Towards a Peer-to-Peer Based Global Software Development Environment

Patrick Mukherjee[1], Aleksandra Kovacevic[2], Michael Benz[3], Andy Schürr[1]

[1]Real-Time Systems Lab, Technische Universität Darmstadt
Merckstr. 25, 64289 Darmstadt, Germany
mukherjee, schuerr@ES.tu-darmstadt.de

[2]Multimedia Communications Lab, Technische Universität Darmstadt
Merckstr. 25, 64289 Darmstadt, Germany
sandra@KOM.tu-darmstadt.de

[3]Research Group IT-Security, Technische Universität Darmstadt
Hochschulstr. 10, 64289 Darmstadt, Germany
benz@SEC.informatik.tu-darmstadt.de

Abstract: Nowadays, large projects are developed by globally distributed developer teams. Global Software Development (GSD) is currently not supported by appropriate tools but with the tools designed for on-site development. In this work we analyze benefits of a peer-to-peer approach to integrated environment for GSD, analyze its requirements from selected industrial field studies, and present the architecture of our solution – **P**eer-to-Peer based **I**ntegrated **P**roject-support **E**nvironment (PIPE).

1 Introduction

Development of most of today's software projects is organized into globally distributed developer teams in different locations around the world. Outsourcing (to obtain better expertise or to decrease cost) and efficient time management (taking advantage of different time zones) are just a few motives for it [MH01]. Global software development (GSD) is not a single phenomenon but, rather, the norm for industrial projects [MHK05].

In addition to cultural differences [PAE03], GSD has to cope with the lack of appropriate support tools. Currently there is no development environment specifically designed for GSD; the available tools are designed for on-site development where multiple clients are using one server, i.e. the client/server communication paradigm (C/S).

In this paper we discuss the drawbacks of the C/S approach for GSD and focus on the challenges facing a peer-to-peer based GSD environment. We analyze the general requirements for GSD based on various case studies and derive requirements specific for the peer-to-peer approach. Additionally, we introduce the architecture of our solution, the **P**eer-to-Peer based **I**ntegrated **P**roject-support **E**nvironment (PIPE), and analyze how the requirements affected our design decisions.

This paper is organized as follows: After motivating why a peer-to-peer approach to GSD is promising (Section 2), we give all identified requirements for a GSD with derived requirements specific for a peer-to-peer approach (in Section 3). The architecture of our solution, PIPE, is presented in Section 4. Related work is summarized in Section 5 and a conclusion is given in Section 6.

2 Motivation

In order to clarify the motivation for this work, we first describe GSD, then analyse drawbacks of existing, C/S solutions, and finally show why a peer-to-peer based system would overcome them.

A Description of a Global Software Development Environment

GSD is typically organized as follows: The company leading the project is referred to as *project leader*. It employs *multiple subcontractors*. In IT-projects it is not unlikely that open-source software will be integrated so that the open-source developer community becomes part of a project. The developers of the mentioned parties are static, meaning they are working at a single location. It is necessary that developers have the option of meeting face-to-face. Additionally, GSD includes experts who consult teams directly on-site. Both of them are modeled as *nomadic developers* that are assumed to work in different places, even during their travels. A GSD environment is supposed to support a large number of participants, whereas a global company handles multiple projects using the environment to share knowledge and expertize. The opposite should be supported as well, where a few developers spontaneously meet for a small project and a low start-up time is important.

B Drawback of Existing Client/Server Based Solutions

The C/S solution to GSD has numerous drawbacks. Communication is often slow with poor scalability since all communication the system supports has to be transmitted through a single server. Numerous field studies [HPB05, HM03, Šmi06, PAP06] and a recent survey [ISHGH07] concluded that communication is most critical for the success of distributed projects. Communication refers not only to exchange of files but of messages as well.

Unlike on-site development, some developers in GSD are geographically distant from the server and, therefore, have to cope with delay in data transfer [HM03]. Even if a cluster of servers is utilized, it just offers more power, thereby extending the scalability limits. Distributing the servers across different locations can not decrease the communication delay for all developers. The distributed servers remain limited and they are not omnipresent.

If cooperating teams are using their own server environments, significant synchronization problems may arise. These problems occur mainly in version control systems, but may

also affect access rights. In practice, these problems are ignored and *artifacts* (all products of the steps in software development - from specifications and documentation to source code) are exchanged through numerous other channels like email or USB-sticks.

Another important issue of the C/S approach are its high maintenance cost. Resources (computation power, memory, bandwidth, and storage space) in the system are not optimally used: servers provide all their resources while resources on client machines remain unused most of the time. The cost of servers is especially problematic for non-profit projects (like open source projects) as they are normally founded by unreliable donations.

C Motivation for a Peer-to-Peer Approach

The centralization on a server is orthogonal to the natural structure of GSD. Developer teams normally manipulate the same artifacts and communication will most likely happen between them. This communication (including message exchange and file transfer) obviously does not need a remote server and should proceed directly from peer to peer.

A peer-to-peer based approach is naturally suited for GSD as it is inherently distributed; There is no central point of communication. After a distributed routing algorithm connects the participants they communicate directly with each other. Thus the network delay is as minimal as possible.

The costs of peer-to-peer solutions are significantly lower than in C/S solutions as the network is self-organized and resources from all participants are used [SE05].

3 Requirements and Challenges for Peer-to-Peer GSD Environment

In this section we elaborate on the requirements derived from selected industrial field studies ([HPB05, HM03, PAP06, Šmi06, Sou01, ISHGH07]) and additional requirements emerging from usage of peer-to-peer technology. The requirements are grouped into four categories, depending on their main goal: *system properties*, *services*, *required applications*, and *security aspects*.

A System Properties

A.1 One conjoined system A GSD environment should be one unitary system meaning it should be sufficient for a participant to join only once per working session. This includes single sign-on as well as connection sharing among the different applications running in the system, where the data should be managed by a single component of the system. A study on nine GSD projects [HPB05] showed that each site had its own version control server so as to avoid the constant delay introduced by using a common one. The produced artifacts, therefore, had to be synchronized, which involved manual work where errors occurred.

A.2 Adequate communication speed The delay of communication and transfer in the system should be minimal. Difficulties introduced by communication delays are described in [HPB05] and marked as critical to the project success in [HM03]. Depending on the underlay topology awareness, peer-to-peer overlay networks can significantly decrease such delays.

A.3 Robust to changing number of participants Participants join and leave the system at any time. Due to different time zones, it is likely that a large number of participants from one area will leave and participants from another will join within a short time frame. This high fluctuation in number and geographical distribution of participating peers should not influence system performance. Replication and update mechanisms of peer-to-peer overlay networks are crucial here.

A.4 Scalability The system should be able to support a large, dynamic number of participants without decreasing performance.

A.5 Availability There should be no constraints to accessing the system regarding / depending on the network a participant is connected to, time of access, or working platforms (e.g. OS).

B Services

B.1 Version management The most important service in a GSD environment is keeping track of the changes and variants (branches in version management) of artifacts – version control [Šmi06]. These requirements are one of the biggest challenges for our peer-to-peer approach, as distributed and replicated artifacts have to be coherent. The following three requirements are crucial for version management in GSD:

B.1.i Coherency A participant requesting the latest version of some data should not be provided with an outdated copy. It is not necessary that all replicas in the system are up-to-date (consistency), but outdated replicas should be marked.

B.1.ii Interoperability of different version management systems Numerous tools use their own version control system, which is often proprietary. The version management in a GSD system needs to be able to integrate such systems. In the ideal case only one system should be used, as pointed out by [PAP06, HPB05, Šmi06].

B.1.iii Offline version control Nomadic developers, in particular, need to be able to work offline (e.g. during travels). Modified artifacts should be transparently synchronized as soon as the developer comes online as demonstrated by, for example, the C/S based system [SVK].

B.2 Resource management In a distributed system it is important to manage the available distributed resources (e.g. data, storage, memory, etc.). This includes sharing and discovery.

B.2.i Transparent offering of resources Services (e.g. memory and hard disc space or data) should be offered transparently to the user. The system is responsible for balancing the load and for offering the resources to other participants.

B.2.ii Transparent resource discovery All shared resources should be easy to find. The system has to manage resource discovery without the involvement of the user. It should not make a difference for the user whether a resource is stored locally or on a remote location.

B.2.iii Full retrievability In spite of the fact that peers can go offline at any time, all stored artifacts should always be retrievable.

B.2.iv Traceability Artifacts may contain links to other semantically related artifacts. It should always be possible to trace and locate those linked artifacts in the system. Linking artifacts provides needed context as demanded by [Sou01]. It also enables a user to navigate from one artifact to another as demonstrated in [ADDK03], where the linked artifact is retrieved and its appropriate editing tool is opened automatically.

B.3 Awareness If the participants are not able to physically see each other, it is important that their presence in the GSD environment is visible, as shown in the empirical study [HM03]. For example, applications like instant messengers can directly show directly whether somebody is absent or free to respond. Group calenders can help in planning (virtual) meetings and tracking conversations; As soon as a participant appears online, similar to hallway conversations, other participants are reminded about a deferred conversation.

C Required Applications

C.1 Knowledge management system In project development, sharing expertize and knowledge are often crucial in order to fulfill certain sub-tasks. Therefore, a knowledge management application (knowledge database) should be offered by the GSD environment (see [HM02, HPB05]). Additional information, e.g. about available experts on specific topics can also be stored here [HM03, Sou01].

C.2 Requirements engineering tool An important initial step in a project is identifying all necessary requirements; thus, a requirement engineering tool has to be part of a GSD environment.

C.3 Modeling tool The design phase requires various modeling tools (e.g. UML for specifying the software architecture) which consequently has to be part of a GSD environment.

C.4 General purpose IDE The main tool in any software development is an Integrated Developing Environment (IDE) that supports writing source code.

C.5 Instant communication At least one tool that enables instant communication between all participants in a GSD environment should be provided. Several investigations of distributed projects ([Šmi06, HM03]) pointed out that communication is one of the most critical factors for success. Communication should take place directly between participants (i.e. without an intermediate like a project manager) [HPB05, Sou01].

C.6 CSCW tools To support collaboration, Computer Supported Cooperative Work (CSCW) tools should be easy to integrate in a GSD environment. These include calenders, discussion boards, event scheduling support and similar tools that allow multiple participants to communicate efficiently.

D Security Aspects

A highly distributed system, where confidential data is shared between business partners, requires sophisticated security mechanisms. Many new security challenges arise when the tasks that are usually fulfilled by a single trusted node are divided among multiple entities [Wal02]. One of the central challenges is the issue of trust [MGM06], which has spawned numerous research studies regarding trust management in distributed systems [AD01, ADV+06, KSGM03, SL03]. In the following, we will describe the security requirements we deem most important for a GSD environment:

D.1 Access and usage control Access control plays an important role in a software development process. Security critical documents must be protected from unauthorized access without impairing the overall system efficiency. In a GSD environment, selected group administrators are responsible for defining security policies for their respective user groups. The participants have the option to further restrict access to their files. The peer-to-peer approach introduces even more problems, since there are now multiple entities responsible for defining security policies for their respective groups. Enforcement of those policies has to be deferred to the participant [SZRC06], since availability of central authorities can not be guaranteed.

D.2 Attribute based access control In a large software development process, it is often unnecessary to control the access on an individual basis. For most tasks it suffices for participants to be identified via their respective attributes. An administrator can issue a signed certificate to a group member, certifying his developer status. Other participants can then base their access decision solely on the presented user credentials. Park et al., for example, developed a role-based access control approach for a collaborative enterprise in peer-to-peer computing environments [PH03].

D.3 Authentication Several security goals such as confidentiality, data integrity, access control and non-repudiation depend on proper authentication. If the system for user identification fails, the mentioned security goals cannot be met in a satisfactory manner. On top of that, it is not always easy to decide whom to trust in a highly distributed system. Employees of a large company might benefit from certificates issued by a company CA, whereas nomadic developers might have to depend on a web of trust with self-signed certificates. Thus, the development environment has to offer and support many different authentication methods.

D.4 Dynamic user groups The composition of the user groups is highly dependent on the current project setup. For example, open source projects often have a fluctuating developer team. Therefore, efficient methods for handling group collaborations [PWF+02] are required. Numerous efforts aimed at providing efficient group keying mechanisms have already been carried out in the research community. Ever since the point to point Diffie-Hellmann key exchange protocol was first proposed in 1976, there have been efforts to extend its simplicity to a group setting. Steiner et al. [STW96], for example, proposed an extension of the Diffie-Hellman protocol to an n-party setting and showed that the security of their protocol is equivalent to the security of the original 2-party protocol. A couple of years later they proposed another extension to support highly dynamic group collaborations [STW98].

D.5 Secure communication The communication between the participants has to be confidential and data integrity needs to be guaranteed. This requirement is tightly coupled with the previous requirement. Efficient and secure group communication mechanisms have to be provided by the development environment.

D.6 Low administration overhead The administration overhead introduced by additional security mechanisms has to be kept as low as possible. A software developer participating in the system should not have to deal with defining specific access rights for his files. Though this should be transparent to him, it should be the task of a group administrator.

4 Peer-to-Peer Based Integrated Project-support Environment (PIPE)

In this section we present the architecture of our peer-to-peer approach for a GSD environment and discuss how the previously described requirements (in Section 3) affected the design of our solution. **P**eer-to-Peer based **I**ntegrated **P**roject-support **E**nvironment (PIPE) follows the modular component architecture shown in figure 1.

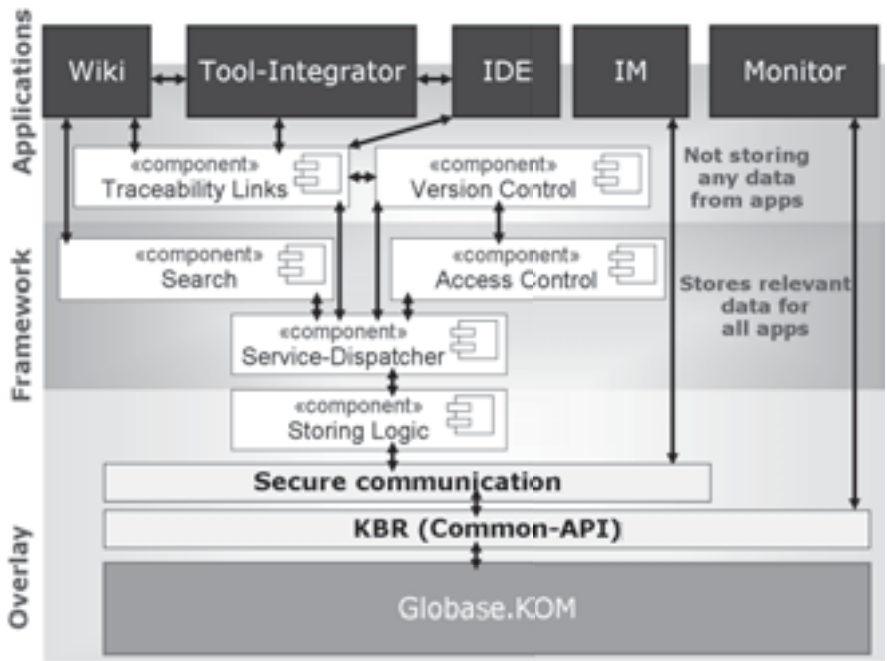

Figure 1: PIPE: Components

As PIPE should support any project development it will feature applications which are not project specific. That includes a Wiki Engine, Tool Integrator, CSCW, IDE, and Monitor. The *Wiki Engine* can be used for requirements engineering (C.2) as shown by [GHHS07] in addition to its usual usage as a knowledge database (C.1). It utilizes the version con-

trol, traceability and search components and uses access control. As an *IDE* we chose to integrate Eclipse, currently the most popular IDE, in order to meet requirement C.4. The *IDE* relies on the same components as the *Wiki Engine*. We will provide an *IM plug-in* for the choosen *IDE*, which, in addition to standard instant communication, highlights the user who last edited the currently opened sourcecode (B.3). Messages that are meant for an offline peer would automatically be sent by other peers as soon as it appears online. In existing peer-to-peer IMs (e.g. [sky]), such a message would arrive only when both participants are online meaning that the message could be delayed as long as the sender is offline. There will be other integrated *CSCW*-tools like a shared calender, which is coupled with the *IM*. As soon as a user appears online, all scheduled events related to that user will pop up to notify other connected users.

To enable users to get a global view of the resources and the current state of the system (how many peers are connected, etc) we will implement a *Monitor* application. Additional project-specific tools like, for example, a modeling tool (C.3) can be easily integrated into PIPE with the Tool-Integrator [KS06b], which will be an integral component. As the developers in GSD projects are usually working with different tools and, therefore, creating artifacts in different proprietary formats, conversion and keeping artifacts consistent is the focus of this component. Therefore, we use [KS06a], which is based on triple graph grammars (TGG) [Sch94] that enable extracting the semantic information of a structured document and transforming it in another structured document, thereby making their editing tools interoperable.

The *Search* component will provide data indexing and localization, which helps fulfilling B.2.ii. If an application, like the *Wiki Engine*, needs this service it will index its artifacts in order to provide a full-text search. Tracing a semantic link between artifacts is another way to retrieve information. The service which fulfills this need described by requirement B.2.iv is the component *Traceability Links*. The link information can be explicitly or implicitly set by the user (e.g. when linking wiki articles) or automatically by the system as a result of the *Tool-Integrator* as done in [ADDK03]. Additionally, any other metadata can be stored in traceability links, like the last authors name or the link status. This component takes versioning into account and retrieves a linked artifact in the corresponding version though it is not necessarily the latest one. Therefor it utilizes the *Version Control* component, which fully meets requirement B.1 and will be implemented similar to [BLS06]. It will feature offline version control to support nomadic developer (and statisfy A.5).

The component *Access Control* is responsible for fulfilling requirements D.1, D.2, D.3, and D.4, taking into account requirement D.6. Whenever information should be retrieved, this component ensures that the requesting user is authorized. This applies to querry results as well. Confidential data is exchanged via the *Secure Communication* component (fulfilling requirement D.5). Standard secure communication methods like SSL encryption will be used here.

The *Service Dispatcher* configures the overlay network for all applications, keeps track of the resources (data, services) an individual peer offers to the network, and sets/defines all overlay specific parameters as well as the peer ID for a peer. This component computes hashes for the data-keys obtained from an application, as the mapped key-space depends on the overlay configuration/implementation. Additionally, it keeps track of all running

applications and their offered services in order to direct messages not only to another peer, but to a specific application on that peer. If we further equip the system with the ability to discover offered services, an indexing mechanism would be implemented in this component. It meets requirements B.2.i and B.2.ii as well as supporting A.1.

The Component *Storing Logic* provides permanent availability of the data (requirements A.3 and A.5) by creating and maintaining replicas independently of the used overlay. In spite of the fact that replication and storage is part of most overlay designs, it can be separated from routing in structured overlays [DZD+03].

The overlay layer provides the communication logic, i.e. message exchange, storage, and resource discovery. In order to fulfill requirements A.2, A.3, and A.4, we use Globase.KOM [KLS07], a superpeer-based peer-to-peer overlay that forms a tree. It enables a fully retrievable location-based search. Each superpeer is responsible for all peers in an assigned rectangular zone. Globase.KOM proved to have short response time due to a high degree of underlay topology awareness and to be logarithmically scalable. In PIPE we use the available servers as superpeers in order to improve robustness and stability (A.3). We use the advantage that a superpeer is geographically close to the peers it is responsible for, as it is likely that developers within one subcontractor would more frequently exchange data and communicate with each other than with geographically distant developers.

However, by using the *Key-Based Routing Common API (KBR)* [DZD+03] our solution supports any peer-to-peer overlay network that can be adapted to it, like, for example, FreePastry [FP] or the superpeer overlay network for GSD proposed by Bischofs et al. [BH04]

The **P**eer-to-Peer based **I**ntegrated **P**roject-support **E**nvironment (PIPE) is highly available (A.5) by using the platform independent programming language Java and due to the inherent properties the peer-to-peer technology offers [SE05]. Currently it consist of a wiki engine, which utilizes components for traceability, version control, (full-text) search and storing logic. Other described components are under development.

5 Related Work

There are numerous C/S based tools that support GSD. Most of them cover only a particular aspect of GSD, sometimes integrated into a single platform, e.g. IBM Rational [IBM]. Jazz [HCRP04] is a tool under development, intended specifically for GSD. It will be based on the C/S paradigm. According to the best authors' knowledge, there is no integrated tool environment that fulfills the needs listed in section 3.

Groove Virtual Office [Gro] is a collaboration environment that is partly peer-to-peer-based. In its first version it had serious scalability problems, hardly supporting 20 developers in the same workspace [Har01]. When Microsoft bought Groove Networks in March 2005 in order to save the project [Mil], it became evident that the technology was still unready for the market. The software was restructured to improve scalability using the C/S approach. The current version fulfills several security requirements but still does not support version control management.

Code co-op [Rel] is a version management system that is partly peer-to-peer based. In local area networks, it uses peer-to-peer-communication. For longer distance communication and for storing data, it relies on mail servers.

The only peer-to-peer-tool that was designed to support GSD is MASE P2P [Bow03]. Basically, MASE P2P is an extension of the C/S based MASE environment [Mau] which was developed in a diploma thesis. However, this peer-to-peer-extension was discontinued after the thesis was finished. The development of MASE continues on a C/S basis.

6 Conclusion

In this paper we point out the needs for a global software development (GSD) environment, as there is no suitable solution available. Further more, we discussed the benefits that a peer-to-peer approach brings to this purpose in contrast to a client/server solution. Based on selected industrial field studies and taking into account issues emerging from the usage of peer-to-peer technology, we derived four sets of requirements: system properties, services, required applications, and security aspects. We proposed a peer-to-peer GSD environment design which implements the aforementioned requirements. The consistent modular design of our architecture should help in exchanging or adding components, starting from an overlay network and applications and going to version control, replication, and access control mechanisms. PIPE will be developed as a proof-of-concept within the DFG funded research group QuaP2P aiming to improve quality properties of peer-to-peer systems.

Acknowledgment

The authors would like to thank colleagues from DFG funded research group QuaP2P and Real-Time System Lab for valuable discussions and future collaboration.

References

[AD01] Karl Aberer and Zoran Despotovic. Managing Trust in a Peer-2-Peer Information System. In *Proceedings of the Conference on Information and Knowledge Management (CIKM)*, pages 310–317, 2001.

[ADDK03] Frank Altheide, Sven Dörfel, Heiko Dörr, and Jan Kanzleiter. An Architecture for a Sustainable Tool Integration. In *Proceedings of the Workshop on Tool Integration in System Development (TIS)*, pages 29–32, September 2003.

[ADV+06] Roberto Aringhieri, Ernesto Damiani, Sabine De Capitani Di Vimercati, Stefano Paraboschi, and Pierangelo Samarati. Fuzzy Techniques for Trust and Reputation Management in Anonymous Peer-to-Peer Systems. *Journal of the American Society for Information Science and Technology*, 57(4):528–537, February 2006.

[BH04] Ludger Bischofs and Wilhelm Hasselbring. A Hierarchical Super Peer Network for Distributed Software Development. In *Proceedings of the Workshop on Cooperative Support for Distributed Software Engineering Processes (CSSE)*, September 2004.

[BLS06] Elizabeth Borowsky, Andrew Logan, and Robert Signorile. Leveraging the Client-Server Model in P2P: Managing Concurrent File Updates in a P2P System. In *Proceedings of the Advanced Int'l Conference on Telecommunications and Int'l Conference on Internet and Web Applications and Services (AICT-ICIW)*, 2006.

[Bow03] Seth Bowen. Using a Peer-to-Peer Architecture to Support Distributed Software Development. Master's thesis, University of Calgary, Department of Computer Science, November 2003.

[DZD+03] Frank Dabek, Ben Zhao, Peter Druschel, John Kubiatowicz, and Ion Stoica. Towards a Common API for Structured Peer-to-Peer Overlays. In *Proceedings of the 2nd International Workshop on Peer-to-Peer Systems (IPTPS03)*, pages 33–44, February 2003.

[FP] FreePastry. `http://www.freepastry.org/FreePastry/`.

[GHHS07] Michael Geisser, Hans-Joerg Happel, Tobias Hildenbrand, and Stefan Seedorf. Einsatzpotentiale von Wikis in der Softwareentwicklung am Beispiel von Requirements Engineering und Traceability Management. *Social Software in der Wertschöpfung*, March 2007.

[Gro] Groove Virtual Office. `http://www.groove.net/index.cfm/pagename/VirtualOffice/?home=hp-overview`.

[Har01] Ann Harrison. The Promise and Peril of P2P. `http://www.networkworld.com/buzz2001/p2p/`, September 2001.

[HCRP04] Susanne Hupfer, Li-Te Cheng, Steven Ross, and John Patterson. Introducing Collaboration into an Application Development Environment. In *Proceedings of the Conference on Computer Supported Cooperative Work (CSCW)*, pages 21–24, 2004.

[HM02] Harald Holz and Frank Maurer. Knowledge Management Support for Distributed Agile Software Processes. In *Proceedings of 4th International Workshop of Advances in Learning Software Organizations (LSO)*, pages 60–80, 2002.

[HM03] James D. Herbsleb and Audris Mockus. An Empirical Study of Speed and Communication in Globally Distributed Software Development. *IEEE Transactions on Software Engineering*, 29:481–494, June 2003.

[HPB05] James D. Herbsleb, Daniel J. Paulish, and Matthew Bass. Global Software Development at Siemens: Experience from Nine Projects. In *Proceedings of the 27th International Conference on Software Engineering (ICSE)*, pages 524–533, 2005.

[IBM] IBM Rational Software. `http://www-306.ibm.com/software/rational/`.

[ISHGH07] Timea Illes-Seifert, Andrea Herrmann, Michael Geisser, and Tobias Hildenbrand. The Challenges of Distributed Software Engineering and Requirements Engineering: Results of an Online Survey. In *Proceedings of the First Global Requirements Engineering Workshop (GREW)*, pages 55–66, 2007.

[KLS07] Aleksandra Kovacevic, Nicolas Liebau, and Ralf Steinmetz. Globase.KOM - A P2P Overlay for Fully Retrievable Location-based Search. In *Proceedings of the Seventh IEEE International Conference on Peer-to-Peer Computing*, pages 87–96, September 2007.

[KS06a] Alexander Königs and Andy Schürr. MDI - a Rule-Based Multi-Document and Tool Integration Approach. *Special Section on Model-based Tool Integration in Journal of Software&System Modeling*, 5(4):349–368, December 2006.

[KS06b] Alexander Königs and Andy Schürr. Tool Integration with Triple Graph Grammars - A Survey. In R. Heckel, editor, *Proceedings of the SegraVis School on Foundations of Visual Modelling Techniques*, volume 148 of *Electronic Notes in Theoretical Computer Science*, pages 113–150, Amsterdam, 2006. Elsevier Science Publ.

[KSGM03] Sepandar D. Kamvar, Mario T. Schlosser, and Hector Garcia-Molina. The Eigentrust algorithm for reputation management in P2P networks. In *Proceedings of the 12th International Conference on World Wide Web (WWW)*, pages 640–651, 2003.

[Mau] Frank Maurer. MASE. `http://ebe.cpsc.ucalgary.ca/ebe/Wiki.jsp?page=MASE`.

[MGM06] Sergio Marti and Hector Garcia-Molina. Taxonomy of Trust: Categorizing P2P Reputation Systems. *Computer Networks*, 50(4):472–484, March 2006.

[MH01] Audris Mockus and James D. Herbsleb. Challenges of Global Software Development. In *Proceedings of the IEEE Seventh International Software Metrics Symposium*, pages 182–184, 2001.

[MHK05] Eve MacGregor, Yvonne Hsieh, and Philippe Kruchten. Cultural Patterns in Software Process Mishaps: Incidents in Global Projects. In *Proceedings of the Workshop on Human and Social Factors of Software Engineering (HSSE)*, pages 1–5, 2005.

[Mil] Phil Milford. Suit challenges Microsoft's deal for Groove. `http://seattlepi.nwsource.com/business/218502_msftgroove02.html`.

[PAE03] Rafael Prikladnicki, Jorge Luis Nicolas Audy, and Roberto Evaristo. Global Software Development in Practice Lessons Learned. *Software Process: Improvement and Practice*, 8(4):267–281, October 2003.

[PAP06] Leonardo Pilatti, Jorge Luis Nicolas Audy, and Rafael Prikladnicki. Software Configuration Management over a Global Software Development Environment: Lessons Learned from a Case Study. In *Proceedings of the International Workshop on Global Software Development for the Practitioner (GSD)*, pages 45–50, 2006.

[PH03] Joon S. Park and Junseok Hwang. Role-Based Access Control for Collaborative Enterprise in Peer-to-Peer Computing Environments. In *Proceedings of the Eighth ACM Symposium on Access Control Models and Technologies (SACMAT)*, pages 93–99, 2003.

[PWF+02] Laura Pearlman, Von Welch, Ian Foster, Carl Kesselman, and Steven Tuecke. A Community Authorization Service for Group Collaboration. In *Proceedings of the 3rd International Workshop on Policies for Distributed Systems and Networks (POLICY)*, page 50, 2002.

[Rel] Reliable Software. Code Co-Op - Affordable Peer-to-Peer Version Control System for Distributed Development. `http://www.relisoft.com/co_op/index.htm`.

[Sch94] Andy Schürr. Specification of Graph Translators with Triple Graph Grammars. In *Proceedings of the 20 International Workshop on Graph-Theoretic Concepts in Computer Science*, June 1994.

[SE05] Ralf Steinmetz and Klaus Wehrle (Eds.). *Peer-to-Peer Systems and Applications*. Springer, Sep 2005.

[sky] Skype - A VoIP Instant Messenger. http://www.skype.com/.

[SL03] Aameek Singh and Ling Liu. TrustMe: Anonymous Management of Trust Relationships in Decentralized P2P Systems. In *Proceedings of the 3rd International Conference on Peer-to-Peer Computing (P2P)*, 2003.

[Šmi06] Darja Šmite. Requirements Management in Distributed Projects. *Journal of Universal Knowledge Management*, 1(2):69–76, 2006.

[Sou01] Cleidson R. B. De Souza. Global Software Development: Challenges and Perspectives, 2001. Available online http://citeseer.ist.psu.edu/457465.html.

[STW96] Michael Steiner, Gene Tsudik, and Michael Waidner. Diffie-Hellman Key Distribution Extended to Group Communication. In *Proceedings of the 3rd ACM conference on Computer and communications security (CCS)*, pages 31–37, 1996.

[STW98] Michael Steiner, Gene Tsudik, and Michael Waidner. CLIQUES: A New Approach to Group Key Agreement. In *Proceedings of the 18th International Conference on Distributed Computing Systems (ICDCS)*, pages 380–387, 1998.

[SVK] The SVK Version Control System. http://svk.elixus.org/view/HomePage. Last checked: September 2007.

[SZRC06] Ravi Sandhu, Xinwen Zhang, Kumar Ranganathan, and Michael J. Covington. Client-side Access Control Enforcement Using Trusted Computing and PEI Models. *Journal of High Speed Networks*, 15:229–245, August 2006.

[Wal02] Dan S. Wallach. A Survey of Peer-to-Peer Security Issues. In *Proceedings of the International Symposium on Software Security (ISSS)*, 2002.