ones by either sending asynchronous messages or by inviting to join a chat.

2.2. Interview tool

Interviews are frequently used in requirements elicitation to gather detailed information from stakeholders. It also allows the requirements engineer to hear different viewpoints that might need to be reconciled.

In this tool an interview is set up through a wizard. There are two types of available interviews: structured or unstructured; in the former case the requirements engineer can decide whether to prepare the interview from scratch or not. If not, the tool supplies a set of interview templates for the elicitation of high-level system goals and architecturally significant requirements [5]. Using these templates, it is also possible to trace questions with respect to a requirements taxonomy.

Another feature of this tool is the availability of multiple views of the interview according to the interviewee's role. In this way the same templates can be customized for different stakeholders.

2.3. Requirements tool

The *Requirements* tool is used as a shared repository of elicited requirements. The tool exploits the P2P feature of data sharing: data are replicated (and encrypted) on everyone's computer and made accessible even offline.

A requirement is described by the following attributes: ID, version, name, rationale, description, priority, difficulty, stability, and status. An additional attribute, requirement type, was added accordingly to the requirements taxonomy used in the interview templates. In this way it is possible to trace stored requirements with related questions from the *Interview* tool. Users can also define new requirements categories.

2.4. Workshop tool

The *Workshop* tool replaces the usual face-to-face workshop in which a group of experts and stakeholders

works together with the goal of information discovery and creation. Live discussions can be conducted either through a chat or an audio link.

A workshop requires an agenda to let participants follow a discussion flow, and two specific roles: facilitator and scribe. The facilitator leads the process by monitoring the discussion and managing group's dynamics. The scribe documents the group's work by taking notes as the discussion proceeds.

The tool helps to prepare the workshop through a wizard which lets to define the target, the discussion topics (they can be derived from interview logs), the participants (and their role) and the meeting schedule.

2.5. Vote tool

The *Vote* tool can be used in the context of a workshop, if controversies arise during discussion, or as a standalone tool as well.

Voting includes a preparation phase in which it is possible to add open or closed-answer questions to a shared list. Voting can be set up as a secret or evident ballot, depending whether or not participants' choices have to be kept anonymous.

3. Decentralized platform

We have implemented the requirements elicitation toolset on the basis of Groove, an extensible decentralized platform intended for communication, content sharing, and collaboration [6].

Collaboration activities with Groove take place in a shared application space, which is accessed from a rich application client, called *transceiver*. A shared space, including tools and persistent data, is duplicated on every space member's computer. Data within a shared space are encrypted, both on disk and over the network, to assure confidentiality and integrity. Both data and commands are transformed, stored and transmitted as XML documents.

Every modification made to a shared space is propagated to the other peers. If users do not stay connected at the same time, the shared space gets synchronized when a peer goes online. In this way the state of the shared space remains the same for all peers.

As other P2P systems, such as Napster, the actual interaction model of Groove is hybrid because peers can establish direct connections using a Groove's native P2P protocol called SSTP (Simple Symmetrical Transmission Protocol), while the following services are supplied through central servers:

- Presence awareness: when a peer running Groove goes online, it registers with a presence server. In this way other peers can scan the presence server if interested in collaborating with other ones.
- Relay: when a peer is offline, communications destined to it are sent to a relay server which will deliver data to the peer when it reconnects. Relay servers are also used when peers reside behind a firewall that only allows outbound connections. Since firewalls are usually configured to allow employees to access the web, Groove leverages this existing configuration to send and receive messages through HTTP tunneling.
- Fanout: when the same information must be sent to many users at the same time, data are transmitted once to a server which retransmits them to other peers.

With Groove, shared space members can use predefined tools supplied by Groove Networks or by third parties. Predefined tools include instant messaging, chat, threaded discussions, audio-conferencing, shared files, shared contacts, group calendaring, group editing, group drawing and co-browsing. Developers can also build their own applications in the form of single tools or integrated toolsets.

To build the distributed requirements elicitation toolset we used Groove as an application framework, by reusing Groove's critical services (storage, transport, encryption, synchronization, messaging, presence awareness) and default tools (e.g., Contact Manager, Outliner, Chat) with the addition of scripts and XML code, accordingly to the Groove Development Kit (at the time of development we used GDK 1.3).

4. Conclusions

P2P is not a new technology but it is now emerging as an alternative or complement to client-server models for designing collaborative development systems. In order to investigate issues that can be encountered when providing P2P tool support for global software development, we developed an integrated toolset for distributed requirements elicitation.

In the field of collaborative software development environments the P2P technology has begun to being introduced. At University of Calgary, a project has recently started [1] to port an existing process-support environment (called MILOS) from a client-server model to a P2P model, thus relaxing centralized control. The migration project intends to use JXTA [7] as a P2P framework., JXTA is a set of protocols for P2P applications without restrictions to particular operating systems or network services. Although the JXTA specification is open to any programming language, the most widely used implementation is in Java.

As for Groove, JXTA uses XML to exchange data between peers. However, JXTA and Groove are not fully comparable: the former is a low-level, general-purpose

platform and, hence, provides only services and no built-in tools, while the latter is a very full-featured collaborative application, which can also be used as a development platform. By the way, the author is taking part to the development of a P2P remote-conferencing tool, based on the JXTA framework. This is an open source project, hosted by jxta.org [8] and owned by Fabio Calefato at University of Bari.

We wish that this paper will rise a discussion on developing and using tools to support collaborative activities for global software development. We also hope that the workshop will give us the opportunity to start collaborations for experimenting with the requirements elicitation toolset.

5. References

[1] S. Bowen, and F. Maurer, "Using peer-to-peer technology to support global software development – some initial thoughts", *Proc. of the ICSE Int. Workshop on Global Software Development*, Orlando, FL, USA, May 2002.

[2] D. E. Damian, A. Eberlein, M. L. G. Shaw, and B. R. Gaines, "Using different communication media in requirements negotiation", *IEEE Software*, 17(3), May/June 2000, pp.28-36.

[3] D. E. Damian, "The study of requirements engineering in global software development: as challenging as important", *Proc. of the ICSE Int. Workshop on Global Software Development*, Orlando, FL, USA, May 2002.

[4] A.M. Davis, *Software requirements: analysis and specification*, Prentice-Hall Press, Upper Saddle River, NJ, 1990.

[5] P. Eeles, "Capturing Architectural Requirements", *The Rational Edge*, Nov. 2001, www.therationaledge.com/content/nov_01/t_architecturalRequirements_pe.html

[6] Groove Networks, http://www.groove.net

[7] jxta.org, *Project JXTA*, http://www.jxta.org

[8] jxta.org, *Project P2PConference*, http://p2pconference.jxta.org

[9] S. Robertson, and J. Robertson, *Mastering the Requirements Process*, Addison-Wesley, Boston, MA, 1999.

[10] CollabNet, http://www.collab.net

[11] SourceForge.net, http://sourceforge.net

A Communication Protocol for Distributed Process Management

Lerina Aversano, Aniello Cimitile, Andrea De Lucia
RCOST - Research Center On Software Technology
Department of Engineering, University of Sannio
Via Traiano, Palazzo ex-Poste – 82100, Benevento, Italy
(aversano/cimitile/delucia)@unisannio.it

Abstract

Large scale software development processes imply the coordination and cooperation of several sites with a large number of people and sub processes. We present an asynchronous communication protocol for distributed process management adopted within the GENESIS (Generalized ENvironment for procESs management in cooperatIve Software engineering) project. The GENESIS process management sub-subsystem enables distributed process modeling and enactment on different organizational sites through an event dispatching architecture.

1. Introduction

Workflow management is a rapidly growing research and development area of very high practical relevance in business applications and software development [CL97; GHS95; ACF98; WfM99]. However, most existing workflow management systems have a monolithic and centralized architecture and therefore are not adequate to cope with the requirements that large scale software development processes pose. Large scale software development processes imply the coordination and cooperation of several sites with a large number of people and sub processes [MDB00, BSK97, GAH98]. The support for distributed process management is a relevant problem for two reasons.

- distributed software processes may involve a large number of concurrent process instances which impose an adequate coordination support. Therefore, for sake of scalability and availability a software process needs to be distributed across multiple workflow engines running on the different sites involved;
- when a process spans multiple sites that generally work in a largely autonomous manner, it may be required that those parts of a process are under the responsibility of a local project manager that can organize the sub-process in a more appropriate way. Thus, the partitioning and distribution of a process may fall out naturally from the organizational decentralization.

In particular, the latter issue also poses problems concerned with the decentralized and autonomous modeling of distributed software processes. Most work on distributed process management focus on developing paradigms and architectures for the enactment of distributed processes and scarcely address decentralized process modeling [RS99, KM99, HK98, EP99, BSK97, GAH98, BDF96, C98, CDF01]. In most cases process modeling is a centralized activity and enactment of portions of the process is distributed on different workflow engines. In some cases, the central process model is collaboratively edited with the contribution of people on different sites [GAH98] and all sites have visibility of the overall process model. A different approach is used in OzWeb [BSK97] where process models are autonomously defined on the different sites and cooperates through specifically designed interfaces.

In this paper we present the GENESIS (Generalized ENvironment for procESs management in cooperatIve Software engineering) approach to distributed process modeling. GENESIS is an on-going research project aiming at designing and developing a non-invasive and open source system to support software engineering processes in a highly distributed environment [RG02]. The GENESIS process modeling language is hierarchical. The global process is modeled and enacted on the coordinator site (that is the technical leader of distributed software project [KM99]), while sub-processes can be autonomously modeled and executed on different organizational sites. The global process model can be collaboratively edited by the project managers of the different sites. This is particularly important to define the interfaces between the global process model and the coordinated sub-processes.

The paper is organized as follows. Section 2 discusses related work. Section 3 describes the overall architecture of GENESIS and the main features of the process modeling language, while Section 4 presents the asynchronous communication protocol for distributed process modeling

and project management. Finally, Section 5 concludes and comments on future development.

2. Related work

In global and virtual enterprises, software processes consist of multiple sub-processes that may span organizational boundaries. Current commercial workflow technology does not provide the necessary functionality to model, enact, and manage distributed processes due to its mostly centralized server architecture. Modern workflow management systems exploit the web as a mean to enable distributed access to the facilities provided by the workflow engine [ABM97, HHM00, BT98, MDB00]. However, most of these systems are still based on a client-server architecture and the problem of designing architectures for distributed process modeling and enactment of the process is still a research issue [RS99, KM99, HK98, EP99, BSK97, GAH98, BDF96, C98, CDF01, WfM99]. However, most approaches do not discuss issues concerned with distributed process modeling.

PROSYT is an artefact based PSEE [C98]. Each artefact produced during the process is an instance of some artefact type, which describes its internal structure and behaviour. All the routing in this model is based on the artefact and the operations on them. Boolean expressions are used to express the constraints under which operations are allowed to start. PROSYT also allows for distributed enactment facilitated by an event-based middleware [CDF01] (the same middleware is also used by the OPSS WfMS [CDF01]).

In [EP99] the authors propose an approach for the distributed execution that exploits the central role of an event notification service, READY. Workflow participants, both workflow engines and agents, can subscribe to events that trigger the start of workflow activities and processes, and events that describe state changes in the workflow processes they are interested in. Therefore, the configuration of the participants in a workflow can be dynamically changed without requiring any modifications to the existing architecture. Moreover, time-related constructs for addressing the time aspects of process management are provided.

The Endevors project [HK98] proposes an approach to provide a coordination mechanism for distributed process execution and tool integration by using the Hypertext Transfer Protocol (HTTP). The system uses a layered object model to provide for the object-oriented definition and specification of process artifacts, activities, and resources. The intent for distribution is to support a wide range of configurations with varying degrees and kinds of distribution. Stand-alone with a base system configuration without distributed components, Multi-user with a single remote data-store, Multi-user with a single remote data-store are the configuration experimented for distribution.

Kötting and Maurer [KM99] propose an extension of MILOS [KM99] which focuses on the process support for virtual corporation by integrating the process modeling with project planning and enactment in distributed environments. They propose three different approaches for distributed process enactment:

- replicating the workflow engine (process model and data) on several servers and propagating a change to the data on one server to all the others;
- maintaining the process model on the central (coordinator) site and giving the possibility to the other servers to exchange the data they need for local process execution with the coordinator site;
- maintaining different portion of process model and data on different workflow engines which exchange data according to a peer-to-peer architecture.

The authors do not address the problem of decentralized process modeling. Grundy et al. [GAH98] focus on problems concerning the distribution in process modeling. The proposed system provides mechanisms for collaboratively editing process models both in a synchronous and asynchronous way, together with version management support. The architecture is based on a central site maintaining the process model and distributed sites enacting portions of the model.

In the Ozweb environment the peer-to-peer paradigm for distribution is adopted [BSK97]. Here a decentralized system consists of independent sub systems spread among multiple sites. In particular, the authors focus on the process autonomy of each sub system that should be self contained and operationally independent. To this aim they introduce the concept of “treats” to guarantee compliance of the artifacts exchanged between sub-processes.

Our approach mixes both these features: we have the notion of a coordinator site where a global process can be defined in a collaborative way by the project managers of the different cooperating sites of a virtual corporation. Sub-processes executed on different sites are autonomously defines and only have to respect the interfaces defined at the global level. Moreover, the dept of the hierarchical process model is not limited at only two levels, thus allowing the possibility for a partner of the virtual corporation to have further sub-contractors.

3. GENESIS architecture for distributed process management

Traditional workflow management systems do not provide adequate support for the evolution of software organizations towards distributed virtual corporation

models. The main open problem remains the systematic definition of distributed process models and their enactment across multiple sites using appropriate abstractions and mechanisms. GENESIS environment has been developed with the aim to cope these problems. The environment provide a special support for distributed scenario, from the modeling of a distributed process to its enactment.

Distributed projects in GENESIS are organized in a hierarchical way. For example, a two level project will include a project coordinator site, managing the project at the global level and a number of local sites, managing specific project workpackages. The coordinator is in charge of modeling and executing the global process, while the local sites are in charge of modeling and executing the sub-processes concerning their workpackages. Therefore, each GENESIS site includes different components (see Figure 1):

- a workflow management system to model and enact software processes;
- an artifact management system to store and retrieve the artifacts produced within a process;
- a resource management system to allocate resources, in particular human resources, to a project;
- an event engine to collect and dispatch events raised during process management, such as the termination of an activity or the production of an artifact, in particular between the coordinator and local sites;
- a notification and communication system to enable users of the environment to communicate and to send notifications about particular events of interest.

Distributed process modeling is made in an asynchronous way (see next section for the communication protocol). The global process model includes super-activities corresponding to workpackages that are associated to the sites where the actual work is performed. The local site independently creates the process model for the management of the workpackage. The only requirements are that a sub-process has been created when the corresponding super-activity in the global process model has to be enacted and the sub-process interface (in terms of input / output artifacts) is conform to that of the macro-activity (see Figure 2). Besides super-activities, a process model can also contains global activities, i.e. activities that can be collaboratively performed by workers distributed across different sites. The project manager of each site involved in a global activity is in charge of providing the needed human resources for the activity.

In GENESIS the process modeling language is the same both at global level, to model the global software process with the coordination of the composing sub-processes, and at local level, to model the sub-processes at the single GENESIS sites. At both levels, our concern has been to create a language for defining processes in a way general enough to respect the single organization rules, in order to keep low the intrusiveness of the platform. In this respect, each activity (or work item) of the process model is essentially described by the artifacts that will produce, and freedom is left to the worker(s) to decide how to actually perform that activity.

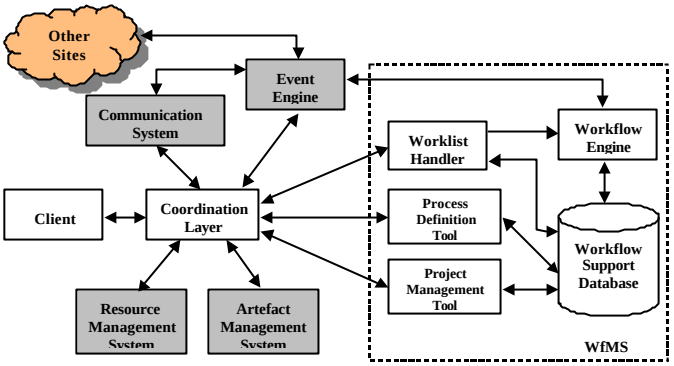


Figure 1 – GENESIS site architecture

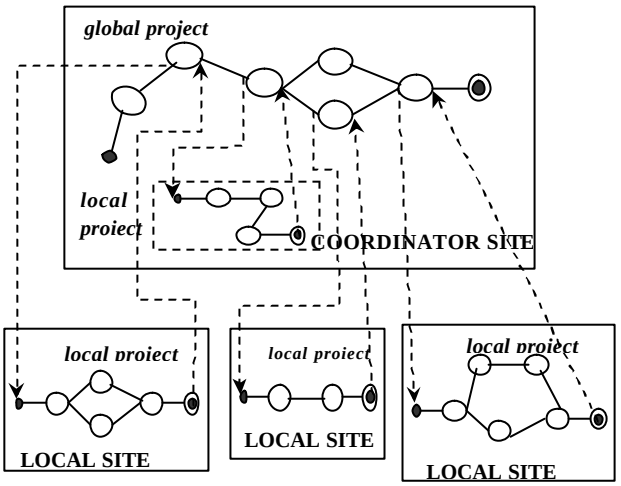


Figure 2 – Hierarchical project decomposition

GENESIS provides two process modeling levels. At the first level a process designer is able to create abstract process models, through the Process Definition Tool, according to specific organization standards. Abstract process models include description of activities (including roles of people performing an activity and types of input and output artifacts) and enactment rules (or transitions) that basically describe control and data (artifact) flow between activities and are expressed according to the Event-Condition-Action paradigm. Abstract process models have to be customized for specific projects by specifying project data, such as actual people performing activities and actual artifacts. Project managers can use them as templates to create concrete process models, through the Project Management Tool, for a specific project that can be enacted by the workflow engine.

Different process instances may be created and enacted according to the same concrete process model.

Further details concerning the process modeling language are described in [AC03].

4. Distributed process modeling

In the GENESIS environment, an asynchronous protocol has been defined for the communication between the global coordinator level and the local coordinated sites during the instantiation of a distributed software project.
The protocol used for the communication is described in the following. We distinguish three main phases: the creation of the project on both the coordinator site and the local sites, where the resource managers associate people to the project and select the project managers; the definition of the global process involving project managers of the different sites; and the definition of the local processes, independently defined by the different local project managers.

4.1 Project creation

At the starting of a new project the resource manager of the global site creates a project using the resource management tool provided by the platform. This means that s/he selects the human resources allocated on the global site and the local sites participating to the project. A "Global Project Creation" event is sent to the involved sites and notified to the global project manager.
The resource management tools that locally receive this event automatically store this information locally. Then, each resource manager of an involved local site decides the allocations of the human resources and the local project manager. The corresponding event is "Local Project Creation", which is sent to resource management tool of the coordinator site, to store this information at the global level. The event is also notified to the global and local project managers.

4.2 Global Process Definition

Once the global project manager has received the notification concerning the "Global Project Creation", s/he can start defining the needed concrete process models for the project , starting from available abstract process models (if a suitable abstract process model is not available, it has to be created first). The global process model includes super-activities to be assigned to local sites and global activities, carried out by groups of people distributed among different sites. Local project managers can collaborate with the global project manager for the definition of the global process, as soon as they are selected by the local resource manager.

Each super-activity has to be assigned by the project manager to a site participating in the project. In this case a "Super Activity Creation" event is sent to the local site together with information concerning the super-activity (start and end date, artifact types, etc.). The project management tool of the local site automatically stores this information and associates them locally to the project. The event is also notified to the project manager of the local site, as soon as s/he will be available.
For each global activity, the GPM can select for each site the number of required people and send this information together with the role associated to the global activity and the work modality (collaborative/single user) to the project manager of the local sites. A "Global Activity Creation" event is sent to each site involved in the global activity.
A concrete global process can start when it is completed, independently of the local process definition status (see next sub-section). Checks are made at enactment time to make sure that all super-activities and global activities have the needed resources allocated.

4.3 Local Process Definition

For each super-activity assigned to a local site, the project manager creates the corresponding concrete local process model (again, starting from an available abstract process model). When the concrete local process is completed, a "Local Process Model Creation" event is sent to the project management tool of the coordinator site which stores this information and associates it to the corresponding super-activity. The event is also notified to the global project manager.
The project manager of a local site involved in a global activity must select the human resources that will participate to the activity. When this is done a "Global User Assigned" event is sent to the project management tool of the global site, to store this information at the global level. The event is also notified to the global project manager.

5. Conclusion

In this paper we have presented the GENESIS approach to distributed process modeling. The definition of the GENESIS platform requirements for distribution, especially for the process modeling facilities, followed a strict interaction with the pilot users (the industrial partners) of the GENESIS project [BC03]. At the starting of the project we collected detailed description of their work modalities on distributed processes and emerging needs. We considered all the problems express by them when organizing and conducting distributed processes and translated them in formal requirements for the

implementation of the workflow management system of the GENESIS platform.

Currently, a single site implementation of GENESIS is already available, while the implementation of the features for distributed modeling and enactment is in progress. Each GENESIS site is realized as a web application: the user interface and the coordination layer are realized using JSP and servlets (Tomcat being the web server), while the other components are developed using the Java 2 Platform Standard Edition. The communication between the coordination layer and the different subsystems composing a GENESIS site is based on Java RMI. The WfMS supporting database is based on MySQL Server.

ACKNOWLEDGMENTS

This work has been partially supported by the European Commission under Contract No. IST-2000-29380, Project GENESIS - http://www.ist-genesis.org).

References

[ABM97]C.K. Ames, S.C. Burleigh, and S.J. Mitchell, "WWWorkflow: World Wide Web based workflow", *Proceedings of the 13th International Conference on System Sciences*, 1997, pp. 397–404.

[ACF98] V. Ambriola, R. Conradi, and A. Fuggetta, "Assessing Process-Centered Software Engineering Environments", *ACM Transaction on Software Engineering and Methodology*, vol. 6, no. 3, 1998, pp. 283-328.

[AC03] L. Aversano, A. Cimitile, A. De Lucia, S. Stefanucci, M. L. Villani, "Workflow Management in the GENESIS Environment", 2nd *Workshop on Cooperative Supports for Distributed Software Engineering Processes*, Benevento, Italy, 2003.

[BC03] D. Ballarini, M. Cadoli, M. Gaeta, T. Mancini, M. Mecella, P. Ritrovato and G. Santucci, "Modeling Real Requirements for Cooperative Software Development: A Case Study", 2nd *Workshop on Cooperative Supports for Distributed Software Engineering Processes*, Benevento, Italy, 2003.

[BDF96] S. Bandinelli, E. Di Nitto, and A. Fuggetta, "Supporting Cooperation in the SPADE-1 Environment", *IEEE Transactions on Software Engineering*, vol. 22, no. 12, 1996, pp. 841-865.

[BSK97] I. Z. Ben-Shaul and G.E. Kaiser, "Federating Process-Centered Environments: the Oz Experience", *International Journal of Automated Software Engineering*, 1997.

[BT98] Gragory Alan Bolcer and Richard N. Taylor, "Advanced workflow management technologies", *Software Process Improvement and Practice*, vol 4 n. 3, pp 125-171, 1998.

[CL97] D. Chan, and K.R.P.H. Leung, "A workflow Vista of the software Process", *IEEE 8th International Workshop on Database and Expert Systems Applications*, 1997.

[C98] G. Cugola, "Tolerating Deviations in Process Support Systems via Flexible Enactment of Process Models", *IEEE Transactions on Software Engineering*, vol. 24 no. 11, 1998, pp. 982-1001,.

[CDF01] G. Cugola, E. Di Nitto, and A. Fuggetta, "The JEDI Event-Based Infrastructure and Its Application to the Development of the OPSS WFMS", *IEEE Transactions on Software Engineering*, vol. 27, no. 9, 2001, pp. 827-850.

[EP99] J. Eder, E. Panagos, "Towards Distributed Workflow Process Management", AT&T Research Labs, 1999.

[GAH98] J. C. Grundy, M. D. Apperley, J. G. Hosking and W. B. Mugridge, "A decentralized architecture for software process modeling and enactment", *In IEEE Internet Computing*, v 2 no 5, 1998, p 53-62.

[GHS95] D. Georgakopoulos, H. Hornick, and A. Sheth, "An Overview of Workflow Management: from Process Modelling to Workflow Automation Infrastructure" *Distributed and Parallel Databases*, vol. 3, no. 2, 1995, p.119-153.

[HHM00] G.Q. Huang, J. Huang, and K.L. Mak, "Agent-based workflow management in collaborative product development on the internet", *International Journal of Computer Aided Design*, vol. 32, no. 2, 2000, pp. 133-144.

[HK98] Hitomi A. S., Kammer P. J., Bolcer G. A., Taylor R. N., "Distributed Workflow using HTTP: Example using Software Pre-requirements", *Proceedings of the International Conference on Software Engineering*, 1998.

[KM99] B. Kötting, F. Maurer, "Approaching Software Support for Virtual Software Corporations", *Proceedings of the International Conference on Software Engineering ICSE '99*, Los Angeles, California, 1999.

[MDB00] F. Maurer, B. Dellen, F. Bendeck, S. Goldmann, H. Holz, B. Kötting, and M.Schaaf, "Merging Project Planning and Web-Enabled Dynamic Workflow for Software Development", *IEEE Internet Computing*, vol. 4 no.3, 2000, pp. 65-74.

[RG02] P. Ritrovato and M. Gaeta, "Generalised Environment for Process Management in Co-operative Software", *Proceedings of the 1st Workshop on Cooperative Supports for Distributed Software Engineering Processes*, Oxford, UK, IEEE Computer Society Press, 2002.

[RS99] F. Ranno and S. K. Shrivastava, "A Review of Distributed Workflow Management Systems", *Proceedings of the international joint conference on Work activities coordination and collaboration*, San Francisco, California, United States, 1999.

[WfM99] Workflow Management Coalition, "Workflow Management Coalition Interface 1: Process Definition Interchange Process Model", Document no. WFMC-TC-1016-P, 1999, accessible from http://www.aiim.org/wfmc/standards/docs/if19910v11.pdf.

Investigation of Repository Replication Models in Globally Distributed Configuration Management

Kazuhiro Fujieda Koichiro Ochimizu
Japan Advanced Institute of Science and Technology
1-1 Asahidai, Tatsuhokuchi, Nomi, Ishikawa, Japan
{fujieda,ochimizu}@jaist.ac.jp

Abstract

The repository replication often takes place in open source software development (OSSD) projects for distributed configuration management. Its style varies with each project. On investigation of these styles, we found tools or systems supporting them should have the following functions: partial replication, both of pull and push model in change propagation, and condensation of successive versions into one version. There is no tool or system satisfying these requirements. This paper shows the investigation, and then proposes a primitive tool to satisfy them. It grafts version trees between repositories. Developers can write scripts with it to automate or easily perform various styles of the repository replication.

1. Introduction

Open source software development (OSSD) projects are typical cases of global software development. Most of all such projects use CVS [1] for their software configuration management. CVS allows developers to access its repository via the Internet. It doesn't provide file locking, so developers can check out the same part of the repository. They work on it separately, and resolve inconsistency in their work afterward. This style of development can decrease the cost of negotiation among developers separated globally. It has supported the rapid evolution of open source software.

CVS, however, provides only a single repository model. It leads two problems. 1) It requires a server connected to the Internet with wide bandwidth and an organization maintaining it. 2) It coerces a single set of management policies of the repository. Such a set includes how to give the write permission to developers, how to commit changes to the repository, how to use branches, and so on. You can see the examples of such policies in [2][3][4]. The former problem can be solved by hosting services typical of SourceForge[1]; the latter problem can decrease concurrency of the work.

Developers or teams in one project sometime need to advance their work to a considerable extent separately with the overall project. Although they need configuration management, they can't use the project's repository unless their work meets the management policy in the project. They often create full or partial replicas from the repository without the negotiation with the project's manager(s) to moderate the policy. They can work on the replicas inconsistently with the original although they may need to incorporate changes in the original sporadically. Afterward they propagate their own changes from their replicas to the original according to the policy. This style of development with replicas can augment concurrency of the development.

The repository replication also takes place because of network bandwidth or server resources. CVS allows read-only access to the repository by anonymous users. The server hosting it can have too heavy load to trouble developer's work. In this case, developers like to separate repositories from the read-only access with the replication. They also like to settle replicas in their local hosts when they have only sporadic or narrow connectivity to the server.

Although the repository replication is important in OSSD projects, CVS doesn't support it at all. Developers create replicas by the supplementary tool named CVSup [5] or by hand. CVSup support only one-way mirror naturally, so it can't fully support the above style of development with replicas. There are two configuration management systems supporting the replication: BitKeeper [6] and ClearCase Multisite [7]. Each of them supports only one style of the repository replication. They can't necessarily support various styles of it in OSSD projects.

In this paper, we first investigate some styles of the repository replication, and then discuss problems in the above two configuration management systems. Finally,

[1] http://sourceforge.net/

we will show our approach supporting various styles of it with a primitive tool grafting version trees between repositories for CVS.

2. Replication Models in OSSD

2.1. Basic Scenario

We present the basic scenario of the replication and the change propagation taking place practically in OSSD projects using CVS. Figure 1 illustrates this scenario.

1. Create a replica.

   A developer creates a replica from the project's repository by hand or CVSup. He/she often creates it from part of files and/or part of versions in the original to save costs of network bandwidth and storage.

2. Create a branch in the replica.

   Branches are traditionally used in version control systems to isolate changes from the main line of development. Although he/she works on the replica, a branch is necessary to easily incorporate changes from the original.

3. Work on the branch.

   He/she adds his/her own changes to the branch in the replica.

4. Pull changes from the original and merge them into the replica.

   He/she sporadically pulls changes from the original and adds them to the trunk in the replica. CVSup can support this operation as long as the version number of the branch doesn't crash any branch in the original. He/she merges them into his or her branch afterward when occasion demands.

   It is hard to reproduce the version history in the replica by hand without CVSup. When he/she doesn't or can't use CVSup, he/she often adds one version including all changes in the history to the trunk. He/she takes such condensation of versions intentionally when he/she doesn't need intermediate changes between specific versions, for example, stable or snapshot releases.

5. Push changes from the replica to the original.

   He/she negotiates with other members of the project about merging his/her changes into the original when occasion demands. If the negotiation succeeded, he/she adds one change condensed from his/her changes on the branch to the trunk in the original. In most cases, there is no need to reproduce the branch in the original because such a branch rarely meets the management policy of the original. Even though he/she wants to do so, it is hard because CVSup can't support this operation at all. There isn't another tool supporting it.

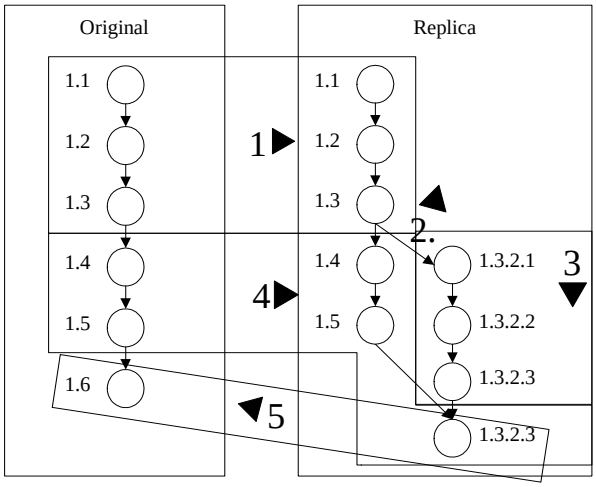


Figure 1. Basic scenario of replication and change propagation

ClearCase MultiSite can't support this scenario at all. It performs automatic change propagation in the push model to keep consistency between the original and replicas. It can't help policy separation, but solve the problem about network resources. BitKeeper and CVS with CVSup can't support this style of replication.

BitKeeper naturally performs this scenario. Any developer must create a replica when he/she intends to make any change in the repository. He/she doesn't need to create a branch on the replica at the step 2. BitKeeper automatically creates a branch. It merges the trunk into it at the step 4. Additionally it makes version trees consistent between the replica and the original at the step 5.

2.2 Advanced Scenario

BitKeeper can work on the above basic scenario. But it always replicates all files in a repository and synchronize all files between a replica and its original. So it can't support the following styles of repository replication in OSSD.

When a development project uses outcome of another ongoing project, it is desirable its repository contains the full or partial replica of another repository. For example,

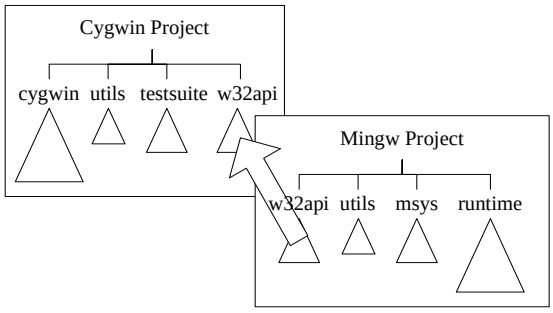


Figure 2. An example of partial replica

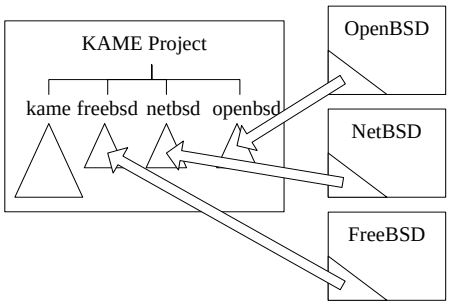


Figure 3. An example of multiple parents

Cygwin project [2] uses a part of outcome by Mingw project[3]. Its repository contains the partial replica of the repository of Mingw project (Figure 2).

When a project depends on outcomes of multiple ongoing projects, its repository can be composed full or partial replica of the repositories of these projects. For example, KAME Project[4] depends on FreeBSD, NetBSD and OpenBSD. Its repository contains partial replicas of the repositories of those projects (Figure 3).

3. Our Approach

On the above investigation, a tool or a configuration management system needs the following functions to cover various style of repository replication.

- Partial replication in files and versions.
- Both of push and pull models to propagate changes in both directions between a replica and its original.
- Condensation of several changes into one change in the change propagation.

We believe it isn't a proper solution to realize a configuration management system supporting all of the above functions. Developers working on replicas need a specific style such as provided by BitKeeper and ClearCase MultiSite to handle them without confusion.

Our approach is rather simple. We provide a supplement tool grafting version trees between repositories for CVS. It obtains whole or part of a version tree of a file from a repository, and condenses it on demand, and then grafts it on a version tree of an existing file or simply stores it in another repository. It records the information of grafting and grafted version trees. At the next time, it can graft growing part of the tree in the source repository onto the corresponding tree on the destination repository based on the record. This operation corresponds to the step 4 in the basic scenario.

It can support not only basic scenario but also advanced scenarios. CVS supports some triggers to invoke other tools on specific timing. The combination of these triggers and our tool can realize the automatic propagation like ClearCase MultiSite. It is, however, only primitive to realize these scenarios. Developers have to write scripts with it to automate or easily perform them.

Conclusion and Future Work

OSSD projects need a tool supporting various styles of repository replication. We proposed a tool grafting version trees between repositories as a primitive to cover these styles. Now we are implementing it in the object-oriented scripting language `Ruby'. We will provide the environment allowing developers to write their own style of replication with it in Ruby.

References

[1] K. Fogel. *Open Source Development with CVS.* CoriolisOpen Press, 1999.

[2] The Apache Software Foundation. Apache project guidelines and voting rules. http://httpd.apache.org/dev/guidelines.html, 2002.

[3] The FreeBSD Documentation Project. Committer guide. http://www.freebsd.org/doc/en_US.ISO8859-1/articles/committers-guide/, Feb 2003.

[4] The Mozilla Organization. Mozilla hacking in a nutshell. http://www.mozilla.org/hacking/, Feb 2003.

[5] J. D. Polstra. CVSup: The CVS-optimized general-purpose network file distribution system. http://www.cvsup.org/, 2002.

[6] BitMover, Inc. BitKeeper - the scalable distributed software configuration management system. www.bitkeeper.com, 2002.

[7] Rational Software Corporation. *Administrator's Guide for Rational ClearCase MultiSite*, Oct 2001.

[2] http://www.cygwin.com/
[3] http://www.migw.com/
[4] http://www.kame.net/

A Software Architecture for Knowledge Acquisition and Retrieval for Global Distributed Teams

Amdreas Braun
Accenture
Maximilianstr. 35
80539 München, Germany
andreas.braun@accenture.com

Allen H. Dutoit and Bernd Brügge
Technische Universität München
Institut für Informatik/ I1, Boltzmannst. 3
85748 Garching b. München, Germany
{dutoit, bruegge}@cs.tum.edu

Abstract

Communication and knowledge building are challenging in distributed contexts: participants do not all know each other and work at different times and locations; the number of participants and their organization change during the project; participants belong to different communities. Hence, to deal with the global market place, it is critical to provide teams with distributed collaboration skills and tool support. To improve the collaboration in global software development (GSD), we propose iBistro [2], an augmented, distributed, and ubiquitous communication space. iBistro aims to overcome problems resulting from miscommunications and information loss in informal or casual meetings. In this paper, we specifically focus on the technical architecture for iBistro, called the distributed concurrent blackboard architecture (DCBA). We developed and tested an experimental prototype of the DCBA between the National University of Singapore and Technische Universität München (TUM),Munich, Germany.

1. Introduction

Distributed projects leverage off tools, such as groupware, distributed repositories, and videoconferencing utilities, to accumulate and distribute knowledge and artifacts. Distributed projects, however, introduce many technical and social barriers. In addition to being geographically distributed, participants come from different corporate cultures, use different tools, follow conflicting standards, and often speak different languages. Such challenges are difficult to meet and often cause the failure of the project.

Our goal in software engineering labs at Technische Universität München (TUM) and at Carnegie Mellon University (CMU) has been to provide a realistic software engineering experience to students. We have done this by immersing students in a single, team-based, system design project to build and deliver a complex software system for a real client. Since Fall 1997, we had the opportunity to teach distributed software engineering project courses in TUM and CMU [4]. Teams of students at CMU and TUM were taught to collaborate using groupware (e.g., web sites, Lotus Notes, Email) and configuration management systems (e.g., CVS) to design and build a system for a client. Client reviews and internal reviews were conducted using videoconferencing facilities, enabling each site to present its progress and obtain feedback from the client and from the other site. While all projects were completed successfully and students acquired a number of skills for dealing with distribution, we experienced many difficulties in the areas of communication and collaboration among the sites. In particular, participants at both sites spent much more effort during solving unexpected problems and interface mismatches than would have been the case in a single site setting. Distribution made the four following obstacles especially difficult:.

Inability to find stakeholders quickly--- Since participants were distributed and did not know each other, finding the author of a piece of code or of a subsystem could take several days. Similarly, finding a project participant who had an area of expertise to help with a specific problem could likewise take several days.

Inability to access knowledge--- Since many decisions taken by teams were taken during meetings or informal conversations, participants at the other site could not easily access the rationale of the system. Hence, participants encountered unexpected problems when enhancing or modifying components produced by the other site. While meetings were documented in meeting minutes that were available via the groupware, such records were organized chronologically and were difficult to search when looking for a specific problem.

Inability to find artifacts quickly--- Even though participants used the same repository for tracking versions of their components, it was difficult to identify when new versions were checked in and which problems new versions addressed. Similarly, a site was usually not aware of whether a new version was under test and about to be released. Consequently, sites worked often on outdated versions and produced version conflicts by solving the same problems twice.

Inability to build ``group memory''--- During the different stages of the lifecycle, many different and altering communication media, such as email, bboards, team web pages, databases, were used. People collaborate by communicating both formally and informally. Much of the information and knowledge, however, only resides only in the people's mind or somewhere unlinked in the databases or applications; the knowledge is lost for the organization or team.

Note that all four problems noted above were caused, at least in part, by some type of communication breakdown. Researchers distinguish between informal and formal communication and recognize their application to different types of issues [8]. Formal communication is typically non-interactive and impersonal and includes, for example, formal specifications, written documentation, structured meetings. Informal communication is typically peer-oriented and interactive and includes, for example, hallway conversations, lunch breaks, and informal conversations that follow formal meetings. While formal communication is useful for coordinating routine work, informal communication is needed in the face of uncertainty and unexpected problems. Note that all three problems noted above were caused, at least in part, by lack of informal communication, typical of distributed projects [6, 8, 1]

In this paper, we describe the technical architecture for iBistro [2]. iBistro is an experimentation environment that allows distributed teams to capture, structure, and retrieve information and knowledge produced in global distributed software development projects. iBistro focuses on the integration of various sources of information, including informal meetings held in specifically equipped rooms. The technical architecture, the *distributed concurrent blackboard architecture* (DCBA) for iBistro has been implemented and evaluated during a distributed project at the National University of Singapore and TUM.

This paper is structured as follows. Section 2 provides an overview of iBistro and details how iBistro can be used to capture, structure, and retrieve knowledge, and more generally, address problems such as finding stakeholders, accessing knowledge, finding artifacts, and building a group memory, as we identified above. Section 3 lists our results achieved so far and concludes this paper by outlining the outlook for iBistro.

2. The iBistro System

We start our overview of iBistro's architecture and design by recapitulating the specific characteristics of both (informal) meetings and software engineering to substantiate our design decision for the architectural pattern chosen.

Software development is a problem-solving activity. In a software project, many different stakeholders contribute to the resolution with their individual knowledge of how to find a (partial) solution of (parts of) the problem. During the process of finding a resolution or partial solutions, stakeholders gather and contribute many different types of information. For instance, one single source file that builds a partial solution for the whole system is built using many different types of contributions, such as programming expertise, application domain knowledge, social skills, and many others. The final version of the artifact (the source file) eventually contains many, but not all, of the contributions made. These contributions, however, are often not used as contributed initially, but in some improved version. Further, the flow of events and contributions made is not predictable. Thus, finding a resolution is an *opportunistic*, as opposed to *systematic*, process.

In other words, the process of building software is reminiscent of the process of building up a wall with little stones ``step-by-step''. It can be seen as *knowledge assembly*, in contrast to *search for solutions*. This view of the problem domain suggests a blackboard-based [17] approach. The blackboard model is originally used in opportunistic problem-solving to deal with non-computable and diverse problems in AI. In this section, we introduce our modified blackboard which we call a *distributed concurrent blackboard architecture*.

The DCBA is an approach to deal with the variety of events and context that occur during (informal) meetings. In our architecture, various *capture components* record contextual events (such as people entering or leaving the meeting room or tracking of electronic events), track discussions and bboard activity, as well as audio, video, and whiteboard content. Several specialized *knowledge sources* seize the captured events to create new types of information or knowledge, for instance a hypothesis (for instance argumentation or intermediate work products; not part of the deliverables) or a solution (e.g., artifacts that are part of the deliverables). This process of knowledge construction creates an abstraction hierarchy of knowledge that is stored in several *layers of abstraction* within the blackboard. Figure 1 displays the simplified overview of the iBistro system.

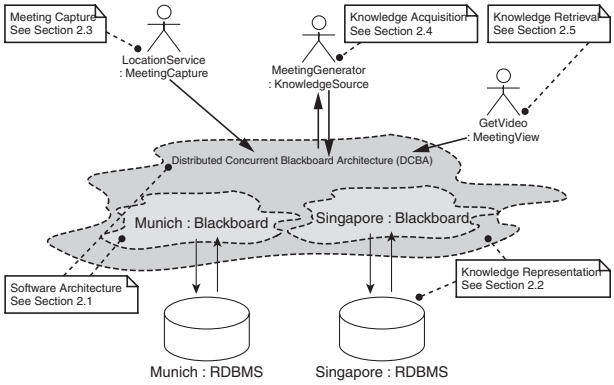


Figure 1: Overview of the iBistro System.

The single distributed concurrent blackboard space is build up from local blackboard installations (e.g., ``Munich" and ``Singapore"). The distributed concurrent blackboard provides a transparent means for knowledge sources and users to access one single portal, regardless of its actual technical implementation.

Figure 1 displays five fundamental concepts for iBistro. In the next sections, the design of iBistro and the usage of these concepts are detailed as described in the following overview.

3. Software Architecture

The *distributed concurrent blackboard* serves as a primary data repository. It is a virtual portal for all users and components which is made from many interconnected and local blackboards. Each local blackboard, however, provides transparent access for local tools to the complete knowledge base. Hence, tools do not need to know where the data are stored. In Figure 1 for example, blackboard systems at Munich and Singapore together form the global DCBA.

Small data objects are completely replicated over time, enabling faster access and concurrent access. Larger data objects, such as video files or other artifacts, are not completely replicated but accessed directly in smaller chunks. This saves blackboard storage space and transfer time. The blackboard is *concurrent* in that several components can access the repository at the same time, either using read or write operations. This also implies that concurrent users or components likely will create several concurrent *versions* of a data item (see next section for more information on version control).

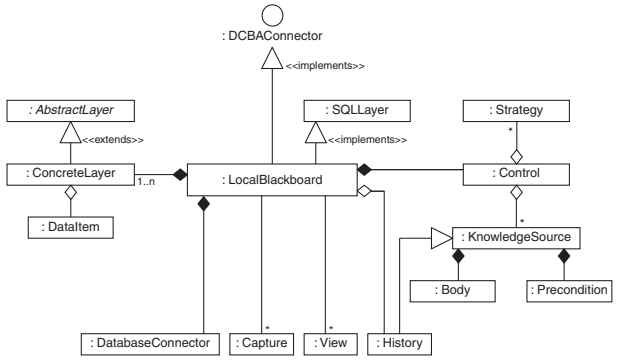


Figure 2: The DCB Architecture in Detail.

Figure 2 shows a class diagram representing one single iBistro system. The following components relate to their corresponding principles in iBistro: The **blackboard** (including the **layers** and **database**) serves as a global data structure for knowledge representation and storage. **Control** and **strategy** components orchestrate the overall knowledge acquisition process by directing the knowledge sources (KS). The **knowledge sources** achieve the knowledge building process on a certain level of abstraction. The **precondition** of a KS implements a rule for the control component to decide whether to execute the **body** (the actual code) of the KS. **Capture** and **view** components access the repository without being directed by the control and simply write data items to the repository or read from out to present the information to the users.

3.1. Knowledge Storage & Representation

The blackboard is a global data structure and serves as a medium for all communication within the architecture. The blackboard can store any kind of information, called *data item*. However, the information stored is categorized into several *levels of abstraction*, which are stored in certain *layers* in the blackboard repository. The blackboard is designed flexible to allow for different representation schemes and organization of layers. For now, we assume that the blackboard contains three types of data items, represented in three layers accordingly:

Layer n:	Solutions and decisions.
Layer *n-1*:	Hypothesis and partial solutions.
Layer *0*:	Context and raw data.

The data items stored in the blackboard layers are incrementally modified and built as the blackboard operates. Data and knowledge stored in the knowledge space is recorded accordingly to a taxonomy of data items, hypothesis, and solutions. The hierarchy used is represented in the layers of the blackboard model. Basic types of information (data items) are stored at lower levels of the blackboard, while higher-level information

(hypothesis, solutions) are stored at higher-levels. Such partitions are necessary to maintain the organization of distributed domain knowledge, which is represented in the collection of knowledge sources. The structure of the levels is also necessary to control the data on the blackboard and to organize the levels of hypotheses.

The layers represent the concept of knowledge acquisition: data is transformed to information, which is transformed to knowledge. The layers also represent the search for solutions, as explored alternatives, regardless of their later use in a solution, are considered as an important contribution and called *hypothesis*. All possible hypothesis and solutions that might be derived from a given set of data items in layer 0 are called the *solution space*. For an iBistro meeting, the solution space would consist of all possible interpretations of a meeting between a certain set of individuals. However, many of these potential solutions are not found by the system. Instead of that, the system might identify a subset of the solution space.

The elements of each layer of the blackboard are composed of elements of the layer below or from the same layer. For instance, layer 0 might contain a bitmap of a whiteboard snapshot depicting the topic and agenda of a meeting. A specific knowledge source, such as an OCR component, then might be used to interpret the bitmap and create a machine-readable object, such as a text-object. The resulting object typically is stored within the same or one level higher layer than the source object (*bottom-up analysis*), or vice-versa (elements at lower layers are created from higher-level objects, *top-down analysis*). In top-down analysis, for instance, a changed line in the source code could be linked back to the related position in the source inspection meeting video.

In addition to the organization in layers and in contrast to existing blackboard systems, all data items also may exist in several versions. Similar to version control in software development, two knowledge sources may create two concurrent successors of a data item. Each of them later can be reconsidered or dropped independently.

3.2. Information and Meeting Capture

Capture components simply capture a particular type of contextual information in a meeting or electronic communication, for instance sensor-based data. This information is then offered to the system, independent of its potential use. Specific capture components can be implemented for various types of context, such as people entering or leaving the meeting room (location-based), people using specific equipment in the electronic meeting room, such as the electronic whiteboard (activity-based), or access to project-relevant artifacts, such as source-code from a workstation, or many others. All capture components have in common that they track information that can be electronically recorded. The resulting information is stored as a *data item* at a low level of abstraction. The video capture component, for instance, simply records audio and video of a meeting and puts the resulting video-stream (as an artifact) to the knowledge space.

3.3. Knowledge Acquisition

Knowledge sources, pick up basic data items and work on them, potentially by using and combining the information captured by several different meeting capture components. A knowledge source is responsible for knowledge acquisition at a certain level of abstraction. Knowledge sources closely conform to the related concept introduced with generic blackboard architectures. Most of the rules introduced there do apply in the DCBA as well: knowledge sources exclusively work with information stored on the DCBA. They are only able to communicate with other knowledge sources using the DCBA. A knowledge source typically works on information from a specific layer and rises the information to the next level higher.

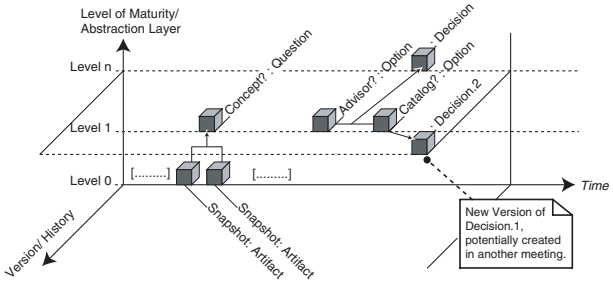


Figure 3: Stored Knowledge as a 3D-model

In meetings in iBistro, the MINUTEGENERATOR is a specific knowledge source to work on the diverse information created during a informal meeting, including the recorded video stream, and to create linkage of knowledge with external data items (e.g., knowledge from other meetings or data items in general). The tool is used exclusively by a dedicated *meeting champion* after the meeting. The MINUTEGENERATOR implicitly creates indices which are used by the user to navigate through the knowledge space. Knowledge is linked accordingly to its logic cohesion and by version. By logic cohesion we understand that, for instance, a question rised by one person is logically linked to that originator; later answers to the question are also interlinked with the question. Alternate versions of the question, for instance rephrased or more precise versions of the questions, are linked as a new version to the initial question. This builds up a complex network of knowledge, dependencies, and versions.

After the post-mortem process, the hierarchy of information and knowledge now stored in the meeting minutes can be translated to a three-dimensional model as shown in Figure 3. The three axes represent the timeline (x-axis), level of abstraction (represented in the blackboard layer, y-axis), and version or knowledge-interlinkage (z-axis).

3.4. Knowledge Retrieval and Navigation

Knowledge views provide access to the contents and structure of the knowledge space. Similar to the model-view-controller paradigm [5], a variety of knowledge views provides different visualizations of the state of the knowledge space. Knowledge views are used in particular to provide a human-computer-interface (HCI) to the information and knowledge stored in the DCBA.

The meeting minutes stored in the repository represent the natural flow of the meeting, including external artifacts, events, or annotations from other sites or an individual's personal computer. A self-evident way to view such a meeting is to playback the meeting as a multimedia archive, thus enabling non-participants to access the raw data. In iBistro, the SMIL MEETINGVIEWER generates on-demand a SMIL[1] [9] file (or data stream) to represent the meeting along with the captured requirements, context, argumentation, and so on. This allows interested people to navigate through a meeting using any SMIL compliant video player, such as RealPlayer™ or Quicktime™ to view the meeting. As the content of the meeting follows a common timeline, the "Clip Position" slider is used to navigate through the captured audio, video, as well as other content such as requirements. Alternatively, the history events can be used to jump to specific segments of the meeting minutes, for example, navigating an option will move the position slider to the frame where the option was first suggested. Graphical views of requirements or rationale can be displayed using HTML or by generating bitmaps on demand.

Displaying the multi-dimensional structure of knowledge, such as context-links between stored entries which allow navigation, is non-trivial. Thus, a specific 3D-meeting view facilitates the *n*-dimensional navigation through the captured knowledge from various sites. As knowledge in iBistro is stored along with its related contextual information, navigation is possible using various types of input. The minutes consist of contextual information (e.g., location, identity, activity, history, and time) which can serve as keys for searching. For example, a minute may be sorted by requirements authored by a certain participant, by time, or any other key. Navigation is possible on any of those keys: the stakeholder of an issue is found by clicking on that issue. Related information, like time or location where the meeting took place, is displayed accordingly and might be used for further navigation. Thus, iBistro's database can be used to find stakeholders over various meetings or even projects. While a MEETINGVIEW provides a meeting-based index into the knowledge base, other knowledge sources can provide an artifact-based view into the knowledge base.

[1] SMIL™ enables simple authoring of multimedia presentations over the Web. A SMIL presentation can be composed of streaming audio, streaming video, images, text or any other media type.

4. Status and Conclusion

The iBistro DCBA has been developed and evaluated in a distributed setting between the National University of Singapore and TUM. This setup revealed some technical difficulties and deficiencies, especially regarding ``live'' audio and video quality due to limited bandwidth and camera orientation problems. This shows the importance of local post-meeting processing and information structuring, as communication then is based on the electronic meeting minutes. The distributed setup also showed the strengths of iBistro compared to simpler electronic communication (such as email).

The DCBA builds a rich *group memory* by integrating artifacts and surrounding information and knowledge (rationale, stakeholders, …) information into a common knowledge space. This allows participants to:

1. *Find stakeholders* by looking at related efforts, discussions, or material.
2. *Access knowledge* by browsing the linked structure of the knowledge space.
3. *Find artifacts quickly* by issue, stakeholder, topics, location, or any other node in the knowledge space.
4. Understand and learn from the *history* of the ongoing project or former projects by seeing rationale entries, which include argumentation, alternatives, and decisions.

In this paper, we motivated the need to integrate various sources of information and knowledge, including informal communication, in GSD. We illustrate how a group memory is build from various knowledge sources. We propose to address some of the issues surrounding informal communication by supporting the efficient capture, structure, and navigation of meeting minutes and their integration into the long term project memory embedded in tools and documents. We described the distributed concurrent blackboard architecture as a connecting technical architecture to achieve these goals. We finally introduce our experimental environment used and conclude by presenting our current status.

10. References

[1] A. Al-Rawas and S. Easterbrook. Communication problems in requirements engineering: A field study. In *Proc. First Westminster Conf. Professional Awareness in Software Engineering*, Univ. Westminster, London, 1996.

[2] A. Braun, B. Bruegge, and A. H. Dutoit. Supporting informal requirements meetings. In *7th International Workshop on Requirements Engineering: Foundation for Software Quality. (REFSQ'2001)*, volume 7, Interlaken, Switzerland, June 2001.

[3] A. Braun, B. Bruegge, A. H. Dutoit, T. Reicher, and G. Klinker. Experimentation in context-aware applications. Submitted to HCI Journal for publication in special issue on context-aware computing, 2000.

[4] B. Bruegge, A. H. Dutoit, R. Kobylinski, and G. Teubner. Transatlantic project courses in a university environment. In *7th Asia-Pacific Software Engineering Conference*, Singapore, Dec. 2000. APSEC.

[5] S. Burbeck. Application programming in Smalltalk-80: How to use Model-View-Controller (MVC), 1987. [6] B. Curtis, H. Krasner, and N. Iscoe. A field study of the software design process for large systems. In *Communications of the ACM*, volume 31(11), Nov. 1988.

[7] L. D. Erman, F. Hayes-Roth, V. R. Lesser, and D. R. Reddy. The hearsay-ii speech-understanding system: Integrating knowledge to resolve uncertainty. In *ACM Computing Surveys*, volume 12 (2), pages 213–253, 1980.

[8] R. Kraut and L. Streeter. Coordination in software development. In *Communications of the ACM*, volume 38(3), Mar. 1995.

[9] W3C. SMIL. Technical report,WorldWideWeb Consortium, 1998.

Supporting an Explicit Organizational Model in Global Software Engineering Projects

Oliver Creighton, Allen H. Dutoit, Bernd Brügge
Technische Universität München
{creighto,dutoit,bruegge}@cs.tum.edu

Abstract

In this paper, we propose the integration of relevant support tools for a global software development project based on a shared organizational model. By providing a single directory service where consistency and accuracy of this model can be better controlled, we intend to achieve several benefits. In particular, the ability to automate some of the tasks associated with initiating a distributed software project, thereby reducing the latency between the setup phase and the development phase.

1. Introduction

A software engineering project requires several different tools at the same time: communication tools, like email clients or web-based bulletin boards, workflow applications for process enactment, and traditional CASE tools such as integrated development environments. Many depend on a model of the project organization. The simplest form has been traditionally to store user names and passwords, a more complicated form, which also supports workflows, includes notions of teams, roles, and resources. The tools have their own scheme to store the models, as a consequence much of the organizational information is duplicated across tools and development sites. This leads to problems of inaccuracy, redundancy, and incompleteness. In the past these were accepted by developers in local environments, as they could be resolved through informal communication.

The problems, however, have become harder in a distributed environment: If the organization is best-effort driven, as e.g. many open source projects are, change occurs frequently and spontaneously, making one person the key contact for an entire system, where most subsystems might still state a previous person as the contact. Updating this information in all associated tools across potentially several development sites quickly becomes too much of an effort for any organization, the frequent solution is therefore to simply not store this information explicitly in the first place. In general, distributed organizations suffer from reduced informal communication, which in turn results in the need to make implicit organizational knowledge explicit [2,5]. Ideally, the information about the organization (e.g., "Who is programmer, tester, or maintainer for what? ") is made explicit for every artifact in every tool, especially in a globally distributed project.

The key issue that we address in this paper is how to minimize redundancy in the organizational knowledge stored across tools in order to minimize inconsistencies. If more knowledge about the organization is made explicit and shared among many tools, the value of this knowledge increases dramatically. Moreover, if all project participants can update the part of this knowledge that is relevant to them, the chance that this knowledge is up-to-date increases accordingly. Hence, we propose a central project directory service supporting the authentication of users, the storage of user-specific attributes (e.g., email address, web home page), team compositions, roles, and access control lists. This project directory can be conveniently accessed by any tool in the project for storing and retrieving organizational knowledge, across sites, tools, and users.

We identify four main properties that such integration efforts should expose: Solutions should be

open: Several entry levels of abstraction (from basic access protocols such as LDAP or HTTP to higher levels of abstraction such as a Java API) should be provided.

encapsulated: All integrated components (the tools and infrastructure extensions) should be exchangable by using standardized interfaces and protocols.

secure: Distribution of personal data should be definable by users, following the informational self-determination principle (making it necessary to use encryption for network communication and possibly storage).

scalable: A single directory service should be distributed across server clusters or even globally. The unity needs only to exist for data integrity, but response times and firewall issues need to be considered. Adding new tools and new sites should be easy, so a single directory hosted on a single server is not an acceptable solution.

Such a solution has several benefits. Take for example the startup phase of new projects in a project-based organization. When a large number of new hires are assigned to a newly initiated project, the project setup also faces social barriers as informal communication requires team-building and a relaxed atmosphere for getting to know each other. We compare such highly condensed project start-ups to a big-bang, where suddenly a large mass of active elements are introduced in a formerly empty environment. The formation of a clear structure on these elements is key to good collaboration. It is common practice to begin projects with some ice-breaking activities that will help participants memorize their names or get a feel for background and interests of each other. But in the case of global software development, this is sometimes impossible to organize, as sending all developers to meet everyone else is too expensive.

As the various support tools need to be installed and configured, projects usually begin with a preparation phase where a small group of project initiators try to generate as much of the anticipated required structure as possible. This task is typically on-going during the entire project duration, as changes in organization frequently also lead to changed requirements for the support infrastructure.

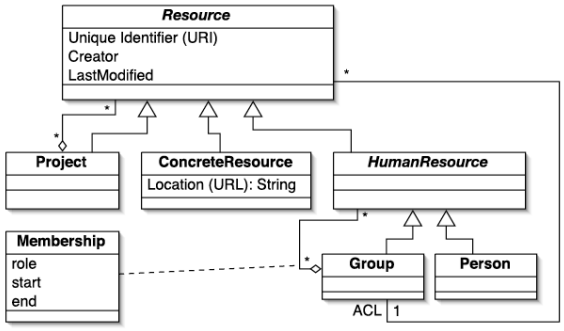
As mentioned, almost all tools that aid in software engineering contain some form of user management for access control, contact information, or representing the roles that various project members have in the project. By unifying some of the required installation steps through tool integration, we expect to shorten the setup phase and begin the development phase of projects earlier.

2. Single Directory Service based on an Organizational Model

The typical use of a centralized directory service in organizations is to provide a phonebook (the "White Pages" of the organization) containing basic contact information for every employee. While this is a useful and important service in itself, when it comes to supporting CASE tools in a broad sense, the standard idea of an alphabetical list of people is not sufficient. Providing information about the properties of each employee, in an organized fashion that makes it straightforward to look for related work areas, would be another use (the "Yellow Pages" of the organization).

But even if a central directory service for these purposes is installed, and is provided by standard access mechanisms such as LDAP, the benefit is not yet reaching the CASE tools directly. To provide integrated user and resource management, it is necessary to first make the organizational model explicit and then to share it across tools and sites for the consistency reasons explained before.

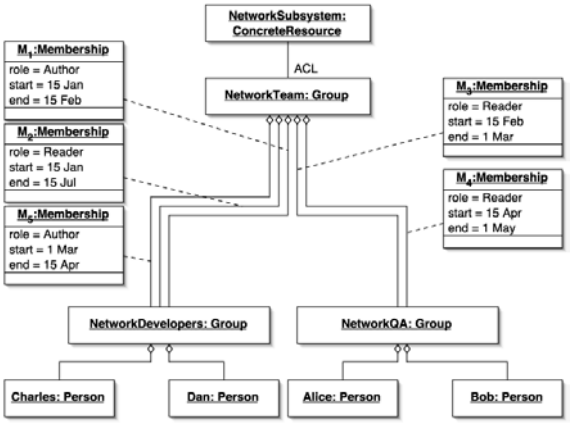
The basis of our directory is the following object model, which we believe can express all organizational models we encounter in the various project courses that we test our tools in (cf. section 3.):



In short, human resources can be grouped in arbitrary depth, and the associations have attributes to support changes in the organizational structure over time. Projects are represented as the sum of all resources they contain, and on the abstract level of *Resource*, the basic functionality for authorization is integrated by associating it with one group as its Access Control List (ACL). The exact access role is represented by the role attribute in the Membership class, which states, for example in the directory itself, roles like "Administrator", "Author", or "Reader". But since this attribute is interpreted by the individual applications, it can be extended with other roles as they require.

With this model, the directory service is predestined to provide a service for authentication, to verify the identity of users, as it then can be assured that if applications want to control access to specific entity objects they control, and they are concrete subclasses of *Resource* in our model, the directory service can provide verification of access (authorization) for users.

Consider the following example, illustrated by an instance diagram of the directory object model:



The NetworkSubsystem, a ConcreteResource controlled and modified by different tools (such as a version control system for the source code and a workflow support tool for quality assurance scheduling) is associated with the NetworkTeam. This Group is interpreted as the ACL for the resource, which causes that the role attribute in the Membership association objects specifies the access rights of the contained *HumanResources*, in our case the two groups NetworkDevelopers and NetworkQA. Our example shows how one would model a source code promotion strategy, where after the code freeze deadline of February 15th, developers can no longer modify the subsystem, as it has been handed over to the quality assurance (QA) group for testing. The QA group's deadline for testing is in our example March 1st, after which developers are again allowed to modify the subsystem, but the QA people have no access until the next cycle, starting April 15th.

For the entire duration of the project, developers are allowed to read source code; note that Membership objects can exist multiple times for the same time span and *HumanResource* combination, only varying in role. The Membership objects between the Groups and Persons that we omitted in the instance diagram could be used for temporarily assigning more people to the testing group, but the role attribute will be insignificant for the NetworkSubsystem as only the NetworkTeam group is interpreted as its ACL. In other words, the containment relation is transitive, whereas the *is-ACL-for* relation is not.

By storing this information in a single directory and providing mechanisms for modifying it, while retaining a consistent access restriction as to who is allowed to, it becomes possible to integrate functionality in the workflow tool for project management to delay the date of code promotion to QA stage, and automatically assure that developers will continue being able to modify the source code without changing anything in the version control system.

3. Evaluation Environment

In researching distributed software engineering, we have taken the approach of "learning-by-doing." We have taught several global software engineering (GlobalSE) project courses in which teams of students located in Pittsburgh, PA and in Munich, Germany collaborated on developing a system for a single industrial client [1,3]. We follow a sawtooth process in which developers present a sequence of incrementally more refined prototypes to the client and to project management, so that the scope and the direction of the project can be refined at regular intervals.

Distributed reviews are done using a combination of video conferencing, phone, application sharing environments, shared slides, and the web. The infrastructure also included asynchronous tools such as Lotus Notes bulletin boards, CVS for version control, a UML modeling tool, and an integrated development environment. More recently, we have also introduced a requirements management tool [4], an awareness infrastructure [6], and a workflow tool for process enactment. As most tools do not share the same user information, developers need a user name and password for each tool. Moreover, while the infrastructure available to students includes the same set of tools in both sites, these tools are usually administered locally, resulting in duplication of user and project organization information, as described in the introduction. Worse, not all knowledge for administrating this information is available in any one site, resulting in inconsistencies and information that is out of date (e.g., users who left the project, teams whose name and purpose changed). To address these issues and to evaluate the value of a single project directory approach, we have been adapting the project infrastructure so that it uses the directory described in the previous section. In particular, the following aspects of the environment are being revised:

User authentication. The most immediate benefit of the single directory is that users have a single account across all tools in the infrastructure. If they change their password, the new password will take into effect immediately. As the directory is shared across sites, users also have a single account across all sites. Similarly, all attributes associated with users, such as email addresses used for notifying developers, are stored only once.

Directory user interface. We implemented the directory in our Lotus Notes infrastructure. As a consequence, users can update their directory entries

as before, by editing their record in the Lotus Notes address book. As every user has write access to their own data, the attributes associated with each users are kept up to date more often.

Team structure. The team membership of each user is represented in the directory as groups, which are also resources associated with each project. As developers change teams and often take part in more than one projects, this enables us to track team membership over time, when assessing a developer's skills. From the infrastructure stand point, this also enables tools to leverage off the team organization. For example, the workflow tool we use allows tasks to be assigned to a team or to an individual. When the task responsibility changes, an email notice is sent both to the participants who are responsible and to those who were responsible for the task. The workflow tool retrieves the list of team members from the directory. As other tools which provide group notifications (e.g., Lotus Notes) also retrieve the same information, the actions of these different tools are consistent. Moreover, this encourages participants to keep this information up-to-date.

Role and access control. The Group and Membership classes in the directory enable us to represent role information. By using a Group as an access control list, each tool can offer a different behavior depending on the role of the user. For example, the workflow tool only allows users with a manager role to change the process model. Users with a coach role can change the responsibility of each task. We are currently modifying other tools to take advantage of this information. For example, in our requirements tool, only analysts would be able to modify the requirements while both analysts and reviewers would be able to annotate the requirements with questions.

We plan to complete these changes and evaluate the single directory concept by our next distributed project course, which will take place during the summer between Munich, Germany and Otago, New Zealand.

4. Conclusion

In this paper we propose a technology-driven approach, as opposed to a business administration-driven approach, for enabling software engineers to build better software through rationale capture and knowledge management.

Key challenges in project organization include identifying incentives for developers to accept the tools the organization wants to employ and addressing individuality and privacy concerns.

When identifying potential incentives, we note that lack of sharing of information among sites leads to adverserial relationships and lack of trust. We anticipate that sites can benefit from the system by offering a greater transparency into their activities. Such transparency can then lead, for example, to certification frameworks for supplier sites and reinforce long term relationships among sites.

When addressing privacy concerns, we propose that providing a unified storage and access control model that is powerful enough to individually set the data distribution scope while at the same time, through unification, being simple enough to allow every individual to determine the scope themselves, we can offer enough flexibility for people to trust a centralized storage of their personal information.

In general, the above issues are difficult to predict and anticipate, as they relate to complex organizational and human processes. Only an experimental approach will enable us to assess the impact of the system with respect of these issues and design solutions to address them.

We intend to continue our approach to first integrate the organizational model across tool boundaries with integrating entity objects that CASE tools work with by identifying common data structures that could then also be shared in a similar fashion. The description of our underlying meta-model encompassing not only the GlobalSE organizational models, but also the system models and rationale models is out of scope for this paper and can be found in Kobylinski et al.[6].

References

[1] B. Bruegge, A. H. Dutoit, R. Kobylinski, and G. Teubner. Transatlantic project courses in a university environment. In *Asian Pacific Software Engineering Conference*, Dec. 2000.

[2] B. Curtis, H. Krasner, and N. Iscoe. A field study of the software design process for large systems. *Communications of the ACM*, 31(11), Nov. 1988.

[3] A. H. Dutoit, J. Johnstone, and B. Bruegge. Knowledge scouts: Reducing communication barriers in a distributed software development project. In *Asian Pacific Software Engineering Conference*, Dec. 2001.

[4] A. H. Dutoit and B. Paech. Rationale-based use case specification. *Requirements Engineering Journal*, 2002.

[5] R. E. Grinter, J. D. Herbsleb, and D. E. Perry. The geography of coordination: Dealing with distance in R&D work. *ACM*, 1999.

[6] R. Kobylinski, O. Creighton, A. H. Dutoit, and B. Bruegge. Building awareness in distributed software enginering: Using issues as context. In *International Workshop on Distributed Software Development, International Conference on Software Engineering*, May 2002.

D-Meeting: an Object-Oriented Framework for Supporting Distributed Modeling of Software

Naoufel Boulila, Allen H. Dutoit, Bernd Brügge
Technische Universität München, Applied software engineering Chair,
{boulila, dutoit, bruegge}@in.tum.de

Abstract

The distributed development of software is increasingly common, driven by the globalization of companies and business and enabled by the improvements in communication and computing.

The distributed development of software introduces new aspects of cooperative work in which a greater emphasis is placed upon technological support of the software development process. Tools to support distributed collaboration are at present limited to general-purpose groupware involving video, audio, chat, shared whiteboards and shared workspaces. However, we are not aware of any group support framework specialized for distributed software engineering.

In this paper we focus on the development, evaluation, and refinement of the D-Meeting framework for supporting synchronous collaboration among distributed groups of developers.

A prototype called D-UML groupware supports distributed software-modeling meetings by enabling real-time sharing and manipulation of information, the capture and management of rationale knowledge, in which different groups have access to different terminals (e.g., live board, desktop machine, handheld, interconnected over fixed or wireless local area networks). D-UML shows a potential enhancement for supporting distributed meetings.

Keywords: Framework, Distributed Collaboration, Global Software Engineering, CSCW, UML, and Rationale

1. Introduction

Software development performed by traditional co-located teams is essentially a difficult task. The complex nature of the activities being carried out requires strong coordination, collaboration, and communication among developers through numerous meetings. Meetings play typically a critical role in collaboration, as it is much easier to build consensus and reach compromises in face-to-face situations. Moreover, large amount of implicit knowledge is exchanged during negotiation and conflicts resolution.

In a distributed context, where developers are dispersed across different sites and even countries, several problems arise due to the physical, social, and cultural barriers [1]. Coordination of the activities is much more difficult, informal communication among team member cannot happen, and a lot of knowledge is lost due to misunderstandings. Consequently, meetings become difficult, rare, and expensive, as participants have to schedule these meetings and travel. Hence, efforts in supporting distributed development should focus on better supporting distributed meetings.

Previous efforts to develop distributed groupware applications that are interoperable across diverse environments, both research prototypes and products, have encountered difficulties and significant cost [2].

This paper is structured as follows. Section 2 describes a general model of the software engineering meeting activities. Section 3 discusses the issues and challenges in distributed collaborative development. Section 4 introduces the D-Meeting framework. Section 5 describes D-UML, an instance implementation of the framework. Section 6 concludes with future research direction.

2. Meeting activities

Software development activities require strong team collaboration and cooperation. Much of this collaboration occurs during meetings, where designers discuss, argue, negotiate, and reach decisions via compromise and consensus. These meetings represent critical points in the project when knowledge is created, conflicts are identified and resolved and social networks are formed. Figure 1 shows the different software development activities.

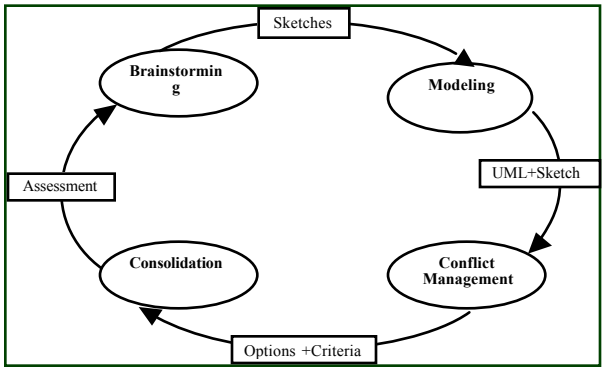


Figure 1: Meeting Activities

2.1. Brainstorming

The early stages of modelling require brainstorming and idea exploration. During this activity, team members explore a wide range of solutions using informal drawings and sketches.

2.2. Modeling

Once team members have explored a sufficient number of ideas, they detail a small number of promising ones. This is usually done with formal artiefacts, e.g. UML diagrams. However, the discussions surrounding the architectural diagrams are often non-linear and sometimes chaotic. Since this is a communication-intensive activity, everyone attempts to talk at the same time. In distributed settings, floor control is the main issue that participants will encounter during these tasks. Moreover, during the modeling activity, team members discuss several options and suggest different design views for each option.

2.3. Conflict identification and resolution

During modeling, participants raise several issues in the form of questions and propose options with different argumentations. Resolution of these issues is done after the evaluation of the pro and cons with respect to criteria. Due to different opinions, conflicts are often raised and need to be addressed. Therefore, structuring issues and the different options and criteria enables team members to quickly identify the source of the conflict and focus on new options to address them. Hence using techniques to capture and maintain rationale can be used to support this negotiation [3].

2.4. Consolidation

At the end of a meeting or after the meeting, team members review their decisions, examine open issues and eventually close them. Finally, this activity ends up with restructuring and documenting the meeting for the following modelling session.

3. Issues in distributed collaborative development

In distributed settings, supporting the collaboration of different individuals and teams over distance and time becomes a increasingly challenging issue.
Many CSCW researchers have investigated the issues associated with collaborative work. Baecker [4] provides several requirements for collaborative writing systems. Nunamaker [5] presents research in developing and using same-time/ same-place and same-time/different-place electronic meeting systems technology; Greenberg [6] describes the issues and experiences in designing and implementing group drawing tools, and Olson [7, 8] presents details of how real groups of expert designers engage in the early stage of software design meetings, Damian [9,10] presents initial studies in investigating groupware support for the interaction in Requirement Engineering processes.
We address the following issues and challenges in building the D-Meeting framework:

Informal/formal communication: knowledge is constructed through group communication and conversation. Communication is a social activity and takes different forms, formal and informal. During brainstorming, a substantial amount of informal knowledge is produced, while during consolidation activity, the generated knowledge is formalized.

Knowledge capture and management: meetings produce a substantial amount of discussion, arguments, and rationale knowledge. The outcome of the meeting, i.e., the knowledge embedded in conversations, should be reusable, organized, and shared within the workgroup to ensure that all participants are working in the same context. In particular, design rationale and its concepts, methods, and techniques will be addressed by the D-Meeting framework.

Awareness: people need to know who else is present at the meeting to guide their work. Peripheral awareness (low-level monitoring of others' activities) is pivotal factor in collaborative work [11]. The tradeoffs inherent in awareness versus privacy and in awareness versus interrupting others will be addressed.

Inflexible floor control policies: in a meeting, there are potential problems if several participants decide to access the same artifact and to manipulate it at the same time, so different systems adopt different floor control policies to determine which participant can take control of an artifact at any time. Most of applied policies are likely to cause the frustration of the users and leads to misuse or to abandon of the system. D-Meeting addresses this issue by providing several control policies.

Heterogeneity: in distributed environments, it is rarely the case that all participants share the same computing environment. A collaborative groupware must be flexible enough to span a variety of devices, from small handhelds to live-board systems.

4. D-Meeting Framework architecture

The key idea behind the D-Meeting framework is that, all the components building a meeting, i.e., Floor Control, Awareness, Design Rationale and the standard meeting activities, are formally modelled as software components that could be used as pluggable strategies (Figure 2).

4.1. D-Meeting framework components

Meeting: is the core engine of the framework, built as a mediator class that defines an interface for communicating with the *MeetingComponent*. Built as black box component that could be subclassed. The reuse of D-Meeting however, is done through composition and delegation more than inheritance. This class implements a cooperative behavior by coordinating the elements of a meeting.

MeetingComponent: represents a generic component that makes up a meeting. It communicates with *the Meeting* object whenever an event of interest occurs. By subclassing and overriding its behavior, new components could be added to the system without changing the behavior of the *Meeting* component.

Awareness: is an essential component in the D-Meeting framework, as it is always required to coordinate group activities. This component cooperates closely with the *Floor Control* component. A default implementation is provided.

Modeling: supports the creation and manipulation of informal structures such as free-hand diagramming and textual annotations to communicate, as well as formal artifacts such as UML models.

Floor control: Default behavior is provided which consist in allowing users to manipulate objects without any kind of locking relying only on socially accepted practices to ensure consistency.

Design Rationale: this component addresses an important issue such as linking the design rationale to the concrete and visible artifacts through embedding communication and history in the design process. This enables developers to collaboratively build a UML diagram and attach additional knowledge to the diagram. Techniques like QOC [12] are intensively used to support knowledge management activity.

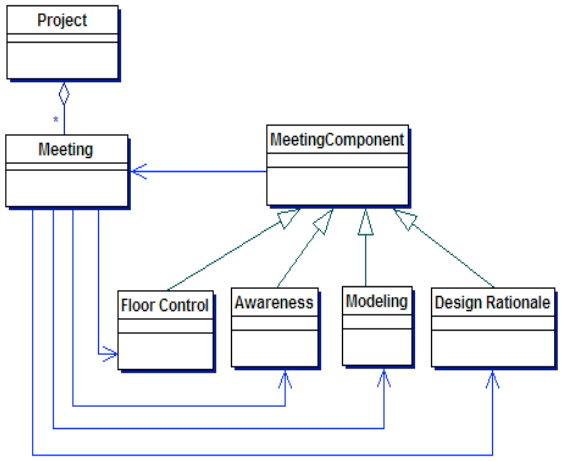


Figure 2: D-Meeting Architecture

In designing D-Meeting, we considered the flexibility of extending it with new components or behavior without breaking its structure. The Meeting class localizes behaviour that otherwise would be distributed among the concrete *MeetingComponent* subclasses. This enforces decoupling of the different components, in that the *Meeting* class as well as *MeetingComponent* class can vary independently.

4.2. D-Meeting Components Collaboration

The D-Meeting framework provides default behavior that can be used for experimentation with. Every component subclassing the *MeetingComponent,* implements a *RemoteObservable* interface and the *Meeting* class implements a *RemoteObserver* interface to support distributed real-time meetings. Whenever a component changes state (after a remote user initiates an action, such as adding an object, marking it, scribbling etc), it sends a notification to the *Meeting* object, which responds by propagating the effect of change to the concerned *MeetingComponent*s(Figure 2,3).

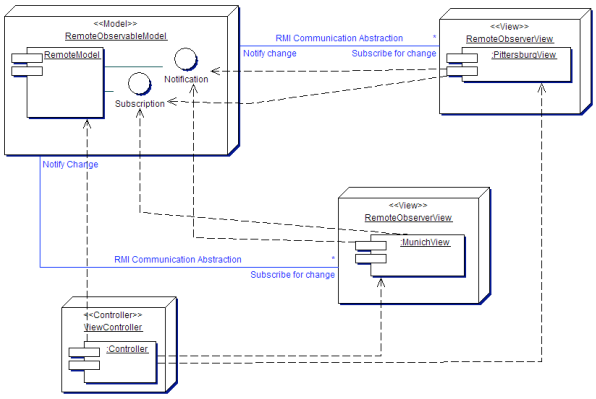


Figure 3: Distributed Architecture

A typical scenario deployment is that small group-work members are physically dispersed in different places and working around a common task to build an architectural design.

5. D-UML: an instance of the D-Meeting framework

D-UML (Distributed UML) is a Java implementation of the D-Meeting framework, based on a Distributed MVC pattern (Figure 3). The Model resides in the *Meeting* object, and can be shared across a number of views composing the D-UML groupware (Figure 4,6):

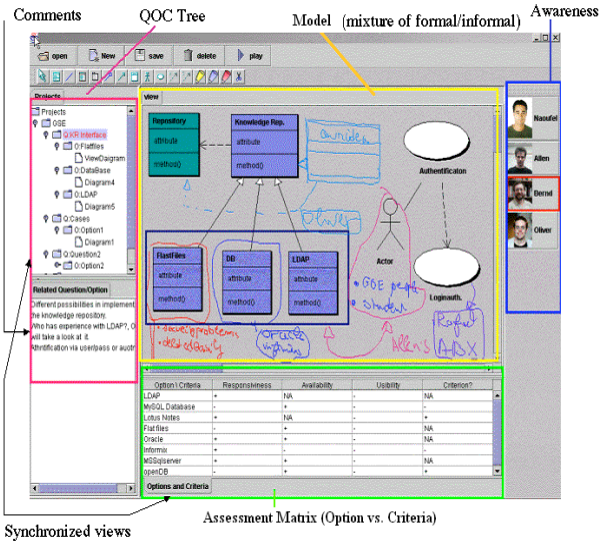


Figure 4: D-UML Groupware

- *UML view*: represents a high-level architecture of a system being modelled which is composed of use cases, class diagrams, objects etc…. The users manipulate the UML view in a similar way as a CASE tool. However the interaction is simpler, enabling the use with a touch screen or a smart board.
- *Scribbling view*: composed of free-hand drawing, handwritten annotations, scribbling and any mean that enables a communication in a team. The scribbling view and the UML view are superimposed, so that developers can use scribbling to draw attention to specific parts of the model.
- *D-UML* is collaboration-aware; several users can simultaneously interact with the application and with each other. Each user that joins a modeling session, his icons is added to the list of the participants and the moment he starts an action his icon is selected with a red square to let the all participants know who is doing what. It was a default implementation of the awareness component.
- *Rationale view*: where we describe in a structured way the Questions that are raised while a brainstorming session, the different Options that could be envisaged and the Criteria that influence a decision. To each artifact being modeled, a related rationale view is created to track its history and the knowledge behind its existence and its relation to other artifacts. D-UML supports conflict resolution activity through an assessments matrix that regroups the different options versus criteria. Criteria are used to selectively identify the acceptance or differentiation of an option. Positive assessment indicates an option satisfies a criterion. A negative assessment indicates an option hurts a criterion (Figure 5).

O/C(examples)	**Criterion1**(platform independent)	**Criterion2**(fast execution)
Option1(C++)	-	+
Option2(Java)	+	-

Figure 5: Matrix option vs. criteria

- *D-UML* provides some history features: replay allows a user that didn't attend the meeting or simply joined it late can replay the session to see what have been done so far, who did what and who were present. The replay functionality is simple and interesting; it shows all the steps the meeting went through till the current status of the brainstorming session. Undo/Redo feature that can be initiated in single as well as in distributed settings.

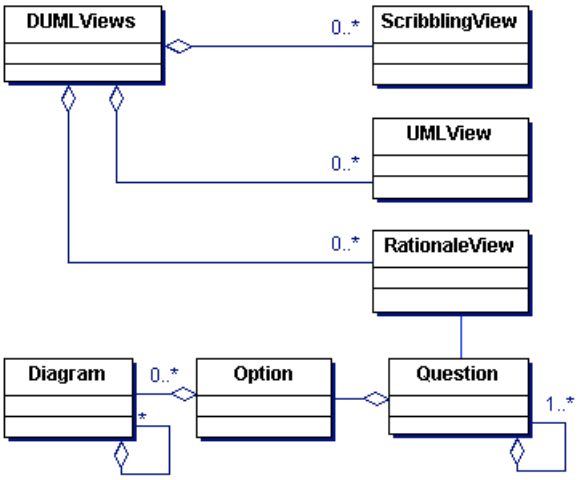


Figure 6: D-UML Views

6. Conclusion and future work

In this paper, we described a generic framework architecture to support synchronous distributed software development meetings. We described D-UML as an instance of this framework. D-UML allows us to investigate the effectiveness and usability of distributed software modelling activity. D-UML addresses the following issues:

- D-UML based on a What You See Is What We Share (WYSIWWS) paradigm in that users don't share windows or applications but rather a model. The advantages are that it provides potentially a lot of flexibility in view sharing and respecting privacy, although we visualize the same view in a given time, scrolling, zooming and moving the view don't involve the rest of the users but rather only the initiator.
- Support for integrated software modeling (UML), rationale capture and management through communication and history embedding.
- Support of replay sessions, where a user, who was not present, can replay the meeting history, as well as Undo/Redo functionalities.
- Support for distributed collaboration over formal and informal artifacts

D-UML has been refined in the context of a small group of researchers. For further usability evaluation, D-UML will be used in an experiment with two sets of groups, one group who uses D-UML and the other who does not, both trying to accomplish the same task to collaborate over a system model from different locations.
In the near future we plan to support the following features:

- Adding an interaction component that enables the use of Augmented Reality technology by augmenting desks with virtual UML objects and video streams to experience the collaboration in another way and to evaluate its usability.
- Automate event trace for post-mortem structuring of design rationale capture.

7. References

[1] R.E. Grinter, J.D. Herbsleb, & D.E. Perry. "The Geography of Coordination: Dealing with Distance in R&D Work". ACM. 1999.

[2] I.Marsic. Data-centric collaboration in heterogeneous environments. Submitted for publication.

[3] A.H. Dutoit, B. Paech, "Rationale management in Software Engineering", In S.K. Chang (ed.) Handbook of Software and Knowledge Engineering, World Scientific Publishing, 2002

[4] R. M. Baecker, Readings in Groupware and Computer Supported Co-operative Work, Assisting Human-Human Collaboration, Morgan Kaufmann Publishers, 1993

[5] J. F. Nunmaker et al., "Electronics Meetings Systems to Support Group Work", Communication of the ACM, 34, 7, 1991.

[6] S. Greenberg et al., "Issues and Experiences Designing and Implementing Two Group Drawing Tools", in Proc. Of the 25th Annual Hawaii International Conference on The System Science, pp. 139-250, Jan 1992

[7] G. M. Olson et al, "Small Group Design Meeting: An Analysis of Collaboration", Human-Computer Interaction, 7, 347-374, 1992

[8] G. M. Olson et al, "Designing Software For A Group's Needs: A Functional Analysis of Synchronous Groupware", in User Interface Software, Edited by Bass and Dewan, John Wiley & Sons Ltd, 1993

[9] Herlea, D. (1997). A groupware system for negotiating software requirements. M.Sc. thesis, Dept of Computer Science, University of Calgary, Alberta, Canada.

[10] D. Damian. An empirical study of requirements engineering in distributed software projects: Is distance negotiation more effective? In Asian Pacific Software Engineering Conference, Dec. 2001.

[11] Dourish, Paul, and Bellotti, Victoria. "Awareness and Coordination in Shared Work Spaces." Proceedings of ACM Conference on Computer-Supported Cooperative Work, Toronto, Canada, November 1992

GNOME, a case of open source global software development

Daniel M. German
Department of Computer Science
University of Victoria
dmgerman@uvic.ca

Abstract

The GNOME Project is an open source project which main goal is to create a GUI desktop for Unix systems, and encompases close to two million lines of code. It is composed by a group of more than 500 different contributors, distributed across the world Some companies employ several of these contributors with the hope of accelerating the development of the project, but many other contributors are volunteers. The project is divided into several dozen modules, ranging from libraries (such as GUI, CORBA, XML, etc) to core applications (such as email client, graphical editor, word processor, spreadsheet, etc). This paper describes the organization and management of the project and describes the infrastructure needed by a contributor, how contributors work as independently together, but still with a common goal. It also describes how requirement gathering takes place, and its unique administration structure, rooted in the GNOME Foundation, a body created solely to oversee the current and future development of the project.

1. Introduction

Bruce Perens describes open source as software that provides the following minimal rights to their users: 1) the right to make copies of the program, and distribute those copies; 2) the right to have access to the software's source code; and 3) the right to make improvements to the program [11]. Some of the classical examples of open source software are the Linux operating system, the Apache Jakarta project, or the GNU toolkit of software development tools (gcc, emacs, make). The GNU Network Object Model Environment (GNOME) Project (www.gnome.org) is an attempt to create a free (as defined by the General Public License, and therefore open source) desktop environment for Unix systems. It is composed of three main components: an easy-to-use GUI environment, a collection of tools, libraries, and components to develop this environment, and an "office suite" [5]. The GNOME Project was founded by Miguel de Icaza in 1996 as a loosely coupled group of developers, scattered all over the world. The project currently involves around five hundred developers. The first version (0.10) was posted in 1997. Version 1.0 was released in March 1999, a point in which it was integrated into Red Hat Linux as its default desktop. Version 2.0 is the latest stable version, released in 2002, and development of Version 2.2 is on its way. GNOME is composed of a large collection of programs and libraries, comprising almost two million lines of code [1, 2].

2. Infrastructure

One of the main requirements for distributed development is agreement on a common toolkit for software development, as each of the members of the team is expected to have access to these tools. In a private development, this is not an issue, as it is expected that the organization will provide the necessary software. Given the philosophy behind OSS and because volunteers are an important driving force in OSS, the toolkit of choice is usually OSS itself (one notable exception to this rule is the use of *bitkeeper* a commercial product for configuration management used by the Linux project, free to be used by Linux developers, with certain restrictions).

GNOME uses, like many OSS projects, the GNU toolkit (gcc, make, autoconf, automake, emacs, vi, etc), CVS for software configuration management, Bugzilla for bug management, GNU Mailman for its mailing lists, and of course, because its goal is to provide a desktop for Linux, it uses Unix as its development platform. Potential developers, by just installing a recent version of Linux, have an environment that allows them to start contributing to the project. The cost of entry is therefore minimized.

It is important to note that contributions to an OSS project are not only restricted to working code (patches, as they are commonly called); they can include documentation, bug reports, artwork, translations to other human languages, and many others. In this paper, we will use the term developer to refer to any type of contributor to the project.

3. Project Management

In order to handle a project of this magnitude, the code base is divided into *modules*. There are four main groups of modules: a) required libraries (19 modules); b) core applications (4 modules); c) applications (16 modules), and d) other (several dozen modules and growing, these modules represent individual applications that are not considered part of the core of GNOME). Each module has one or more maintainers, who oversee the development of their corresponding module and coordinate and integrate the contributions of other developers to their module [3]. In a way, a module represents an independent product that can have its own maintainer, sets of requirements, development timeline. These modules are interrelated between themselves, but there relationships are kept to the minimum, so each module can evolve as independently as possible from the rest.

In an analysis of SourceForge projects, Krishnamurthy found that most OSS projects are composed of a handful of developers [8]. GNOME is one of the few projects that appears to break this rule, given that more than 500 people have "write access" to its CVS repository. Zawinsky, a Mozilla developer, provides insight to this phenomenon: "If you have a project that has five people who write 80% of the code, and a hundred people who have contributed bug fixes or a few hundred lines of code here and there, is that a '105-programmer project?" (as cited in [7]).

A preliminary analysis of the development logs appears to agree with Zawinsky's view. Most modules have few developers who write most of the code. For example, Evolution (the mail client of the GNOME project) is composed of approximately 160kLOCS (Feb. 2003) and almost 50% of the times the source code has been modified, the change can be attributed to one of 5 developers (see figure 1) [4].

The success of the GNOME project seems to lie in the division of the project into manageable modules in which a handful of developers (a team) can concentrate. The amount of communication between developers is therefore minimized.

As some modules grow into large entities themselves, they are then broken into smaller modules. For example, Evolution is broken into slightly more than 20 submodules, each as independent as possible from the rest. In includes modules dedicated to user interface, different mail libraries, filters, importers, documentation, translations, etc.

4. Requirements

As described in [12], most OSS projects do not have a traditional requirement's engineering phase. Specially at the beginning of GNOME, the only stakeholders were the developers who acted as users, investors, coders, testers,

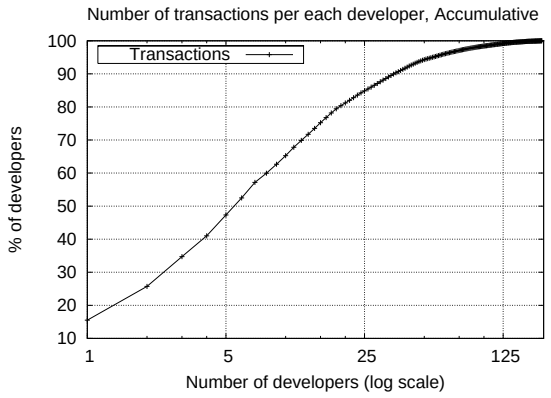


Figure 1. This plot shows the number of CVS transactions committed to Evolution per developer. From a total of 196 developers, 5 account for 47% of the CVS transactions, while 20 account for 81% of the transactions, and 55 have done 95% of them.

documenters, etc., with little interest in the commercial success of the project, but who, at the same time, wanted to achieve respect from their peers for their development abilities. One of the main goals of the developers is to produce software that is used by its associated community.

In particular, we can identify the following sources of requirements in GNOME:

- Vision. One or several leaders provide a list of requirements that the system should satisfy. In GNOME this is epitomized by the following non-functional requirement: "GNOME should be completely free software" (free as defined by the Free Software Foundation).
- Reference Applications. Many of its components are created with the goal of replacing similar applications. The GNOME components should have most if not the same functionality as these reference applications. For example, gnumeric uses Microsoft Excel as its reference, ggv uses gv and the kghostview, Evolution uses Microsoft Outlook and Lotus Notes.
- Asserted Requirements. In few cases, the requirements for a module or component are born from a discussion in a mailing list. In some cases, a requirement emerges from a discussion whose original intention was not to do requirement analysis. In other instances (as it is the case of Evolution), a person posts a clear question instigating discussion on the potential requirements that a tool should have. Evolution was born when several hundred messages were created describing the requirements (functional and non-functional) that a *good* mailer should had before coding started.

- A prototype. Many projects start with an artifact as a way to clearly state some of the requirements needed in the final application. Frequently a developer proposes a feature, implements it, and presents it to the rest, who then decides on its value and chooses to accept the prototype or scrap the idea [6]. GNOME started with a prototype (version 0.1) created by Miguel de Icaza as the starting point of the project.

- *Post-hoc* requirements. In this case, a feature in the final project is added to a module because a developer wants that feature and he or she is willing to do most of the work, from requirements to implementation and testing. This feature might be unknown by the rest of the development team until the author provides them with a patch, and a request to add the feature to the module.

Regardless of the method used, requirements are usually gathered and prioritize by the leader of the project, the maintainer or maintainers of the module and potentially the Foundation (see section 6). A maintainer has the power to decide which requirements are to be implemented and in which order. The rest of the developers could provide input and apply pressure on the maintainers to shape their decisions (as in *post-hoc* requirements). Sometimes a subset of the developers group might not agree with the maintainer's view, and could potentially jeopardize the project, and create what is known as a fork (see section 5). So far this has not happened within GNOME.

5. Module Maintainers

Module maintainers serve the roles of leaders for their module. [9] identified the main roles of a leader in an OSS project as:

- to provide a vision;

- to divide the project into parts in which individuals can tackle independent tasks;

- to attract developers to the project;

- to keep the project together and prevent forking.

The success of an OSS project is dependent on the ability of its maintainer to divide it into small parts in which developers can work with minimal communication between each other and, with minimal impact to the work of others [9]. GNOME has been able to attract and maintain good, trustworthy maintainers in its most important modules due mainly by being employees paid by different companies to work on GNOME.

5.1. The Paid Employees

As we described in [3], several companies have been subsidizing the development of GNOME. RedHat, Sun Microsystems, and Ximian are some of the companies who pay full time employees to work on GNOME. Paid employees are usually responsible for the following tasks: project design and coordination, testing, documentation, and bug fixing. These tasks are usually less attractive to volunteers. By taking care of them, the paid employees make sure that the development of GNOME continues at a steady pace. Some paid employees also take responsibility (as maintainers) for some of the critical parts of the project, such as gtk+ and CORBA (RedHat), the file manager Nautilus (Eazel, now bankrupt), Evolution (Ximian), etc. Paid employees contribute not only in the form of code. One of the most visible contributions of Sun employees is the proposal of the GNOME Accessibility Framework, which aims at guaranteeing that GNOME can be used by a vast variety of users, including persons with disabilities.

In the case of Evolution, the top 10 contributors (which account for almost 70% of the CVS commits) are all Ximian employees.

Volunteers still play a very important role in the project and their contributions are everywhere: as maintainers and contributors to modules, as bug hunters, as documenters, as beta testers, etc. In particular, there is one area of GNOME that remains done mainly by volunteers: internationalization. The translation of GNOME is done by small teams of volunteers (volunteers who usually speak the language in question and who are interested in seeing support for their language in GNOME).

As with any other open source project, GNOME is a meritocracy, where people are valued by the quality (and quantity) of their contributions. We are currently evaluating the commit logs to analyze the amount and type of contributions from these paid employees to the project and how they compare to the contributions of volunteers. Most of the paid developers in GNOME were at some point volunteers. Their commitment to the project got them a job to continue to do what they did before as a hobby.

6. The GNOME Foundation

Until 2000, GNOME was a run by a "legislature" where each of its developers had a voice and a vote and the developer's mailing list was the "floor" where the issues were discussed. Miguel de Icaza served as the "constitutional monarch" and "supreme court" of the project, and had the final say on any unsolvable disputes. This model did not scale well, and was complicated when Miguel de Icaza created Helixcode (now Ximian), a commercial venture aimed

at continuing the development of GNOME, planning to generate income by selling services around it.

In August 2000 the GNOME Foundation was instituted. The mandate of the Foundation is "to further the goal of the GNOME Project: to create a computing platform for use by the general public that is completely free software." [13]. The Foundation fulfills the following four roles [10]: 1) it provides a democratic process in which the entire GNOME development community can have a voice; 2) it is the responsible for communicating information about GNOME to the media and corporations; 3) it will guarantee that the decisions on the future of GNOME are done in an open and transparent way; 4) it is a legal entity that can accept donations and make purchases to benefit GNOME.

The Foundation is composed of four entities: its members (any contributor to the project can apply for membership); the board of directors (composed of 11 democratically elected contributors, with at most four with the same corporate affiliation), the advisory board (composed of companies and non-for-profit organizations), and the executive director. As defined by the Foundation's charter, the board of directors is the primary decision-making body of the GNOME Foundation. The members of the board are supposed to serve in a personal capacity and not as representatives of their employers. The Board meets regularly (usually every two weeks, via telephone call) to discuss the current issues and take decisions in behalf of the entire community. The minutes of each meeting are then published in the main GNOME mailing list.

6.1. Committees

Given the lack of a single organization driving the development according to its business goals, OSS projects tend to rely in volunteering to do most of the administrative tasks associated with that project. In GNOME, committees are created around tasks that the Foundation identifies as important. Developers then volunteer to be members of these committees.

Examples of committees are: "GUADEC" (responsible for the organization of the conference),, "Web team" (responsible for keeping the Web site up-to-date), "sysadmin" (responsible of system administration of the GNOME machines, the "release team" (responsible for planning and releasing the official GNOME releases), the "Foundation Membership" (responsible for maintaining the membership list of the foundation), and several others.

6.2. The Release Team

Each individual module has its own development timeline, and objectives. Planning and coordination of the overall project is done by the Release Team. They are responsible for developing, in coordination with the module mantainers, release schedules for the different modules, and the schedule of the overall project. They also keep track of the development of the project and its modules, making sure that everything stays within schedule. Jeff Waugh, a GNOME Foundation member, summarized the accomplishment of the team and the skills required (in his message of candidacy to the Board of Directors in 2002):

> *[The Release Team] has earned the trust of the GNOME developer community, madly handwaved the GNOME 2.0 project back on track, and brought strong co-operation and "the love" back to the project after a short hiatus. It has required an interesting combination of skills, from cheerleading and Maciej-style police brutality to subtle diplomacy and 'networking'.*

7. Communication

Developers are located in many different places all around the world. As described above, some are volunteers and the rest works for different organizations (and even within those organizations, they might still be located in different parts of the world, as it is the case with some Ximian developers). The GNOME communication relies on the following:

- Mailing Lists. The GNOME project has extensively used mailing lists. These lists have a wide range of purposes: some are intended for final users, some for particular components of the software, some for announcements, etc. Mailing lists provide a trail of decision making for the project.

- IRC: Internet Relay Chat. The "watercooler conversations" have been mimicked by using irc. Developers connect to a common IRC server/channel, and wait for other users to connect. The conversation is informal, with no real agenda.

- Web sites. The Web sites of the project comprise a large amount of information, intended for every type contributor to the project (coders, bug reporters, bug hunters, documenters, translators, etc).

- GUADEC, the GNOME Conference is a Foundation's effort to get developers together. Its goal is to provide a venue for discussion, interaction, and training. The Foundation attempts to support many of the developers who cannot afford their own traveling expenses. This year's GUADEC will last 5 days and will take place in Dublin, Ireland.

- The GNOME Summaries. Every two weeks a summary is published in the GNOME mailing list. It usually contains the most relevant events of the period, links to new or improved documentation, news specific to different GNOME modules, a "Hacker Activity" section, enumerating the most active modules and the most active developers in the project, and a bug-hunting section, listing the number of current bugs per module including the progress made during the period.

8. Conclusions

After almost 6 years of development, GNOME has demonstrated to be a success. One of the major accomplishments of the project was the decision of Sun to replace its outdated CDE with GNOME. Its "free software" nature has created special requirements in the way that the project is organized and managed. Furthermore, GNOME is a project where people employed by different companies and volunteers work together with a common goal. Developers contribute in a wide range of ways (code, testing, bug reports, documentation, artwork, bug-hunting, system administration) and are located across the world, relying on the ability of its leaders and maintainers to manage the project, on the Internet as its communication channel and on several tools (such as mailing list, Web pages, CVS, Bugzilla) to maintain a good-enough communication that allows for the project to proceed. Recently, the Foundation has taken the responsibility of giving the project a coherent vision for the present and into the future, aimed at guaranteeing that GNOME continues to fulfill its main goal: "to create a computing platform for use by the general public that is completely free software."

About the author

Daniel M. German is a member of the GNOME Foundation and was the maintainer of ggv, the PostScript viewer for the GNOME project. He is currently assistant professor of computer science at the University of Victoria, in Canada. In his spare time he enjoys hacking free software.

References

[1] J. Charles. Linux Support Ranges from GUI to Big Blue. *Computer*, 32(5):20–22, May 1999.

[2] M. de Icaza. GNOME History. http://primates.helixcode.com/~miguel/gnome-history.html, 2002.

[3] D. M. German. The evolution of the GNOME Project. In *Proceedings of the 2nd Workshop on Open Source Software Engineering*, 2002.

[4] D. M. German and A. Mockus. Automating the Measurement of Open Source Projects. In *Proceedings of the 3rd Workshop on Open Source Software Engineering*, May 2003. To appear.

[5] T. Gwynne. GNOME FAQ. http://www.gnome.org/faqs/users-faq/, 2003.

[6] S. Hissam, C. B. Weinstock, D. Plakosh, and J. Asundi. Perspectives on open-source software. Technical Report CMU/SEI-2001-TR-019, Software Engineering Institute - Carnegie Mellon, November 2001.

[7] P. Jones. Brooks' law and open source: The more the merrier? does the open source development method defy the adage about cooks in the kitchen? IBM developerWorks, August 20, 2002.

[8] S. Krishnamurthy. Cave or Community? An Empirical Examination of 100 Mature Open Source Projects. *First Monday*, 7(6), June 2002.

[9] J. Lerner and J. Triole. The Simple Economics of Open Source. Working Paper 7600, National Bureau of Economic Research, March 2000.

[10] D. Mueth and H. Pennington. GNOME Foundation FAQ. http://foundation.gnome.org/faq.html, 2002.

[11] B. Perens. *Open Sources: Voices from the Open Source Revolution*, chapter The Open Source Definition. O'Reilly, 1999.

[12] W. Scacchi. Understanding the requirements for developing open source software systems. *IEE Software*, 149(1):24–39, February 2002.

[13] The GNOME Foundation. GNOME Foundation Charter Draft 0.61. http://foundation.gnome.org/charter.html, October 2000.

Experiences in Distributed Development: A Case Study

Lori Kiel, P.Eng.
Interdisciplinary M.A. Candidate, Departments of Anthropology and Computing Science
University of Alberta, Edmonton, Canada
lkiel@ualberta.ca

Abstract

The findings from a case study involving a mid-sized software development organization illustrate the complex interaction of factors common to many global software development projects. The focus of the study is a single product development group that was distributed between two international divisions of the company, one in Canada and one in Germany, for a twenty-month period. When the distribution ultimately failed, it was a web of social, cultural, linguistic and political factors, rather than use or misuse of specific tools or techniques, that emerged as being most significant in the project's ultimate demise. A summary of these factors is presented here.

1. Introduction

Despite a recent decline in the international software industry, there is every reason to believe that there will be continuing pressures towards the adoption of globalized approaches to software creation. These approaches may take the form of formalized outsourcing agreements; they may emerge as collaboration among various divisions of international organizations; or they may consist of a small group of individual programmers who work together but live in separate cities.

Consequently, managers are frequently asked to organize software development projects that draw upon a mix of personnel in multiple locations. The technical barriers to such practices are diminishing rapidly. What about the human factors? Given the newness of the phenomenon, internationally distributed software development projects, whether they work, and why they either succeed or fail are little understood.

The study described here was undertaken in order to contribute to the small but growing body of empirical knowledge of global software development.

This research was supported in part by the Natural Sciences and Engineering Research Council of Canada, the Social Sciences and Humanities Research Council of Canada, the Alberta Software Engineering Research Consortium and the Departments of Anthropology and Computing Science at the University of Alberta.

2. The Company

The subject company is a medium sized software development firm with offices in Canada, the United States, Germany and Malaysia. While some of the international offices were originally created within the corporate umbrella, others were acquired within the last two or three years. Consequently, the company has a diverse and dispersed set of resources, and managing that diversity represents a significant challenge.

Given this international structure, it is not surprising that the organization has undertaken distributed projects in one form or another for much of its history. Many of these projects have been short term, or structured on an *ad hoc* basis according to shifting delivery and development pressures. One project, however, represented a departure from this pattern.

For a period of roughly twenty months, spanning October 1999 to May 2001, the company structured all development of one of its core projects according to a fully distributed model. Before that time, they used multiple teams in their German and Canadian offices creating customized versions of this core application for each client. Teams in Canada developed for North American clients, while teams in Germany handled clients in Europe. In order to reduce duplication of efforts and facilitate a comprehensive code re-use strategy, they pooled their expertise in the two offices. A single distributed team was created; it was given the task of producing a new version of the product that would be the base of all future developments.

Ultimately, the distribution failed, and the company consolidated all development for the base product in a single office. While a base version was successfully created, the significant overhead required to support the distribution was deemed to be too expensive.

The purpose of my study was to answer the question, how did the distribution of software development for this product fail?

3. Method

In order to answer this question, I completed a qualitative case study in which interviews constituted the primary data collection technique. Senior managers in the company developed a comprehensive list of all project

participants from which seventeen individuals were selected for participation in the study. These seventeen participants represented a cross-section of the development process, including managers, business analysts, trainers, team leads and developers. I interviewed all seventeen individuals, recorded those interviews where possible, and used the transcripts and interview notes as the basis of analysis.

The strength of qualitative research is its ability to engage with multiple perspectives as experienced by participants in a particular set of circumstances, and to isolate recurring patterns or characteristics. To that end, I created codes that reflected the patterns that emerged from a preliminary review of the data. These codes were further refined upon a second review of the data.

In total, I applied twenty-three codes to the data. They fit into three main categories that form my analytical framework. The first consists of the main *themes* of distributed development that have emerged from the data. These themes represent the abstract or broad conceptual challenges to distribution that participants described.

The second category consists of the *practices* or specific ways of doing things on the ground, on a day-to-day basis within the company as described by the participants. It is at the level of practices where the themes are realized and reinforced.

Finally, a third category contains general observations made by the participants about the process of distributed development. This category contains the participants' own reflections on the overall practice of distribution, not necessarily their own experiences with a particular distributed project or projects.

4. Themes

I identified five main themes in the interview data: time, language, power, culture and trust. These themes define the broad, conceptual challenges to distributed software development that emerged in this case study. Taken individually, none of them is particularly surprising. The cumulative impact is, however, remarkable.

4.1. Time

Overwhelmingly, participants reported that the eight-hour separation in time zones between the two offices presented a substantial challenge to the project. When people were arriving at the office in Germany (8am), people in Canada (midnight) had long since left for home. The workday was just starting for the Canadian developers as things wound down in Germany. Depending on the schedules of particular individuals, it was possible for there to be little or no overlap in the workday.

The most often reported consequence of this temporal separation was a reliance on asynchronous communication techniques, primarily e-mail. Problems that should have been simple to resolve often dragged on for days. What might have been settled by a quick conversation was often blown out of proportion because the information needed in order to resolve the situation had to be communicated through e-mail, a message that might not be read for as much as 16 hours. Furthermore, the conversion of ideas and arguments into e-mail form introduced great opportunity for misunderstanding, particularly problematic when the content of the communication was contentious or argumentative.

4.2. Language

The main business language of the company is English. Not surprisingly, language emerged as a point of tension for almost all of the German participants, as well as for those participants from the Canadian office who had spent considerable time in Germany. The most common observation was that it was very difficult to fully participate in a teleconference in English. Often such meetings were oriented to some sort of problem solving or dispute resolution and, as such, could be dynamic and highly charged. Voices were raised, and people spoke rapidly. German participants reported frustration at not being able to follow or participate in the discussion. Canadians often interpreted the silence coming from the other office as an indication that no one in Germany wanted to participate or add to the discussion, and carried forward with the meeting. Many of the Canadian participants described being in meetings in Germany where they could see that the majority of the people participating in the teleconference were having great difficulty following the discussion.

In addition, many of the German participants reported a reluctance to engage in argument over the telephone. When technical or methodological debates arose – a necessary component of any software development activity – German speaking participants reported that they preferred to have the time to formulate their position, write it down, check it, ensure that they were saying what they meant to say and, finally, send it off in an e-mail. While this addressed their discomfort, it introduced the potential for misunderstanding and stretched out the problem-solving exercise over an extended serious of asynchronously exchanged e-mails.

4.3. Culture

Many participants – from both offices - suggested that people in the two offices could be characterized as exhibiting a particular set of attributes they identified with the term *culture*. They indicated that people perceive things differently, people say things differently, and people make things differently. Participants identified cultural differences as having posed a challenge to the project.

For example, the Germans were described as being blunt, efficient, stubborn, and to the point, but reluctant

to speak out and criticize openly. On the contrary, the Canadians were described as being laid back, chatty, comfortable with open criticism, lax and indecisive. Several participants reported that the Canadians often interpreted the Germans as being rude, and that the Germans were often frustrated by the Canadian way of doing things. Most individuals acknowledged the differences in national culture, and were aware of how their own behavior might be interpreted by people in the other office. Despite this awareness, frustration with the behavior of the other group persisted.

4.4. Power

One issue that was not necessarily explicitly stated, but emerged in some form in many conversations was the issue of power. Many participants reported that, particularly in the early days of the distribution of the project, decisions were made in the Canadian office and flowed to the German office. According to one participant, the Canadian office has historically seen itself as the 'brains of the operation'. Managers were typically located in the Canadian office. On many occasions, managers from the Canadian office temporarily relocated to Germany, but there were no managers from within the ranks of the German office.

The flow from Canada to Germany was not restricted to management personnel and management decisions: technical standards and architectural decisions usually originated in the Canadian office and the developers in Germany were expected to adopt these standards and accept the decisions. Many participants reported a strong resistance on the part of the German developers. These developers often refused to follow standards or use tools developed in other offices. One senior manager reported with frustration on the outright intransigence on the part of some in the German office. Another offered that not following standards was almost a point of pride, and suggested that managers failed to recognize the importance of local ownership of standards. The history of the company is characterized by a marked imbalance of power between the Canadian and German offices.

4.5. Trust

Temporal separation, language gaps, cultural differences and inequality between the two offices all contributed to the challenge of building an atmosphere of trust, respect and cooperation that characterizes a cohesive software development team. Common to participants from both offices were stories of misunderstandings, angry exchanges, and complete dismissal of people in the other office as incompetent. Indeed, such tensions seem to extend well beyond the specific boundaries of the project of concern for my case study. Many participants reported general problems in dealing with the other office, regardless of the particular project. One participant described the friction between the two offices.

Several participants suggested that when you are physically separated from co-workers, it is easy to ignore them and devalue their contributions and abilities. At great distance it is difficult to empathize with those in the other office, and this in turn makes it challenging to maintain an atmosphere of mutual respect and shared understanding. Several people commented that it often seemed that simple choices – for example, the timing of a meeting - completely failed to consider the impact on those in other offices.

Participants generally acknowledged that the people in the other office were not, in fact, incompetent: everyone recognized the abilities of their co-workers. Nonetheless, several people noted that when denied access to the context in which a decision was made and the detailed reasoning that entered into the decision, it became very easy to dismiss an apparently poor choice as being the work of an idiot. This was particularly the case in a crunch situation when everyone was under pressure to meet a deadline.

5. Practices

Along with the five themes identified above, I have isolated several practices that articulated with the themes, dramatically impacting the experience of distributed development for the project participants. Many of these practices involve specific tools such as e-mail, telephone, teleconference, StarTeam (configuration management tool), net-meeting, translation tools, and the company intranet. Others refer more to business processes or ways of doing things, like software process, project team size, project scope, management practices and travel for face-to-face meetings. There is a complex interaction among the various themes and practices that must be explored if anything is to be learned about this particular distributed software development exercise.

For example, while the practice of using e-mail has several basic characteristics that exist in any circumstance, certain properties become critical when it is employed in a predominantly asynchronous and politically charged communication environment. Other qualities emerge when messages are exchanged between people from different cultural or linguistic backgrounds. And still other factors are important when messages are exchanged between close friends. Examination of these contextual elements is fundamental to understanding how e-mail and other practices shape and are shaped by the distributed software development project.

Teleconferencing is another practice that deserves careful consideration. Anyone who has participated in a teleconference can identify with the awkwardness of this form of dialogue. It's often difficult to control speaking order, people frequently talk over one another, and equipment problems can make people in the other location difficult to understand. The data in this study indicate that when linguistic and cultural differences are

present, along with inter-office power struggles, these problems are dramatically amplified.

Managers at the organization recognized the challenges presented by the project's linguistic, cultural and political context, and tried many things to mitigate the emerging problems. For example, English language training was made available in the German office, with some success. In addition, developers were repeatedly encouraged – "prodded" – to phone their colleagues when problems needed resolving, rather then send them messages via e-mail. Unfortunately, the time zone separation limited the effectiveness of this approach.

The project team leaders employed a simple but powerful tool to create links among team members: the company intranet. They created a site where they posted photographs of everyone working on the project. Their goal was to decrease the personal distance between the teams. It was generally well received, and many participants commented on the positive impact that this had on the team.

Another practice that many respondents had positive comments about was the opportunity to travel to the other office. Such travel provided an occasion to get to know and work with individuals from that office. They made high-bandwidth, face-to-face meetings possible, and provided an excellent opportunity for what one participant called cross-pollination between the offices: an opportunity to exchange ideas about how the thing that everyone is building should be built. When people spent time with one another, the cultural, and linguistic barriers began to break down, leading to less conflict. According to one senior level manager, these exchanges were 'like gold'. Many people reported that whenever team members spent time in the other office it was a successful team-building occasion.

Despite these efforts, it was team-building that emerged as one of the biggest challenges for this project. Numerous respondents indicated that, while they put many processes into place to coordinate and control day-to-day activities, and while these processes were successful for the most part, process was not enough. Throughout the twenty-month term of the distributed development experiment, a strong sense of a single team never emerged.

6. Participants' General Comments

Most participants eagerly voiced their general opinion about their experience with distributed software development. They talked about communication in a very broad sense, incorporating language, time-zones, culture, as well as one or more practices. It was generally recognized that the success of any programming activity depends on successful communication and doing global work requires a certain overhead to maintain communication channels. According to one participant, when times are tight economically, it is these maintenance activities that are easiest to cut. Sooner or later, however, there are problems in taking this approach. Issues will pop up down the road.

Several participants used the term *bandwidth* to describe the character of the communication channel between various teams and offices. A high bandwidth link exists between individuals located in the same city, or between cities separated by a minimal number of time zones, whereas the link between the Canadian office and the German office has a very low bandwidth. Bandwidth is further degraded by cultural or linguistic distance.

Acknowledging the extra overhead inherent with distribution, several people remarked that it was simply too expensive for a single team, working from different offices, to develop a single product.

7. Conclusion

It is always compelling to try and isolate the one factor - the silver bullet [1] – that will solve a problem. In this case, the problem is how to use globally distributed technical personnel to create software in an effective and economical manner. For this study, no single factor can be isolated as the cause of the failure. For example, while temporal separation on its own might not present an insurmountable barrier to successful distribution, the cumulative impact of an eight-hour time zone difference, a subsequent reliance on asynchronous communication, and a poor inter-office relationship appears to have been a significant obstacle. Similarly, while teleconference meetings are admittedly a challenge for most of us, they nonetheless function as a powerful and effective communication tool for many international organizations. This effectiveness is severely diminished when language or cultural barriers hinder the degree to which all people involved in the meeting can participate fully.

In future work, I will analyze this data from an anthropological perspective, using the anthropology of technology as a general framework from which to consider the technology of distributed software development. This analysis will employ a decidedly expanded view of technology that considers not only the servers, telephones, teleconferencing equipment, code repositories, management practices and software processes that structured the daily activities of the software developers, but the social and cultural context in which these tools and processes were employed.

In the absence of this analysis, it is my hope that this summary of study findings will make a useful contribution to discussions of global software development.

8. References

[1] Brooks, Frederick P, *The Mythical Man-Month*, Addison Wesley, Berkeley, 1995.

Defect Detection in a Distributed Software Maintenance Project

Alessandro Bianchi, Danilo Caivano, Filippo Lanubile, Giuseppe Visaggio
Dipartimento di Informatica – Università di Bari - Via Orabona, 4, 70126 Bari – Italy
{bianchi, caivano, lanubile, visaggio}@di.uniba.it

Abstract

A large software project may be distributed over multiple sites when the organization needs resources which are available on a single site. However, previous empirical research in the context of telecommunication organizations has shown a number of disadvantages. In this paper we continue our comparative postmortem analysis on data from a large software massive maintenance project in the information systems domain, which in part has been carried out on a single site, and in part across multiple sites of the same organization. Results show that no significant differences exist among the distributed and collocated work with respect to the ability to detect defects.

Keywords: Global Software Development, Empirical Study, Massive Maintenance

1. Introduction

The new forms of competition and cooperation that have arisen in software engineering as a result of the globalization process have had an impact on the whole software process. Software development and maintenance are often distributed across sites, thus involving an increasing number of people with different cultural backgrounds. Carmel and Agarwal [1] report that at present, 50 different nations are collaborating in different ways in software development.

However, global software development has a number of drawbacks, which have been recognized by many studies, such as the need to apply ad hoc management methods [2], the need to use knowledge sharing tools [3, 4], and the overhead derived from staff communication interchanges [5]. Herbsleb and Moitra [6] classified the main drawbacks in global software development in a set of issues:

- *strategic issues*, concerning the decisions on how to divide the tasks among sites, so as to be able to work as independently as possible while maintaining efficient communication among sites;
- *cultural issues*, that arise when the staff come from different cultural backgrounds;
- *inadequate communication*, caused by the fact that geographical distribution of the staff over several sites increases the costs of formal communications among team members and limits the possibility of carrying on the informal interchanges that traditionally helped to share experiences and foster cooperation to attain the targets;
- *knowledge management*, that is more difficult in a distributed environment as information sharing may be slow and occur in a non uniform manner, thus limiting the opportunities for reuse;
- *project and process management issues*, having to do with all the problems of synchronization of the work at the various different sites;
- *technical issues*, that have an impact on the communication network linking the various sites.

Previous investigation on how geographical distribution affects software development and validation activities, have been carried out, respectively, at Lucent Technologies [7] and Alcatel [8]. Main findings were that distance negatively affects cost, time and quality. However, those studies were both conducted in the context of a telecommunication application domain and involved complex tasks.

Our research takes its rise from the acknowledgement that the application domain and the software engineering task are both fundamental drivers of global software development costs and benefits. For projects involving massive, well-defined and stable activities, we hypnotize that the distribution over different geographical sites would present just a project management overhead.

In this context, previous papers by the same authors concerned an explorative analysis [9] and an investigation on communication and project management issues [10]. In this paper, we investigate significant differences, if any, in detecting defects when maintenance activities are executed on a single site rather than on multiple sites.

The paper is organized as follows: section 2 presents the maintenance project and the metrics used in the analysis; section 3 illustrates the data analysis; the results are discussed in section 4, and section 5 draws some conclusions.

2. Case Study Setting

2.1. Project Characterization

Our research can be characterized as a post mortem analysis on data concerning a maintenance project carried out by EDS-Italia. In the following, we only summarize the main features of the maintenance project; interested readers can refer [10] for a more detailed presentation.

The project consisted in a massive, non-routine maintenance of a large information system to solve the Y2K problem. To this end, the software system had been decomposed into 100 *work-packages* (WP), each being assigned to a working team. The maintenance effort had to deal with 52 of them. The job was partitioned between 2 different geographically distant sites, both settled in Italy.

The size of each WP is expressed by the number of items, where an item can be a program, a library element or a Job Control Language (JCL) procedure, i.e., a procedure written in a scripting language to control the program execution in batch systems.

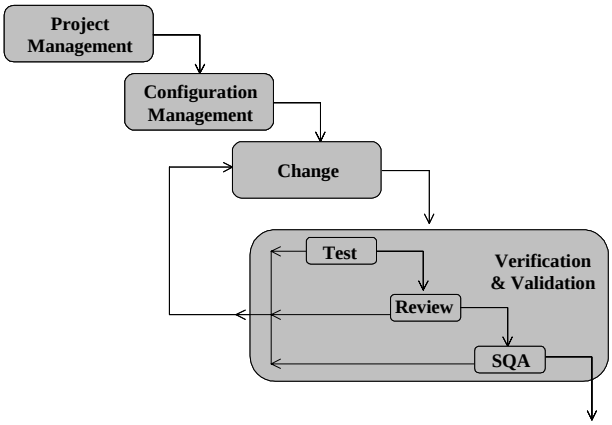


Figure 1. The process adopted for each WP in the maintenance project.

The maintenance project was executed according to the following process (Fig. 1) that was enacted for each WP:

- a *Project Management* phase, aimed at managing and scheduling the activities for the WP;
- a *Configuration Management* phase, aimed at collecting and identifying all the artifacts produced within the WP;
- a *Change* phase, aimed at executing the maintenance of the items belonging to the WP;
- a *Verification & Validation* phase, aimed at looking for defects into the maintained artifacts.

When defects are identified, the maintained items are reworked looping from the Corrective phase.

The Verification & Validation phase, in turn, includes three sequential activities:

- a *Test* activity, aimed at looking for failures and related faults into the maintained items
- a *Review* activity, aimed at looking for defects into the maintained artifacts through inspection meetings;
- a *Software Quality Assurance* (SQA) activity, aimed at verifying that the maintained artifacts comply with the company's Quality System.

For all the WPs, the Project Management established to start process execution on a single site (hereinafter referred to as *Site1*) but, depending on both rework needs and currently available resources, the execution of Change and Defect Detection phases could also be switched to another site (hereinafter referred to as *Site2*). According to [5], we consider the WPs entirely executed at Site1 as part of a *collocated project*; conversely the WPs executed both at Site1 and Site2 as belonging to a *distributed project*.

2.2. Data Collection

The post-mortem analysis included all the work packages and covered the entire WP life cycle. In the following we only focus on the Defect Detection phase; the measures taken into account are:

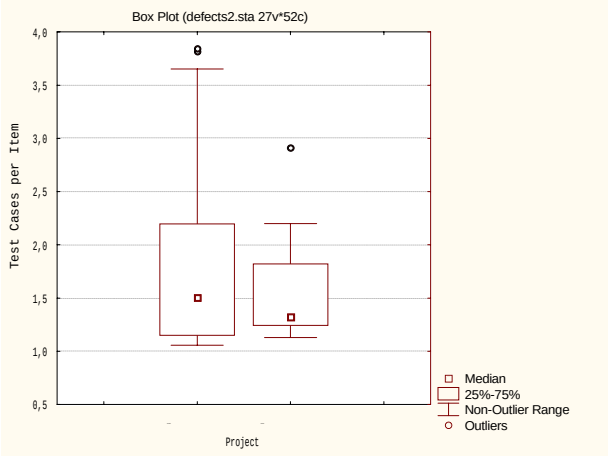
- number of executed test cases and the number of faults that caused failures: in the following these will be referred to as *faults from testing*;
- number of reviews and the number of defects they found out (in the following, number of *faults from review*);
- number of audits and the number of issues they found out (in the following, number of *non conformities*);
- size of the WPs, expressed as number of *items*.

Unfortunately, the number of failures has not been recorded by the organization, but only the number of faults generated by those failures.

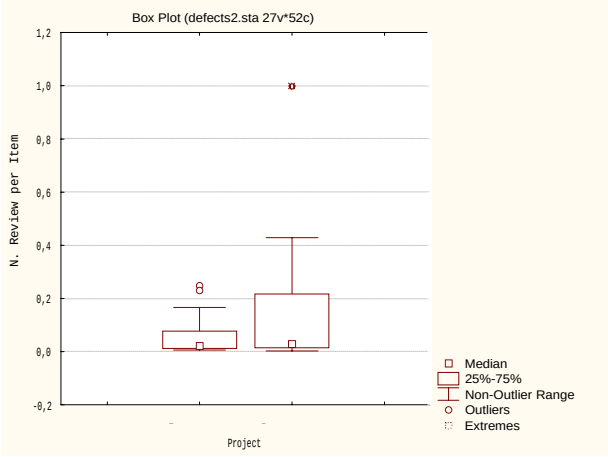
It is worth noting that the number of executed test cases, reviews and audits as well as the size of WPs are used only for verifying the comparability of the two projects. The dependent variables taken into account in our investigation are the number of faults from testing and from review and of the number of non conformities.

Since the variation of WPs size is quite high, ranging from 6 items to 8337 items and with quartile values ranging from 68.5 items to 533 items, our analysis was based on the metric values normalized with respect to WP size.

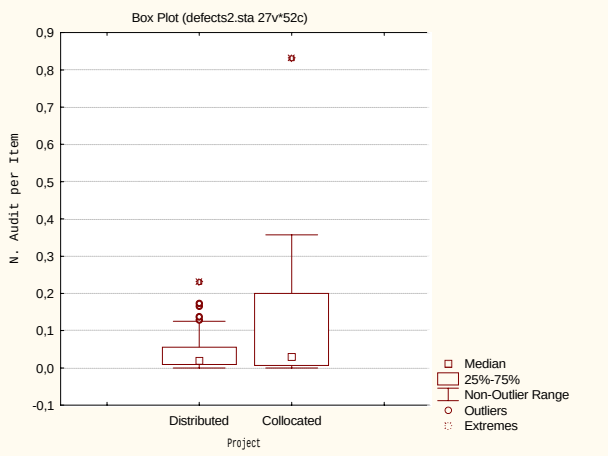
Having normalized, the tasks executed in the collocated and distributed project did not present technical differences. In fact, the number of items maintained was approximately the same in the two projects. The total number of maintained items was 26,739: among these, 14,163 items (53%) were maintained in the collocated project, and 12,576 items (47%) in the distributed one.



2.a)



2.b)



2.c)

Figure 2 Boxplots of the normalized number of test cases (a), reviews (b), and audits (c), executed in collocated and distributed projects.

For what concerns the activities of the Defect Detection phase:

- the density of test cases executed in the collocated project (median 1.327) is comparable to the normalized number of test cases executed in the collocated project (median 1.506); these results are summarized in Figure 2.a;
- the density of reviews executed in the collocated project (median 0.029) is comparable to the normalized number of reviews executed in the collocated project (median 0. 022); these results are summarized in Figure 2.b;
- the density of audits executed in the collocated project (median 0.030) is comparable to the normalized number of audits executed in the collocated project (median 0.020); these results are summarized in Figure 2.c.

3. Data Analysis

Available data led to two samples from possibly different populations, and the samples taken into account were not normally distributed. Moreover:

- both samples are random samples from their respective populations;
- in addition to independence within each sample, there is mutual independence between the two samples;
- the measurement scale is at least ordinal.

Since these assumptions allow to apply the Mann–Whitney *U* test [11], we used this nonparametric test to analyze defect metrics.

In order to investigate whether the distribution between sites does affect defect metrics, for each metric M_i the null and alternative hypotheses are formulated as follows:

Hi_0: There is no difference between the values of metric M_i for collocated WPs and for distributed WPs.

Hi_a: There is a difference between the values of metric M_i for collocated WPs and for distributed WPs.

3.1. Number of Faults from Testing

The first analysis made on defects data assessed the number of faults discovered through the execution of the test activity. Figure 3 shows the boxplots of the distribution of number of faults for both collocated and distributed projects.

For both the collocated and distributed WPs, the median is 0; the WPs in collocated case does not present any outlier, and they have three extreme values (0.004, 0.021 and 0.071); conversely, the WPs in distributed case present two outliers (0.013 and 0.020) and two extremes (0.028 and 0.029).

The non parametric Mann-Whitney U test failed to reveal a significant difference between the two groups (p-level = 0.489).

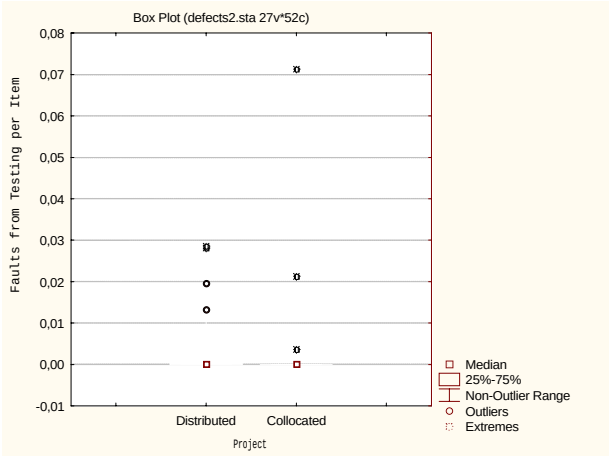


Figure 3 Boxplots of the faults from testing in collocated and distributed projects.

3.2. Number of Faults from Review

Figure 4 shows the boxplots of the distribution of number of faults discovered during the execution of the review activity for both collocated and distributed projects.

For the collocated WPs, the median is 0.020 and for the distributed WPs the median is 0.040; the WPs in collocated case present an extreme value (0.429) and they have not any outlier; conversely, the WPs in distributed case have an extreme value (0.20), and two outliers (0.139 and 0.154).

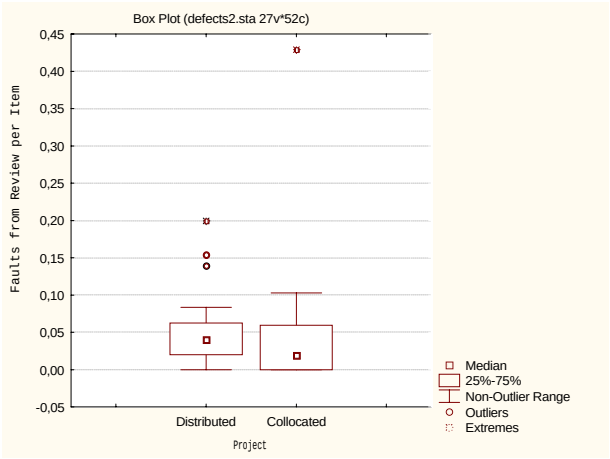


Figure 4 Boxplots of the number of faults from review in collocated and distributed projects.

The non parametric Mann-Whitney U test failed to reveal a significant difference between the two groups (p-level = 0.212).

3.3. Number of Non Conformities

Figure 5 shows the boxplots of the distribution of number of non conformities for both collocated and distributed projects., The median is 0.0 for the collocated WPs and it is 0.005 for the distributed WPs; the WPs in collocated case present two outliers (0.063 and 0.071) and two extremes (0.089 and 0.111); the WPs in distributed case present one outlier (0.032) and one extreme value (0.077).

The non parametric Mann-Whitney U test failed to reveal a significant difference between the two groups (p-level = 0.633).

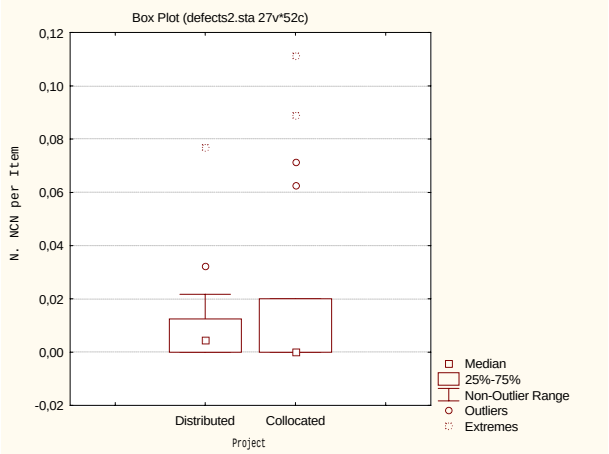


Figure 5 Boxplots of the number of non conformities discovered in collocated and distributed projects.

4. Discussion and Conclusions

In general, collocating the maintenance activities or splitting them over two sites did not differ with respect to defect metrics. In both cases, the observed differences were all not statistically significant at the conventional 0.05 *p* level. We postulate that these results can be explained by considering the context, which characterizes this case study.

The specific maintenance task carried out was conceptually simple and it is characterized by a massive and repetitive nature. The main skills required to execute the maintenance were generic programming skills for the Y2K problem, and knowledge of the application domain and the software system to maintain. Therefore, the choice of the most adequate maintenance team to assign a WP

was straightforward, even when teams were geographically separated.

The majority of maintainers had a deep knowledge of both the application domain and the system, because of previous experience maintenance related to the same system. Moreover, all of them had been trained on the Y2K problem, and many maintainers had been already involved in other Y2K activities.

Moreover, there was a strong organizational and cultural cohesion between the two sites because they were part of the same company and located in the same country, at a distance no more than 300 Km.

Finally, since it was a massive maintenance project, the project components were loosely coupled and therefore the need to manage a common knowledge was kept to a minimum.

As a consequence of these features, even the followed defect detection strategy was quite straightforward: the loose coupling of the project components to be maintained allowed project managers an easy partition and distribution of the items to test and WPs to inspect across sites. So, each site could operate on each WP as an independent (sub)system. In this way, the distribution of the Verification & Validation phase between sites did not determine any statistically significant difference with respect to the execution of the same phase in a collocated environment.

Nevertheless the cultural homogeneity of the teams involved in the collocated and in the distributed project, extremes and outliers are encountered in all sets of data. This can be explained by the human-centric nature of software processes: maintainers adopted different tactics to execute the assigned tasks, even if simple.

These results confirm the hypothesis we made in our previous analysis [10] about the need of an adequate management of the strategic, cultural, and technical issues in order to make effective the distribution of software process. If so, the distribution of the process over geographically distant teams makes it possible to include skilled people, wherever they are available, without significant loose in technical aspects of the process as well as the Defect detection.

This study is one step towards a model of impact of geographical distance on critical factors of software development and evolution, which still needs further empirical investigation.

References

[1] E. Carmel, R. Agarwal, "Tactical Approaches for alleviating Distance in Global Software Development", *IEEE Software*, Mar-Apr 2001, pp. 22-29.

[2] A. Cockburn, "Selecting a Project's Methodology", *IEEE Software*, July-August 2000, pp.64-71.

[3] K. Nakamura, Y. Fujii, Y. Kiyokane, M. Nakamura, K. Hinenoya, Y.H. Peck, S. Choon-Lian, "Distributed and Concurrent Development Environment via Sharing Design Information", *Proc. of the 21st Intl. Computer Software and Applications Conference*, 1997.

[4] J. Suzuki, Y. Yamamoto, "Leveraging Distributed Software Development", *Computer*, Sep 1999, pp.59-65.

[5] C. Ebert, P. De Neve, "Surviving Global Software Development", *IEEE Software*, Mar-Apr 2001, pp.62-69.

[6] J.D. Herbsleb, D. Moitra, "Global Software Development", *IEEE Software*, Mar-Apr 2001, pp. 16-20.

[7] J.D. Herbsleb, A. Mockus, T.A. Finholt, R.E. Grinter, "An Empirical Study of Global Software Development: Distance and Speed", *Proc. Intl. Conf. on Software Engineering*, 2001, pp. 81-90.

[8] C. Ebert, C.H. Parro, R. Suttels, H. Kolarczyk, "Improving Validation Activities in a Global Software Development", *Proc. Intl. Conf. on Software Engineering*, 2001, pp.545-554.

[9] A. Bianchi, D. Caivano, F. Lanubile, F. Rago, G. Visaggio, "Distributed and Colocated Projects: a Comparison", *Proc. of the IEEE Workshop on Empirical Studies of Software Maintenance*, 2001, pp. 65 – 69.

[10] A. Bianchi, D. Caivano, F. Lanubile, F. Rago, G. Visaggio, "An Empirical Study of Distributed Software Maintenance", *Proc. of the IEEE Intl. Conf. on Software Maintenance*, Montreal–Canada, October 2002, pp. 103–109.

[11] W.J. Conover, *Practical Nonparametric Statistics*, John Wiley and Sons, 1980

Requirements Management in Global Software Development: Preliminary Findings from a Case Study in a SW-CMM context[i]

Rafael Prikladnicki, Jorge Audy, Roberto Evaristo
School of Computer Science, Pontifical Catholic University of Rio Grande do Sul, Porto Alegre, Rio Grande do Sul, Brazil; University of Illinois, Chicago, United States
rprik@inf.pucrs.br, audy@inf.pucrs.br, evaristo@uic.edu

Abstract

The requirements analysis is an important phase in the software development process. In geographically distributed environments (Global Software Development), requirements management becomes critical due to the characteristics of the distributed development (physical distance, cultural differences, trust, communication, etc). The objective of this paper is to analyze the requirements management in geographically distributed environments, identifying the main challenges. The results are based on a case study carried on at Dell Computers, a multinational organization that has offshore software development centers in Brazil, India and Russia, and was recently certified in SW-CMM Model level 2 in the Brazilian unit. The results suggest the necessity to adapt the requirements management to the distributed software development environment, addressing the main existing challenges. The problems and the solutions adopted are presented, aiming to relate these solutions to the organization distribution level, considering the project team, users and customers.

1. Introduction

Software development has become part of the business globalization. This is mainly due the need for cost reduction, increased competitiveness and the possibility to share resources in a global scale [5]. As a consequence, the communication between the project team, users and customers occurs in a geographically distributed way. In this case, the requirements management seems to be an even more critical activity. Normally, the requirements' gathering occurs in meetings having all the participants (project team, users and customers) in the same place. This facilitates the communication, becoming easier any negotiation or existing conflict resolution. In the Software Engineering literature, software requirements represent the interests of customers and users and are the heart of any project [8]. The quality and the capacity of analyzing and managing the requirements of a software project not only affect the final product quality, but also the time necessary to satisfy the requirements. A badly managed requirement can mean a loss for the project and can compromise it success, generating delays or the cancellation of projects.

In distributed software development environments the challenges become even more significant. This paper has as objective to understand what kind of problems the project teams has faced when managing requirements in physically distributed environments and how these problems have been addressed. With this objective, a case study was conducted in a multinational organization with software development centers in Brazil, India and Russia, identifying the difficulties to analyze and to manage the requirements of a software project in this type of environment. The results are analyzed and the existing challenges are identified. Some of the solutions that are being implemented with the objective of minimizing the problems found are presented. Our contribution is in the identification of these problems and the addressing of the solutions. This paper has the following structure: section 2 presents the theoretical base; section 3 describes the research method; section 4 describes the case study; section 5 discuss the results found in the case study; section 6 presents the conclusions, future studies and the research limitations.

2. Theoretical Base

2.1. Requirements Management

Requirements engineering plays an important role in the software development. As said by [8], a requirement is the condition or capacity that a system that is being developed must satisfy. Therefore, the compliance with requirements determines the success or the failure of a project. The requirements are identified, registered, organized and verified during the project development. And that is what it called requirements management, a process that establishes and keeps the agreements firmed

between the project team, users and customers related to the changes of requirements in a specific system.

The literature states that the problems related with requirements engineering are one of the main reasons for software projects failures. This means that the final product does not have all the requirements gathering from users and customers [13]. Research identified that 70% of the requirements were difficult to identify and 54% were not clear and well organized. Also, it can be identified that [8]:

- Requirements are not easy to be described in words;
- There are different types of requirements in different levels of details;
- It can be impossible to manage the requirements if they cannot be controlled;
- Most requirements change during the project time.

Therefore, it is not difficult to find errors in the requirement specifications, and they can have a large impact in the project costs. An estimative shows that 40% of the requirements generate rework during the project life cycle [13]. It is evident that the earlier a problem is detected and solved (especially during the requirements phase), many other problems are minimized in the following project phases. But in contrast, what it is observed is a short time for the requirements phase in a project, not considering the project type or environment where this phase occurs.

2.2. Global Software Development (GSD)

As said by [9], software process is defined by a set of activities, methods, practices and technologies that people and companies use to develop and to keep related software and products. The interest in the software process is based on the following premises:

- The software quality is strongly dependent on the quality of the process used in its preparation;
- The software process can be defined, managed, measured and improved.

However, it is not a simple task to develop software using a well-defined development process. Such process has become increasingly more complex, whereas the software demands of companies increase according to the strategic importance for its operations.

As part of the globalization efforts currently pervading society, software project teams have also become geographically distributed on a worldwide scale. This characterizes Global Software Development (GSD).

Tools and technological environments have been developed over the last few years to help in the control and coordination of the development teams working in distributed environments. Many of these tools are focused in supporting procedures of formal communication such as automated document elaboration, processes and other non-interactive communication channels.

Moreover, [3], [4], [5] and [10] point out that GSD is one of the biggest business-oriented challenges that the current environment presents under the software development process point of view. Many companies are distributing its software development process in countries such as India, Russia and Brazil. Frequently this process occurs in only one country, particularly in regions with tax incentives or critical mass in some skill or resource areas.

Organizations search for competitive advantages in terms of cost, quality and flexibility in the area of software development [10], looking for productivity increases as well as risk dilution [7]. Many times the search for these competitive advantages forces organizations to search for external solutions in other countries (offshore outsourcing). This epitomizes the traditional problems and the existing challenges in GSD.

2.3. The Capability Maturity Model (CMM)

The Capability Maturity Model for Software (CMM or SW-CMM) has been developed by the software community with stewardship by the SEI (Software Engineering Institute). The first version was released in 1992 and describes the principles and practices underlying software process maturity and is intended to help software organizations improve the maturity of their software processes in terms of an evolutionary path from chaotic processes to mature, disciplined software processes. The CMM is organized into five maturity levels [6]:

1 - Initial: The software process is characterized as ad hoc. Few processes are defined, and success depends on individual effort and heroics;

2 - Repeatable: Basic project management processes are established to track cost, schedule, and functionality. The necessary process discipline is in place to repeat earlier successes on projects with similar applications;

3 - Defined: The software process is documented, standardized, and integrated into a standard software process for the organization for both management and engineering activities;

4 - Managed: Detailed measures of the software process and product quality are collected. Both the software process and products are quantitatively understood and controlled;

5 - Optimizing: Continuous process improvement is enabled by quantitative feedback from the process and from piloting innovative ideas and technologies.

Except for Level 1, each maturity level is decomposed into several key process areas that indicate the areas an organization should focus on to improve its process [6].

3. Research Method

This research is characterized as a study mostly exploratory, since the main research method was the case

study. It is possible to justify the use of qualitative methods since it involves the study of the system development process in its real context, with description and the understanding of the state of the art in those situations where practice precedes theory [12].

4. Case Study

4.1. Characterization of the Organization

The organization is a global software development center (GDC) located in Brazil, owned by Dell Computers, a multinational organization with worldwide activities (one of the largest computer manufacturers in the world). This center was created in 2001 using incentives based on the Brazilian Law on Information Technology that stimulates companies located in the country to invest part of their earnings on research and development institutions providing tax exemption on manufactured products (IPI). The GDC aims to perform technological development for the organization in worldwide scope and since July of 2002 it is located inside of the technological park of a university in the South of Brazil (PUCRS). Many research projects are being developed, like the SW-CMM level 2 certification process monitoring and the study related to Global Software Development. All research projects are performed using both the organization professionals and the researchers and students of the host university.

Figure 1 shows the context of this study: the organization acts in global software development environment, having the Microsoft Solutions Framework (MSF) and the SW-CMM model as base for the software development processes definition. The Brazil GDC is a SW-CMM level 2 certified organization since January of 2003, with 2 years of work done to achieve this certification.

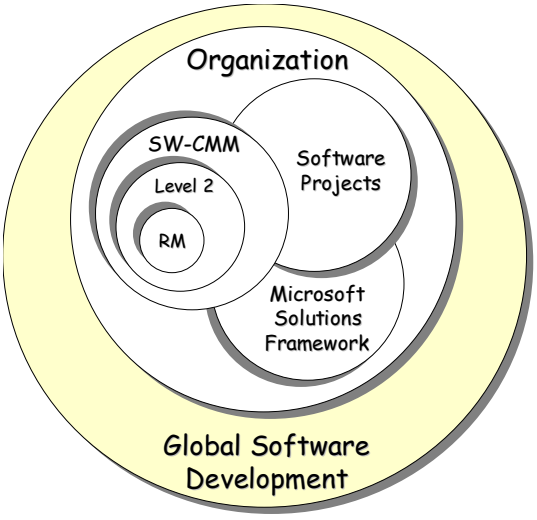


Figure 1. Organization context.

4.2. Defining the two projects evaluated

The objective of this case study is to analyze two projects developed in Brazil GDC, aiming at the identification of problems, advantages and disadvantages considering the requirements management (RM) in both projects in a geographically distributed context at the same time where the organization was working to obtain the SW-CMM level 2 certification. These projects were developed in the second semester of 2002.

Project 1: The objective of this project was to develop a new version of a tool related to employee compensation for the human resources area of the worldwide organization. This project lasted nine months. According the classification proposed by [11], the project team, customer and users had the following distributed level:

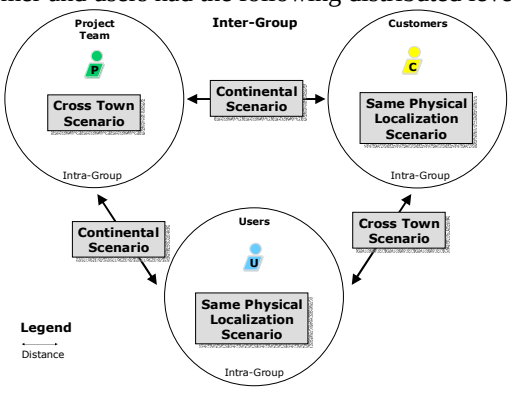


Figure 2. Project 1.

The three members of the project were located in different buildings of the Brazilian unit. The customers and users were located in the organization headquarters in the U.S., each one in its building.

Project 2: The objective of this project was to integrate and consolidate two versions (Latin American and Canadian) of an application of the manufacturing area into a single application. This project lasted one year. According the classification proposed by [11], the project team, customer and users had the following distributed level:

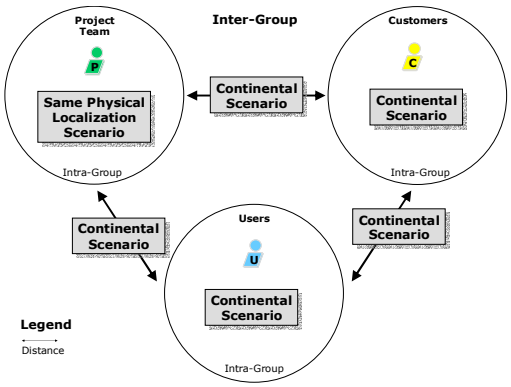


Figure 3. Project 2.

The project team had thirteen members and was located in the Brazilian unit, in the same physical space. The customer's team were distributed in a continental way, with members in Brazil, Mexico, Argentina, Canada and the U.S. The user's team were located in Latin America and Canada.

4.3. Case Study Results

During the project development, considering the requirements management, many observations were done and interviews with the Program Managers were performed. The results are presented below.

4.3.1. Project 1. The project was developed very well, even though there were both highlights and problems. Many solutions had been implemented in common agreement between the geographically distributed teams.

Considering just the geographic team distribution, the following issues were found:

Problems:

- Communication problems in the project beginning;
- Bad distribution of the activities between teams of Brazil and the U.S.;
- Bad planning in the beginning of the project;
- Inexistence of an evaluation of the impact that an activity would have for being done in a distributed way;
- Videoconference resources had not been used.

Highlights:

- The initial problems had motivated the definition of working standards;
- The distance facilitates the formalism;
- The Brazil team spent six weeks in the U.S. for the knowledge transfer process and to start the requirements gathering;
- The customer had visited Brazil to know the team and to approve the requirements specification document;
- Social programs had been done during the trips;
- Before the U.S. team came to Brazil, the Brazilian team finished all the pending activities aiming a good impression;
- The customer was not American and had already have problems related to cultural differences;
- A weekly meeting was performed with the customer.

Considering the implementation of the SW-CMM Model level 2, there was a great contribution of the certification process to the use of the requirements management process in a geographically distributed environment. The following issues were found:

Problems:

- The U.S. team was not involved in the certification process. Therefore, the Brazilian team had an additional task to explain why each activity was done that way.

Highlights:

- The implementation of process based on SW-CMM Model helped in the organization and standardization of the activities between the geographically distant teams;
- The director of the customer area in the U.S. gave total support in the understanding of the necessity to follow the process defined by the Brazilian team;
- The U.S. team, although not involved in the certification process, absorbed all the knowledge related to the process.

Considering the tools used beyond the meetings done physically in the same place, the teams communicated through e-mails, teleconferences and net meeting.

Finally, the requirements phase, considered a critical phase in any project, was performed to satisfaction. This was possible because of the work that all teams did in order to minimize cultural differences, communication problems, trust problems and the work related to the SW-CMM Model level 2 certification process. In the end, the project was delivered before the planned date.

4.3.2. Project 2. The project was developed without problems. It had highlights and problems, and many solutions had been implemented in common agreement between the geographically distributed teams.

Considering just the geographic team distribution, the following issues were found:

Problems:

- Lack of customer work standardization, due to its geographic distribution (teams in many countries);
- Lack of trust in the project beginning related to the U.S. team, due to the competition between the teams;
- The U.S. team did not have a well-defined process;
- The time zone confused in the accomplishment of requirements gathering meetings;
- Videoconference resources had not been used.

Highlights:

- The initial problems had motivated the definition of working standards;
- The Brazilian Program Manager had participated since the beginning of the project, due to the project and the team sizes;
- The requirements specification was standardized;
- The Brazilian Program Manager spent one week in the U.S. in the project beginning;
- Two members of the U.S. customer team had visited Brazil to know the environment and they were surprised;

- The cultural differences had been absorbed delegating tasks in accordance with the profile and the culture of a team member (i.e. the Canadians were very good in something and the Americans in other things.);
- Three weekly meetings were performed with the customer.

Considering the implementation of the SW-CMM Model level 2, it was also verified a great contribution of the certification process and the use of the requirements management process in a geographically distributed environment. The following issues were found:

Problems:
- The U.S. team was not involved in the certification process. Therefore, the Brazilian team had an additional task to explain why each activity was done that way;
- To be involved in a certification process caused a work overhead.

Highlights:
- The requirements management process was well defined and was completely incorporated by other teams;
- The implementation of process based on SW-CMM Model helped in the organization and standardization of the activities between the geographically distant teams;
- The use of process was mandatory and helped in the trust acquisition between the teams;

Considering the used tools, beyond the meetings physically the same place, the teams had communicated through e-mails and teleconferences.

Finally, the requirements phase, considered a critical phase in any project, was done very well, despite of some problems due to the project distribution. Many meetings were performed and some communications problems were identified, but all teams worked very hard in order to minimize the problems, mainly those problems related to cultural differences, communication problems and the work related to the SW-CMM Model level 2 certification process. With all this effort, the first phase of the project was delivered on time and without errors.

5. Impacts of GSD in Requirements Management

After the analysis of these two projects, it can be conclude that to manage requirements in a global software development context can become an arduous task if the process will not be well defined and if the teams will not be previously prepared to work in this scenario. Some studies [1], [2] and [13] point to problems such lacks of communication, cultural differences, collaboration, knowledge management, context sharing, lacks of contact between people and the lack and difficulty to use tools to give support to the activities in distributed environments.

What was perceived in this case study is that all the work involving the SW-CMM Model level 2 certification in the Brazilian unit collaborated in a big scale to minimize some problems found in this scenario. The definition of a software development process based on the CMM model brought excellent results related to the distributed environments problems. Also, the teams were able to standardize all the work and to converge in a common understanding about the best approach to develop both projects. So, it can be concluded that many of the efforts spent in the SW-CMM Model level 2 certification contributed to minimize problems like organization, standardization and, sometimes, communication. The training that was applied concerning soft skills minimized the distance impact and some problems related to these things (trust, cultural differences, etc). All these issues take to believe that, despite the existing difficulties in working this way, a good training is a key to success.

5.1. Lessons Learned

Many lessons were learned during the developing of these two projects, considering the requirements management phase. In the following table there is a list of the main lessons learned:

Table 1. Lessons Learned.

No.	Lesson
#1	Training the team in soft skills (trust, cultural differences, communication, collaboration, context sharing, knowledge management, etc.) is essential.
#2	Work standardization is mandatory.
#3	Frequent meetings with people geographically distant are very important to track the project.
#4	A well-defined process is a key to success.
#5	If it's possible, travels can occur to meet each team involved in the project.
#6	The time zone can act as an advantage and a disadvantage at the same time.
#7	The use of tools like email, conference calls and videoconferences are very important.
#8	To have a certification process like SW-CMM Model level 2 in parallel can increases the overhead, potentially leading to overload.
#9	The SW-CMM Model level 2 certification process helped to define a standard way to work.
#10	It's very important to know about the people that your are working, considering the way to communicate, cultural differences, etc.

6. Conclusions

This paper advances the knowledge in the GSD area when identifying some important characteristics of the requirements management phase in a distributed environment, in parallel with a SW-CMM Model level 2 certification process. As result, many important issues were identified and many lessons were learned. A comparison can be done between the two projects analyzed, in order to understand the way that each team did his work, considering the distribution level and the teams profile.

This study enables a better understanding of the GSD area and the relationship between the project team, customers and users related to the requirements management phase. It is also applies in projects the standard created for comparison between different organizations [11], opening space for new research in this area. Due to the small number of case studies, the results cannot be generalized. In this phase of the study can be adopted the analytical generalization principle, proposed by (Yin 1994).

This preliminary set of results that we are finding give us trustful indication that the search for greater formalism in the development process and the selective utilization of international pattern will provide full conditions to overcome the linguistic and cultural differences, particularly in requirements management, which is the focus of this paper. As contributions of this study, it can be highlighted the lessons learned and the main advantages in having training in soft skills and a well-defined process to work in distributed environments. Moreover, a certification process in a quality model and a continuous software process improvement are very important to succeed.

This study was not considered an analysis of the reasons than can take an organization to adopt strategies of distribution, nor the software development process by itself. Planned follow up studies in this topic will analyze the changes in a general way, considering not only requirements management, but also in all project phases. Some alternatives will be searched and solutions related to the GSD process identified, considering all difficulties and critical success factors like culture, communication, coordination, trust and cooperation.

7. References

[1] Damian, D., The study of requirements engineering in global software development: as challenging as important, *Proceedings of International Workshop on Global Software Development – ICSE 2002*, Florida, USA, 2002.

[2] Evaristo, R., and Scudder, R., Geographically Distributed Project Teams: A Dimensional Analysis, *Proceedings of the Thirty-third Hawaii International Conference on Systems Sciences*, 2000.

[3] Grinter, R. E., Herbsleb, J. D., and Perry, D. E, *The Geography of Coordination: Dealing with Distance in R&D Work.* ACM, 1999.

[4] Herbsleb, J. D., Mockus, A. Finholt, T. A., and Grinter, R. E, *An Empirical Study of Global Software Development: Distance and Speed*, IEEE, 2001.

[5] Herbsleb, J. D., and Moitra, D., *Global Software Development*, IEEE Software, pp. 16-20. March/April/2001.

[6] Mark C. Paulk, Bill Curtis, Mary Beth Chrissis, and Charles V. Weber, "Capability Maturity Model, Version 1.1," IEEE Software, Vol. 10, No. 4, July 1993.

[7] McConnel, S., *Rapid Development***.** Microsoft Press, 1996.

[8] Oberg, R., Probasco, L., and Ericsson, M., "Applying Requirements Management with Use Cases", *Rational Software White Paper*, Cupertino, CA, 2000, pp. 3-5.

[9] Pressman, R. S., *Software Engineering: A Practitioner's Approach.* Fifth Edit, 2001.

[10] Prikladnicki, R., Peres, F., Audy, J., Móra, M. C., and Perdigoto, A., Requirements specification model in a software development process inside a physically distributed environment, *Proceedings of ICEIS 2002*, Ciudad Real, Spain, 2002.

[11] Prikladnicki, R., Audy, J., and Evaristo, R., Distributed Software Development: Toward an understanding of the relationship between project team, users and customers, *to be presented in ICEIS 2003*, Angers, France, 2002.

[12] Yin, R. K, Case study research: design and methods, Sage, 1994.

[13] Zowghi, D., Does Global Software Development Need a Different Requirements Engineering Process?, *Proceedings of International Workshop on Global Software Development – ICSE 2002*, Orlando, Florida, USA, 2002, 53-55.

[i] Paper developed in the Dell Brazil Global Development Center (GDC), sponsored by DELL Computers through the Brazilian Law on Information Technology.

Communication Needs, Practices and Supporting Structures in Global Inter-Organizational Software Development Projects

Maria Paasivaara
Helsinki University of Technology
Software Business and Engineering Institute
P.O.B. 9600, FIN-02015 HUT, Finland
Maria.Paasivaara@hut.fi

Abstract

This paper presents communication needs, supporting structures and communication practices collected from global software development projects. The data was gathered by 32 interviews from seven global inter-organizational projects. We identified four important communication needs: problem solving, informing and monitoring, relationship building, and decision-making and coordination. Structures supporting communication were: organizational structure with communicating roles, partial synchronization of intra-organizational processes, and project level coordination. Communication practices are built upon and facilitated by these structures. A surprising finding was that companies rarely had any company level practices that were used in all inter-organizational projects. Instead, the practices were formed by trial and error and were mainly project specific.

1. Introduction

Global inter-organizational software development projects, including outsourcing, subcontracting or partnership relations, are becoming increasingly common [4, 5]. The fact that such projects cross both country and organizational borders makes them extremely challenging. Advice for outsourcing and acquiring large projects or modules with well-defined requirements can be found in literature (e.g. [6]). However, in many new product development software projects a lot of uncertainties exist and subcontractors or partners are needed long before these uncertainties can be resolved and the requirements thoroughly specified. Therefore, in such projects parties usually cannot receive clear requirement specifications at the beginning. Instead, close cooperation and communication between parties are required during the whole project. Problems often arise, since practices needed for collaborating and communicating across distances and organizations are not well established. Companies often underestimate the need for specific practices when collaborating across distances, and start global inter-organizational projects without first planning how to work together. This often leads to quite problematic situations. Most of the problems are related to communication difficulties (e.g. [1, 7]), which mainly arise due to geographical distance, which e.g. limits the number of face-to-face meetings [2, 5].

Current literature does not provide much help for managers planning their projects; only a few articles can be found presenting practices used in case projects (e.g. [1, 3, 4]). We believe that collecting successful practices, especially to support communication could help managers better plan and execute global inter-organizational software development projects.

In the research presented in this paper, we studied global, inter-organizational software projects, which used parallel development and had lots of uncertainties and interconnections between tasks. Since pure partnership projects were difficult to find, we concentrated on projects involving subcontractors, and focused on the structures and practices between the customer and the subcontractor(s) in parallel development situations. The projects chosen had also a global distribution aspect, either inside or between the companies. The focus of this study is illustrated in Figure 1.

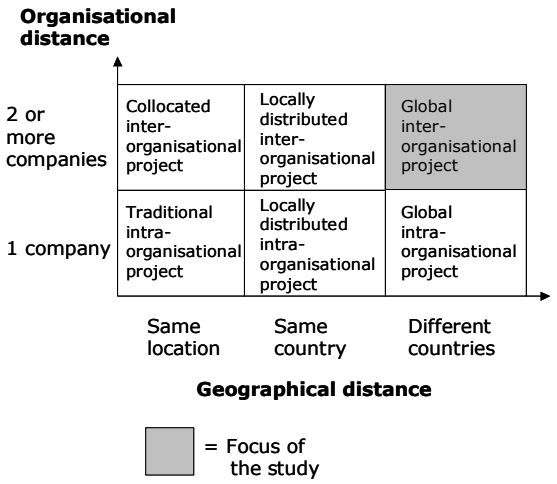


Figure 1. Project type classification

The aim of this paper is to present communication needs, structures that support communication, and communication practices collected from globally distributed inter-organizational projects.

2. Methodology

The research presented in the paper is based on a multiple-case study approach [10]. Seven successful, Finnish companies that develop software were chosen for the study. Three of the companies developed software products, one customer specific systems and three embedded systems. All of these companies used software subcontractors and were expected to be quite experienced in inter-organizational software development. All companies, except one, were large and well-known in Finland.

From every company we chose one globally and organizationally dispersed project that was studied closely. The chosen projects had sites or partners in two or three different countries. Four projects were distributed between continents, two of them between Europe and Asia, and two between Europe and North America. The rest three projects had a bit shorter inter-site distances, since all their sites were located in Europe.

We gathered data from 32 interviews. In each customer company we interviewed, if possible, both a partnership manager responsible for software subcontracting, and a process developer involved in subcontracting process development. From a chosen case project we interviewed project manager and, if possible, also one or more team members and a representative from the supplier company. We tape recorded all interviews, transcribed them and used Atlas/TI for grouping and analyzing the results.

3. Results and discussion

The most surprising result of our study was that companies did not have almost any clear structures and practices that were commonly used in all their inter-organizational software development projects. The practices we encountered were mainly project specific and created by trial and error. The structures and practices found and presented in this paper might seem quite basic. However, in our experience, they are often not implemented in real life projects, even though a lot of problems could be avoided by using them. Next, the observed communication needs, supporting structures, and communication practices are presented.

3.1. Communication needs

We identified four main types of communication needs: 1) problem solving, 2) informing and monitoring, 3) relationship building, and 4) decision-making and coordination. This classification is very close to classification presented by Stahl et al. [9] about communication in distributed product development. Our classification adds monitoring and relationship building. Monitoring is needed to give transparency of the project progress. Relationship building includes all kinds of social communication, which is especially important in a distributed project and therefore needs to be emphasized.

The identified communication needs where found in all projects studied. However, the importance of each need and the suitable communication practices depended on the type and phase of the project. The most important finding was that communication needed for problem solving was almost totally forgotten when planning projects. This type of communication was needed especially in projects involving a lot of uncertainties, since problems demanding communication just cannot be totally avoided.

The purpose of this communication need classification is to bring out communication needs that managers should take into account when planning their own distributed projects. Next, each communication need is briefly discussed.

3.1.1 Problem solving. Problem solving communication is easily forgotten in project planning, even though it is commonly needed in distributed projects, especially when facing a lot of uncertainties, e.g., concerning new technologies. If channels for problem solving communication are not agreed upon at the beginning of the project, it might take a long time before problems are solved and this delays the whole project. If there does not exist a suitable communication practice, project members will ask around, and hopefully find a person who can help them, but a lot of time and energy is lost.

3.1.2 Informing and monitoring. The customer normally remembers to monitor how the supplier's work is progressing, even though it is difficult if only time reports are used. However, the supplier's personnel and other distant sites would also like to get information about the progress of the whole project. This information would, besides helping personnel in distant sites to accomplish their tasks, motivate them, e.g., to keep-up the schedule, when they know why it is important. For a customer it is also very easy to forget to inform supplier about decisions and changes made, or new documents produced. The informing and monitoring should happen in both directions from the customer to the supplier and the other way around.

Besides informing, suppliers also expect feedback from their work, e.g., about the quality of the work. They would like to get comments also when they are doing something right, not only when things go wrong.

3.1.3 Relationship building. It is easier to communicate with a person that you have met at least once. Therefore, face-to-face meetings are crucial, especially in the beginning of the project. These meetings facilitate later electronic communication. Moreover, it is important that distant sites and companies have ”faces”. Otherwise they are easily forgotten and, e.g., their questions might not be regarded as important and urgent to answer.

Building a good relationship with suppliers requires also that they are treated more like partners and experts in their field, not like second class citizens. Normally, even suppliers want to do high quality work.

3.1.4 Decision making and coordination. Coordination and decision making is in a networked project concentrated to a network level steering group, the project mangers and the team level meetings. All these should take part of the responsibility. Define what kind of decisions each of them can make and how the whole project is informed about those decisions.

3.2. Supporting structures

We identified tree aspects that create supporting structures for an inter-organizational project: 1) a clear organizational structure with communicating roles, 2) partial synchronization of intra-organizational processes, and 3) structures for project level coordination (Table 1).

Table 1. Supporting structures

Structure	Actions	Support for communication
Organizational structure with communicating roles	- Create roles - Link communicating roles between organizations - Make the organization chart, with roles and contact info easily available	- Roles include communication requirements and identify which roles need to communicate with each other between companies - Roles and the organization chart make it easier to know whom to contact
Inter-organizational process	- Synchronize the main process milestones between organizations - Use iteration cycles of similar length and frequent builds	- Milestones synchronize communication - Several iteration cycles and builds create transparency, and facilitate follow-up and communication
Project level coordination	- Create a project level steering group with members from all organizations and sites - Arrange inter-organizational groups with weekly (teleconference) meetings	- Meetings (face-to-face / video- / teleconference) facilitate problem solving and decision making, they provide transparency and facilitate later electronic communication

When these structures are planned and implemented carefully and used constantly during a project, they support work and communication. Next, we presented the structures in more detail.

3.2.1 Clear organization structure with communicating roles. Creating roles, assigning the roles to team members and indicating which roles need to communicate with each other between companies, was a successful practice and it also stabilised the project structure. Defined roles make the inter-organizational project structure more clear to all participating team members and helps them to find the correct person to contact.

Each role description includes tasks to perform, decision-making rights, responsibilities, and identified communication contacts. The roles and their descriptions can be similar in all projects. Each project chooses the roles needed and names persons to the roles. At the beginning of a project it is easier to give team members roles than many separate tasks. Moreover, it is important that some roles have comparable roles at the customer’s and the supplier’s side. These roles take care of tasks demanding a lot of communication between companies.

At the management level in both companies there could be one named person, e.g., a subcontracting responsible who communicates with the other company’s corresponding role about future projects, prices, infrastructure needs, etc. At the project level, project managers communicate on a daily basis. At the team level, there are often experts on both sides who need to communicate with each other, e.g., persons responsible for related modules, software architects, etc.

Such a simple thing as an organization chart of the whole inter-organizational project was often missing. This kind of a chart makes it easier to find the correct persons to contact when questions emerge. A simple web page with information about project personnel, including names, roles, photos, and contact information can also help a lot.

3.2.2 Partial synchronization of intra-organizational processes. Our study showed that it is possible for both customer and supplier to use their own development processes in inter-organizational projects. Only the main

phases and milestones need to be synchronized between companies.

Many of the projects we studied had iteration cycles and builds. In some project phases even weekly builds were used. Frequent iterations and builds were noticed as a very suitable practice for distributed use, since they prevented different sites and partners from developing totally incompatible parts for long time periods. Frequent iteration cycles also bring partners transparency of the work done in a project. However, if all parties do not have the same interval between builds, problems will arise. Therefore, frequent iterations and builds with cycles of similar length in every company and site can be recommended.

3.2.3 Project level coordination. A project steering group at the inter-organizational level having members from all participating companies has been a good practice. It could meet, e.g., once a month and discuss important high level matters. This meeting can be either face-to-face or using video/teleconference.

3.3. Communication practices

The case companies did not agree upon communication practices at the beginning of their projects, a fact that caused problems later. Even many basic guides recommend doing a project communication plan first, e.g. the PMBOK® Guide [8], but that just did not seem to be a common practice in our case projects. Especially the need for problem solving communication was huge in the case projects. However, agreeing about it was often neglected partly because anticipating when and who would need it seemed to be difficult. Other important, but neglected, needs were relationship building and monitoring communication between distributed team members. These communication gaps limited transparency and caused, e.g., team members not always knowing whom to contact and made following the progress of the project difficult. Next, communication practices related to each four types of communication needs are presented (Table 2).

3.3.1 Problem solving. If a project does not have a suitable communication practice for problem solving, project members will ask around when they have questions, and might finally find a person who can answer their questions. After sometime one specific person, e.g. a system architect, might end up receiving a huge number of questions just because other team members have noticed that he or she can help them. However, answering questions and finding the answers takes time and this person's own duties suffer easily. This practice is not a very good one, but many projects use it.

Chat between developers was regarded as a very useful way of communicating in problem solving situations, since when chatting clarifying counter questions can be posed easily and chat session can be open all the time.

Discussion lists about specific technological areas were used in some larger projects and were found helpful, since know-how and experiences might exist somewhere in a large project.

Project wide mailing lists were used in smaller projects for asking questions. In an email questions asked need to be explained very carefully, otherwise readers do not understand questions and they have to send several mails asking clarifying questions before the question is understood correctly.

3.3.2 Informing and monitoring. Weekly meetings are a good arena in which to inform and monitor the project progress in both directions, from the customer to the supplier and the other way around. Team level weekly face-to-face meetings are often difficult to arrange in a distributed project, therefore, e.g., video- or teleconferences have been a very good alternative. Weekly meetings should be arranged among a group small enough, e.g., a project team or a subteam, to be efficient. It is important that everyone participates. The length of these meetings vary, half an hour can be enough. Inter-organisation representation is needed in these meetings, if there are dependencies across companies. The agenda could concentrate on tasks done, tasks to be done, problems and open issues. In a larger project subteam leaders could have their own meeting to get information about other teams and the whole project progress.

Table 2. Communication needs and practices identified

Communication need	Important	Practices
Problem solving	- Often neglected -> lack of answers delays the project - Organization chart and roles help to find the correct person to contact	- A person who "solves problems" - Mailbox for questions - Chat between developers - Discussion lists - Project wide mailing list with well explained questions
Informing and monitoring	- Follow-up in both directions, inform also the subcontractor - Customer should comment all points in the follow-up report	- Weekly meetings inside a subgroup (teleconference) - Follow-up reports including tasks done open questions, problems, and future outlook.
Relationship building	- Give a "face" to distant sites - All communication	- A common kick-off meting - Circulating meetings

	affects relationship building especially face-to-face meetings	or trainings - Planning / problem solving meetings
Decision making and coordination	- Define the correct forum for different type of decisions - Inform about decisions	- Network level steering group meetings - Weekly project/team level meetings

3.3.3 Relationship building. A common kick-off meeting for the whole project or a sub-project is often a good idea. If it is impossible to arrange due to large project size and long distances, you should arrange other face-to-face meetings for important communication link persons. For example, project architects or other key persons can go to the supplier's site to train them, or some supplier's key persons can be invited to the customer's site for training or a short collocated working period. When major problems arise they are best solved face-to-face.

3.3.4 Decision making and coordination. In an inter-organizational project coordination and decision making is concentrated to an inter-organizational steering group, project managers and weekly team meetings. All these should take part of the responsibility. Define what kind of decisions each of them can make and how the whole project is informed about those decisions.

4. Summary and conclusions

This paper presented communication needs, structures that support communication, and communication practices collected from globally distributed inter-organizational software development projects. The most surprising result was that case companies, even though successful in their field, did not have clear structures and practices that were commonly used in all inter-organizational projects. The practices encountered were mainly project specific and created by trial and error. The structures and practices found and presented in this paper might seem to be quite basic. However, in real life projects a lot of problems could probably be avoided by using them constantly. For example, the case companies did not agree upon communication practices in the beginning of their projects, which caused them problems later. Especially the need for problem solving communication was recognized to be huge in the case projects, but agreeing about it was often neglected. Other important, but neglected, needs were relationship building and project monitoring communication between distributed team members.

5. Future work

In the future we plan to extend this study and concentrate especially on communication, since it seems to be the biggest problem and is related to almost everything in global software development. We plan to study more projects and collect successful communication patterns and practices used in them. Furthermore, we plan to classify the communication patterns and practices according to projects type and communication needs. This collection should help managers choose suitable communication practices for their projects.

6. References

[1] R.D. Battin, R. Crocker, J. Kreidler, and K. Subramanian, "Leveraging resources in global software development", *IEEE Software,* March/April 2001, pp. 70-77.

[2] E. Carmel, and R. Agarwal, "Tactical approaches for alleviating distance in global software development*", IEEE Software*, March/April 2001, pp. 22-29.

[3] C. Ebert, and P. De Neve, "Surviving Global Software Development", *IEEE Software*, March/April 2001. pp. 62-69.

[4] R. Heeks, S. Krisna, B. Nicholsen, and S. Sahay, "Synching or sinking: global software outsourcing relationships", *IEEE Software*, March/April 2001, pp. 54-60.

[5] J. Herbsleb, A. Mockus, T. Finholt, and R. Grinter, "An Empirical Study of Global Software Development: Distance and Speed", *Proceedings of the 23rd International Conference on Software Engineering*, ICSE 2001. Pages, 81-90.

[6] IEEE Recommended practice for software acquisition, Institute of Electrical and Electronics Engineers, Inc. 1994

[7] A. Mockus, and J. Herbsleb, "Challenges of Global Software Development", *Proceedings of the Seventh International Software Metrics Symposium*, METRICS 2001. IEEE. Pages, 182-184.

[8] Project Management Institute, *A Guide to the Project Management Body of Knowledge (PMBOK® Guide),* 2000 Edition.

[9] J. Stahl, S. Killich, and H. Luczak, "Co-ordination, Communication, and Co-operation in Locally Distributed Product Development", *Proceedings of the 5th International Product Development Management Conference*, Como, Italy, May 25-26. 1998, pp. 947-959.

[10] R.K. Yin, *Case Study Research, Designs and Methods*, Sage Publications, Thousand Oaks, California, 1994.

Modeling Coordination Costs Due to Time Separation in Global Software Teams

J. Alberto Espinosa and Erran Carmel
American University, Washington DC
Kogod School of Business
alberto@american.edu; carmel@american.edu

ABSTRACT

Research to date has not attempted to model coordination in global software teams. We formulate a preliminary collaboration model for a dyad to help us understand the consequences of time separation. We first describe the model and its theoretical foundations and we then evaluate the model by simulating several thousand observations and running regression models to inspect the effect of different variables on coordination costs. We then make suggestions for further extension of the model to include more complex scenarios with multiple collaborators and fewer assumptions. Our evaluation shows that the consequences of time separation are complex and that we need to understand them well before we can make claims about coordination outcomes in larger software teams that are separated by time zones.

1. Introduction

New team configurations are increasingly carried out across global locations and time zones [1]. Such complex configurations have led to interest in the effect of distance on coordination in global software teams (GSTs). However, difficulties due to geographic dispersion often correlate with those of time zone differences. With some exceptions [2, 3], most research has not distinguished between the two. So, our objective in this paper is to present a model that represents coordination costs in which we distinguish distance from time separation. In this article we begin to lay the groundwork for a more rigorous inquiry on this topic. We introduce a mathematical model of interdependent work between two actors and measure the coordination costs of work that moves from one actor to another in all 4 conditions of the classic time-place matrix (Figure 1) [4]. We simulate different conditions using this model.

2. Time separation

Team members are separated by time when there are differences in working hours, time zones, and/or working

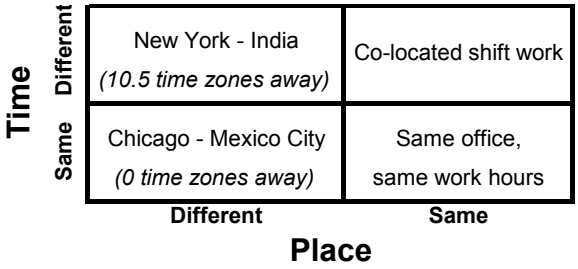


Figure 1: The time-place matrix with examples

rhythms that reduce the time available for same-time (i.e., synchronous) interaction [2, 3]. For example, teams separated on an east-west axis have fewer overlapping work hours than teams separated on a north-south axis [5], making it more difficult for the former to coordinate and communicate. Even co-located teams can be separated by time if their members work in different shifts.

Effective coordination is strongly influenced by communication, but human beings communicate more effectively when in close proximity [6]. While geographic distance affects coordination, coordination problems are often the result of time separation, which makes it difficult for members to interact synchronously. Even small time zone differences can bring surprising difficulties [7]. In the absence of face-to-face communications, GSTs have a menu of asynchronous and synchronous technologies to choose from. Media Richness theory [8] suggests that rich communication media (face-to-face, video conference) is more effective, but when GSTs are separated by time zones they are forced to use less rich, asynchronous media (e-mail, voice-mail).

The degree of task dependency plays a key role on coordination. When two team members with tightly coupled task dependencies collaborate, time zone differences can disrupt coordination. Not being able to pick up the phone and call other members can slow down a group's progress. Frequently, requests are not clear, requiring further communication. When team members are working face-to-face, the clarification may be nearly instantaneous. However, when team members are distant,

clarification may introduce delay. Furthermore, unclear communication exposes the team to "vulnerability costs" – e.g., misunderstandings, rework. On the other hand, time zone differences could actually be beneficial. For example, *Follow-the-sun* work [9] takes advantage of time zone differences to speed up project work. A team in New York can hand off work at the end of their day to be continued by team members in India, who can then continue the task while the New York staff sleep overnight. In fact, GSTs often adjust their work to overcome time zones differences – e.g., overlap work hour windows; liaisons whose work hours are the same as the other site; batch work delivered toward the end of the day; and periodic travel to interact face-to-face.

3. The model

3.1 Theoretical foundations

Coordination is "the management of dependencies" in a task. If work can be done independently, then there is no need to coordinate. Conversely, when two members carry out a task with tightly coupled dependencies, these dependencies need to be managed either by structuring task activities or by communicating [10-12]. However, coordination theory [13-15] thus far has not taken into account delays resulting from time zones differences. We focus in this article on coordination via communication and try to begin to fill this gap by formulating a simple dyad model, influenced by Malone [13]. However, our model departs from his in a number of accounts:

First, Malone's model analyzes different coordination structures based on different patterns of communication and decision-making that a set of actors can use. Our model employs only two actors that need to carry out a task with tightly coupled dependencies. Second, Malone's model assumes that actors employ their production capacities optimally and that different agents have different capacities to produce. We don't make such assumption in our model because there are only two actors, one (R) who requests a task from a task provider (P) because of a dependency (i.e., R cannot continue the task until P carries out the requested task). Third, Malone's model does not incorporate time and distance separation among actors, while we specifically model such time and distance separation.

3.2 Model Formulation

There are only two actors in our simple dyadic model, R and P. R has a workflow dependency with P. A single collaboration act in this context consists of the following: (1) R communicates a request to P; (2) P carries out the requested task; and (3) P communicates completion of the task to R. The model is constructed with *cost* as the dependent variable which, in turn, is composed of three costs: (1) Production costs – due to the actual time necessary to complete the task; (2) Coordination costs – due to delay; and (3) Vulnerability costs – due to unclear messages. A message can be unclear, with some probability, which can lead to one of two conditions: (a) a request for clarification, which results in an additional cost due to delay; and (b) rework, which leads to both additional production costs and a cost of further delays.

We developed different formulas for the 8 different possible task conditions (2x2x2), depending on whether the work time overlap occurs at the beginning or end of the requestor's work day; whether the request arrives during or outside of the overlapping time; and whether the task is completed and notified during or outside of the overlapping time. For simplicity of illustration (the formulas don't change too much), we only analyze the model with overlapping time occurring at the end of R's work day. We model time separation based on an overlap index [5, 16] between the two actors. In Appendix A we present the resulting conditions and formulas in detail.

3.3 Assumptions

We made a number of simplifying assumptions to the model in order to test its robustness, which we can later relax to evaluate more complex collaboration models: (1) A task is composed of individual and shared portions. Actors are equally capable of doing their individual tasks. The shared portions contain dependencies that are coordinated via communication; (2) Coordination failures are due to unclear communications, creating vulnerability costs (i.e., further communication to clarify the message or re-work); (3) The probability of unclear messages increases as the richness of the communication medium used decreases. Only one clarification message is necessary to resolve unclear messages; (4) The task is a software task and the production object is digital and it can be sent across a network in 0 time units. Similarly, messages sent arrive instantly; (5) There is only one synchronous and one asynchronous link between R and P; (6) The task is high priority and time constrained; (7) Non face-to-face communication is conducted electronically, and when working hours overlap actors prefer to communicate synchronously (e.g., telephone, video conference); they communicate asynchronously (e.g., e-mail, shared databases) otherwise; (8) All tasks requested by R are immediately accepted and carried out competently by P and there is no parallel multi-tasking. Once P has full information about the requested task, P's production costs are the same regardless of time or distance separation; (9) Time is measured from R's perspective. If P is processing a task during R's non-work hours it has no time delay consequences for R.

	Coordination Costs				Vulnerability Costs			
	Main Effects		+ Interaction		Main Effects		+ Interaction	
Variable	Coefficient	P-Value	Coefficient	P-Value	Coefficient	P-Value	Coefficient	P-Value
Constant	**-390.64**	**<0.001**	**-409.64**	**<0.001**	**-55.18**	**<0.001**	**-69.65**	**<0.001**
Request Time	**-353.81**	**<0.001**	**-49.51**	**<0.001**	**-133.62**	**<0.001**	**-27.13**	**<0.001**
Task Duration	**721.50**	**<0.001**	**942.53**	**<0.001**	**71.88**	**<0.001**	9.80	0.143
Overlap Index	**-159.64**	**<0.001**	-2.51	0.492	**-135.13**	**<0.001**	0.21	0.923
Distributed	**594.14**	**<0.001**	**600.09**	**<0.001**	**24.71**	**<0.001**	**26.82**	**<0.001**
Time Separated	**208.39**	**<0.001**	**205.34**	**<0.001**	**70.91**	**<0.001**	**75.21**	**<0.001**
Distributed & Time Separated	**749.31**	**<0.001**	**759.58**	**<0.001**	**123.90**	**<0.001**	**122.93**	**<0.001**
ReqTime x TskDur			**-321.26**	**<0.001**			**-26.33**	**0.020**
ReqTime x Overlap			**471.20**	**<0.001**			**255.99**	**<0.001**
TskDur x Overlap			**496.89**	**<0.001**			**-28.37**	**0.009**
ReqTime x Distr			5.87	0.279			4.79	0.136
ReqTime x TimeSep			**-610.74**	**<0.001**			**-170.78**	**<0.001**
ReqTime x Distr&Time			**-608.48**	**<0.001**			**-257.61**	**<0.001**
TskDur x Distr			1.52	0.922			3.87	0.676
TskDur x TimeSep			**-416.15**	**<0.001**			**87.36**	**<0.001**
TskDur x Distr&Time			**-454.54**	**<0.001**			**154.38**	**<0.001**
Overlap x Distr			*9.32*	*0.071*			4.14	0.177
Overlap x TimeSep			**-366.62**	**<0.001**			**-235.66**	**<0.001**
Overlap x Distr&Time			**-262.92**	**<0.001**			**-305.90**	**<0.001**
R-sq	0.854		0.974		0.571		0.901	
R-sq Change			0.120				0.330	
R-sq Change P-Value			<0.001				<0.001	

Table 1: Regression Analysis Results

4. Model evaluation

We evaluated the robustness of the model with a simulation of 11,000 observations and then exploring the effect of the timing of requests, task duration, and time overlap on coordination and vulnerability costs when the overlap occurs at the end of the requestor's work day. The request time (Rt) variable was generated randomly from a uniform distribution (0,1), with 1 being a full work day. The task duration (Tt) variable was also generated randomly from a normal distribution with an average of 0.25 (1/4 of a work day) and a standard deviation of 0.1. We fixed all other parameters as follows (see Appendix A): Cla=$100 and Cls=$500 per day; Cma=$10 and Cms=$50 per message; Cd=$1,000 and Cp=$1,000 per day. These costs are arbitrary, but they serve the purpose of helping illustrate and evaluate the model. Further evaluations of this model will incorporate variable costs.

The probability that a request was unclear was fixed at 10%, 30%, 50% and 70% for the four conditions, respectively, face-to-face, distributed, time-separated and time/distance separated. These probabilities are arbitrary but based on the expectation that, as the richness of the communication media diminishes the probability of unclear messages increases. The probability differences are purposefully wide to make their effects on coordination costs more noticeable. Also, there is a probability of 30% that unclear messages will lead to re-work and 70% that it will simply lead to a request for further clarification with no re-work. If re-work is necessary, it is assumed that, on average, 30% of the work completed will have to be redone, thus increasing production costs. Finally, we assume one request per day.

Regression results from Ordinary Least Squares models (Table 1) suggest that the model is robust and that it behaves as expected. Both models were run first including only main effects and then adding interaction variables. The interaction variables significantly increased the explained variance (R^2) in both models, suggesting that these interaction terms are important. The base models with only main effects yielded intuitive and similar results. Both, coordination and vulnerability costs increase with longer tasks and with time and/or distance separation. Both costs decrease when requests come later in the day (i.e., closer to overlapping hours) and when there are more overlapping hours in the day. The coefficients are larger in absolute value for coordination costs than for vulnerability costs, but this difference will change as we change cost parameters in the future.

Most main effects remained significant and retained their signs when the interaction variables were added, with a few exceptions. The main effect of overlapping hours became non-significant in both models and the main effect of task duration became non significant in the vulnerability costs model. The sign and significance levels of the interaction coefficients for the overlap variable indicate that the amount of overlapping time has a significant effect for teams that are separated by time or by time-and-distance, as one would expect, but, naturally, it does not have an effect on face-to-face and distributed-same time conditions. This is an expected result since these teams have full overlap in their working hours.

The negative interaction between task request time and time separation suggest, intuitively, that issuing task requests closer to overlapping working hours reduces coordination costs when actors are separated by time. Interestingly, task duration increases coordination costs, as expected, but this effect diminishes with time separation, because the task provider T can work during the requestor R's off work hours. This is consistent with the benefits of *"follow-the-sun"* noted in Section 2. The negative interaction between request time and task duration suggest that issuing task requests closer to overlap time reduces coordination and vulnerability costs, but more so for tasks of longer duration. On the other hand, the positive interaction between request time and work overlap time suggests that the benefits of making task requests later in the day are diminished as the work overlap hours increase. In time-separated work contexts, request timing is critical in reducing coordination and vulnerability costs, but this becomes less important with less time separation. Finally, the interaction between task duration and overlap time was positive for coordination costs and negative for vulnerability costs. This suggests that coordination costs increase with task duration, especially when there is less time separation (i.e., more overlap), but this is offset by lower vulnerability costs because it is less costly to clarify miscommunication when there is less time separation.

5. Discussion and future research

Coordination and vulnerability costs in time-separated contexts are affected by the time of the day when a task is requested but this effect diminishes as overlapping work time increases. This and the other results discussed in the prior section suggest that the model we have formulated in this paper is robust. Collaborations in which more than two actors are separated by time are much more complex than the model we have presented here. But the robustness of our dyadic model gives us confidence that this model can be expanded to more complex coordination structures with more variable cost and operational parameters. For example, delay costs are higher where time-to-market is critical. On the other hand, production costs may be much higher in situations where software production requires expensive resources (e.g., sophisticated testing labs, scarce expertise). Other interesting manipulations include: giving actors a choice of communication technologies, each with different costs, and then evaluate the tradeoffs among coordination, vulnerability and communication costs; assessing new collaboration tools by reducing the probability of unclear messages due to better communication effectiveness; actors could also reflect different production costs (as is typical today with offshore work). In sum, we plan to expand our model by progressively relaxing assumptions.

6. References

[1] Orlikowski, W., *Knowing in Practice: Enacting a Collective Capability in Distributed Organizing.* Organization Science, 2002, 13(3): p. 249-273.

[2] Espinosa, J.A., et al., *Team Boundary Issues Across Multiple Global Firms.* Journal of Management Information Systems, Spring 2003, 19(4).

[3] Watson-Manheim, M.B., K. Crowston, and K.M. Chudoba. *A New Perspective on Virtual: Analyzing Discontinuities in the Work Environment*. in *35th. Hawaiian International Conference on System Sciences,* 2002, Big Island, Hawaii: IEEE.

[4] Bullen, C. and J. Bennett, *Groupware in Practice: An Interpretation of Work Experiences*, in *Groupware and Computer-Supported Cooperative Work: Assisting Human-Human Collaboration*, R. Baecker, Editor, 1993, Morgan Kaufman Publishers: CA. p. 69-84.

[5] O'Leary, M.B. and J.N. Cummings. *The Spatial, Temporal, and Configurational Characteristics of Geographic Dispersion in Teams*. in *Presented at the Academy of Management Conference,* 2002, Denver, Co.

[6] Kiesler, S. and J.N. Cummings, *What Do We Know About Proximity in Work Groups? A Legacy of Research on Physical Distance*, in *Distributed Work*, P. Hinds and S. Kiesler, Editors, 2002, MIT Press: Cambridge, MA. p. 57-80.

[7] Grinter, R.E., J.D. Herbsleb, and D.E. Perry. *The Geography of Coordination: Dealing with Distance in R&D Work*. in *International ACM SIGGROUP Conference on Supporting Group Work (Group 99),* 1999, Phoenix, Arizona: ACM Press.

[8] Daft, R. and R. Lengel, *Organizational Information Requirements, Media Richness and Structural Design.* Management Science, 1986, 32(5).

[9] Carmel, E., *Global Software Teams,* 1999, Upper Saddle River, NJ: Prentice Hall.

[10] March, J. and H. Simon, *Organizations,* 1958, John Wiley and Sons.

[11] Thompson, J., *Organizations in Action,* 1967, McGraw-Hill.

[12] VanDeVen, A.H., L.A. Delbecq, and R.J. Koenig, *Determinants of Coordination Modes Within Organizations.* American Sociological Review, 1976, 41(April), p. 322-338.

[13] Malone, T., *Modeling Coordination in Organizations and Markets.* Management Science, 1987, 33(10), p. 1317-1332.

[14] Malone, T. and K. Crowston. *What is Coordination Theory and How Can it Help Design Cooperative Work Systems*. in *Computer Supported Collaborative Work,* 1990, ACM Press.

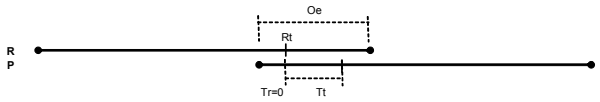
[15] Malone, T. and K. Crowston, *The Interdisciplinary Study of Coordination.* ACM Computing Surveys, 1994, 26(1), p. 87-119.

[16] O'Leary, M.B. *Varieties of Virtuality: Separate but not Equally*. in *FIU Workshop on Distributed Work and Virtuality,* 2001, Miami, Fla.

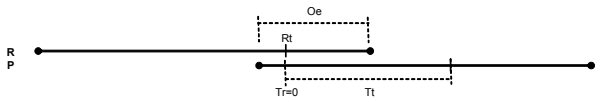
Appendix A: Model Variables and Formulas

When overlapping working hours are at the end of R's workday

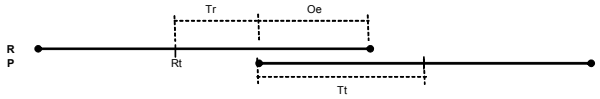
1. R(Oe)C(Oe) Request comes during overlapping hours and is also finished during overlapping hours



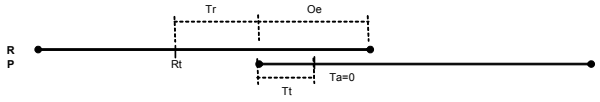
2. R(Oe)C(after-Oe) Request comes during overlapping hours, but is finished after overlapping hours



3. R(bef-Oe)C(after-Oe) Request comes before overlapping hours and is finished after overlapping hours



4. R(bef-Oe)C(Oe) Request comes before overlapping hours and is finished during overlapping hours



Variables:

λ = number of tasks per day that R requests from P
Cp = P's production cost (per time unit)
Cd = R's cost of delay (waiting) for task to be processed and getting confirmation
Cl = Cls + Cla = Cost per day of maintaining a (synch + asynch) connection between R and P
Cm = Cms + Cma = Cost of sending a (synch + asynch) message from R to P or viceversa
Ob = Overlap index (0-1), proportion of overlap time in work hours, at beginning of R's work day
Oe = Overlap index (0-1), proportion of overlap time in work hours, at end of R's work day
Rt = Time when R requested task from P, expresed as a fraction (0-1) of time from the start of the work day
Tt = Task duration time or time it takes P to complete the requested task, as a fraction (0-1) of the work day
Td = Delay time between R's request and P's acknowledgement, measured from R's perspective as a fraction
Pn = Probability that a message is not clear; varies with collaboration mode: Pn(TP)>Pn(T)>Pn(P)>Pn(O)
where, TP=separated by time and place, T=by time only, P=by place only, O=face-to-face

O. SameTime-SamePlace -- Co-located

Assumptions: Cl = 0; Cm = 0; Tr=0; Ta=0; Pn(O) is very low
Production Costs = λCpTt (same across all 4 collaboration modes O, P, T and TP
Coordination Costs = λTtCd
Vulnerability Costs = λPn(O)[(Pr)(RwCpTt)+(1-Pr)(0+0)(Cd)] = λPn(O)[(Pr)(RwCpTt)]; Pn(O) is very low, on average

P. Same Time-Different Place -- Distributed Synchronous (separated by place)

Assumptions:
Cla = 0 or Cl = Cls; Cma = 0 or Cm = Cms; T2=0; T5=0
Pn(TP)>Pn(P)>Pn(O); probability of unclear messages is higher when separated by more boundaries
Production Costs = λCpTt
Coordination Costs = Cls + 2λCms + λTtCd
Vulnerability Costs = λPn(P)[(Pr)(RwCpTt)+(Pf)(2Cms+0+0)] = λPn(P)[(Pr)(RwCpTt)+(1-Pr)(2Cms)]

T and TP. Different Time-Same Place/Different Place -- Asynchronous (Co-Located and Distributed)
With Overlap at the End of R's workday (Oe)

1. R(Oe)C(Oe) Request comes during overlapping hours and is also finished during overlapping hours

(T) Td = Tt; Tr=0; Ta=0; Cm=0 -- (PT) same, except that ***Cm=Cms***
Coordination Costs (T) = Cla + λTtCd -- (PT) = Cla + ***Cls*** + ***2λCms*** + λTtCd
Vulnerability Costs (T) = λOePn(O)(Pr)(RwCpTt) -- (PT) = λ***Pn(P)***(Pr)(RwCpTt)

2. R(Oe)C(after-Oe) Request comes during overlapping hours, but is finished after overlapping hours

Td = 1-Rt; Tr=0; Ta=1-Rt
(T) Cm=Cma (for both request and acknowledgement) -- (PT) Cm = Cma + ***Cms***
Coordination Costs (T) = Cla +λCma + λ(1-Rt)Cd
(PT) = Cla + ***Cls*** + λ(Cma+***Cms***) + λ(1-Rt)Cd
Vulnerability Costs (T) = λPn(O)[(Pr)RwCpTt+(1-Pr)(Cma+(1-Rt)Cd)
(PT) = λ***Pn(P)***[(Pr)RwCpTt+(1-Pr)(Cma+***Cms***+(1-Rt)Cd)

3. R(bef-Oe)C(after-Oe) Request comes before overlapping hours and is finished after overlapping hours

Td = 1-Rt; Tr = 1-Rt-Oe; Ta=0
(T) Cm=Cma (for both request and acknowledgement) -- (PT) Cm = Cma + ***Cms***
Coordination Costs (T) = Cla +2λCma + λ(1-Rt)Cd
(PT) = Cla + ***Cls*** + 2λCma + λ(1-Rt)Cd
Vulnerability Costs (T) = λPn(T)[(Pr)RwCpTt+(1-Pr)(2Cma+(1-Rt)Cd)
(PT) = λ***Pn(PT)***[(Pr)RwCpTt+(1-Pr)(2Cma+(1-Rt)Cd)

4. R(bef-Oe)C(Oe) Request comes before overlapping hours and is finished during overlapping hours

Td = 1-Rt-Oe+Tt; Tr = 1-Rt-Oe; Ta=0
(T) Cm=Cma (for request only) -- (PT) Cm=Cma+***Cms***
Coordination Costs (T) = Cla +λCma + λ(1-Rt-Oe+Tt)Cd
(PT) = Cla + ***Cls*** + λ(Cma+***Cms***) + λ(1-Rt-Oe+Tt)Cd
Vulnerability Costs (T) = λPn(T)[(Pr)RwCpTt+(1-Pr)(Cma+(1-Rt-Oe)Cd)]
(PT) = λ***Pn(PT)***[(Pr)RwCpTt+(1-Pr)(Cma+***Cms***+(1-Rt-Oe)Cd)]

Building Trust in Global Inter-Organizational Software Development Projects: Problems and Practices

Jarkko Pyysiäinen
Helsinki University of Technology
Software Business and Engineering Institute
P.O.B. 9600, FIN-02015 HUT, Finland
Jarkko.Pyysiainen@hut.fi

Abstract

This paper explores problems and potential practices for trust building in global inter-organizational software development networks. The concept and traditional sources of trust are briefly reviewed, and the special problems on trust building in networks are analyzed on the basis of the theoretical framework. Our empirical findings from nine global software development networks show that such networks are facing problems because the traditional sources of trust do not exist in networked conditions. In such networks, trust may emerge occasionally, but maintaining it seems especially challenging. Consequently, building and maintaining trust in globally and organizationally dispersed networks seems to require supportive practices that compensate the deficient sources of trust. On the basis of our empirical data, some successful practices in trust building are outlined.

1. Introduction

Global inter-organizational networks have become increasingly popular in software development [3]. Such networks may include, e.g., several subcontractors or partners working concurrently with customers across distances and relying primarily on communication technologies instead of face-to-face meetings. Work across teams and companies is rather interdependent than independent and the need to orchestrate the work across the whole network is often great. This kind of coordination of work in networks creates new challenges and demands: coordination is simply not possible based on such traditional features as direct face-to-face feedback, common experiences, similarity of backgrounds and co-located decision making [4]. Instead, alternative ways for facilitating cooperation and communication must be utilized. In the field of organizational behavior a growing attention has been paid to the role of trust in such processes: a growing body of literature demonstrates the important benefits of trust for organizations and their members [1,5,8]. Even more importantly, trust is seen as a necessary element in facilitating the functioning of networked organizations [4,7,11] and global software outsourcing relationships [2,9].

While on the one hand trust building seems to be a promising mechanism for overcoming many difficulties related to global software development, it may on the other hand be precisely the virtual and global contexts that constrain the development of trust between companies and teams [e.g. 4]. Lack of face-to-face interaction and informal communication seem especially troubling.

The aim of this paper is to first present a theoretically motivated empirical analysis of the problems encountered in trust building in nine Finnish, global and inter-organizational software development networks. Second, on the basis of the data successful practices in overcoming these problems in trust building are outlined.

2. Theoretical framework

2.1. Concept and sources of trust

For the purposes of this study the concept of trust may be best understood as a relationship between parties, not as a property of an individual: trust prevails if each of the interacting parties acknowledges the right of the other parties to assess the competence and the intentions of their acts [11]. Trust prevails, if parties after this kind of assessment are willing to be vulnerable to – or cooperate with – each other based on the belief that the other is competent, open, concerned and reliable [8]. In this sense trustworthiness of an individual may be a source of trust for others, but trust is realized only in action. Without shared experiences trust is not likely to survive.

This type of assessment of trust between parties also requires a dialogical, negotiating mode of communication. In addition, grounds for this kind of trusting orientation are likely to develop in organizational contexts, where work is evaluated and control located at the level of the joint project, not at the level of individual contributions. [11] When these antecedents are evaluated from the perspective of global software development networks, it seems that shared experiences at the network level may be rare, but that a negotiating mode of communication and evaluation of work at the level of a joint project are characteristic to some networks.

According to Kramer [5] traditional sources of trust within organizations can be summarized as follows:

Dispositional trust: Predispositions to trust or distrust others tends to be correlated with other personal orientations and styles. Certain features in the behavior of others become associated with stable expectations and it is possible to extrapolate from earlier trust-related experiences.

History-based trust: Trust thickens or thins as a function of cumulative interaction. Individuals' judgments about others' trustworthiness are partly anchored on a priori expectations about others' behavior, and these expectations change as subsequent experience either validates or discredits them. Reciprocity in exchange relations enhances trust while violation of reciprocity erodes it.

Third parties as distributors: Third parties are important because of their ability to diffuse trust-relevant information via informal communication and gossip. On the basis of this kind of mediated information it becomes possible to transfer expectations of existing embedded relationships to newly formed ones.

Category-based trust: Common categories function as vehicles for perceiving common identities and common goals. A shared membership in a salient category (woman, researcher) can provide basis for presumptive trust and a sense of familiarity. Membership in a category is associated with a tendency to attribute positive characteristics to other ingroup members.

Role-based trust: often it is not so much the person in the role that is trusted but the system of expertise that produces and maintains role-appropriate behavior. In this sense trust can be seen also as depending on the system that is represented in the role – roles lessen the need for repeatedly negotiating trust when interacting with others. A related issue here is that serious failures of cooperation can occur if novel situations break down role-based habits e.g. in organizational crisis

Rule-based trust: trust based on internalized rules rests not on an explicit contract but on socialization into the structure and practices of the organization. If socialization is high and common principles are well internalized, mutual trust can acquire a taken-for-granted quality.

2.2. Developmental stages of trust

Luhmann [6] has presented a useful distinction of the antecedent conditions for the development of trust. The development of trust can be seen as depending upon two previous stages, namely familiarity and confidence. All three stages represent qualitatively different modes of asserting expectations towards the behavior of others. Familiarity is the first necessary condition for the development of trust, because no stable expectations can be formed towards the strange, which remains mentally uncontrollable. The second condition is the stage of confidence, which depends in turn upon a certain amount of familiarity of the target. Confidence is based upon expectations of normal practices, standard operations and definite norms that are supported by sanctions. Confident expectations mean that no alternative ways of doing things are actively thought about; instead, a certain scenario is taken for granted. Finally, trust is the stage where open negotiation and active search for alternatives become possible, but only if the stages of familiarity and confidence are fulfilled and do not let the trusting parties down. [6, also 10]

3. Data and methods

The data was collected in an interview study that aimed at exploring working practices and problems in global inter-organizational software development projects. The focus of the study was on networked projects that involve at least two companies: a customer and a supplier. Nevertheless, most networks studied involved more than two organizational parties.

The data was collected from nine distributed software development networks. In each case the customer company was Finnish, and all of them, except one, were large and nationally well known. Altogether eight customers, five subcontractors and ten projects were studied. The data consists of taped and transcribed thematic interviews of the project personnel and managers (N=44).

Of the networks, four were developing software products, two developed bespoke systems, and three developed embedded systems.

4. Results and discussion

On the basis of the interviews it seems that most of the major problems in networked projects are related to communication and the arrangement of cooperation between companies. Due to problems in these areas,

projects were easily delayed or even failed. In our interview study we noticed that companies were very interested in networked product and software development, but because of the fear of possible problems and the lack of concrete working procedures they were sometimes hesitant to start that kind of projects.

Section 4.1 presents the results from the analysis of the problems encountered in trust building. The classification of problems is based on the framework of potential sources of trust presented in section 2.1.

Section 4.2 outlines the practices found in interviews that proved to be useful in tackling the problems. These successful practices are classified according to the developmental stages of trust described in section 2.2.

4.1. Problems in trust building

4.1.1. Personal dispositions. The receiver did not always know how to interpret the messages (e.g. e-mail) from senders in different companies, because the personalities and interaction styles of parties remained ambiguous. Some persons preferred short-worded mails that went straight to the point, but if the receivers had never really got acquainted with the sender, they easily thought that the sender was not satisfied with their work or did not respect their ideas. Situations got worse, if receivers interpreted the messages as including unwarranted commanding functions, when after all the case in point was that the personal ways of expressing oneself in mediated communication differed so significantly.

4.1.2. Common history. Because of the temporary nature of software development networks, the exchange of background information was often not sufficient, especially if companies had not properly planned and documented what kind of information would be required in each development phase. The companies often forgot to discuss the available documentation and whom to contact in specific issues. If clear organizational charts and face-to-face meetings were lacking, people did not get to know each other's roles, responsibilities and competences. Thus people hesitated to spontaneously give and ask for help. In some cases customers made impossible demands to subcontractors, who lacked the required background information; as the subcontractor then presented their own solutions to the tasks, the customer claimed that the subcontractor was incompetent and had to redo the whole task. These kinds of violations of reciprocity led easily to a decline in trusting attitudes and motivation.

4.1.3. Mediating third parties. In collocated projects tacit knowledge, positive experiences and reasons for successes and failures are spread automatically in various informal occasions and conversations increasing the awareness of the progress of the project and of possible sources of risk and opportunity. However, in the case of networks spontaneous transfer of knowledge via third parties was often blocked, because no mediating link persons between companies were available. Parties remained ignorant about reasons for delays in deliveries and testing. Similarly, the causes for changes and some troubling bugs remained unclear. This kind of uncertainty aroused suspicions about the positive intentions and motives of other parties, ending up in unwarranted accusations towards remote teams. The absence of mediating third parties also manifested as question overloads. When people did not know whom to ask for help, it was typically one salient key person (e.g. a system architect) who received a huge load of questions. In such cases the work of the contacted link person suffered heavily, and crucial information was left untransmitted.

4.1.4. Shared category membership. One obvious problem was to create a sense of togetherness at the network level. In many cases, people from different companies did not actually feel that they were working towards a common goal. Rather, the sub-goals of each site tended to conflict with commitment to a common goal. Uncertainty existed concerning the limits of confidential information that should be withheld from other companies. When in doubt, team members preferred to withhold all information and not to exchange ideas that would have helped the progress at the whole network level. Further, when, e.g., subcontractors did not get feedback on the quality of their work and could not perceive how their contributions affected the progress of the whole project, it became even more difficult for them to identify with a common goal. As a consequence, the commitment of the subcontractor was weakened and in a couple of cases collapsed totally. Problems were raised by the mere dissimilarity of the deliverables coming from "alien" sites: in one case testers who encountered deficiencies in code coming from another site intentionally made impossible change requests and finally refused to test the code from that particular site.

4.1.5. Predictable role behavior. No one of the studied networks had established role definitions at the level of the entire network. Companies may have had indicated roles in their internal processes, but at the network level a clear prediction of the behaviour of other people on the basis of their roles was not possible. Difficulties were also related, e.g., to the contested justification of decisions, because the roles did not suggest who had the right to decide on issues, especially at lower levels in the organization. Sometimes developers made confusing changes to the core modules of the product, even though that kind of tasks were not ascribed to them. At times information passed over crucial persons (e.g. an architect) because task dependencies were not reflected in the agreed communication relations between roles.

4.1.6. Internalized common rules. It was quite surprising that such basic issues as common terms to be used in the development process were not always clearly stated. Similarly, some parties were for a long time ignorant of the true nature of the development process and development cycle in other companies. The lack of binding principles manifested, e.g., as lacking communication and change-request protocols. There was only seldom agreed reaction times to received mails, which caused confusions in communication. An unclear threshold for changes caused unnecessary and overlapping changes.

4.1.7. Discussion. On the basis of the presented classification of problems in trust building it seems that the traditional sources of trust cannot properly function in networked conditions. There are simply very few natural sources of trust that would facilitate the cooperation between parties. Especially commanding communicative acts (e.g. giving orders, asking for help, delegating tasks) between parties may lead to difficult situations in absence of underlying trusting attitudes. In this sense, the building of trust in global software development is something that must be intentionally arranged and taken care of.

4.2. Practices for successful trust building

The interviewed companies had managed to establish some practices that proved to be successful in building trust. The practices were rather unsystematically implemented and, accordingly, the results for trust building were not optimal.

4.2.1. Practices supporting the development of familiarity. A common kick-off meeting in the beginning of the project was a successful way to create initial familiarity between the members. Successful kick-off meetings did not have to include all the members participating in a single event; instead there could be several kick-off meetings at different times associated to, e.g., interdependent sub-projects. The important thing was that those individuals who would be changing information or cooperating with each other, would get to know each other at the beginning of the project.

In some cases cooperation problems were solved only after one party (e.g. the customer) had visited the other party's (e.g. the subcontractor's) premises and got acquainted with the working process and the nature of the encountered problems. In this sense collocated reviews, training occasions and joint planning meetings were invaluable for both parties, because in these meetings great amounts of background information and tacit knowledge could be exchanged.

Other practices that facilitated the development of familiarity included the establishment and updating of an organizational chart for all the members to see, e.g., on the project web pages. A useful organizational chart included information about project members in all companies, e.g., names, roles, pictures, and contact information. Additionally, salient informing about the project goals in the beginning of the project – e.g., in the form of project plans or start-up meetings – also provided a ground for forming a common identity and vision of the project.

Together these practices function to create a basis for the first step in trust building, namely familiarity. Some of these practices may seem quite trivial, but precisely because of the lack of traditional sources of trust, a proper implementation of the aforementioned practices proved to be crucial in networked conditions.

4.2.2. Practices supporting the development of confidence. A tricky issue regarding the development of confidence in the studied cases was to establish binding and clear inter-organizational processes and stabilizing structures across the network. A useful practice in the beginning of the project was a collocated training of the development process to be used: things progressed more fluently after the exact meaning of the terms to be used in the development process was clearly agreed on between parties. Issues that were informed only in writing were often improperly internalized. Another related step was to give detailed feedback of the encountered deficiencies in code. If illustrative feedback of the deficiencies was given, perhaps face-to-face by a liaison person, the initial failures turned out to be strengths in some cases. Internalization of the coding and documentation principles was better when trained after some insufficient trials.

Another major area in need of stabilizing practices was the arrangement of inter-organizational communication. A promising practice that was being developed in a couple of cases was the allocation of, e.g., task descriptions, decision-making rights and responsibilities to specific roles. These roles could be linked to matching roles in other companies. This way, the interdependencies between tasks and the exchange of crucial information became more clearly structured, providing a more predictable environment for the development work and helping people find the correct persons in important issues, preventing effectively both information overloads and information blocks. Further, clearly agreed reaction times to received e-mails, messages and questions decreased confusion between parties.

When applied systematically, these practices created expectations of normal routines, standard operations and definite norms that guaranteed that the process kept on going solidly even in the face of problems and unexpected changes. In other words, when properly implemented, the practices established confidence, which

can be seen as the second step on the way to successful trust building.

4.2.3. Practices supporting the maintenance of trust. In the cases studied, a successful practice in maintaining a trusting orientation was proper informing about the project progress to all contributing parties. Mere follow-up based on reported working hours did not suffice, since the parties wanted to know how their contributions affected the progress of the whole project. Instead, feedback about the quality and concrete contributions of the deliverables was appreciated. When the delivering parties could recognize, what had gone especially well and what were the reasons for possible dissatisfaction, their working morale and motivation remained high. Also, the exchange of experiences in the development work across team and company borders (e.g. chat, phone, e-mail lists) helped to create a common understanding and lowered the threshold for spontaneous offers of helping acts. When, e.g., developers learned about the circumstances and difficulties on other sites, their suspicions towards the alien deliverables were lowered. However, without a supporting management policy an open atmosphere was not likely to develop.

4.2.4. Discussion. When the identified practices are viewed from the perspective of trust building, practices providing familiarity work as compensators for the first four sources of trust by providing knowledge of personal dispositions, helping to build a common history, introducing mediating third parties and identifying salient memberships in shared categories. Second, the practices that establish confidence compensate the most obvious deficiencies in the last two source categories by facilitating predictable role behaviour and the internalization of common rules. Third, if open and negotiating communication prevail and the parties are willing to inform each other about the project progress while also accepting the right to assess the contributions of each other, a basis for maintaining a trusting orientation is laid down.

5. Conclusions and future research

On the basis of both literature and the empirical findings it seems that the conditions associated with distributed global software development are very demanding, because most of the traditional sources of trust don't exist in networked conditions. Consequently, trust in networks may emerge occasionally, but maintaining it is especially challenging. However, by concentrating properly on the practices that support the development of antecedent conditions of trust – familiarity and confidence – in the beginning of projects, trust building may bring out successful results also in the case of geographically and organizationally dispersed networks.

The paper provides only initial outlines of the nature of problems and possible solutions in trust building. In the future our aim is to analyze the nature of the problems more thoroughly, e.g., by studying the typical sources of problems in different kinds of software development project types. We believe that when the challenges of cooperation in different project types are understood better, it also becomes possible to formulate adequate supporting practices for different kinds of networks and projects. Nevertheless, since it seems that the traditional sources of trust do not exist in networked conditions, it is probable that all types of global inter-organizational software development projects will benefit from some basic and general practices that support trust building.

6. References

[1] Creed, D.W.E., and R.E. Miles, "Trust in organizations", in Kramer, R.M., and T.R Tyler, (eds.) *Trust in organizations*, Thousand Oaks, Sage, 1996, pp. 16-38.

[2] Heeks, R., S. Krishna, B. Nicholsen, and S. Sahay, "Synching or sinking: Global software outsourcing relationships", *IEEE Software*, March/April, 2001.

[3] Herbsleb, J., A. Mockus, T. Finholt, and R. Grinter, "An empirical study of global software development: Distance and speed", *Proceedings of the 23rd International Conference on Software Engineering*, ICSE 2001, pp. 81-90.

[4] Jarvenpaa, S.L., K. Knoll, and D.E. Leidner, "Is Anybody Out There? Antecedents of Trust in Global Virtual Teams", *Journal of Management Information Systems*, Vol. 14, No. 4, 1998, pp. 29-64.

[5] Kramer, R.M., "Trust and distrust in organizations: Emerging perspectives, enduring questions", *Annual Review of Psychology*, Vol. 50, 1999, pp. 569-597.

[6] Luhmann, N., "Familiarity, confidence, trust: problems and alternatives", in Gambetta, D. (ed.) *Trust: Making and Breaking Cooperative Relations*, Basil Blackwell, Oxford, 1988, pp. 95-107.

[7] Miles, R.E., and C.C. Snow, "Causes of failure in network organizations", *California Management Review*, Vol. 34, No. 4, 1992, pp. 53-72.

[8] Mishra, A.K., "Organizational responses to crisis", in Kramer, R.M., and Tyler, T.R., (eds.) *Trust in organizations*, Thousand Oaks, Sage, 1996, pp. 261-287.

[9] Sabherwal, R., "The role of trust in outsourced IS development projects", *Communications of the ACM*, Vol. 42, No. 2, 1999, pp. 80-86.

[10] Seligman, A.B., "Trust and sociability: On the limits of confidence and role expectations" *American Journal of Economics & Sociology*, Vol. 57, No. 4, 1998, pp. 391-404.

[11] van der Smagt, T., “Enhancing virtual teams: social relations vs. communication technology”, *Industrial Management & Data Systems*, Vol. 100, No. 4, 2000, pp. 148-156.

Evaluating Effectiveness of Global Software Development Using the eXtreme Programming Development Framework (XPDF)

Samantha J. Butler, Sian Hope
School of Informatics
University of Wales, Bangor
sbutler@informatics.bangor.ac.uk, sian@informatics.bangor.ac.uk

Abstract

This paper presents an ongoing study into the productivity of the eXtreme Programming software development approach when carried out globally. XP is evaluated using a framework that has been developed using multiple data collection techniques. This study is in the early stages, at this point a pilot study is underway with a view to a more rigorous study once the analysis is complete and framework assessed. The paper also covers the future work that could be completed using this framework.

1. Introduction

Due to the changing nature of Software Engineering a greater number of people than ever are working in non-traditional development environments. This is pushing the existing Software development methods to new limits. These methods have been proven over a number of years to work to at least a satisfactory level in more orthodox settings, the question that my research poses is do they also work to the same satisfactory level when used in a geographically disparate manner.

This paper presents a theory that I proposed, the reasoning behind this investigation and an initial set of results and future experiments. This work is in its early stages and the results give an intermediary view of the investigation.

2. Why I think XP can work in a Global development environment

The eXtreme Programming (XP) approach has been proven by previous studies to be a successful development methodology in an orthodox development environment [1][2]. In addition these studies show that although XP isn't for everyone the majority of developers who have developed software using both traditional software development methods and XP believe that this is a more satisfying and productive way of writing software.

There are now many tools available to encourage communication and numerous other tools available to allow two people to view and amend the same code at the same time from different locations, commonly grouped under the 'groupware' heading [3][4]. However the use of tools alone will not be sufficient. This work focuses on using XP as the process model with an eXtreme Programming Development Framework (XPDF) to evaluate this approach.

One of the four XP principles is communication and the importance of this to effective software development. Encouraging the level of communication suggested by XP in development teams working remotely has proved to be one of the most difficult areas to replicate. With the tools that are readily available and the XP approach, this work is evaluating whether it is possible to have a similar level of communication whilst working in remote locations to more orthodox software development teams.

3. What is XPDF?

XPDF is an evaluation framework, which incorporates small, easily implemented, aspects from three, previously tried and tested investigation techniques. These include Software Process Improvement [5] evaluation processes, such as aspects from the Capability Maturity Model [6][7][8], which focuses heavily on the repeatability and reliability of the development method in question, in this case XP.

Qualitative Research techniques focus on the subjective opinions of those participating in the study; in

this context how people feel about working in remote locations in place of a more traditional development environment or using different development methodologies. Data will be collected using questionnaires and interviews [9][10].

Finally Software Metrics were collected, software metrics are the bare statistics of the team, number of lines of code produced in the time given, the number of errors in the final deliverable, any change from previously delivered work to name a few [11][12][13].

During the data collection phase data collected using each technique was treated in isolation, as indicated by Figure 1. However during the evaluation phase of the investigation several of the results will be analysed using two or more methodologies simultaneously.

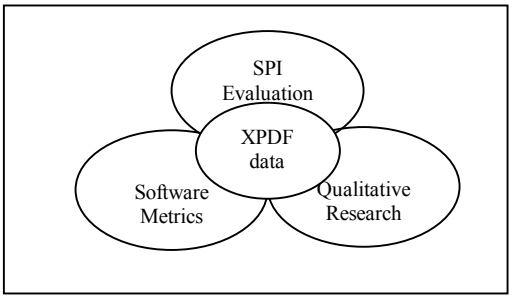


Figure 1: The Evaluation Framework used.

Several XP practices can be carried out in the same way independently of the location of the developers. This study concentrates on the evaluation of the practices that must be adapted, these include pair programming, collective code ownership, customer onsite and always available, and high levels of communication and pair swapping [14][15]. XPDF aims to evaluate the productivity of teams using XP in remote situations in comparison to those in more traditional settings.

4. Preliminary results

Preliminary results show that XP is a very effective development methodology when used in an orthodox setting [1][2]. There are no official results for the effectiveness of XP when used on remotely conducted projects as this research is in its early stages. However early indicators of the analysis of data are that with the correct tools in place to aid communication XP can be just as effective when carried out in this manner as in more orthodox settings.

5. Experiments in Progress

5.1. Experiment 1

This experiment involves a number of volunteers. Each experiment requires two participants. After the pilot investigation it would be beneficial to have more participants involved throughout each iteration. People participating in this experiment are given an overview to eXtreme Programming, in the form of a handout for them to read. An in-depth knowledge is not required, as specific tasks are presented guiding the participants through XP in a stepwise manner.

A relatively simple exercise, involving the programming of a 'Lego brick' is presented for completion following XP principles. The main challenges of this task are the implementation of XP practices on this scale and with participants in different locations. The two participants are located in adjoining rooms. The person conducting the experiment is able to view the activities of both participants and give instructions to both participants or to each individual. Communication is by means of a telephone, NetMeeting and a code-sharing tool. All tasks are be presented by the customer as realistically as possible in the form of tasks and story cards decided at a planning meeting, again conducted remotely, with changing requirements and priorities.

5.2. Experiment 2

Software Hut is a module taken by all second year students taking a Computer Science / Studies degree course. During this module the students are asked to write a piece of software to fulfil a specific brief. All the students are given the same brief and all have to follow the same development process. To carry out the investigation effectively the group is split into two sub groups, both are presented with the same task. One group follows the XP principles in an orthodox manner, the other follows XP remotely. Both groups are evaluated using XPDF.

It could be argued that these students have insufficient experience in both developing software using any development process, or carrying out eXtreme Programming software development in particular.

However at no point in the specification of this research is it mentioned that the subjects under investigation are either experienced software developers or in mainstream industry. The criteria are that the participants are developing software on some scale following the eXtreme Processing methodology in a remote fashion. When this implementation is carried out the lack of experience will have to be taken into consideration when analysing the results. Similar studies evaluating XP in undergraduate modules have been carried out and the effect on the final conclusions shows to be negligible [16][17].

6. Future experiments – Why Use XPDF?

Small development teams carrying out all or some of their development using XP in a remote manner would be required to participate in the experiment. Companies would be required to supply similar information to that required in the experiments currently running. This would include the completion of qualitative and quantitative questionnaires, participation in an interview. As well as the above each company would be required to provide a summary of productivity during the course of the study.

In return for their participation the company would gain feedback on the emotions of the development team regarding the process and productivity, suggestions on how the current process could be improved based on the information gathered from their company and the findings of others. All information supplied would be confidential.

As well as this companies would be included in any research papers resulting from the studies.

7. References

[1] Gittins, R.G., Hope, S., Williams, I. 'Qualitative Studies of XP in a Medium Sized Business' UPGRADE The European Online Magazine for the IT Professional. Vol. III, No.2, April 2002 http://www.upgrade-cepis.org

[2] Gittins, R Qualitative Studies of XP in a Medium Sized Business'
Proceedings of the 2nd International Conference on Extreme Programming and Flexible Processes in Software Engineering. 20-23 May 2001, Sardinia, Italy

[3] http://www.usabilityfirst.com/groupware/intro.txl

[4] http://simon.cs.vt.edu/jamm/

[5]Systemic Software Engineering; Software Process Improvement White Paper; 2000

[6] Mark C. Paulk, Bill Curtis, Mary Beth Chrissis, and Charles V. Weber, "Capability Maturity Model, Version 1.1," IEEE Software, Vol. 10, No. 4, July 1993, pp. 18-27.

[7] http://www.sei.cmu.edu/cmm/cmm.html

[8] http://www.sei.cmu.edu/cmm/

[9] Patton, M.Q; Qualitative Evaluation and Research Methods (2nd Edition; SAGE Publications

[10] Seaman, C.B; Qualitative methods in Empirical Studies of Software Engineering; IEEE Transitions in Software Engineering; Vol25 (4): 55-572 July/Aug 99

[11]Fenton, NE and Pfleeger, Shari Lawrence; Software Metrics, a Rigorous and Practical Approach; International Thompson Computer Press; 1996

[12]Goodman, Paul; Practical Implementation of software metrics; McGraw-Hill; London; New York: McGraw-Hill, c1993

[13] Cultivation and engineering of a software metrics program; Iversen J, Mathiassen L INFORMATION SYSTEMS JOURNAL 13 (1): 3-19 JAN 2003

[14]Beck, K; eXtreme Programming Explained: Embrace Chnge; Addison Wesley; 2000

[15] Jefferies, R; Anderson, A; Hendrickson, C; eXtreme Programming Installed; Addison Wesley; 2000

[16] Macias, F, Holcombe, M; Empirical Experiments with XP, University of Sheffield, 2002

[17] Holcombe, M, Gheorghe, M, Macias, F; Teaching XP for Real: Some Initial Observations and Plans; University of Sheffield; 2002

Taking Global Software Development from Industry to University and Back Again

Igor Cavrak
University of Zagreb
Faculty of Electrical Engineering and Computing
HR-10000 Zagreb, Croatia
+385 1 6129 861
igor.cavrak@fer.hr
http://www.rasip.fer.hr/icavrak

Rikard Land
Mälardalen University
Department of Computer Science and Engineering
PO Box 883, SE-721 23 Västerås, Sweden
+46 21 10 70 35
rikard.land@mdh.se
http://www.idt.mdh.se/~rld

Abstract

As global software development (GSD) establishes itself as a software engineering practice, it should be taught to the software developers of the future. But can the problems of GSD really be transferred to the controlled environment of a university course? Will the students attending the course be able to cope with the problems associated with GSD as professionals? This paper elaborates on these questions by presenting approaches, methods, and goals in a planned GSD university course.

1. Introduction

Universities should equip computer science students with not only theory and technology skills but also knowledge in the engineering environment they will be faced with as future software engineers. Many such issues are addressed by specific courses in software engineering, often organized as practical projects in which the students will have to tackle "real" problems [1,2,5,6,7,8,9,12]. But as global software development (GSD) is becoming widespread in today's international enterprises and virtual organizations [3,10], universities should incorporate this trend into their curriculum [11]. To our knowledge, university courses or student projects involving global software development are very rare, and are restricted either to existing software engineering courses [2] or case studies [1]. Perhaps the problem to give a course in this area is due to the inherent properties of global software development: it is difficult to present a university course that faces the students with global development specifics such as telephone- and videoconferencing, distributed configuration management, using a foreign language (English), coping with another culture, and collaborating across time zones. Despite this, we have accepted the challenge and are developing a course in distributed software development, to be simultaneously held in Västerås, Sweden, and Zagreb, Croatia.

2. Course Description

The students will be introduced to the problems of GSD through lectures and/or self-studies but the larger part of the course will consist of a joint software development project. In the project, the students will encounter many GSD problems but not all: Sweden and Croatia are e.g. located in the same time zone. Besides combining theory and practice through lectures and the joint project, the problem domain addressed by the project will as well be in the GSD domain: we plan to develop and/or enhance tools to support GSD. Each time the course is held, experiences and suggestions of more product features will be "inherited" from the previous year. In this way requirements and products are "bootstrapped" along subsequent courses.

2.1. Relation to Community Development

As products will be primarily based on free software (due to the financial reality), we plan to submit course products back to the public community thus employing a large number of users in testing the product usability and quality, and gathering feedback to use in next year's requirements. Maintenance of the product will be (at least partially) ensured by assigning product-related graduation thesis to some of the students participating in the project.

The type of development the students will perform only vaguely resembles community development: although the work is distributed and somewhat individualized, there will be a much stronger project

management, and some of the developers will meet physically.

2.2. Relation to GSD in Industry

If the course involves a real customer, he will presumably invest both time and money [2], enabling both a real-world problem and traveling possibilities [2]. However, there is a risk that focus will be more on the end product than on the educational elements of the project.

We have therefore chosen to avoid industrial involvement in favor of a slight shift towards community software development. This means that we can focus on education, but it also means that we will have to use low-cost communication media and will not be able to let the students travel. Especially the latter is a great disadvantage, since being able to meet face to face has been emphasized as the most important means to enable successful GSD [3,4,10]. However, by the start of the course some of the course leaders will have worked together for six months, which will help alleviating the cultural and physical distance [2,3,4]. Also, since this is a university course, should the worst happen and there is no product at all at the end of the course, this is evidence that the students encountered problems – which is educational (at least to some extent).

The students will form groups at each university and work part of the time together, as is the case in the office environment at most companies, but will also work at different hours and locations (due to their other obligations and us not providing personal workplaces at the university).

2.3. University Specifics

There are other university environment specifics as well such as students having different background and experiences, adjusting the course to the existing curriculum at two universities (which can present a major problem as duration of the course, start and end dates must be coordinated [2]). Each university also has a responsibility towards its own students in the first place – how should students working on the same project/product be graded in different rating systems? And what happens if there are not enough students at one of the universities involved? A careful balance must be struck not to overwhelm students with project work on one of the sites due to differences in obligations regarding the number of other courses held at their universities.

3. Main Issues

We can discern two major challenges for our course. First, how do you transfer the problem domain of GSD from industry environment to a university course? Second, how useful will the outcomes of the course be to industry: solutions and software professionals?

4. References

[1] Brereton P., Lees S., Bedson R., Boldyreff C., Drummond S., Layzell P., Macaulay L., Young R., "Student Collaboration across universities: A Case Study in Software Engineering", *Proceedings of 13th Conference on Software Engineering Education and Training (CSEE&T)*, IEEE, 2000.

[2] Bruegge, B., Dutoit, A.H., Kobylinski, R. and Teubner, G. "Transatlantic project course in a university environment", *Proceedings of 7th Asia-Pacific Software Engineering Conference (APSEC)*, 2000.

[3] Carmel E., *Global Software Teams: Collaborating Across Borders and Time Zones*, ISBN 0-1392-4218-X, Prentice-Hall, Upper Saddle River, NJ, 1998.

[4] Carmel E., Agarwal R., "Tactical Approaches for Alleviating Distance in Global Software Development", *IEEE Software, volume 18, issue 2*, IEEE, 2001

[5] Crnkovic I., Larsson M., and Lüders F., "Implementation of a Software Engineering Course for Computer Science Students", *Proceedings of 7th Asia-Pacific Software Engineering Conference (APSEC)*, 2000.

[6] Crnkovic I., Land R., and Sjögren A., "Is Software Engineering Training Enough for Software Engineers?", *Proceedings of 16th Conference on Software Engineering Education and Training (CSEE&T)*, IEEE, 2003.

[7] Daniels M., Faulkner X., and Newman I., "Open ended group projects, motivating students and preparing them for the 'real world'", *Proceedings of 15th Conference on Software Engineering Education and Training (CSEE&T)*, IEEE, 2002.

[8] Dawson R.J., Newsham R. W., and Fernley B. W., "Bringing the 'real world' of software engineering to university undergraduate courses", *IEE Proceedings In Software Engineering, volume 144, issue 5*, 1997.

[9] Dawson R., "Twenty Dirty Tricks to Train Software Engineers", *Proceedings of 22nd International Conference on Software Engineering (ICSE)*, ACM, 2000.

[10] Karolak D., *Global Software Development: Managing Virtual Teams and Environments*, ISBN 0-8186-8701-0, IEEE Computer Society Press, Los Alamitos, CA, 1998.

[11] Shaw M., "Software Engineering Education: A Roadmap", *Proceedings of the 22nd International Conference on Software Engineering*, ACM Press, New York, NY, 2000.

[12] Wohlin C. and Regnell B., "Achieving industrial relevance in software engineering education", *Proceedings of 12th Conference on Software Engineering Education and Training (CSEE&T)*, IEEE, 1999