Collaboration in Distributed Software Development

Filippo Lanubile

Dipartimento di Informatica, University of Bari
Via Orabona 4, 70126 Bari, Italy
lanubile@di.uniba.it

Abstract. Software development is an intense collaborative process where success depends on the ability to create, share and integrate information. Given the trend towards globalization in the software development industry, distance creates an additional challenge to development processes, as fewer opportunities for rich interaction and lower frequencies of direct communication. The chapter introduces a taxonomy of software engineering tools for distributed projects and presents collaborative development environments, ranging from classic platforms for dispersed developers in open source software projects to modern environments for flexible and distributed processes. Moreover, it introduces computer-mediated communication theories which reveal some patterns of tool usage to overcome the challenges of distance. Building upon the theoretical background of media selection, the chapter summarizes research activities aimed to build an evidence-based model of task-technology fit for communication-intensive activities such as distributed requirements workshops.

Keywords: Distributed software development, global software development, computer-supported cooperative work, collaborative development environments.

1 Introduction

Working across distances has become commonplace for software projects today. Nevertheless, distance creates an additional challenge to development processes, because of fewer opportunities for rich interaction and lower frequencies of direct communication [27].

In order to support collaborative work on their projects, software engineers communicate both directly, through meetings and informal conversations, and indirectly, by means of software artifacts. Adequate tool support is paramount to enable collaboration in distributed software development. However, most work in collaborative environments for distributed development has focused on code-specific tasks rather than on other software engineering activities at a higher level of abstraction like requirements engineering or software design [35].

This chapter focuses on requirements engineering as an appropriate domain for studying distributed engineering teams. In fact requirements engineering is essentially a collaborative endeavor which involves a large pool of stakeholders processing significant amount of information about the problem domain and solution space. Hence requirements engineering involves a complex set of communication-intensive tasks

A. De Lucia and F. Ferrucci (Eds.): ISSSE 2006–2008, LNCS 5413, pp. 174–193, 2009.

that are significantly affected by the stakeholders' geographical distribution which impedes collocated meetings. Videoconferencing is generally considered the first choice communication medium to conduct requirements workshops between remote stakeholders. However, while videoconferencing sessions come with an additional overhead (e.g., the costs of infrastructure setup and maintenance), even when everything runs smoothly, it is still hard to conduct a long-running and productive discussion during a videoconference, especially when more than a few people are involved.

There is a need to further our knowledge of what is the most appropriate collaboration toolset to achieve a shared understanding among distributed project stakeholders. Distributed software development thus can benefit from an interdisciplinary approach in which the research areas of software engineering and computer-supported cooperative work (CSCW) converge.

This chapter is organized as follows. In Section 2 we provide first the motivation for distributed software development and then we introduce the challenges which rise from the negative effects of distance. Section 3 introduces a taxonomy of software engineering tools for distributed projects and presents collaborative development environments, ranging from classic platforms for dispersed developers in open source software projects to modern environments for flexible and distributed processes. Next, in Section 4 we present computer-mediated communication (CMC) theories from the field CSCW. Building upon the theoretical background of media selection, Section 5 summarizes our research activities aimed to build an evidence-based model of task-technology fit for communication-intensive activities such as distributed requirements workshops. The chapter concludes with a brief summary and some remarks.

2 Distributed Software Development

Distributed software development (DSD; or GSD for global software development; or GSE for global software engineering) means splitting the development of the same product or service among globally distributed sites.

In his seminal book [6], Erran Carmel lists the six main 'catalyst' factors, or potential benefits, which have driven to distributed software development.

Mergers and acquisitions. The global demand for software products and services that began in the 80s lead to a rush for mergers and acquisitions, as ICT firms strived to penetrate new markets and adjust or complement their products lines. As a result, software teams, no longer independent but yet in their own sites, are forced to collaborate as an overall global software team.

Position as global organizations. Software firms also began in the 80s to position themselves as 'global players,' to increase business opportunities with other global organizations that prefer comprehensive software suppliers for all their global subsidiaries rather than an heterogeneous network of vendors in different countries.

Increase proximity to the market. The business advantages of proximity to the market includes knowledge of customers and local conditions, such as localization, customization, and after-sale services, as well as the goodwill engendered by local investments such as a favorable tax treatment from governments.

Access the most talented developers. The quality of the programmers is mentioned as the most important factor in software work [25]. This statement is supported by multiple

sources of empirical evidence other than common wisdom. The implication is that organizations that want to deploy market-winning software products have to hire the most talented developers throughout the world, regardless of their geographical location.

Reduce development costs. Software companies in high-cost countries try to reduce development costs by outsourcing development work to programmers in low-cost countries (e.g., India, China, Brazil, and East Europe). Most managers mention cost reduction by offshoring (i.e., global outsourcing to contracting staff located offshore) as the first driving factor for GSD.

Reduce time to market. Since programmers are scattered across multiple sites, dispersion allows for 'round-the-clock', or 'follow-the-sun', development, which has the potential to permit the reduction of development cycles by increasing the amount of time in a day that software is being developed. However, there are very few studies on the effects of time separation [21] and then the benefits of follow-the-sun development are more a claim than a fact based on empirical evidence.

In spite of the benefits described above, the success of a globally distributed project is not guaranteed by just opening a development center in another region of the world [17]. Developing software as a team is a challenging task, but developing as a global software team is even more challenging due to distance [26]. Distance has an impact on the three main forms of cooperation within a team [7]: communication coordination, and control. *Communication* is the exchange between the members of information, whether formal or informal, occurring in planned or impromptu interaction. *Coordination* is that act of orchestrating each task and organizational unit, so that they all contribute to the overall objective. *Control* is the process of adhering to goals, policies, standards or quality levels, set either formally (e.g., formal meetings, plans, guidelines) or informally (e.g., team culture, peer pressure). Distributed teams create further burdens on communication, coordination and control mechanisms, primarily the informal ones.

Because of distance, people cannot coordinate and control by just visiting the other team members. The absence of management-by-walking can result in coordination and control issues, like misalignment and rework. When control and coordination needs of distributed software teams rise, so does the load on all communication channels available. In fact, software projects have two complementary communication needs. First, the more formal, official communications is used for crucial tasks like updating project status, escalating project issues, and determining who has responsibility for particular work products. Secondly, informal 'corridor talk' allows team members to keep a 'peripheral awareness' of what is going on around them, what other people are working on, what states the various parts of the project are in, and many other essential pieces of background information that enable developers to work together efficiently. In collocated settings, communication is taken for granted and then, its importance often goes unnoticed. When developers are not located together, they have much less opportunities of communication. There is empirical evidence that the frequency of communication drops off with the physical separation among developers' sites [27]. As Fig. 1 shows, distance exacerbates coordination and control problems directly or indirectly through its negative effects on communication. In other words, communication disruption due to distance further increases and aggravates coordination and control breakdowns [7].

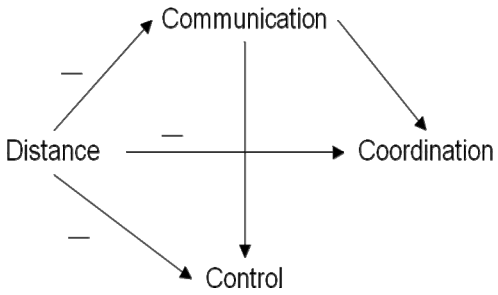

Fig. 1. Impact of distance on distributed software development (from [7])

Table 1. GSD threats and dimensions of distance (adapted from [1])

	Temporal Distance	**Geographical Distance**	**Sociocultural Distance**
Communication	- Reduced opportunities for synchronous communication	- Face-to-face meetings difficult	- Cultural misunderstandings
Coordination	- Typically increased coordination costs	- Reduced informal contact can lead to lack of critical task awareness	- Inconsistent work practices can impinge on effective coordination - Reduced cooperation arising from misunderstandings
Control	- Management of project artifacts may be subject to delays	- Difficult to convey vision and strategy - Perceived threat from training low-cost "rivals"	- Different perceptions of authority can undermine morale - Managers must adapt to local regulations

Table 1 summarizes the impact of the three dimensions of distance: geographical, temporal, and socio-cultural. *Geographical distance* is a measure of the spatial dispersion, occurring when team members are scattered across different sites. It can be operationalized as the cost or effort required to exchange visits from one site to another. *Temporal distance* is a measure of the temporal dispersion, occurring when team members wishing to interact. It can be caused by time-zone differences or just time shifting work patterns (e.g., one site having a quick lunch break at noon and another site a two-hour lunch time at 1:00 pm). *Socio-cultural distance* is a measure of the effort required by team members to understand the organizational and national cultures (e.g., norms, practices, values, spoken languages) in remote sites.

3 Collaboration Tools and Environments

Tools provide a considerable help to software development activities. Software engineering tools to assist distributed projects may fall into the following categories:

Software Configuration management. A software configuration management (SCM) tool includes the ability to manage change in a controlled manner, by checking components in and out of a repository, and the evolution of software products, by storing multiple versions of components, and producing specified versions on command. SCM tools also provide a good way to share software artifacts with other team members in a controlled manner. Rather than just using a directory to exchange files with other people, with an SCM tool developers can make sure that interdependent files are changed together and control who is allowed to make changes. Further, SCM tools make it possible to save messages about what changed and why. Open-source SCM tools, such as Subversion and its predecessor CVS, have become indispensable tools for coordinating the interaction of distributed developers.

Bug and change tracking. This function is centered around a database, accessible by all team members through a web-based interface. Other than an identifier and a description, a recorded bug includes information about who found it, the steps to reproduce it, who has been assigned on it, which releases the bug exists in and it has been fixed in. Bug tracking systems also define a life cycle for bugs to help team members to track the resolution of defects. Fig. 2 shows the life cycle of a defect in Bugzilla [3], the bug tracking system originally developed and used by the Mozilla project. Trackers are a generalization of bug tracking systems to include the management of other issues such as feature requests, support requests, or patches.

Build and release management. It allows projects to create and schedule workflows that execute build scripts, compile binaries, invoke test frameworks, deploy to production systems and send email notifications to developers. The larger the project, the greater the need for automating the build and release function. Build and release management tools can also provide a web-based dashboard to view the status of current and past builds (see Fig. 3). Build tools, such as CruiseControl and its ancestor like the UNIX make utility, are essential tools to perform Continuous Integration [22], an agile development practice which allows developers to integrate daily thus reducing integration problems.

Product and process modeling. This function encompasses the core features of what was called Computer Aided Software Engineering (CASE), from requirements management to visual modeling of both software artifacts and customized software processes. Collaboration in software development tends to be around the creation of formal or semiformal software artifacts. According to [38], model-based collaboration is what distinguishes software engineering collaboration from more general collaboration activities which lack the focus on using the models to create shared meanings.

Knowledge center. This function is mostly document-driven and web-enabled, and allows team members to share explicit knowledge across a work unit. A knowledge center includes technical references, standards, frequently asked questions (FAQs) and best practices. Recently wiki software for collaborative web publishing has emerged as a practical and economical option to consider for creating and maintaining group documentation. Wikis are particularly valuable in distributed projects as global teams may use them to organize, track, and publish their work [30]. Fig. 4 shows the home page of the Fedora project wiki where both developers and users may contribute other than find information. Knowledge centers may also include sophisticated knowledge management activities to acquire tacit knowledge in explicit forms, such as expert identification and skills management [34].

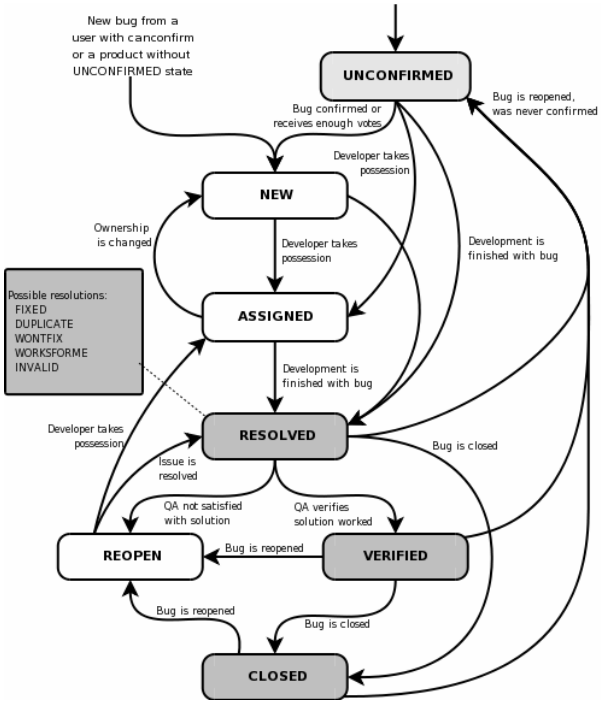

Fig. 2. Bug's life cycle (from [3])

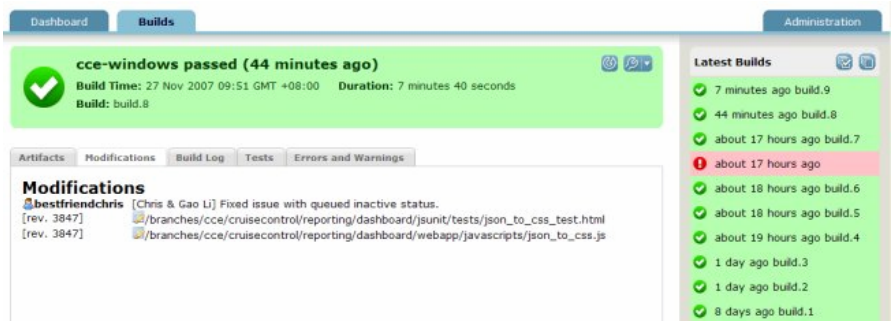

Fig. 3. Project build information within a dashboard (from [12])

Communication tools. Software engineers have adopted a wide range of mainstream communication technologies for project use in addition or replacement of communicating face-to-face by speech. Asynchronous communication tools include email, mailing lists, newsgroups, web forums and, more recently blogs; synchronous tools include the classic telephone and conference calls, chat, instant messaging, voice over IP, and videoconferencing. Email is the most-widely used and successful collaborative application. Thanks to its flexibility and ease of use, email can support conversations, but also operate as a task/contact manager. However, one of the drawbacks of

email is that, due to its success, people tend to use email for a variety of purposes and often in a quasi-synchronous manner. In addition, email is 'socially blind' [20] in that it does not enable users to signal their availability. Nevertheless, before becoming an indispensable tool ubiquitous in every workplace, email was initially used by the niche of research community and opposed by management. Currently, chat and instant messaging are following a similar evolution path. At first mostly used by young people for exchanging 'social' messages, these synchronous tools have been recently spreading more and more in the workplace. While email is socially blind, these tools, in contrast, provide a lightweight means to ascertain availability of remote team members and contact them in a timely manner.

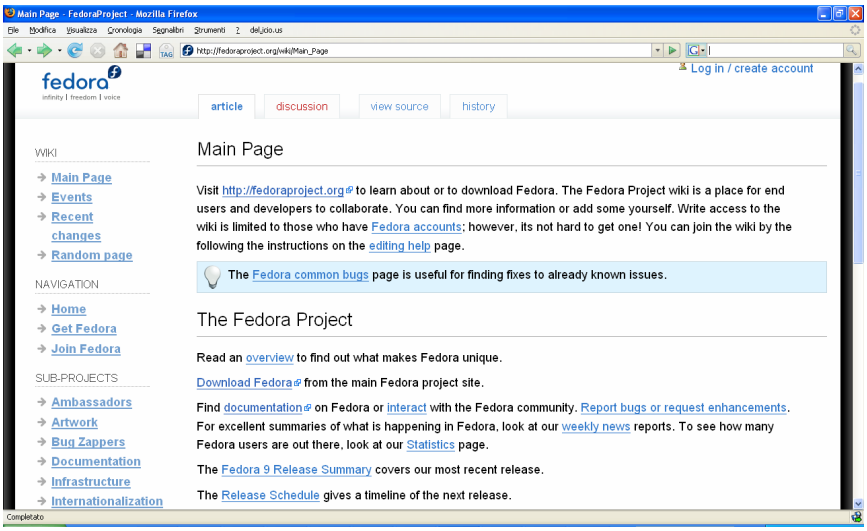

Fig. 4. Fedora Project documentation based on wiki

General communication tools (i.e., non software engineering specific) fall in the category of 'groupware' which refers to the class of applications that support groups of people engaged in performing a common task [19]. However, the term groupware is nowadays almost disused in favor of preferred wordings such as 'collaborative software', 'social software,' or 'Web 2.0' [32] which also include systems used outside the workplace (e.g., blogs, wikis, instant messaging).

Interoperability and a familiar user interface provide strong motivations to integrate task-specific solutions and generic groupware into collaborative development environments (CDE). A *CDE* provides a project workspace with a standardized toolset to be used by the global software team. Earliest CDE were developed within open source software (OSS) projects because OSS projects, from the beginning, have been composed of dispersed individuals. Today a number of CDE are available as commercial products, open source initiatives or research prototypes to enable distributed software development.

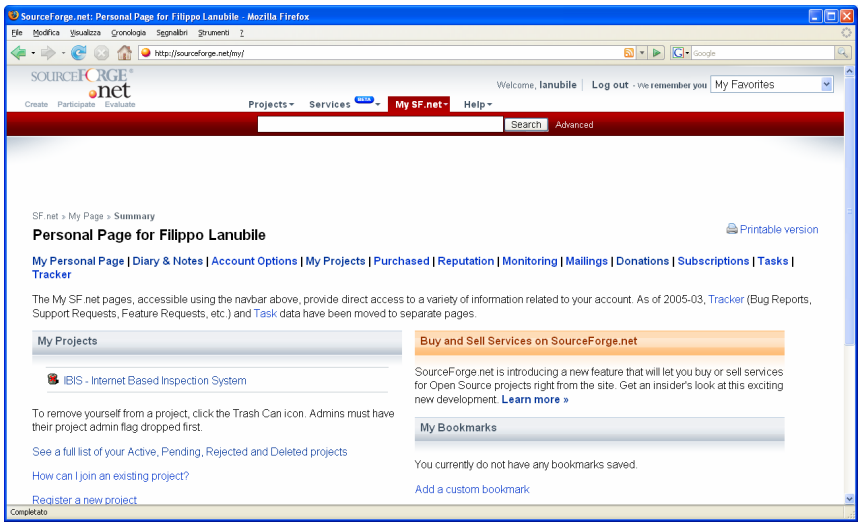

Fig. 5. Personal SourceForge portal

SourceForge, from CollabNet, is the most popular CDE with over 170.000 hosted projects and 1.800.000 registered users [11]. The original mission of SourceForge was to enrich the open source community by providing a centralized place for developers to control and manage OSS projects. SourceForge offers a variety of free services: web interface for project administration, space for web content and scripts, trackers (for reporting bugs, submitting support requests or patches to review, and posting feature requests), mailing lists and discussion forums, download notification of new releases, shell functions and compile farm, and CVS-based as well as Subversion-based configuration management. Fig. 5 shows the personal page of the author which provides access to a standard toolset which can be used on every project. The commercial versions for corporate use, called SourceForge Enterprise Edition and CollabNet Enterprise Edition, add features for tracking, measuring and reporting on software project activities.

GForge [24] is a fork of the 2.61 SourceForge.net project. It has been downloaded and configured as in-house server by many industrial and academic organizations (see Fig. 6 from [10]. Like SourceForge it also offers a commercial version, called GForge Advanced Server.

Trac [18] provides an integrated wiki, an issue tracking system and a front-end interface to SCM tools, usually Subversion. Project overview and progress tracking is allowed by setting a roadmap of milestones, which include a set of so-called “tickets” (i.e., tasks, feature requests, bug reports and support issues), and by viewing the timeline of changes. Trac also allows team members to be notified about project events and ticket changes through email messages and RSS feeds. Fig. 7 shows a screenshot of a research project at University of Bari with active tickets grouped by milestone and colored to indicate different priorities.

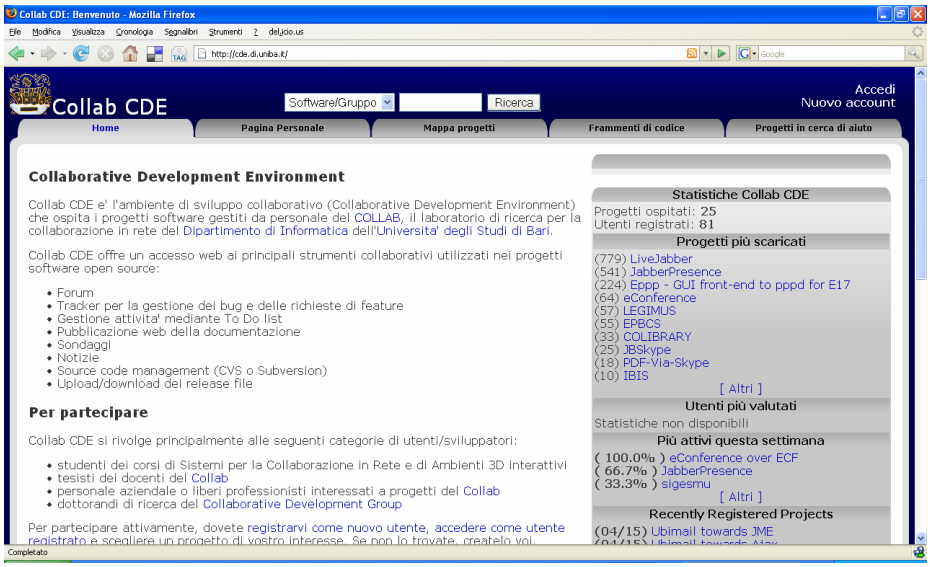

Fig. 6. A GForge-based CDE

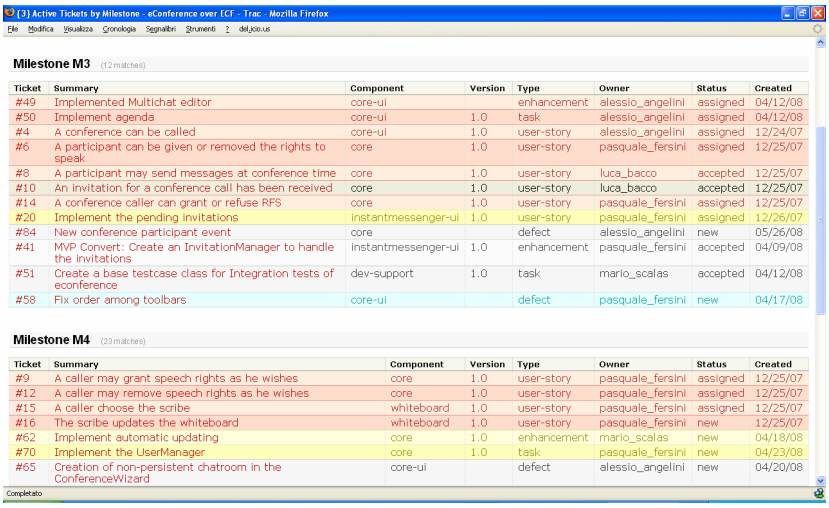

Fig. 7. Active tickets in Trac grouped by milestone

ADAMS is a CDE developed at University of Salerno [15]. ADAMS puts a great emphasis on creating and storing traceability links among software artifacts for the purpose of impact analysis and change management during software evolution. Events concerning changes to an artifact are automatically propagated along traceability links to other artifacts which have been impacted directly or indirectly by the change, and then to developers who have been assigned the responsibility of those artifacts.

Jazz [23] is an extensible platform which leverages the Eclipse's notion of plug-ins to build specific CDE products like the IBM Rational Team Concert. The present version has a wide-ranging scope but in the former version of Jazz [8, 28] the goal was to integrate synchronous communication and reciprocal awareness of coding tasks into the Eclipse IDE. The development of Jazz has been inspired to the Booch and Brown's vision of a "frictionless surface" for development [2], which was motivated by the observation that much of the developers' effort is wasted in switching back and forth between different applications to communicate and work together. According to this vision, collaborative features should be available as components that extend core applications (e.g., the IDE), thus increasing the users' comfort and productivity. Fig. 8 shows two Jazz client-side plug-ins installed into the Eclipse IDE.

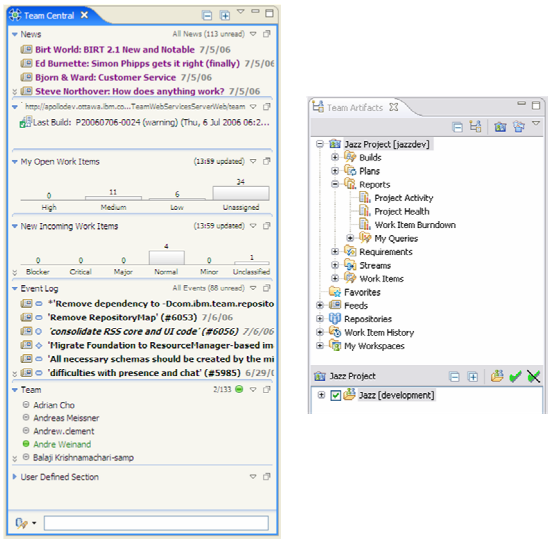

Fig. 8. Jazz plugins

4 Computer-Mediated Communication

Communication media are usually classified in the classic time/space matrix, according to both the spatial dimension (collocated/distributed, i.e., where the interaction occurs) and the temporal dimension (synchronous/asynchronous, i.e., when the interaction occurs). Media can also be classified according to another dimension, 'richness', which can be defined as the ability of media to convey a large amount of information in different forms. Fig. 9 shows the media within the time/space matrix and along the media richness continuum. Face-to-face (F2F) is the richest form of communication, since it conveys information via audio and video channels, but also through cues like gesture and posture. Consequently, videoconference is richer than telephone, since the latter lacks video as information channel, whereas email is richer than letter, since electronic mail can also attach multimedia content.

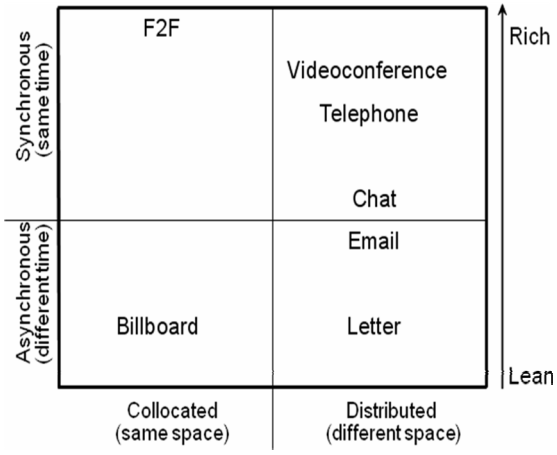

Fig. 9. Media-Richness continuum in the Time/Space Matrix (adapted from [19])

Some CMC theories have agreed on the inadequateness of text-based communication for complex, collaborative tasks, arguing that, as complexity increases, so should the level of richness of the media used, thus suggesting the use of a video link in distributed contexts. Nevertheless, the last decade has witnessed the success of many open-source projects which are coordinated through the almost-exclusive use of text-based technologies, such as wikis, email, and instant messaging. Further, nowadays the practicality of organizing videoconferencing sessions still remains low due to the additional overhead (e.g, infrastructure expensive to setup and maintain at the remote sites).

In the following we review the most prominent, and often conflicting, CMC theories.

The *Social Presence* (SP) [36] and *Media Richness* (MR) [13] theories posit group effectiveness to decrease when media other than F2F are used to accomplish equivocal tasks that require relational cues to be exchanged. Compared to rich media like F2F and video, 'lean' media like email and instant messaging lack the ability of conveying nonverbal cues (e.g., gaze, tone of voice, facial expressions) that contribute to the level of social presence, which in turns fosters individuals' motivation and mutual understanding. Hence, rich media are recommended for accomplishing equivocal tasks (i.e., when multiple and conflicting interpretations exist about a situation), for which also communicating relation content is relevant. In contrast, lean media are sufficient for executing tasks of uncertainty (i.e., the difference between the amount of information required and already possessed about a situation), which need only task-focused communication.

Common Ground (CG) theory [9] posits that people should attempt to achieve common ground (i.e., mutual understanding) with techniques available in a communication medium that lead to the least collaborative effort. CG theory presents eight properties that a medium may impose as constraints on the grounding process: copresence, visibility, audibility, cotemporality, simultaneity, sequentiality, reviewability, revisability. No medium has all the attributes at the same time. When a medium

lacks one of these characteristics, it forces people to use alternative grounding techniques with different costs for the speaker, the receiver or both. Participants in a F2F conversation usually establish common ground on the fly, as they have access to cues like facial expression, gestures and voice intonation. Instead, when participants communicate over media, the fewer cues they have, the harder to construct it. As a consequence, people who do not share mutual knowledge largely benefits from using audio/video channels for completing collaborative tasks, whereas those who have an extensive preexisting common ground can communicate effectively also on lean media such as email.

The *Time-Interaction-Performance* (TIP) theory [31] hypothesizes that communication that occurs in four tasks categories (generating, intellective, judgment, and negotiation) can be ordered by complexity and the amount of information required. Fig. 10 illustrates the task-media fit attempted by the theory, with respect to the communication media. TIP theory argues that rich media do not always provide the best-fitting combination regardless of the task type. There are in fact two possible types of poor fit: when a task requires more information that the medium can deliver and when the medium provides more information than the task requires (i.e., information overload).

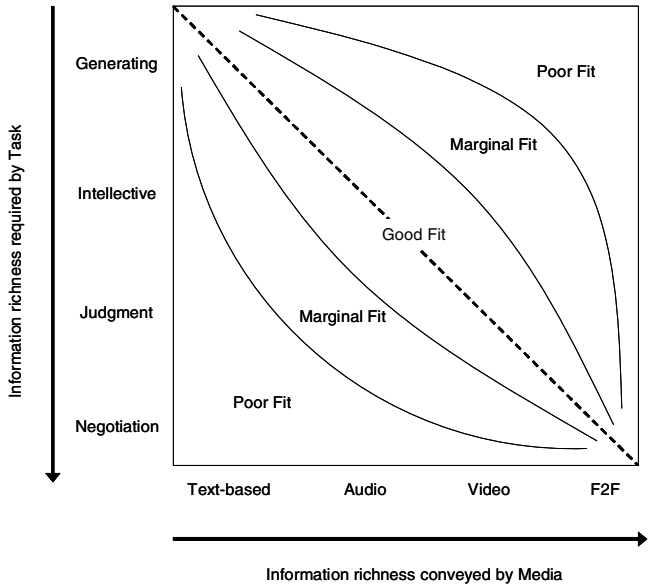

Fig. 10. Task-media fit as suggested by TIP theory (adapted from [31])

Media Synchronicity (MS) theory [16] distinguishes between the interplay of two different communication processes (the conveyance of additional information, and the convergence to shared views), which vary with the degree of synchronicity of the medium. The level of 'medium synchronicity' measures the extent to which it supports two inversely proportional properties, that is, immediacy of feedback (i.e., the ability of a medium to have rapid bidirectional communication) and parallel input

(i.e., the ability of a medium to allow for more simultaneous conversations at one time). F2F conversation and audio/video conference are high-synchronicity media because they grant high immediacy of feedback, but no parallel input (i.e., just one speaker at a time). In contrast, email, chat and instant messaging are low-synchronicity media because they ensure high input parallelism and low immediacy of feedback. MS theory suggests that when extra-information (i.e., conveyance) is needed, low-synchronicity media are to be preferred due to the support of input parallelism, whereas high-synchronicity media are better able to support convergence because of the higher degree of immediacy of feedback ensured. However, in media selection one must take into account that most tasks require individuals to both convey information and converge on shared meanings, and media that excel at information conveyance are often not those that excel at convergence. Thus, choosing one single medium for any task may prove less effective than choosing a medium, or set of media, which the group uses at different times in performing the task, depending on the current communication process (convey or converge).

Cognitive-Based View (CBV) [33] looks at communication as a cognitive process: Not only must the sender's comfort with the communication medium be taken into account, but also the motivation of receivers and, above all, their ability to process the message properly. Rich media, require participants to be present at the same time, if not at a same place. This requires a high level of commitment to participate in the communication process, which in turn provides the ability to reciprocally monitor attention. In contrast, lean media allow a receiver to gain time for thinking at their will, as well as finding additional information sources, until comprehension is fully achieved. However, lean media must compete with other activities on the receiver's side and can be easily ignored with no feedback for the sender. As a result, rich media ensure that motivation and attention stays up, while lean media provides a higher ability to process information. On the other hand, when faced with highly complex messages sent with high social presence, a receiver can be overwhelmed with information, thus delaying or biasing the decision. Fig. 11 shows the so-called 'media richness paradox', that is the inverse relationship between motivation/attention and the ability to process. CBV argues that different media are needed for complex tasks where information overload may be generated. In such cases, the use of mixed media, or 'media switching', is motivated by the need to balance attention and motivation required by senders with the ability to process information of receivers. Depending on the task at hand, when senders want to get the attention of the receiver and motivate them for an immediate response, they should use a rich medium, high in social presence. In contrast, when deep thought and deliberation are needed to process the information, the sender should use a lean medium, low in social presence, to give the receiver time to objectively elaborate on messages.

The common denominator of the many existing CMC theories is that the interaction of individuals is deeply influenced not only by media characteristics, but also by other factors such as tasks requirements. Drawing upon these theories, we argue that, by understanding the driving factors of CMC, groups may be better able to select and use the most appropriate sets of media to accomplish their goals.

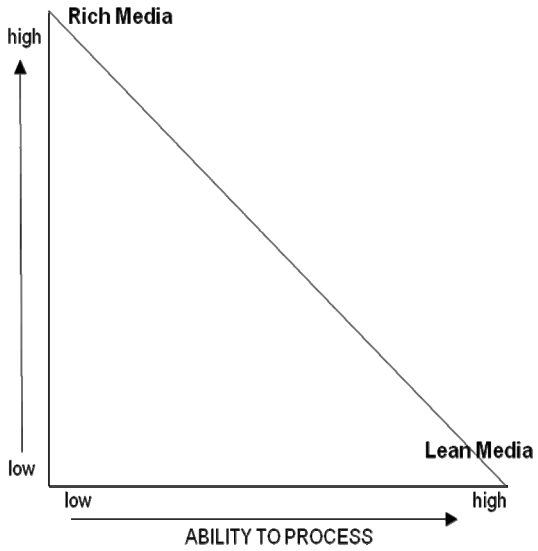

Fig. 11. Media richness paradox according to CBV (adapted from [33])

5 Media Switching in Distributed Requirements Engineering

Recent CMC theories from the literature suggest that relying on rich media alone may not yield the best results in terms of performance, and that a combination with lean media would facilitate a more rational approach to decision making. These theories form the theoretical basis for our central thesis: a mix of lean and rich media is needed to improve requirements engineering activities when stakeholders are geographically dispersed.

The rest of this section describes a family of empirical studies [4, 14] in distributed requirements engineering which have been conducted jointly by the Collaborative Development Group (CDG) at University of Bari, Italy, and the Software Engineering Global interAction Laboratory (SEGAL) at University of Victoria, Canada.

Asynchronous Discussions as a Complement to Videoconference Requirements Negotiation Meetings

To improve the effectiveness of distributed requirements engineering, drawing upon the postulates of modern theories on media selection, we have investigated ways to increase the effectiveness of videoconference requirements negotiation meetings by means of asynchronous text-based discussions as a useful complement for the preparation of such meetings.

We conducted a replicated case study of six academic global projects with the participation of students from three universities of three different countries: Australia, Canada, and Italy. Cross-university student projects were structured as outsourcing projects in which work was allocated to a developer group in a different organization and country. The project outcome was a requirements specification (RS) as a

negotiated software contract between the developer group and the outsourcing company (client group). Teamwork was critical in completing the software project as the developer group had to frequently interact with the client group to understand the required software features.

Reaching a mutual understanding between clients and developers mean reducing equivocality and uncertainty by resolving all open issues found by clients in an evolving RS throughout the process. Fig. 12 illustrates the RS development workflow over a period of 7 weeks. It consisted of eleven phases of continuous requirements discovery and validation through which the understanding and documentation of requirements had to be improved. Each of these stages consisted of tasks for either the client/developer groups, or project team tasks.

Project teams negotiated requirements during one-hour synchronous videoconferencing meetings. In our research design, we created two process variants in which half of the groups were involved in asynchronous discussions of open issues, using the IBIS inspection tool [29], prior to the synchronous negotiation meeting, while the other half was not. We were thus able to compare the performance of the groups using a mix of media with that of the groups using only videoconferencing meetings.

Fig. 13 provides empirical evidence for our main hypothesis: groups in the mixed media process variant were able to end the requirements engineering process with significantly fewer open issues than the groups that were not involved in the asynchronous discussion. More uncertainties than ambiguities were clarified and then closed in the asynchronous discussions. Consequently, when participants had already discussed asynchronously, they began the videoconference negotiation meetings with a shorter list of open issues to be discussed.

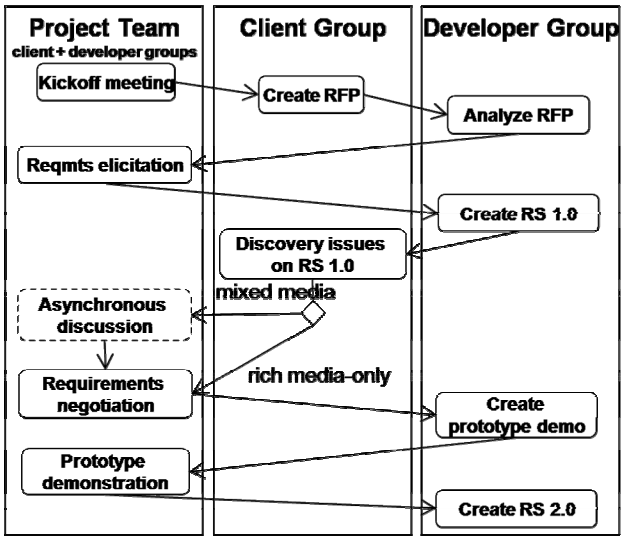

Fig. 12. Global project workflow (adapted from [14])

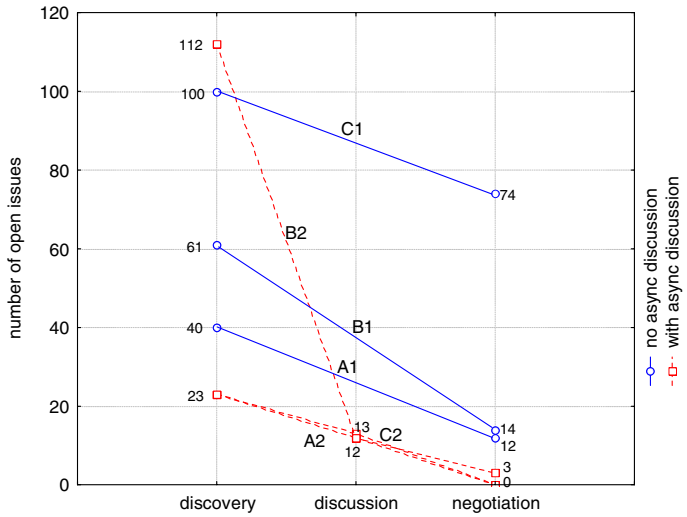

Fig. 13. Open issues in the six projects throughout three stages (from [14])

Comparing Text-Based Synchronous Communication and Face-to-Face Meetings in Distributed Requirements Workshops

Requirements workshops, whether for eliciting or negotiating requirements, are complex tasks that require a constant interplay between idea generation, decision making, and conflict resolution activities, although in different measure: requirements elicitation is more a generative task, whereas requirements negotiation is more oriented to decision making.

To build a body of knowledge about task/technology fit, rooted in the software engineering field, we evaluated the support of synchronous text-based communication for conducting distributed requirements workshops. In particular we investigated two research questions: (1) how synchronous text-based requirements workshops vary from F2F counterparts, and (2) whether both synchronous text-based elicitation and synchronous text-based negotiation represent an appropriate task-technology fit.

The empirical study involved six academic groups, playing the role of stakeholders while completing the project work of a requirements engineering course held at the University of Victoria. Analogously to the previous study, the goal of each project was to develop a Requirements Specification (RS) document through the interaction of a client and a developer team over a period of about ten weeks. Fig. 14 shows the workflow of the requirements development process. In order to perform quantitative analysis, two satisfaction questionnaires, with Likert scale response items, were administered to the students after each workshops session. Table 2 illustrates the experimental design with three factors, each having two levels: communication mode (F2F and CMC); requirements workshop (elicitation and negotiation); and role (client and developer). F2F workshops were held in a classroom while CMC workshops were run using the eConference tool [5], a text-based, distributed meeting system developed at University of Bari.

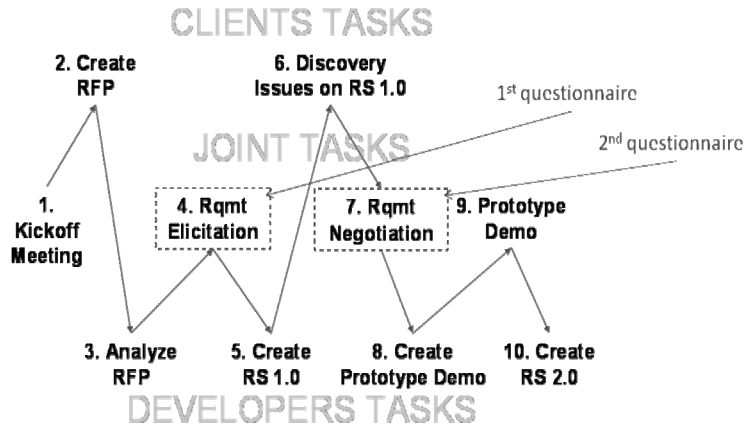

Fig. 14. Project workflow in [4]

The findings from the study were the following:

- during the requirements meetings, the subjects perceived a higher level of comfort with F2F communication mode than with CMC, while keeping an equal level of motivation to participate;
- compared to F2F requirements workshops, CMC workshops grant a higher opportunity to participate in a more structured, equal, and open discussion;
- being a customer or a developer has no effect;
- stakeholders significantly preferred F2F negotiation over F2F elicitation, and F2F negotiation over CMC negotiation;
- CMC elicitation is as good as F2F elicitation

These results suggest that, in order to reduce the negative effects of distance as well as the need and the number of collocated requirements workshops, synchronous text-based elicitations represent a better task-technology fit than synchronous text-based negotiations.

Table 2. Experimental design (adapted from [4])

Communication Mode	Requirements Workshop	Role	Subjects
F2F	elicitation	client	Gr1, Gr3, Gr5
CMC	elicitation	client	Gr2, Gr4, Gr6
F2F	negotiation	client	Gr2, Gr4, Gr6
CMC	negotiation	client	Gr1, Gr3, Gr5
F2F	elicitation	developer	Gr2, Gr4, Gr6
CMC	elicitation	developer	Gr1, Gr3, Gr5
F2F	negotiation	developer	Gr1, Gr3, Gr5
CMC	negotiation	developer	Gr2, Gr4, Gr6

6 Conclusions

Software development is an intense collaborative process where success depends on the ability to create, share and integrate information [37]. We presented a number of tools and collaborative development environments which are available to support distributed teams. We restricted our view to distributed teams of stakeholders when engaged in requirements elicitation and negotiation activities. Collaboration-intensive, these activities largely rely on face-to-face interactions of the project stakeholders and are thus greatly disrupted by distance.

Main media selection theories from the CSCW field assert that rich media alone do not provide the best performance to distributed teams and that switching between rich and lean media would smooth the progress of teamwork for complex activities. The CSCW body of knowledge about media selection, however, is mostly based on experiments which used generic, game-like tasks involving either idea generation or problem solving. Since our interest is in collaborative technologies for distributed development, we have presented our empirical software engineering studies about the effects of media usage on the execution of software engineering specific tasks such as distributed requirements workshops.

Current research in distributed development is very active in academia as well as in industry. Major ACM conferences, such as the International Conference on Software Engineering and the Conference on Computer Supported Cooperative Work, include technical papers and organize workshops about global software development. Further, the IEEE International Conference on Global Software Engineering, now at its third edition, has become the annual event for the community of researchers and practitioners interested in exploring how distributed teams work and how the problems can be solved. We hope that the state of the art and practice presented in this chapter will inspire young researchers to invest their effort into this challenging and interdisciplinary topic.

Acknowledgments

We are grateful to Fabio Calefato and Teresa Mallardo whose doctoral research contributed to the theoretical and empirical work reported in this chapter. I would also like to express my thanks to Daniela Damian who has been an invaluable partner in the research work here reported.

References

1. Agerfalk, P.J., Fitzgerald, B.: Flexible and Distributed Software Processes: Old Petunias in New Bowls? Communications of the ACM 49(10), 26–34 (2006)
2. Booch, G., Brown, A.W.: Collaborative Development Environments. In: Advances in Computers, vol. 59. Academic Press, London (2003)
3. Bugzilla Team: The Bugzilla Guide (2008), `http://www.bugzilla.org/docs`

4. Calefato, F., Damian, D., Lanubile, F.: An Empirical Investigation on Text-Based Communication in Distributed Requirements Workshops. In: Proc. of the Int. Conf. on Global Software Engineering, pp. 3–11. IEEE Computer Society, Washington (2007)
5. Calefato, F., Lanubile, F., Scalas, M.: Evolving a Text-Based Conferencing System: An Experience Report. In: Proc. of the 3rd Int. Conf. on Collaborative Computing: Networking, Applications and Worksharing. IEEE Computer Society, Washington (2007)
6. Carmel, E.: Global Software Teams. Prentice Hall, Upper Saddle River (1999)
7. Carmel, E., Agarwal, R.: Tactical Approaches for Alleviating Distance in Global Software Development. IEEE Software 18(2), 22–29 (2001)
8. Cheng, L., de Souza, C., Hupfer, S., Patterson, J., Ross, S.: Building Collaboration into IDEs. ACM Queue 1(9) (2003-2004)
9. Clark, H.H., Brennan, S.E.: Grounding in Communication. In: Perspectives on Socially Shared Cognition, American Psychological Association, Washington, DC, pp. 127–149 (1991)
10. Collaborative Development Group: Collab CDE (2008), http://cde.di.uniba.it/
11. CollabNet: SourceForge.net (2008), http://sourceforge.net/
12. CruiseControl (2008), http://cruisecontrol.sourceforge.net/
13. Daft, R.L., Lengel, R.H.: Organizational Information Requirements, Media Richness and Structural Design. Management Science 32(5), 554–571 (1986)
14. Damian, D., Lanubile, F., Mallardo, T.: On the Need for Mixed Media in Distributed Requirements Negotiations. IEEE Transactions on Software Engineering 34(1), 116–132 (2008)
15. De Lucia, A., Fasano, F., Oliveto, R., Tortora, G.: Recovering traceability links in software artifact management systems using information retrieval methods. ACM Trans. Softw. Eng. Methodol. 16(4), 13 (2007)
16. Dennis, A.R., Valacich, J.S.: Rethinking Media Richness: Towards a Theory of Media Synchronicity. In: Proc. of the 32nd Annual Hawaii Int. Conf. on System Sciences, p. 1017. IEEE Computer Society, Washington (1999)
17. Ebert, C., De Neve, P.: Surviving Global Software Development. IEEE Software 18(2), 62–69 (2001)
18. Edgewall Software: The Trac Project (2008), http://trac.edgewall.org/
19. Ellis, C.A., Gibbs, S.J., Rein, G.: Groupware: Some Issues and Experiences. Communications of the ACM 34(1), 39–58 (1991)
20. Erickson, T., Kellogg, W.A.: Social Translucence: An Approach to Designing Systems that Support Social Processes. ACM Transactions on Computer-Human Interaction 7(1), 59–83 (2000)
21. Espinosa, J.A., Nan, N., Carmel, E.: Do Gradations of Time Zone Separation Make a Difference in Performance? A First Laboratory Study. In: Int. Conf. on Global Software Engineering, pp. 12–22. IEEE Computer Society, Los Alamitos (2007)
22. Fowler, M., Foemmel, M.: Continuous Integration (2006), http://martinfowler.com/articles/continuousIntegration.html
23. Frost, R.: Jazz and the Eclipse Way of Collaboration. IEEE Software 24(6), 114–117 (2007)
24. GForge Group: GForge project, http://gforge.org/projects/gforge/
25. Glass, R.L.: Software Engineering: Facts and Fallacies. Addison-Wesley Longman Publishing Co., Inc., Amsterdam (2002)
26. Herbsleb, J.D., Moitra, D.: Global Software Development. IEEE Software 18(2), 16–20 (2001)

27. Herbsleb, J.D., Mockus, T.A.: An Empirical Study of Speed and Communication in Globally Distributed Software Development. IEEE Transactions on Software Engineering 29(6), 481–494 (2003)
28. Hupfer, S., Cheng, L., Ross, S., Patterson, J.: Introducing collaboration into an application development environment. In: Proc. of the ACM Conference on Computer Supported Cooperative Work, pp. 21–24. ACM, New York (2004)
29. Lanubile, F., Mallardo, T., Calefato, F.: Tool Support for Geographically Dispersed Inspection Teams. Software Process: Improvement and Practice 8(4), 217–231 (2003)
30. Louridas, P.: Using Wikis in Software Development. IEEE Software 23(2), 88–91 (2006)
31. McGrath, J.E., Hollingshead, A.B.: Groups Interacting with Technology: Ideas, Evidence, Issues and an Agenda. Sage, Thousand Oaks (1994)
32. Murugesan, S.: Understanding Web 2.0. IT Professional 9(4), 34–41 (2007)
33. Robert, L.P., Dennis, A.R.: Paradox of Richness: A Cognitive Model of Media Choice. IEEE Transactions on Professional Communication 48(1), 10–21 (2005)
34. Rus, I., Lindvall, M.: Knowledge Management in Software Engineering. IEEE Software 19(3), 26–38 (2002)
35. Sengupta, B., Chandra, S., Sinha, V.: A research agenda for distributed software development. In: Proc. of the 28th Int. Conf. on Software Engineering, pp. 731–740. ACM, New York (2006)
36. Short, J., Williams, E., Christie, B.: The Social Psychology of Telecommunications. John Wiley and Sons, London (1976)
37. Walz, D.B., Elam, J.J., Curtis, B.: Inside a Software Design Team: Knowledge Acquisition, Sharing, and Integration. Communications of the ACM 36(10), 63–77 (1993)
38. Whitehead, J.: Collaboration in Software Engineering: A Roadmap. In: Int. Conf. on Software Engineering, pp. 214–225. IEEE Computer Society, Washington (2007)