Technical Report DCS-331-IR, Department of Computer Science, University of Victoria, December 2009

Empirical Studies on Collaboration in Software Development: A Systematic Literature Review

Christoph Treude, Margaret-Anne Storey, Jens Weber
Dept. of Computer Science, University of Victoria
ctreude@uvic.ca, mstorey@uvic.ca, jens@uvic.ca

Abstract

Collaboration in software development is a complex issue that has been examined by various researchers over the last decade. This paper presents a systematic literature review of pertinent literature on empirical studies on collaborative software development. We found that a lot of progress has been achieved in the field of global or distributed development. While there are still many challenges, today's projects seem to have overcome at least some of those. Also, first steps have been made in understanding the collaborative aspects of requirements engineering and design. Research on tools for collaboration is advanced, having resulted in several successful tools especially in the area of communication.

1. Introduction and Motivation

Software development is one the most complicated tasks performed by humans. Especially in recent years with growing size and complexity of software systems, the coordination of work, artifacts, and developers has become a challenge. The increasing numbers of software developers involved in a typical software project as well as new opportunities given by network technologies have furthermore led to distributed or global software development. In other words, not only is the number of individuals involved in a software project large, these individuals might also work from different continents and time zones and use different languages.

Along with the trend of software development getting more and more complex, research on issues related to communication, collaboration and coordination in software development has increased significantly over the last decade. Both industry and academia acknowledge the importance of team work in software development. Research is primarily focused on observing how software developers coordinate their work and on building theories based on these observations. Also, several tools to support collaboration have been proposed and partially evaluated.

Since the research field of collaboration in software development is still relatively young, there are only a small number of systematic reviews of research conducted so far. This paper adds to this knowledge base by summarizing and categorizing current research on collaboration in software development. In order to minimize bias, a systematic literature review approach is followed and it is documented how the review was conducted and which data sources were considered.

The remainder of this paper is structured as follows: Section 2 details the methodology used in this systematic literature review by highlighting the steps applied. Sections 3 through 9 summarize and categorize research on collaboration in software development before reflections on the systematic literature review are given in Section 10. The paper is concluded in Section 11.

2. Systematic Literature Review

This section motivates the use of systematic literature reviews in the software engineering domain and details the research protocol used in this study.

2.1. Motivation for Systematic Literature Reviews

The use of systematic literature review methods is common in many disciplines; however, software engineering is not one of them. A notable exception is presented by Brereton *et al.* [9]. In their paper "Lessons from applying the systematic literature review process within the software engineering domain", they promote using systematic approaches for literature reviews in all disciplines for several reasons:

- Systematic literature reviews are reproducible. In other words, the list of references included is not longer arbitrary but can be understood and reproduced by others.
- Missing an important reference is much less likely if all pertinent databases are queried systematically.
- Readers are given a foundation based on which they can either trust or not trust the literature review. If readers know how the results presented in the literature review were produced, they can judge for themselves whether or not they agree to the methodology.

2.2. Conducting a Systematic Literature Review

A systematic literature review consists of ten steps that can be divided into three main phases [9]:

Plan Review. The planning part of a systematic literature review is conducted before the actual review and consists of the following three steps:

- Specify Research Questions. Before a literature review can be conducted, its goal has to be identified. This is done by selecting one or more research questions that are to be answered by the review.
- Develop Review Protocol. The exact steps how literature for the review is selected and summarized have to be specified beforehand.
- Validate Review Protocol. In order to validate the research protocol, a pilot review can be conducted. This will show whether or not the review protocol is feasible.

Conduct Review. The actual review extracts the relevant data from selected data sources using the following steps:

- Identify Relevant Research. In this step, all databases that are to be queried are identified.
- Select Primary Studies. The content of the databases identified in the previous step has to be queried to identify all studies that help answer the research question.
- Assess Study Quality. Primary studies can be assessed regarding their quality and those studies that do not match a certain quality standard can be ignored for the literature review.
- Extract Required Data. The data that helps answer the research question is extracted from all primary studies that match the quality standards.
- Synthesize Data. The extracted data is synthesized to get higher level insights.

Document Review. After data has been synthesized, the report is generated:

- Write Review Report. The synthesized results are presented.
- Validate Report. After the report has been written, it can be validated by other individuals using the same review protocol.

2.3. Research Protocol

In the paragraphs below, the specific instance of a research protocol used in this systematic literature review is given. All choices are justified and the concrete steps are detailed. This protocol should enable other researchers to conduct the same review and to compare their results with the results presented in this paper.

2.3.1 Specify Research Questions

The research question used as starting point for this literature review is

R1: Which insights have researchers gained from empirical studies on collaboration in software development?

This literature review presents all results on collaboration in software development that have been obtained by empirically studying the process of software development in one way or another.

2.3.2 Develop Review Protocol

The review protocol consists of the steps outlined in Sections 2.3.4 through 2.3.10 below. These sections show how relevant research was identified and narrowed down to primary studies and which data was extracted.

2.3.3 Validate Review Protocol

Before conducting the actual review, the protocol was validated by querying the selected databases and looking at a sample of the results. It was established that the specified queries would lead to meaningful results and that the developed protocol was feasible to answer the research question.

2.3.4 Identify Relevant Research

The following five online databases were selected as relevant for research on collaboration in software development:

IEEE Xplore. Available at

`http://ieeexplore.ieee.org/Xplore/dynhome.jsp.`

ACM Digital Library. Available at

`http://portal.acm.org/dl.cfm.`

Google Scholar. The Canadian version available at

`http://scholar.google.ca/.`

CiteSeer. Searching for CiteSeer content using Google available at

`http://citeseer.ist.psu.edu/?form=googlesearch`

Inspec. Available at

`http://www.engineeringvillage2.org/`

The databases were initially visited on December 12th, 2008, and the links were verified on January 2nd, 2009.

This selection was influenced by the databases proposed by Brereton *et al.* [9], by searching the Internet for pertinent databases, and based on the experiences of the author from past literature reviews.

All databases were queried using the same query:

collaboration
AND "software development"
AND study.

The terms *collaboration* and *software development* were chosen to find all research literature in the selected databases related to collaboration in software development. Furthermore, the term *study* limits the choice to papers that describe studies. The quotes around *software development* ensure that only literature is chosen that includes the whole term rather than the single words *software* and *development*.

For all databases, the advanced search interface was used. Still, the search engines work quite differently. In particular, IEEE Xplore searches only meta data by default while the other search engines search full text or at least part of the full text. In addition, Google Scholar can be seen as a meta search engine for the other search engines as Google Scholar itself does not publish any articles but indexes most of the other literature databases. Also, some articles occur in more than one of the aforementioned databases. Due to these differences, the number of articles returned by the query differs significantly between the databases. When the queries were initially run on December 12th, 2008, the following amounts of articles were returned:

IEEE Xplore:	55.
ACM Digital Library:	1,766.
Google Scholar:	45,200.
CiteSeer:	421.
Inspec:	473.

2.3.5 Select Primary Studies

Based on the relevant research articles identified in the previous step, primary studies were selected according to their content. Many papers could be excluded by looking at the full title. That was the case when the topic of the article was obviously different from collaboration in software development. Furthermore, papers could be excluded based on their abstract, in particular when the paper presented a theory, a model, or a tool without evaluating it in a study. Also, a smaller number of articles were excluded based on the full text. In the end, 83 papers remained as primary studies.

2.3.6 Assess Study Quality

For the scope of this systematic literature review, the study quality was not assessed. In other words, all papers that were published and available in the aforementioned databases were believed to meet the quality standards. The only exception is given by papers that claimed to include a study but where the study turned out to be an arbitrary example case.

2.3.7 Extract Required Data

For all primary studies, the study design and the qualitative results were extracted. This was partly possible based on the abstracts, partly based on the conclusions and partly based on the full text.

2.3.8 Synthesize Data

The data extracted from the primary studies was synthesized by categorizing the results and comparing results against each other. Categories were introduced based on different phases of software development, based on different styles of software development, and based on new or existing tools. In particular, the following categories were identified:

Studies on Collaboration during Requirements Engineering. The subject of these studies is collaboration during the initial phase of a software project *(4 studies, see Section 3).*

Studies on Collaboration during Design. These studies look at collaborative design and collaborative development and evaluation of software architectures. The studies are mainly focused on communication and tool support *(8 studies, see Section 4).*

Studies on Collaboration in Global or Distributed Software Development. Most primary studies look at distributed or global software development in general. A lot of research effort has been spent on identifying problems prevalent in distributed development. More recently, studies have focused on recommendations for software projects by giving lessons learned in distributed development. Other focuses are on co-location versus distribution and social networks *(39 studies, see Section 5).*

Studies on the Use of Existing Tools for Collaboration. The use of existing tools for collaboration is examined in several studies. In particular, the uses of configuration management, instant messaging, and annotations have been investigated *(8 studies, see Section 6).*

Studies on the Use of New Tools for Collaboration. Many researchers have proposed new tools to support collaboration with different focuses: process, communication, awareness, sharing, annotations, and recommendations *(17 studies, see Section 7).*

Studies on Collaboration in Open Source. Studies on collaboration in open source development are largely focused on social network issues *(5 studies, see Section 8).*

Studies on the Role of Management for Collaboration. A few studies have looked at the role of management for collaboration in software development *(2 studies, see Section 9).*

2.3.9 Write Review Report

The result of this step is given by the paper at hand.

2.3.10 Validate Report

For now, this systematic literature review has not been validated yet.

3. Studies on Collaboration during Requirements Engineering

3.1. Introduction

The first step in a software project and also the first phase in the classical waterfall model is requirements engineering. A few studies have been conducted on the collaborative aspects of requirements engineering. The following paragraphs outline relevant research.

3.2. Importance of Requirements Engineering

The importance of requirements engineering for software projects was shown through a study conducted by Damian and Chisan [21]. They did a case study over thirty months in a large software development project that was undergoing requirements process improvements at the time of the study. They found several positive impacts of this improvement throughout the project lifecycle. The improvements also led to improvements of other project processes and ultimately to improvements in project negotiation and project planning. Further improvements could be observed in managing feature creep, testing, defects, rework, and product quality.

3.3. Communication during Requirements Engineering

Communication during requirements engineering is essential. A study on the role of communication in requirements engineering was done by Damian *et al.* [23], focusing on asynchronous discussion. In particular, their case study investigated the use of an Internet-based tool for asynchronous discussion of requirements issues prior to synchronous negotiations. The study was conducted in an instructional distributed environment that involved students in a global development task, with the inherent characteristics of geographical distance and multiculturalism. Three universities in Canada, Australia and Italy participated in the project by offering it as part of a software engineering course. The students in all three sites were assigned to six international project teams, each involving two countries. As a result of their study, the authors conclude that collaboration is more successful with asynchronous discussion based on a lower number of open issues after negotiation.

3.4. Global Requirements Engineering

Damian and Zowghi [24] looked at issues prevalent in global requirements engineering in a broader sense. They did a study on the impact of stakeholders' geographical distribution on requirements engineering in a multi-site organization. They collected data through the inspection of documents, observations of meetings and semi-structured interviews. Interviews were conducted with twenty-four stakeholders with several backgrounds ranging from product management and development engineering to customer support management and team leads as well as software engineers. The sites of the study were located in the United States, Australia, New Zealand and Europe.

As a result of their study, the authors were able to identify the following challenges in requirements engineering that can be attributed to the geographical distribution of stakeholders:

- Diversity in customer culture and business,
- Achieving appropriate participation of system users and field personnel,
- Lack of informal communication and diminished awareness of local working context,
- Reduced level of trust,
- Difficulty in managing conflict and having open discussions of interests,
- Difficulty in achieving common understanding of requirements,
- Ineffective decision-making meetings, and
- Delay.

3.5. Changing Requirements

The idea that social rather than technical aspects are the main challenge in requirements engineering is supported by the results of a study conducted by Chudge and Fulton [19]. Their study looked at requirements change practice in the British industry, putting an emphasis on safety-related software development. They conducted two case studies with industrial partners. As a result, they state that problems in the professional relationships between client and developer are mainly caused by social aspects. On a higher level, it is concluded that the opportunity of creating efficient software is marginalized by fixed cost contracts, lack of trust, and the high level of software control required to handle software change.

3.6. Summary

These studies on collaboration in requirements engineering point to the fact that collaboration and social aspects are extremely important in the first phase of every software project. Several challenges have been identified; however, there seems to be a lack of successful approaches of dealing with the aforementioned problems. The increasing geographical distribution of software projects across the globe introduces even more challenges.

4. Studies on Collaboration during Design

4.1. Introduction

After requirements engineering, the next phase in the classical waterfall model is design. Several studies have been conducted on challenges and potential solutions for collaborative design. Those studies are summarized below, categorized by problems in collaborative design, communication in collaborative design, and tools for collaborative design.

4.2. Problems in Collaborative Design

An early study looking at the problems related to collaborative design was described by Catledge and Potts [17]. They studied the conceptual design activities of a software project for three months and did follow-up observations and discussions afterwards to support their results. They found that convergence on a common design was extremely slow. The following reasons were observed for this problem:

- Difficulty in making critical allocation and interface design decisions,
- Repeated failure of reaching closure on key problems, and
- Persistent tension between the desire to follow a prescriptive development process and the urgency of delivering a working product.

The role of artifacts in collaborative design is highlighted by a more recent study by Brown *et al.* [10]. They conducted an ethnographic study of a development team, followed by the application of several kinds of qualitative analysis: activity system analysis, interaction analysis, grounded theory, and contradiction analysis. Focusing on artifacts, they found that sketches and design stories play critical roles in collaborative design and that artifacts support both creation and reflection. Artifacts also facilitate the resolution of contradiction and work at a level of consciousness that is below the level of self-awareness.

While these issues already turn design into a challenging task, the main challenge in collaborative design is communication. The following section looks at studies that focus on communication during design activities.

4.3. Communication during Design

In a study conducted by Wu *et al.* [81], five separate development groups were studied over a period of six weeks. In particular, the study consisted of shadowing, interviews, and communication event logging. A PDA-based application was used to enable data collection in real time. Wu *et al.* found that designers communicated frequently, using a wide variety of means of communication and collaboration. They usually preferred tools for general purpose to applications specific to one domain. Designers changed their physical location frequently throughout the day to support communication. At the same time, designers also frequently changed both their means and styles of communication to accommodate the needs of the situations at hand.

A similar behaviour was observed in a study by Dekel [27] that looked at the issue of distribution versus co-location. Interested in lessons to be learned from co-located meetings, he observed two design meetings with attendees from both academia and industry which were to study a requirements document and to collaborate for the corresponding design. The author found that designers constantly shifted between working with the entire group, working with a smaller team, and working individually. Also, while they were working with others, their focus often shifted away from their team for short periods of time.

In a controlled experiment, Babar *et al.* [1] compared distributed meetings versus face-to-face meetings. The experiment involved 32 teams of three third and fourth year undergraduate students. The quality of scenarios for architecture evaluation was used to determine the quality of the meetings. The authors found that the quality produced by distributed teams using a groupware tool was significantly better than the quality of scenarios produced in a face-to-face setting. However, questionnaires indicated that most participants preferred the face-to-face situations and that distributed meetings were perceived as being less efficient. Babar *et al.* conclude that distributed design meetings are effective but that tool support must be of a high standard as participants will not find these meetings acceptable otherwise.

Another study on collaborative design is given by Ocker and Fjermestad [58]. They looked at communication differences in virtual design teams using a multi-method approach. In particular, they studied four high performing and four low performing virtual design teams that were fully distributed. Data was collected from the asynchronous communication of the team members.

While it was found that the different teams were similar in terms of the number of messages they exchanged and in the amount of communication devoted to team coordination, supportive commentary, and other topics, there were differences in the message contents. In particular, high performing teams communicated more words, i.e. exchanged longer messages. They also spent less time on brainstorming activities. Instead, high performing teams engaged more in critical commentary and active debate than low performing teams. They had more in-depth discussions in the form of argumentation, as ideas were developed through an interactive debate of the advantages and disadvantages of issues. This debate resulted in the need for summaries, which also became intermediate steps in the process of writing the design report.

4.4. Tools for Collaborative Design

The issue of distribution versus co-location in collaborative design is addressed by a study on bridging the gap between physical and virtual media for distributed design collaboration described by Everitt *et al.* [30]. They introduce Distributed Designer's Outpost, a remote collaboration system based on a collaborative web site design tool that employs physical post-it notes as interaction primitives. The authors extended the system for synchronous remote collaboration and introduced two awareness mechanisms, namely transient ink input for gestures and a blue shadow of the remote collaborators. The system was informally evaluated with six professional designers who liked the prospect of physical remote collaboration but found some challenges in the interaction with shared artifacts.

Haynes *et al.* [36] introduced a collaborative environment created to support distributed evaluation of complex system architectures. Their approach couples an interactive architecture browser with collaborative walkthroughs of an evolving architectural representation to facilitate involvement of stakeholders from different physical locations. A preliminary evaluation identified several benefits such as low overhead, support for dynamic development and a common mental model, communication support and increasing architecture clarity. On the other hand, barriers such as cultural norms, reaching a critical mass, and the elicitation of measurable contributions were observed.

4.5. Summary

A lot of challenges come with collaboration in design, and these challenges become even more difficult when designers are geographically distributed. Artifacts play a critical role in collaboration. Several studies found that designers use many different and frequently changing means of communication which requires sophisticated tool support. While some tools for collaborative design have been developed, they still struggle to meet these requirements.

5. Studies on Collaboration in Global or Distributed Software Development

5.1. Introduction

Most of the studies identified during the systematic literature review have a general point of view on distributed or global software development without distinguishing between the different phases. In particular, many studies have been conducted to identify problems related to the geographical distribution of software developers. In that context, the focus of several studies was on co-location versus distribution and on social networks of developers. Other studies highlight lessons learned from distributed software development and look at the progress in this domain over the last decade.

5.2. Problems in Global or Distributed Software Development

A study that summarizes the problems that software development teams face in distributed or global projects is given by Casey and Richardson [12]. They did qualitative research in two organizations in Ireland and found several factors that have become part of everyday development for those involved in global or distributed software development, and that need to be explicitly addressed by management in order to avoid serious problems. These factors are:

- Use of communication tools,
- Project management,
- Process engineering,
- Technical ability,
- Knowledge transfer, and
- Motivational issues.

The main problem in global or distributed software development is that of communication and coordination. In their paper on "Splitting the Organization and Integrating the Code", Herbsleb and Grinter [40] report on a case study on what they identified as the most difficult part of geographically distributed software projects: integration. They conducted ten interviews with managers and technical leads to gather information about perceived challenges of multiple site development, followed by a second round of eight interviews that focused on integration explicitly. Their results show that coordination problems were greatly exaggerated across sites, largely because of the breakdown of informal communication channels. They conclude that distributed development may imply the necessity of stable plans, processes, and specifications. On the other hand, the inherently unpredictable aspects of projects require communication channels that can be invoked spontaneously.

Another aspect of distributed development that has been observed several times is the impact of distribution on the development speed. Herbsleb *et al.* [38] studied both survey data and data from the source code change management system to model the extent of delay in a multi-site software development organization, and to explore several possible mechanisms for this delay. They also measured differences in co-located and distributed communication patterns and analyzed the impact of these variables on delay. Their main finding is that compared to co-located work, distributed work takes much longer and requires more people for work of equal size and complexity.

This study was replicated in a different organization with a different product and different sites to confirm the main findings by Herbsleb and Mockus two years later [37]. Again, data from the source code change management system and survey data was used to model the extent of delay in distributed software development. In accordance with the earlier results, they found that distributed work items appear to take about two and a half times as long to complete as similar items where all developers are co-located. They also found that distributed work items involve more people than comparable co-located work items, and that the number of people involved is strongly related to the calendar time to complete a work item.

Potential reasons for the identified delay are pointed out by Layzell *et al.* [48]. They conducted a study of industrial practice to better understand the communication processes and mechanisms used by practitioners in the software engineering process by doing a series of interviews with participants from distributed software development and maintenance projects. As organizational issues in distributed development, they identified the difficulties to achieve consensus and the variations in tools and support infrastructure. As communication issue, they observed that some sites communicated too much by spending more time communicating than doing useful work. It was also noted that project members tended to be less committed to a project when a lot of the communication took place by email as the relative anonymity of colleagues gave rise to either over-confidence or under-confidence in their perceived level of expertise and ability.

This result is confirmed by a study of Herbsleb *et al.* [41]. They measured site interdependence as well as differences in co-located and distributed communication patterns and found a significant relationship between delay in distributed work and the degree to which remote colleagues were perceived to help out when workloads are heavy. It is concluded that this finding is particularly troubling in light of the finding that workers generally believed they were as helpful to their remote colleagues as to their local colleagues.

Complementary to that, Woit and Bell [79] found that developers believe themselves significantly less effective in a distributed environment because of lack of traditional non-verbal cues. The authors conducted a survey to explore effectiveness of non face-to-face communication with fourth year computer science students as well as graduate education students engaged in distance learning courses that required them to work together to complete software development tasks.

Other difficulties related to the perception of distributed work are pointed out in a study by Begel [2]. He did an interview based study of a product team at Microsoft that was made up of around 300 software engineers working at two locations in the United States and one location in India. 26 members were interviewed using questions that were designed to record the work-related social networks of each participant. Begel found that many of the issues mentioned by different team members were highly asymmetric. In particular, the issues that one group perceived as harming intergroup coordination were not the same issues that the other group felt were important.

However, communication is not the only crucial factor to the success of a distributed software development team. Hause *et al.* [35] found that the timing of specific actions can have a considerable impact on a team's performance as well. They investigated the interactions of distributed student teams that were involved in a software development project that was part of a computer science course at two universities.

Other factors were identified by Panjer *et al.* [61]. They conducted a field study that used interviews and informal observation of a distributed software team and found three key interesting themes in their qualitative analysis: proximity, modification request authoring patterns, and uncooperative behaviours.

Further problems are pointed out by Damian *et al.* [22], who conducted a case study over four months at an IBM software lab and observed the collaboration patterns of a distributed development project team. Data was collected through the inspection of project documentation, interviews with team leads, attendance of project meetings and informal conversations with developers. Damian *et al.* found that the organizational culture has an effect on how developers are made aware of changes and that communication-based social networks revolving around particular work items are dynamic throughout development. Furthermore, they recommend that awareness needs to be maintained in infrastructures of work. From their study, they conclude that information overload and communication breakdowns contributed to problems in cooperation and coordination.

Also looking at the role of awareness in distributed software development is a study conducted by Gutwin *et al.* [32]. They interviewed developers, read project documentation, and looked at project artifacts from three successful open source projects and found that distributed developers need to maintain awareness of one another. In particular, developers maintain both a general awareness of the entire team and a more detailed awareness of people they plan to work with. Although there are several sources of information, this awareness is maintained primarily through text-based communication. These textual channels have several characteristics that help to support the maintenance of awareness, as long as developers are committed to reading the lists and to making their project communication public.

However, making communication public generates a gap between private and public work in collaborative software development, as observed in a study by de Souza *et al.* [26]. They conducted an ethnographic study for eight weeks, making observations and collecting information about several aspects of a software development team. Additional data was collected from manuals and process descriptions as well as training documentation for new developers and problem reports. They conclude that the transition from private to public work needs to be more carefully handled. This transition is currently dealt with using different formal and informal work practices that are adopted by the developers to allow a delicate transition so that developers are not largely affected by the emergent public work.

Finally, globalization of software development leads to cultural differences between the developers which can also impede the work. Some difficulties caused by cultural differences were pointed out in a study by Halverson *et al.* [33]. Using data gained through interviews and the inspection of change request systems at IBM, they found a list of social issues:

- Conflicting work practices whether a bug is really a bug,
- Avoiding breaking another developer's code unnecessarily,
- Figuring out what has caused broken code and who to talk to about it, and
- Wasting time treating something as a technical problem that was really a social or cultural problem.

Additional challenges were identified by Huang and Trauth [45]. They conducted interviews with Chinese IT professionals and found the following cross-cultural challenges:

- Complexity of language,
- Culture and communication styles and work behaviours, and
- Cultural understandings at different levels.

Cultural patterns in software process mishaps were reported by MacGregor *et al.* [50]. They gained data from meetings with project managers and personnel and by exploring the space of global outsourcing and sub-contracting. The patterns they found are: Yes (but no) Pattern, Proxy Pattern, We'll-take-you-literally (Anti) Pattern, We're-one-single-team (Anti) pattern, and The-customer-is-king (Anti) Pattern. Cultural issues were also the subject of a study by Dalberg *et al.* [20], who did a case study on cross-cultural collaboration in the European Union on the requirements and design phase of a software project. They identified a number of collaboration and work process goals as well as culture and context goals.

5.3. Co-Location versus Distribution

Several studies focused on the issue of co-location versus distribution is software development, often through experiments. These studies are summarized here.

To answer the question on how radical co-location helps a team succeed, Teasley *et al.* [72] conducted a field study of six co-located teams by tracking their activities, attitudes, use of technology and productivity. They found that the teams showed a doubling of productivity once they were co-located in so-called warrooms. Among other things, teams had easy access to each other for both coordination of their work and for learning, and the work artifacts they posted on the walls remained visible to all.

In the study design of Huang and Ocker [44], teams were not radically co-located but partially distributed. They conducted a study with twelve student teams within a major university where the majority of team members were co-located on the main campus while the remaining members were located at either one or two branch campuses. The in-group/out-group effect that is associated with geographical distance was identified as one of the unique challenges of partially distributed teams.

On the other hand, a study by Bos *et al.* [8] looked at a phenomenon referred to as "co-location blindness" in partially distributed groups. They did a set of experiments to study how partially distributed teams collaborate when skill sets are not distributed equally. Their experiments revealed that participants whose skills were locally in surplus performed significantly worse as they experienced "co-location blindness" and failed to pay enough attention to collaborators outside of their location. In contrast, remote participants whose skills were scarce inside the co-located location performed well because they were able to charge a high price for their skills.

Another study by Bos *et al.* [7] also examines characteristics of partially distributed teams, in particular the effects of relocation. They conducted an experimental study looking at how relocation affected the collaboration patterns of partially distributed work groups. The locations of some of the participants were switched about halfway through the experiment in order to see what effect that would have. Participants who changed from being isolated telecommuters to co-locators very quickly formed new collaborative relationships. On the other hand, participants who were moved out of a co-located room had more trouble adjusting, and tried unsuccessfully to maintain previous ties. Overall, co-location was a more powerful determiner of collaboration patterns than previous relationships.

Hinds and McGrath [43] studied the social structure in geographically distributed teams compared to the social structure in co-located teams. They used a web-based survey of geographically distributed and co-located development teams within a multi-national corporation. These surveys were followed by interviews intended to provide a richer understanding of the teams, their work processes, and the challenges they faced. It turned out that although flat hierarchy is usually associated with more smooth coordination in co-located teams, the opposite is true for distributed teams. Rather, an informal hierarchical structure was associated with more smooth coordination in distributed teams.

5.4. Social Networks

A recent trend in research on collaborative software development is the analysis and interpretation of social networks of developers. Pertinent literature is summarized in this section.

Milewski *et al.* [52] conducted an interview study investigating the collaborative information seeking and information sharing practices of a global software testing teams in a multi-site organization. Interviews were conducted with 13 global software team members. The authors found that a site located in Europe was used as a temporal bridge to help managing time zone differences between the United States, China and India. All sites utilized this bridge for critical, synchronous information seeking. The interviews also suggested that bridging can be a taxing job and that the success of the bridging arrangement depended upon an intricate balance of temporal, infrastructure and cultural factors.

A similar result was produced by a study by Cataldo and Herbsleb [13]. They obtained data from a geographically distributed software development project covering more than three years of activity. They looked primarily at modification requests and Internet Relay Chat (IRC). It turned out that over time a group of developers emerged as the liaisons between formal teams and geographical locations. In addition to handling the communication and coordination load across teams and locations, those engineers contributed the most to the development effort.

This result was reproduced by the same authors in a study of a geographically distributed software development projects from a distinct company [14]. The authors found that the definition of formal roles had an important impact on patterns of communication across development locations and communication across site was formalized. However, in this second study, the developers involved in the cross site communication and coordination activities were not as productive.

The emergence of liaisons was also observed by Chang and Ehrlich [18]. They did a study that used social network analysis to study the informal communication patterns in three successful global software teams and found that technical leaders acted as brokers to coordinate work across tasks and sites in self-organizing sub groups. Moreover, the communication was influenced by personal networks, the awareness of tasks and accessibility, which in turn affected the social climate of the team.

A study by Urdangarin *et al.* [75] suggests the use of cultural ambassadors and extreme programming to help overcome cultural barriers among remote teams. The study is based on a student-based software development project that was instrumented for data collection. Two questionnaires were used to obtain insights: one with questions related to frequency and importance of the communication with other team members; the other one to collect data related to the experience of one of the remote teams with the use of extreme programming methodologies in a global software development environment. The authors found that remote teams that used extreme programming methods used a more direct and frequent communication style and that extreme programming can help increase trust among the remote teams. Also, cultural ambassadors were able to help overcome cultural barriers among teams.

However, ambassadors and liaisons are not sufficient to guarantee successful team work. Using evidence from globally distributed software development teams from SAP and LeCroy, Oshri *et al.* [59] found that while face-to-face meetings may be invaluable, they do not overcome all the challenges experienced in geographically distributed teams. Managers must also prioritize activities before and after these meetings to help team members stay connected.

Finally, Meneely *et al.* [51] present another use case of social networks in software development. They built and validated a prediction model with data from an industrial product in the telecommunications domain. Data was collected from three annual releases of a large, mature networking product. It turned out that developer networks are useful for failure prediction early in the development phase and that they provide a useful abstraction of the code modification data.

5.5. Lessons Learned

While a lot of research has been conducted on problems and challenges in global and distributed software development, there is also a significant amount of research on lessons learned from geographically distributed projects. This research is presented in this section.

Herbsleb *et al.* [42] report on the experiences of Siemens Corporation in nine globally distributed software development projects. These projects represented a range of collaboration models, from co-development to outsourcing of components to outsourcing the software of an entire project. The authors conducted semi-structured interviews to collect data. The following lessons learned were identified:

- Communicate the work that has to be done,
- Be involved in project management,
- Utilize direct communication,
- Use a single development environment, and
- Travel.

This list can be continued using the results of a study conducted by Purvis *et al.* [63]. They examined a global software project involving 34 students in Germany and New Zealand who worked on a generic framework for real time multi player games. Along with several problems such as difficulties in having a shared schedule and architectural issues due to lack of clear understanding of the requirements at the outset, lessons learned were identified:

- Identify the role of each group member and their responsibilities with regard to the counterpart team,
- Adopt a set of tools acceptable to all groups for communication and development,
- Provide a set of guidelines for communication protocols and the development process, and
- Be flexible and adaptable when new requirements are identified.

Further lessons learned were identified through a study by Thissen *et al.* [74]. Using a case study approach, they examined collaboration processes used by RTI International in three application programming projects. The collaboration mechanisms used consisted of simple conference calls, email through webcasts, and collaboration websites. The following lessons learned were presented:

- Allow teams to choose their own communication tools from a variety of options,
- Insist on frequent communication among all members, including some synchronous interaction,
- Provide shared file storage to facilitate team interaction,

- There is no need for extensive travel, although early face-to-face contact is beneficial,
- Paired programming can take place across long distance, through remote access or collaboration tools, and
- Communication tools do not have steep learning curves; team members adapt quickly, especially if they have had a voice in the selection of tools.

While most of these lessons learned complement each other, there are some contradictions, e.g. regarding the need to travel. Apparently it depends on the project culture whether extensive travel is necessary or not.

A study with stronger focus on lessons learned regarding processes in globally distributed software development is presented by Ramasubbu and Balan [64]. They collected data by gathering information on forty-two completed projects in a period over two years. During this time, one of the authors was present in the field and observed software development processes using an ethnographic observation approach as well as structured interviews. It was found that even in high process maturity environments, distribution significantly reduces development productivity and has effects on conformance quality. However, these negative effects can be mitigated through the deployment of structured software engineering processes.

An example of successful process adjustments in distributed software development was reported by Vitale [78]. He described the two releases of Interactive Solution Marketplace, version 1.0 and 2.0. Version 1.0 was deployed late, provided only a small fraction of the features business stakeholders were expecting and was over budget. For version 2.0, new management was brought in to deliver on the original expectations and to provide a set of new features originally planned for the second release of the offering. The 2.0 project met and exceeded the expectations. The project launch occurred two weeks ahead of schedule and under budget. There were ten times fewer defects uncovered during system test than anticipated and no defects with high severity reported during the first three months in production. As key factors for this improvement, Vitale identified a requirements-driven system engineering methodology, team interactions and project management tools, and on-demand resourcing.

Primarily looking at communication in global software development is a study by Lindqvist *et al.* [49]. They conducted interviews at Ericsson, asking questions about product structure, organizational structure, organizational structure, communication, and different ways of working. An important issue seen in the culture of the organization was the recognition of the need for communication and travel. A lack of continuity in communication and of informal communication made it hard at remote sites to identify important issues. This led to an underestimation of problems at remote sites. Co-workers unaccustomed with distributed development often used mail for communication with remote sites. However, the use of asynchronous tools such as mail created delays in communication.

The same result was produced by an empirical study of a distributed student project located at two different geographical sites conducted by Johansson *et al.* [47]. They conclude that direct contact is more productive than written communication and that communication in general has to be prioritized.

The important role of communication in distributed software development is underlined by a study by Birnholtz *et al.* [6]. Using interviews and observations in four open plan offices, they identified the important role of attention in the management of confidentiality and solitude. The public nature of paying attention allowed developers to build understandings of what objects in a space are legitimate targets for attention and allowed developers to advertise their interest in interaction. The lack of attention in distributed development calls for sophisticated tools that help maintain awareness.

5.6. Global and Distributed Software Development Today

While most of the studies described above identify global distribution as a major problem for software development, there are a few recent studies that suggest that the lessons learned mentioned in the previous section have been applied successfully. In fact, it appears that distributed development does not have to take double time anymore like it was found in studies from the early 2000s.

In a study on the FreeBSD project, Spinellis [69] examined developer location data, the configuration management repository, and records from the issue database to examine the extent of global development and its effect on productivity, quality, and developer cooperation. He found that global development allows round-the-clock work, but that there are some differences between the types of work performed at different regions. However, the effects of multiple distributed developers on the quality of code and productivity are negligible. Mentoring appears to be sometimes associated with developers living closer together, but ad-hoc cooperation seems to work fine across continents.

This result is confirmed by a recent study conducted by Nguyen *et al.* [57]. In their empirical study of communication structures and delay, as well as task completion times in IBM's distributed development project Jazz, they used the number of sites, response time and resolution time as data constructs. The main finding is that distance does not have as strong of an effect on distributed communication delay and task completion as seen in past research.

5.7. Summary

Global or distributed software development introduces many problems for developers, in particular related to communication. Many studies have been conducted to identify and classify these problems. Of particular interest to researchers are the issues of co-location versus distribution in partially distributed teams and social networks of developers and their implications. Several studies have reported on lessons learned in distributed development and a few recent studies suggest that these lessons have been applied as the differences between the quality of co-located and distributed development seem to decrease.

6. Studies on the Use of Existing Tools for Collaboration

6.1. Introduction

Several tools have been proposed to support collaboration in software development in many different ways. Before looking at the development of new tools in Section 7, this section summarizes studies on the use of already existing tools for collaboration. The tools examined include configuration management, instant messaging, annotations, and API.

6.2. Configuration Management

Grinter [31] conducted a naturalistic study of one organization's use of a configuration management tool to coordinate the development of a software product. She found that the developers used the configuration management tool routinely to reduce the complexities of coordinating their development efforts. However, configuration management systems make it difficult to represent work and they offer multiple levels at which they operate. The possibilities for coordination they provide are limited, as is their role in supporting a model of work.

A more recent case study on the use of software configuration management over a global software development environment was conducted by Pilatti *et al.* [62]. The study was carried out at a multinational organization that has offshore software development centres in Brazil, India and Russia. The following lessons were learned from this study:

- The work breakdown in distributed projects should minimize dependencies between geographically distributed teams,
- Distributed development projects should work with only one instance of configuration management,
- Put all configuration items required for a build under configuration management,
- Distributed development projects with centralized configuration management should define one build coordinator,
- Establish and clarify all main concepts on configuration management discipline, before actually starting the development,
- Even with experienced teams in distributed development, the configuration management engagement in the beginning should be prioritized, and
- Always plan baselines and document them in the project's configuration management plan.

6.3. Instant messaging

Instant messaging has been used for quite a while in professional software development and has also been integrated into development environments recently. A study by Isaacs *et al.* [46] logged thousands of workplace chat conversations and evaluated their conversational characteristics and functions. It was found that the primary use of workplace instant messaging was for complex work discussions. Less than a third of the conversations were simple, single-purpose interactions and only about one third were about scheduling or coordination. Moreover, people rarely switched from instant messaging to another medium when the conversation got complex. Isaacs *et al.* also identified two distinct styles of use. Heavy instant messaging users mainly used it to work together, to discuss a broad range of topics via many fast-paced interactions per day, each with many short turns and much threading and multitasking. On the other hand, light users mainly used instant messaging to coordinate and for scheduling, via fewer conversations per day that were shorter, slower-paced with less threading and multitasking.

Similar results were found by Handel and Herbsleb [34]. They did an empirical study of a synchronous messaging application with group-oriented functionality designed to support teams in the workplace. In particular, the tool supported group chat windows that allow members of a group to communicate with text that persists for about a day. The authors examined the experience of six globally distributed work groups who used the tool for more than a year. It turned out that the group functionality was used primarily for bursts of synchronous conversations and occasional asynchronous exchanges. The content was primarily focused on work tasks, and negotiating availability, with some non-work topics and humour. Nearly all groups were remarkably similar in the content of their group chat, although the research group chatted far more frequently than the others.

However, instant messaging is only one of the many ways software developers use for communication and the interplay of different communication tools is highly important. In a study aiming at implications for the design of collaboration and awareness tools, Cataldo *et al.* [16] analyzed data from a software development project of a large distributed system. The data covered a period of almost three years of development activity and the first four releases of the product. Their unit of analysis was a modification request. Developers also used tools such as Internet Relay Chat (IRC) and the modification request tracking system to interact and coordinate their work. The authors looked at the stability of coordination requirements over time, examined the role of congruence in task performance, and examined the evolution of congruence between coordination requirements and actual communication over time. As results, they found that coordination requirements were highly volatile, and frequently extended beyond team boundaries. Congruence between coordination requirements and coordination activities shortened development time. Developers, particularly the most productive ones, changed their use of electronic communication media over time, achieving higher congruence.

6.4. Annotations

Another mechanism that has been adapted by software developers for collaboration are annotations. Cadiz *et al.* [11] conducted a case study of annotations created by members of a large development team using Microsoft Office. Approximately 450 developers created 9,000 shared annotations on more than a thousand documents over ten months. Several issues related to annotations in collaboration were identified:

- The contributions have to be unobtrusive but accessible and they have to inform without being overwhelming,
- Higher and lower priority information for different developers at different times has to be separated, and
- Different roles of developers have to be considered, such as document owner, annotation creator, and respondent.

The use of annotations in source code was analyzed in a study conducted by Storey *et al.* [71]. They conducted an empirical study that explored how task annotations embedded within the source code play a role in how software developers manage personal and team tasks. Data was collected by combining results from a survey of professional software developers, an analysis of code from open source projects, and interviews with software developers. It was found that task management is negotiated between the more formal modification request systems and the informal annotations that developers add to their source code. Also, annotations can have different meanings and are dependent on individual, team and community use.

6.5. API

The use of APIs for collaboration was studied by de Souza *et al.* [25]. They conducted a qualitative study on how practitioners use APIs in their daily work. The methods for data collection comprised non-participant observations and semi-structured interviews, which involved one of the authors spending eleven weeks at the field site. They also collected meeting invitations, product requests for software changes as well as emails and instant messages exchanged among the software engineers. For collaboration, three major limitations of APIs were identified: incompleteness, instability, and the lack of awareness.

6.6. Summary

Software developers have used different kinds of existing tools to facilitate collaboration with mixed success. While tools like APIs and configuration management have major limitations when used for collaboration, annotations and instant messaging seem to support collaboration well.

7. Studies on the Use of New Tools for Collaboration

7.1. Introduction

After looking at the use of existing tools for collaboration in the last section, this section summarizes studies on the use of new tools that were developed to improve collaboration in software development. In particular, tools for process, communication, awareness, sharing, annotations, and recommendations have been proposed.

7.2. Process

In order to support collaboration in software development, Pandey *et al.* [60] proposed a new framework of Tightly Coupled Engineering Team (TCET) process to facilitate collaboration in a team and thereby improve productivity and software quality. They applied this concept in a twenty-one months long software project for the development of a test automation software suite. The evaluation was done according to function points per work month and using interviews. The authors found quantitative and qualitative evidence that the process contributed to increased productivity and software quality, and they also found an increase in the intra team training, ownership, and knowledge flow within the team. Project risk was reduced without harm to schedule contrary to popular perception that such collaboration may lead to redundancies and thus cost and schedule overrun.

Focusing on building consensus in collaborative software development is an extension called CONFER described by Wong *et al.* [80]. In their study, they evaluated the conflict resolution extension through an experiment in which users collaboratively worked on a design problem. This evaluation aimed at assessing how well the model assisted participants in removing conflicts, and their level of satisfaction with this approach. From both empirical and conceptual perspectives, the technique showed promise in providing at least some of the techniques needed to build collaboration tools based on more cooperative models of work.

A framework for the assessment of the impact of technical and work dependencies on software development productivity was proposed and evaluated by Cataldo *et al.* [15]. They used data from the first four releases of a company's main product which was a large distributed system. Overall, the data covered a period of more than three years of development activity. The unit of analysis was the modification request. Their empirical evaluation of the congruence framework shows that when developers' coordination patterns are congruent with their coordination needs, the resolution time of modification requests was significantly reduced. Furthermore, the analysis highlights the importance of identifying the set of technical dependencies that drive the coordination requirements among software developers. On the other hand, call and data dependencies appear to have far less impact than logical dependencies.

7.3. Communication

Several tools for communication in collaborative software development have been proposed. An early introduction of instant messaging into the software development workplace is described by Herbsleb *et al.* [39]. They introduced the tool and gathered usage data via automatic logging on the server, which included logins, logouts, joining and leaving groups, as well as group chat messages. In order to preserve users' privacy, they did not log instant messages. About two dozen semi-structured interviews were conducted with users, and two small focus group sessions were held to get feedback. The evaluation showed that the combination of features had some potential to help distributed teams overcome the lack of context and absence of informal communication, two of the problems that make distributed work difficult. However, there were adoption issues to keep such tools out of many workplaces for some time, in particular the perception that chatting is not real work.

In the years after the study by Herbsleb *et al.*, instant messaging has been adapted more and more. Scupelli *et al.* [68] examined the use of instant messaging by redesigning an instant messenger so that it was able to handle multiple projects and teams. Their redesign used automatic project status logging to show active project related files and team members. In a preliminary evaluation experiment, participants working collaboratively with different partners on two projects found the redesign and the original tool to be equally usable and informative but the participants using the redesign reported less workload stress.

Čubranić and Storey [77] describe a study in which pairs of students used a prototype of a collaborative development environment to work on a programming assignment. The goals of this study were to evaluate the effectiveness and usability of the new features and to determine requirements for future communication support. The authors found that code sharing through upload and download into and from a versioned repository was easy to use even for novice programmers. However, the lack of a repository activity indicator meant that the participants had to compensate by verbally monitoring each other's progress. Frequent references to specific locations in the code suggest that computer mediated communication should be integrated into the development environment for easy remote gesturing and annotation of task artifacts.

Yamashita and Ishida [83] study a tool that is particularly useful for global software development where there are language barriers between developers. Their hypothesis is that machine translation between natural languages works better than communicating in a shared second language. To test their hypothesis, they conducted an experiment that was separated into two phases. The first half was conducted as part of the Intercultural Collaboration Experiment, jointly hosted by Chinese, Japanese, Korean, Malay, and Thai universities and research institutes. The other half was conducted in Japan. The two phases differed only in the site at which the experiment took place and in whether the authors conducted detailed interviews, which were performed only in the second phase. In the experiment, pairs sat in different rooms. Each pair was given tasks that were carried out twice in English and twice in their native languages using machine translation. In the evaluation, it turned out that the tasks were disrupted because the translations did not translate the same terms consistently throughout the conversation. To overcome asymmetries and inconsistencies in machine translation-mediated communication, participants tried to minimize exchanges and used exactly the same referring expressions throughout the experiment. The approach is not sophisticated enough yet to support collaboration effectively.

7.4. Awareness

A tool that has been well researched in the area of awareness is the workspace awareness tool Palantír described by Sarma *et al.* [67]. They performed two user experiments directed at understanding the effectiveness of a workspace awareness tool in improving coordination and reducing conflicts. They evaluated the tool through text-based assignments to avoid interference from the impact of individual differences among participants. Upon this baseline, they conducted a second experiment, with code-based assignments, to validate that the tool's beneficial effects also occured in the case of programming. Time and the number of found and resolved conflicts were used to measure effectiveness. The authors found quantitative evidence of the benefits of workspace awareness in software configuration management. In particular, it improved coordination and conflict resolution without inducing significant overhead in monitoring awareness cues.

Another study on Palantír is also described by Sarma *et al.* [66]. They conducted a preliminary study to find out if Palantír helps developers detect indirect conflicts early on so that they can improve their ability to coordinate their work, and the quality of the code that results from the collaborative effort. Two pilot studies were done to address this question, one in which the authors compared results with and without Palantír and a second in which they compared a version of Palantír with support for both direct and indirect conflicts to a version of Palantír with support for only direct conflicts. Participants in these studies were asked to complete tasks. The authors found that notifications provided by Palantír were actively used by the developers, with the auxiliary result that indirect conflict detection mattered, even when developers were already notified of direct conflicts.

Another awareness tool is given by FASTDash and described by Biehl *et al.* [4]. FASTDash is an interactive visualization that seeks to improve team activity awareness using a spatial representation of the shared code base that highlights team members' current activities. With FASTDash, a developer can quickly determine which team members have source files checked out, which files are being viewed, and what methods and classes are currently being changed. The authors studied the usefulness of FASTDash using a pre/post observation design where two observations were done before the introduction of FASTDash and two observations after it had been introduced. They found that FASTDash improved team awareness, reduced reliance on shared artifacts, and increased project-related communication. Additionally, the team that participated in the field study continued to use FASTDash.

7.5. Sharing

Sharing of artifacts and knowledge is another issue in collaborative software development. Dekel and Herbsleb [28] introduced eMoose, a group memory-aid for software development that addresses the lack of community generated knowledge by visually pushing annotated knowledge from invocation targets into the invoking code. eMoose was evaluated in a small lab study where 15 subjects performed several bug-fixing tasks involving unfamiliar APIs in which the solution was hidden in particular directives while many other directives caused potential distraction. Each subject was allowed to use eMoose in a random half of the tasks. Users without eMoose were more likely to miss important calls and directives in the documentation of these targets, while users with eMoose tended to find these calls and directives faster and were more likely to successfully complete the task. On the other hand, experienced developers were rarely distracted by indicators over calls that appeared irrelevant, or by directives that were not relevant to the task.

A framework for supporting collaboration in multiple display environments was described by Biehl *et al.* [3]. Their interaction framework, IMPROMPTU, allows users to share task information across displays via off-the-shelf applications, to jointly interact with information for focused problem solving, and to place information on shared displays for discussion and reflection. The framework also includes a lightweight interface for performing these and related actions. A three week field study of the framework was conducted in the domain of face-to-face group software development. During this field study, teams utilized almost every feature of the framework in support of a wide range of development-related activities. The framework was used most to facilitate opportunistic collaboration involving task information.

Millen *et al.* [54] introduced ActivityExplorer, a collaboration technology that is based on the support of lightweight, informally structured, opportunistic activities featuring heterogeneous threads of shared items with dynamic membership. A detailed analysis of user behaviour was done during a five month field study. Four patterns of media use were identified: communication, exchanging mixed objects, coordinating, and semi-archival filing.

7.6. Annotations

Dogear and TagSEA are the two tools dealing with annotations that were proposed for collaborative software development. Dogear is a social bookmarking service for companies and was introduced by Millen *et al.* [53]. They describe an eight week field trial of Dogear based on user activity data obtained through log file analysis and a user survey focusing on the benefits of the service. The feedback from the user trial was quite positive and suggested several promising enhancements to the service.

TagSEA is a collaborative tool to support asynchronous software development that was proposed by Storey *et al.* [70]. The authors' goal was to develop a lightweight source code annotation tool that enhances navigation, coordination, and the capture of knowledge relevant to a software development team. The design was inspired by combining waypoints from geographical navigation with social tagging from social bookmarking software to support coordination and communication among software developers. A preliminary evaluation with a small group of programmers at two sites showed encouraging results and patterns of usage already started to emerge.

7.7. Recommendations

Studies on two new tools for recommendations in collaborative software development were identified in the literature review: Hipikat and Emergent Expertise Locator. Hipikat is described by Čubranić *et al.* [76]. It provides developers with efficient and effective access to the group memory for a software development project that is implicitly formed by all of the artifacts produced during the development. This project memory is built automatically with little or no change to existing work practices. The authors performed an exploratory case study evaluating whether software developers who are new to a project can benefit from the artifacts that Hipikat recommends from the project memory. To assess the appropriateness of the recommendations, they investigated when and how developers queried the project memory, how developers evaluated the recommended artifacts, and the process by which developers utilized the artifacts. It turned out that newcomers did use the recommendations and their final solutions exploited the recommended artifacts, although most of the Hipikat queries came in the early stages of a change task.

Emergent Expertise Locator is a tool that recommends emergent teams based on how developers change software artifacts. It was introduced by Minto and Murphy [55]. Emergent Expertise Locator proposes experts to a developer within their development environment as the developer works. It was evaluated using a validation method that involved selecting a bug of interest and recreating the development state at that time by considering only source code revisions that were committed before the bug was closed. The authors used a determination of the files required to fix the bug to populate the matrices and determine the recommendations. Emergent Expertise Locator produced, on average, results with higher precision and higher recall than an existing heuristic for expertise recommendation.

7.8. Summary

Several new tools have been proposed to help software developers with collaboration and coordination. Some of those tools in particular related to awareness, sharing of knowledge and artifacts, annotations, and recommendations are promising. However, most tools have only been evaluated preliminarily. Tools related to communication seem to be most promising if they facilitate instant messaging one way or another and tools for process improvements usually consist of guidelines rather than actual pieces of software.

8. Studies on Collaboration in Open Source

8.1. Introduction

Collaboration in open source projects is particularly interesting to researchers since the infrastructure is significantly different from industry projects and since there is usually only very little project management present. Still, the open source community has produced some remarkable pieces of software. Most studies on collaboration in open source look at the underlying social networks. These studies are summarized below.

8.2. Problems in Open Source

The importance of negotiation in open source development was highlighted in a study by Sandusky and Gasser [65]. They used a systematic random sample of 385 bug reports drawn from an open source bug report repository containing almost 200,000 bug reports. The qualitative analysis of individual bug reports as well as of texts that recorded community responses to reported software problems show how the distributed community uses its process to manage software quality. Focusing on the role of the basic social process of negotiation, the authors found that most bug reports contain negotiation, underlining the importance of communication in open source development.

8.3. Social Networks

The structure of social networks in open source was studied through a topological analysis by Xu *et al.* [82]. They conducted a quantitative analysis of open source software developers by studying the entire development community at SourceForge. Statistics and social network properties were explored to find collaborations and the effects of different members in the OSS development community. Small world phenomena and scale free behaviours were found in the SourceForge development network. Xu *et al.* conclude that these topological properties may potentially explain the success and efficiency of open source development practices. Also, weakly associated but contributing co-developers and active users may be an important factor in open source development.

A study by Bird *et al.* [5] looked at the latent social structure in open source projects by extracting and studying latent sub communities from the email social network of several projects: Apache HTTPD, Python, PostgresSQL, Perl, and Apache ANT. The sub communities were validated with software development activity history. Bird *et al.* found that sub communities do indeed spontaneously arise within open source projects as the projects evolve. The sub communities manifest most strongly in technical discussions, and are significantly connected with collaboration behaviour.

Ellis *et al.* [29] propose a social visualization for open source development. Their visualization is called Social Health Overview and was designed to support open source software development. A preliminary evaluation of the tool was done through interviews intended to identify its strengths and weaknesses and eleven informants in various open source roles were interviewed about their work practices. Generally, the visualization seems to be a promising approach.

Another study to understand the nature of collaboration in open source software development was conducted by Nakakoji *et al.* [56] by examining email traffic in the GIMP developer mailing list. They analyzed individual activities within the community as implied by the message posting to the mailing list over a relatively long period of time, such as who tended to ask questions, who tended to provide answers, and how their roles did or did not change over time. Some preliminary insights are reported.

8.4. Summary

Research on social networks in the open source community is not very far advanced yet. While there is a large interest in the coordination mechanisms and collaboration mechanisms behind successful open source projects, these mechanisms are not well understood yet.

9. Studies on the Role of Management for Collaboration

9.1. Introduction

As pointed out earlier, process and communication between software developers are essential for the success of distributed software development. This implies a high importance of management activities. However, only two studies were identified that directly deal with the role of management in collaborative software development. They can be divided into project management and the coordination between operations and developers.

9.2. Project Management

A study on the activities of project managers in collaborative software development was conducted by Zhang *et al.* [84]. They used a combination of ethnographic techniques including interview and field observation to understand project manager's collaboration activities. Seventeen Chinese project managers from various industries and diverse project teams with different experiences respectively participated in the semi-structured, in-situ interviews for 60 to 90 minutes. Information on project management activities, processes, tool usage, problems, and requirements were discussed. In a second phase, shadowing was used for three months in two projects and project management activities were videotaped for about eighty hours involving 45 individuals including the project managers, customer deputies, project members, and vendors. The authors also accessed the project's artifacts and collected related materials. These data were later transcribed into an activity tracking log and analyzed by two researchers independently.

Project manager's work was observed to comprise fragmented activities, which were interrelated and could be organized into several threads. Project managers reported that it is essential to acquire timely and comprehensive awareness of the project status through frequent ad-hoc communication to assure the project health. They spent about half of their time tracking progress, and another fifth of their time reporting to keep customers and executives' awareness. Project managers had to take extra effort to get project information from external individuals, and update it into the project management system, which made communication inefficient, error prone and out-of-date.

9.3. Coordination Between Operations and Developers

In their study on cooperation between developers and operations in software engineering projects, Tessem and Iden [73] collected data from a focus group of experienced software engineers and project managers and also conducted interviews in two case studies. They found that well performed cooperation between the development team and the operations team was crucial for successful deployment and operations of a new or extensively revised software system. Their data also showed that cooperation can be improved in several development activities like requirements engineering, system design, documentation, testing, training, and deployment planning.

9.4. Summary

These two studies underline the importance of management and effective collaboration between teams in collaborative software development and they also point at places for improvements.

10. Reflections on Literature Review

This section gives a short summary of experiences made while conducting the systematic literature review and reflects on the usefulness of this method in software engineering research.

10.1. General Impression

In general, it seems as if the approach of systematic literature reviews is applicable to the software engineering domain. This was also found by Brereton *et al.* [9]. In addition, the authors state that there are some difficulties such as the lack of appropriate quality measures, the lack of support for advanced searches in digital libraries, and the lack of study protocols in many empirical papers.

10.2. Minor Issues

Below, we detail some of the issues that were found to be difficult during this systematic literature review.

10.2.1 Low Quality of Abstracts

In only a few cases, it was apparent from the abstract what the paper was about, how the study described had been conducted, and foremost which results could be produced. There were many abstracts that did not give results at all and used phrases such as *"We present interesting results"* rather than *"We found A, B, and C"*. Instead of giving a high level overview of the results achieved in a study, it seemed as if many abstracts were trying to get the audience to read the whole paper by withholding information and promising that this information could be found in the paper. This fact made it sometimes difficult to extract the information needed from the primary studies.

10.2.2 Low Quality of Conclusions

Some studies only reported their immediate findings without reaching any conclusions about what these findings meant, e.g. they reproduced transcripts from interviews without saying what these answers actually meant. In those studies, the conclusion section was mainly used to summarize the motivation and study design again instead of explaining the insights reached. It was sometimes tedious if not impossible to extract research results from these studies.

10.2.3 Misuse of the Term "Case Study"

In some studies, the term "case study" was misused. Instead of describing an actual case, some authors used the term equivalent to "illustrative example", i.e. there was no actual evaluation conducted. These studies were usually selected in the early phase of the literature inspection, but disregarded later on.

10.2.4 Categories and Keywords

Categories and keywords seem to be assigned to most papers in a more or less arbitrary way. In particular, there is no consistency over the papers of different authors in their use of categories and keywords. Thus, these mechanisms are not helpful in a systematic literature review.

10.2.5 Differences in Search Engines

As pointed out earlier, different search engines use different mechanisms to handle search queries. In particular there are no specific guidelines on the use of meta search engines such as Google Scholar. While it indexes most databases, the number of search results for most queries is extremely large. However, unlike other search engines, it orders search results by their relevancy, i.e. the first 100 search results are likely to be more relevant than the next 100. While it is virtually impossible to assess all research results by such a meta engine, it seems as if the first several hundred results are already sufficient. This phenomenon will have to be researched more in the future.

11. Conclusion

The introduction of global development has introduced major challenges to collaboration and communication in software development. Thus, most of the studies identified in this systematic literature review reported on challenges in distributed and global software development, with some studies focusing on specific activities such as requirements engineering and design and others focusing on the use of various kinds of tools for collaboration.

For requirements engineering, research on collaboration is still at its beginning. While various challenges have been identified, there is a lack of successful approaches to tackle those challenges. Similarly, collaborative design is not far advanced yet, either. However, some approaches such as interactive architecture browsers seem to be promising.

A lot of studies have been conducted on challenges in global or distributed software development. Especially in the early 2000s, many problems were identified. In a second wave after that, researchers focused on lessons learned in geographically distributed development and there is evidence that some of the challenges have been overcome by software developers.

Tools for collaborative software development mainly focus on communication, in particular instant messaging. Some of them have been applied quite successfully and have found their way into today's development environments. Researchers have also begun to understand the social networks driving open source development and the important role of management for successful collaboration.

References

[1] M. A. Babar, B. Kitchenham, and R. Jeffery. Distributed versus face-to-face meetings for architecture evaluation: a controlled experiment. In *ISESE '06: Proceedings of the 2006 ACM/IEEE international symposium on Empirical software engineering*, pages 252–261, New York, NY, USA, 2006. ACM.

[2] A. Begel. Effecting change: coordination in large-scale software development. In *CHASE '08: Proceedings of the 2008 international workshop on Cooperative and human aspects of software engineering*, pages 17–20, New York, NY, USA, 2008. ACM.

[3] J. T. Biehl, W. T. Baker, B. P. Bailey, D. S. Tan, K. M. Inkpen, and M. Czerwinski. Impromptu: a new interaction framework for supporting collaboration in multiple display environments and its field evaluation for co-located software development. In *CHI '08: Proceeding of the twenty-sixth annual SIGCHI conference on Human factors in computing systems*, pages 939–948, New York, NY, USA, 2008. ACM.

[4] J. T. Biehl, M. Czerwinski, G. Smith, and G. G. Robertson. Fastdash: a visual dashboard for fostering awareness in software teams. In *CHI '07: Proceedings of the SIGCHI conference on Human factors in computing systems*, pages 1313–1322, New York, NY, USA, 2007. ACM.

[5] C. Bird, D. Pattison, R. D'Souza, V. Filkov, and P. Devanbu. Latent social structure in open source projects. In *SIGSOFT '08/FSE-16: Proceedings of the 16th ACM SIGSOFT International Symposium on Foundations of software engineering*, pages 24–35, New York, NY, USA, 2008. ACM.

[6] J. P. Birnholtz, C. Gutwin, and K. Hawkey. Privacy in the open: how attention mediates awareness and privacy in open-plan offices. In *GROUP '07: Proceedings of the 2007 international ACM conference on Supporting group work*, pages 51–60, New York, NY, USA, 2007. ACM.

[7] N. Bos, J. Olson, A. Cheshin, Y.-S. Kim, N. Nan, and N. S. Shami. Traveling blues: the effect of relocation on partially distributed teams. In *CHI '05: CHI '05 extended abstracts on Human factors in computing systems*, pages 1917–1920, New York, NY, USA, 2005. ACM.

[8] N. Bos, J. Olson, N. Nan, N. S. Shami, S. Hoch, and E. Johnston. Collocation blindness in partially distributed groups: is there a downside to being collocated? In *CHI '06: Proceedings of the SIGCHI conference on Human Factors in computing systems*, pages 1313–1321, New York, NY, USA, 2006. ACM.

[9] P. Brereton, B. A. Kitchenham, D. Budgen, M. Turner, and M. Khalil. Lessons from applying the systematic literature review process within the software engineering domain. *J. Syst. Softw.*, 80(4):571–583, 2007.

[10] J. Brown, G. Lindgaard, and R. Biddle. Stories, sketches, and lists: Developers and interaction designers interacting through artefacts. *Agile, 2008. AGILE '08. Conference*, pages 39–50, Aug. 2008.

[11] J. J. Cadiz, A. Gupta, and J. Grudin. Using web annotations for asynchronous collaboration around documents. In *CSCW '00: Proceedings of the 2000 ACM conference on Computer supported cooperative work*, pages 309–318, New York, NY, USA, 2000. ACM.

[12] V. Casey and I. Richardson. Uncovering the reality within virtual software teams. In *GSD '06: Proceedings of the 2006 international workshop on Global software development for the practitioner*, pages 66–72, New York, NY, USA, 2006. ACM.

[13] M. Cataldo and J. D. Herbsleb. Communication networks in geographically distributed software development. In *CSCW '08: Proceedings of the ACM 2008 conference on Computer supported cooperative work*, pages 579–588, New York, NY, USA, 2008. ACM.

[14] M. Cataldo and J. D. Herbsleb. Communication patterns in geographically distributed software development and engineers' contributions to the development effort. In *CHASE '08: Proceedings of the 2008 international workshop on Cooperative and human aspects of software engineering*, pages 25–28, New York, NY, USA, 2008. ACM.

[15] M. Cataldo, J. D. Herbsleb, and K. M. Carley. Sociotechnical congruence: a framework for assessing the impact of technical and work dependencies on software development productivity. In *ESEM '08: Proceedings of the Second ACM-IEEE international symposium on Empirical software engineering and measurement*, pages 2–11, New York, NY, USA, 2008. ACM.

[16] M. Cataldo, P. A. Wagstrom, J. D. Herbsleb, and K. M. Carley. Identification of coordination requirements: implications for the design of collaboration and awareness tools. In *CSCW '06: Proceedings of the 2006 20th anniversary conference on Computer supported cooperative work*, pages 353–362, New York, NY, USA, 2006. ACM.

[17] L. Catledge and C. Potts. Collaboration during conceptual design. *Proceedings of the Second International Conference on Requirements Engineering*, pages 182–189, Apr 1996.

[18] K. T. Chang and K. Ehrlich. Out of sight but not out of mind?: Informal networks, communication and media use in global software teams. In *CASCON '07: Proceedings of the 2007 conference of the center for advanced studies on Collaborative research*, pages 86–97, New York, NY, USA, 2007. ACM.

[19] J. Chudge and D. Fulton. Trust and co-operation in system development: applying responsibility modelling to the problem of changing requirements. *Software Engineering Journal*, 11(3):193–204, May 1996.

[20] V. Dalberg, E. Angelvik, D. R. Elvekrok, and A. K. Fossberg. Cross-cultural collaboration in ict procurement. In *GSD '06: Proceedings of the 2006 international workshop*

on Global software development for the practitioner, pages 51–57, New York, NY, USA, 2006. ACM.

[21] D. Damian and J. Chisan. An empirical study of the complex relationships between requirements engineering processes and other processes that lead to payoffs in productivity, quality, and risk management. *Software Engineering, IEEE Transactions on*, 32(7):433–453, July 2006.

[22] D. Damian, L. Izquierdo, J. Singer, and I. Kwan. Awareness in the wild: Why communication breakdowns occur. *Global Software Engineering, 2007. ICGSE 2007. Second IEEE International Conference on*, pages 81–90, Aug. 2007.

[23] D. Damian, F. Lanubile, and T. Mallardo. The role of asynchronous discussions in increasing the effectiveness of remote synchronous requirements negotiations. In *ICSE '06: Proceedings of the 28th international conference on Software engineering*, pages 917–920, New York, NY, USA, 2006. ACM.

[24] D. Damian and D. Zowghi. The impact of stakeholders' geographical distribution on managing requirements in a multisite organization. *Requirements Engineering, 2002. Proceedings. IEEE Joint International Conference on*, pages 319–328, 2002.

[25] C. R. B. de Souza, D. Redmiles, L.-T. Cheng, D. Millen, and J. Patterson. How a good software practice thwarts collaboration: the multiple roles of apis in software development. In *SIGSOFT '04/FSE-12: Proceedings of the 12th ACM SIGSOFT twelfth international symposium on Foundations of software engineering*, pages 221–230, New York, NY, USA, 2004. ACM.

[26] C. R. B. de Souza, D. Redmiles, and P. Dourish. "breaking the code", moving between private and public work in collaborative software development. In *GROUP '03: Proceedings of the 2003 international ACM SIGGROUP conference on Supporting group work*, pages 105–114, New York, NY, USA, 2003. ACM.

[27] U. Dekel. Supporting distributed software design meetings: what can we learn from co-located meetings? In *HSSE '05: Proceedings of the 2005 workshop on Human and social factors of software engineering*, pages 1–7, New York, NY, USA, 2005. ACM.

[28] U. Dekel and J. D. Herbsleb. Pushing relevant artifact annotations in collaborative software development. In *CSCW '08: Proceedings of the ACM 2008 conference on Computer supported cooperative work*, pages 1–4, New York, NY, USA, 2008. ACM.

[29] J. B. Ellis, S. Wahid, C. Danis, and W. A. Kellogg. Task and social visualization in software development: evaluation of a prototype. In *CHI '07: Proceedings of the SIGCHI conference on Human factors in computing systems*, pages 577–586, New York, NY, USA, 2007. ACM.

[30] K. M. Everitt, S. R. Klemmer, R. Lee, and J. A. Landay. Two worlds apart: bridging the gap between physical and virtual media for distributed design collaboration. In *CHI '03: Proceedings of the SIGCHI conference on Human factors in computing systems*, pages 553–560, New York, NY, USA, 2003. ACM.

[31] R. E. Grinter. Using a configuration management tool to coordinate software development. In *COCS '95: Proceedings of conference on Organizational computing systems*, pages 168–177, New York, NY, USA, 1995. ACM.

[32] C. Gutwin, R. Penner, and K. Schneider. Group awareness in distributed software development. In *CSCW '04: Proceedings of the 2004 ACM conference on Computer supported cooperative work*, pages 72–81, New York, NY, USA, 2004. ACM.

[33] C. A. Halverson, J. B. Ellis, C. Danis, and W. A. Kellogg. Designing task visualizations to support the coordination of work in software development. In *CSCW '06: Proceedings of the 2006 20th anniversary conference on Computer supported cooperative work*, pages 39–48, New York, NY, USA, 2006. ACM.

[34] M. Handel and J. D. Herbsleb. What is chat doing in the workplace? In *CSCW '02: Proceedings of the 2002 ACM conference on Computer supported cooperative work*, pages 1–10, New York, NY, USA, 2002. ACM.

[35] M. Hause, M. Petre, and M. Woodroff. Performance in international computer science collaboration between distributed student teams. *Frontiers in Education, 2003. FIE 2003. 33rd Annual*, 3:S1F–13–18 vol.3, Nov. 2003.

[36] S. R. Haynes, A. L. Skattebo, J. A. Singel, M. A. Cohen, and J. L. Himelright. Collaborative architecture design and evaluation. In *DIS '06: Proceedings of the 6th conference on Designing Interactive systems*, pages 219–228, New York, NY, USA, 2006. ACM.

[37] J. Herbsleb and A. Mockus. An empirical study of speed and communication in globally distributed software development. *Software Engineering, IEEE Transactions on*, 29(6):481–494, June 2003.

[38] J. Herbsleb, A. Mockus, T. Finholt, and R. Grinter. An empirical study of global software development: distance and speed. *Software Engineering, 2001. ICSE 2001. Proceedings of the 23rd International Conference on*, pages 81–90, May 2001.

[39] J. D. Herbsleb, D. L. Atkins, D. G. Boyer, M. Handel, and T. A. Finholt. Introducing instant messaging and chat in the workplace. In *CHI '02: Proceedings of the SIGCHI conference on Human factors in computing systems*, pages 171–178, New York, NY, USA, 2002. ACM.

[40] J. D. Herbsleb and R. E. Grinter. Splitting the organization and integrating the code: Conway's law revisited. In *ICSE '99: Proceedings of the 21st international conference on Software engineering*, pages 85–95, New York, NY, USA, 1999. ACM.

[41] J. D. Herbsleb, A. Mockus, T. A. Finholt, and R. E. Grinter. Distance, dependencies, and delay in a global collaboration. In *CSCW '00: Proceedings of the 2000 ACM conference on Computer supported cooperative work*, pages 319–328, New York, NY, USA, 2000. ACM.

[42] J. D. Herbsleb, D. J. Paulish, and M. Bass. Global software development at siemens: experience from nine projects. In *ICSE '05: Proceedings of the 27th international conference on Software engineering*, pages 524–533, New York, NY, USA, 2005. ACM.

[43] P. Hinds and C. McGrath. Structures that work: social structure, work structure and coordination ease in geographically distributed teams. In *CSCW '06: Proceedings of the 2006*

20th anniversary conference on Computer supported cooperative work, pages 343–352, New York, NY, USA, 2006. ACM.

[44] H. Huang and R. Ocker. Preliminary insights into the in-group/out-group effect in partially distributed teams: an analysis of participant reflections. In *SIGMIS CPR '06: Proceedings of the 2006 ACM SIGMIS CPR conference on computer personnel research*, pages 264–272, New York, NY, USA, 2006. ACM.

[45] H. Huang and E. M. Trauth. Cultural influences and globally distributed information systems development: experiences from chinese it professionals. In *SIGMIS-CPR '07: Proceedings of the 2007 ACM SIGMIS CPR conference on Computer personnel research*, pages 36–45, New York, NY, USA, 2007. ACM.

[46] E. Isaacs, A. Walendowski, S. Whittaker, D. J. Schiano, and C. Kamm. The character, functions, and styles of instant messaging in the workplace. In *CSCW '02: Proceedings of the 2002 ACM conference on Computer supported cooperative work*, pages 11–20, New York, NY, USA, 2002. ACM.

[47] C. Johansson, Y. Dittrich, and A. Juustila. Software engineering across boundaries: student project in distributed collaboration. *Professional Communication, IEEE Transactions on*, 42(4):286–296, Dec 1999.

[48] P. Layzell, O. Brereton, and A. French. Supporting collaboration in distributed software engineering teams. *Software Engineering Conference, 2000. APSEC 2000. Proceedings. Seventh Asia-Pacific*, pages 38–45, 2000.

[49] E. Lindqvist, B. Lundell, and B. Lings. Distributed development in an intra-national, intra-organisational context: an experience report. In *GSD '06: Proceedings of the 2006 international workshop on Global software development for the practitioner*, pages 80–86, New York, NY, USA, 2006. ACM.

[50] E. MacGregor, Y. Hsieh, and P. Kruchten. Cultural patterns in software process mishaps: incidents in global projects. In *HSSE '05: Proceedings of the 2005 workshop on Human and social factors of software engineering*, pages 1–5, New York, NY, USA, 2005. ACM.

[51] A. Meneely, L. Williams, W. Snipes, and J. Osborne. Predicting failures with developer networks and social network analysis. In *SIGSOFT '08/FSE-16: Proceedings of the 16th ACM SIGSOFT International Symposium on Foundations of software engineering*, pages 13–23, New York, NY, USA, 2008. ACM.

[52] A. E. Milewski, M. Tremaine, R. Egan, S. Zhang, F. Köbler, and P. O'Sullivan. Information "bridging" in a global organization. In *CASCON '07: Proceedings of the 2007 conference of the center for advanced studies on Collaborative research*, pages 346–355, New York, NY, USA, 2007. ACM.

[53] D. R. Millen, J. Feinberg, and B. Kerr. Dogear: Social bookmarking in the enterprise. In *CHI '06: Proceedings of the SIGCHI conference on Human Factors in computing systems*, pages 111–120, New York, NY, USA, 2006. ACM.

[54] D. R. Millen, M. J. Muller, W. Geyer, E. Wilcox, and B. Brownholtz. Patterns of media use in an activity-centric collaborative environment. In *CHI '05: Proceedings of the SIGCHI conference on Human factors in computing systems*, pages 879–888, New York, NY, USA, 2005. ACM.

[55] S. Minto and G. C. Murphy. Recommending emergent teams. In *MSR '07: Proceedings of the Fourth International Workshop on Mining Software Repositories*, page 5, Washington, DC, USA, 2007. IEEE Computer Society.

[56] K. Nakakoji, K. Yamada, and E. Giaccardi. Understanding the nature of collaboration in open-source software development. *Software Engineering Conference, 2005. APSEC '05. 12th Asia-Pacific*, pages 8 pp.–, Dec. 2005.

[57] T. Nguyen, T. Wolf, and D. Damian. Global software development and delay: Does distance still matter? *Global Software Engineering, 2008. ICGSE 2008. IEEE International Conference on*, pages 45–54, Aug. 2008.

[58] R. J. Ocker and J. Fjermestad. Communication differences in virtual design teams: findings from a multi-method analysis of high and low performing experimental teams. *SIGMIS Database*, 39(1):51–67, 2008.

[59] I. Oshri, J. Kotlarsky, and L. Willcocks. Missing links: building critical social ties for global collaborative teamwork. *Commun. ACM*, 51(4):76–81, 2008.

[60] A. Pandey, C. Miklos, M. Paul, N. Kameli, F. Boudigou, V. Vijay, A. Eapen, I. Sutedjo, and W. Mcdermott. Application of tightly coupled engineering team for development of test automation software - a real world experience. *Computer Software and Applications Conference, 2003. COMPSAC 2003. Proceedings. 27th Annual International*, pages 56–63, Nov. 2003.

[61] L. D. Panjer, D. Damian, and M.-A. Storey. Cooperation and coordination concerns in a distributed software development project. In *CHASE '08: Proceedings of the 2008 international workshop on Cooperative and human aspects of software engineering*, pages 77–80, New York, NY, USA, 2008. ACM.

[62] L. Pilatti, J. L. N. Audy, and R. Prikladnicki. Software configuration management over a global software development environment: lessons learned from a case study. In *GSD '06: Proceedings of the 2006 international workshop on Global software development for the practitioner*, pages 45–50, New York, NY, USA, 2006. ACM.

[63] M. Purvis, M. Purvis, and S. Cranefield. Educational experiences from a global software engineering (gse) project. In *ACE '04: Proceedings of the sixth conference on Australasian computing education*, pages 269–275, Darlinghurst, Australia, Australia, 2004. Australian Computer Society, Inc.

[64] N. Ramasubbu and R. K. Balan. Globally distributed software development project performance: an empirical analysis. In *ESEC-FSE '07: Proceedings of the 6th joint meeting of the European software engineering conference and the ACM SIGSOFT symposium on The foundations of software engineering*, pages 125–134, New York, NY, USA, 2007. ACM.

[65] R. J. Sandusky and L. Gasser. Negotiation and the coordination of information and activity in distributed software problem management. In *GROUP '05: Proceedings of the 2005 international ACM SIGGROUP conference on Supporting group work*, pages 187–196, New York, NY, USA, 2005. ACM.

[66] A. Sarma, G. Bortis, and A. van der Hoek. Towards supporting awareness of indirect conflicts across software configuration management workspaces. In *ASE '07: Proceedings of the twenty-second IEEE/ACM international conference on Automated software engineering*, pages 94–103, New York, NY, USA, 2007. ACM.

[67] A. Sarma, D. Redmiles, and A. van der Hoek. Empirical evidence of the benefits of workspace awareness in software configuration management. In *SIGSOFT '08/FSE-16: Proceedings of the 16th ACM SIGSOFT International Symposium on Foundations of software engineering*, pages 113–123, New York, NY, USA, 2008. ACM.

[68] P. Scupelli, S. Kiesler, S. R. Fussell, and C. Chen. Project view im: a tool for juggling multiple projects and teams. In *CHI '05: CHI '05 extended abstracts on Human factors in computing systems*, pages 1773–1776, New York, NY, USA, 2005. ACM.

[69] D. Spinellis. Global software development in the freebsd project. In *GSD '06: Proceedings of the 2006 international workshop on Global software development for the practitioner*, pages 73–79, New York, NY, USA, 2006. ACM.

[70] M.-A. Storey, L.-T. Cheng, I. Bull, and P. Rigby. Shared waypoints and social tagging to support collaboration in software development. In *CSCW '06: Proceedings of the 2006 20th anniversary conference on Computer supported cooperative work*, pages 195–198, New York, NY, USA, 2006. ACM.

[71] M.-A. Storey, J. Ryall, R. I. Bull, D. Myers, and J. Singer. Todo or to bug: exploring how task annotations play a role in the work practices of software developers. In *ICSE '08: Proceedings of the 30th international conference on Software engineering*, pages 251–260, New York, NY, USA, 2008. ACM.

[72] S. Teasley, L. Covi, M. S. Krishnan, and J. S. Olson. How does radical collocation help a team succeed? In *CSCW '00: Proceedings of the 2000 ACM conference on Computer supported cooperative work*, pages 339–346, New York, NY, USA, 2000. ACM.

[73] B. Tessem and J. Iden. Cooperation between developers and operations in software engineering projects. In *CHASE '08: Proceedings of the 2008 international workshop on Cooperative and human aspects of software engineering*, pages 105–108, New York, NY, USA, 2008. ACM.

[74] M. R. Thissen, J. M. Page, M. C. Bharathi, and T. L. Austin. Communication tools for distributed software development teams. In *SIGMIS-CPR '07: Proceedings of the 2007 ACM SIGMIS CPR conference on Computer personnel research*, pages 28–35, New York, NY, USA, 2007. ACM.

[75] R. Urdangarin, P. Fernandes, A. Avritzer, and D. Paulish. Experiences with agile practices in the global studio project. *Global Software Engineering, 2008. ICGSE 2008. IEEE International Conference on*, pages 77–86, Aug. 2008.

[76] D. Čubranić, G. C. Murphy, J. Singer, and K. S. Booth. Learning from project history: a case study for software development. In *CSCW '04: Proceedings of the 2004 ACM conference on Computer supported cooperative work*, pages 82–91, New York, NY, USA, 2004. ACM.

[77] D. Čubranić and M. A. D. Storey. Collaboration support for novice team programming. In *GROUP '05: Proceedings of the 2005 international ACM SIGGROUP conference on Supporting group work*, pages 136–139, New York, NY, USA, 2005. ACM.

[78] V. Vitale. Ibm software development leveraging geographically distributed teams - the interactive solution marketplace 2.0 (ism) case study. *Global Software Engineering, 2006. ICGSE '06. International Conference on*, pages 259–260, Oct. 2006.

[79] D. Woit and K. Bell. Student communication challenges in distributed software engineering environments. In *ITiCSE '05: Proceedings of the 10th annual SIGCSE conference on Innovation and technology in computer science education*, pages 286–290, New York, NY, USA, 2005. ACM.

[80] F. Wong, G. Fernandez, and J. McGovern. Confer: towards groupware for building consensus in collaborative software engineering. In *AUIC '07: Proceedings of the eight Australasian conference on User interface*, pages 31–38, Darlinghurst, Australia, Australia, 2007. Australian Computer Society, Inc.

[81] J. Wu, T. C. N. Graham, and P. W. Smith. A study of collaboration in software design. In *ISESE '03: Proceedings of the 2003 International Symposium on Empirical Software Engineering*, page 304, Washington, DC, USA, 2003. IEEE Computer Society.

[82] J. Xu, Y. Gao, S. Christley, and G. Madey. A topological analysis of the open source software development community. *System Sciences, 2005. HICSS '05. Proceedings of the 38th Annual Hawaii International Conference on*, pages 198a–198a, Jan. 2005.

[83] N. Yamashita and T. Ishida. Effects of machine translation on collaborative work. In *CSCW '06: Proceedings of the 2006 20th anniversary conference on Computer supported cooperative work*, pages 515–524, New York, NY, USA, 2006. ACM.

[84] S. Zhang, C. Zhao, Q. Zhang, H. Su, H. Guo, J. Cui, Y. Pan, and P. Moody. Managing collaborative activities in project management. In *CHIMIT '07: Proceedings of the 2007 symposium on Computer human interaction for the management of information technology*, page 3, New York, NY, USA, 2007. ACM.