Continuous Coordination: A New Paradigm to Support Globally Distributed Software Development Projects

David Redmiles, André van der Hoek, Ban Al-Ani, Tobias Hildenbrand*, Stephen Quirk, Anita Sarma, Roberto Silveira Silva Filho, Cleidson de Souza**, Erik Trainer

Donald Bren School of information and Computer Sciences
Department of Informatics, University of California, Irvine
Irvine, CA 92697-3430, USA
{redmiles, andre, balani, squirk, asarma, rsilvafi, etrainer}@ics.uci.edu

* Lehrstuhl für ABWL und Wirtschaftsinformatik
Universität Mannheim
D-68131 Mannheim, Germany
hildenbrand@uni-mannheim.de

** Departamento de Informática, Centro de Ciências Exatas e Naturais
Universidade Federal do Pará
Belém, PA 66075-110, Brazil
cdesouza@ufpa.br

Key Findings

We introduce and explicate a novel development paradigm for distributed software engineering development tools, Continuous Coordination. Continuous Coordination constitutes a paradigm for collaborative systems, which combines elements of traditionally formal, process-oriented approaches with those of the more informal, awareness-based approaches, thus addressing some of the issues of global software development:

- The lack of awareness and informal communication among developers are factors that contribute to problems arising during global software development. The Continuous Coordination paradigm improves awareness and enables a degree of self-coordination by integrating existing tools (based on formal, process-oriented approaches) with awareness for coordination (based on informal, peripheral and visual cues).
- Developers often face challenges when they attempt to integrate artifacts produced by heterogeneous tools. Continuous Coordination is defined in terms of several general design principles which are implemented in new and existing tools. Consequently, practitioners can adopt the paradigm immediately and incrementally; moreover, Continuous Coordination allows them to more readily integrate the artifacts produced.

Abstract (English)

Along with the rapid globalization of companies, the globalization of software development has become a reality. Many software projects are now distributed in diverse sites across the globe. The distance between these sites creates several problems that did not exist for previously collocated teams. Problems with the coordination of the activities, as well as with the communication between team members, emerge. Many collaborative software engineering tools that have been used to date, in global software development projects, exhibit a fundamental paradox: they are meant to support the collaborative activity of software development, but cause individuals and groups to work more or less independently from one another. The underlying issue is that existing software engineering tools, such as

configuration management repositories, issue trackers, and workflow engines, separate time and tasks in concrete but isolated process steps. Designing tools based on the premise that human activities can be codified and that periodic resynchronization of tasks is an easy step reflects poor understanding human nature. We therefore propose a new approach to supporting collaborative work called Continuous Coordination. Underlying Continuous Coordination is the premise that humans must not and cannot have their method of collaboration rigidly dictated, but should be supported flexibly with both the tools and the information to coordinate their activities and to collaborate in their activities as they see fit. In this paper, we define the concept of Continuous Coordination, introduce our work to date in building prototypes that support the Continuous Coordination paradigm in the context of Global Software Development, and set out a further research agenda to be pursued.

Keywords

Global Software Development, Distributed Software Development, Collaboration, Coordination, Awareness.

Abstract (German)

Im Zusammenhang mit der zunehmenden Globalisierung von Unternehmen ist auch die Globalisierung der Softwareerstellung Realität geworden. Viele Softwareprojekte sind bereits über mehrere Standorte rund um den Globus verteilt. Die Distanzen zwischen diesen Standorten rufen viele neue Probleme hervor, die zuvor in räumlich nahe zusammenarbeitenden Teams noch nicht beobachtet werden konnten. Es treten vor allem Probleme bei der Koordination von Aktivitäten sowie in Zusammenhang mit der Kommunikation innerhalb verteilter Teams auf. Die bisher in global verteilten Projekten eingesetzten Werkzeuge zur gemeinsamen Softwareerstellung weisen ein fundamentales Paradoxon auf: Sie sollen eigentlich die Zusammenarbeit im Rahmen der Softwareerstellung unterstützen, führen jedoch dazu, dass Individuen und Teams mehr oder weniger unabhängig voneinander arbeiten. Das grundlegenden Problem hierbei liegt darin, dass existierende Werkzeuge, wie bspw. Konfigurationsmanagement-Repositories, Problemverfolgungs- und Workflow-Managementsysteme, den Prozess zeitlich und aufgabenbezogen diskretisieren, was isolierte Prozessschritte zur Folge hat. Dieser Ansatz enthält die Annahme, dass menschliche Arbeit kodifiziert werden kann und dass die periodische Synchronisation der einzelnen Aufgaben ein trivialer Schritt ist – diese widerspricht jedoch der Natur des Menschen. Daher wird mit „Continuous Coordination“ ein neuer Ansatz zur Koordination von Zusammenarbeitsprozessen vorgestellt. Hinter Continuous Coordination verbirgt sich das Prinzip, dass Menschen die Art und Weise ihrer Zusammenarbeit weder vorgeschrieben werden darf noch strikt vorgeschrieben werden kann. Sie sollten aber sowohl mittels Werkzeugen und Informationen zur Koordination ihrer Aktivitäten als auch zur eigentlichen Kollaboration flexibel unterstützt werden. Dieser Beitrag definiert das Continuous-Coordination-Konzept, stellt unsere aktuellen Arbeiten zur Erstellung von Prototypen vor, die dieses Paradigma im Kontext globaler Softwareentwicklung unterstützen, und präsentiert einen Ausblick auf zukünftige Forschungsarbeiten.

Keywords (German)

Globale Softwareentwicklung, Verteilte Softwareentwicklung, Kollaboration, Koordination, Umgebungsbewusstsein.

1. Introduction

Globalization is a concept that applies in many contexts. Generally, it indicates that the economic, cultural, and social boundaries of countries are transcended. Indeed, the co-authors of this paper have come together from six different countries and from various backgrounds to collaborate, most of us discovering one another's commonalities via the Web and subsequently meeting for the first time via e-mail.

More formally, companies have sought to leverage globalization, especially in the software industry [HeMo01]. To do so, these companies need to adapt their processes, tools, and organizational culture to overcome the disparity among sites. As a consequence, they need to solve a wide variety of problems, the most obvious being the physical distance [Grud94, OlOl00]. In this case, the sense of working in a team decreases due to the lack of interaction among the members of different sites and as a consequence of the reduction in trust among members caused by software developers' lack of knowledge about foreign cultures [JaLe99]. Moreover, relatively simple activities such as discussing requirements in meetings cannot be performed [DCAC03]. Overall, many strategic, cultural, knowledge management, project management (PM), as well as technical issues must be solved [HeMo01]. Existing software tools address some of those problems by adopting more formal process-oriented approaches such as workflow management systems and configuration management tools. Other tools focus on bridging the distance gap between developers by facilitating their communication through, for example, e-mail, instant messengers (IM), and teleconferencing.

In this paper, we introduce a novel paradigm for supporting distributed software development: Continuous Coordination. Continuous Coordination combines aspects of formal, process-oriented approaches, such as configuration management protocols and workflows (used to guide users in their day-to-day high-level activities by coordinating their interactions), with informal, awareness-based approaches such as e-mail and Instant messenger (IM) communication (that provide communication channels to inform users of relevant, parallel ongoing activities). Particularly in the context of global software development (GSD), this paradigm can help to overcome some of the major coordination issues related to the lack of communication, context, and awareness. Through the combined application of the Continuous Coordination design principles of multiple perspectives, non-obtrusive integration, combination of socio-technical factors and the integration of formal and informal coordination approaches, we designed and implemented different software tools discussed in this paper. Those tools allow globally distributed developers to better manage and understand the context in which they perform their work (i.e., their organizational roles) and take action accordingly. Our current empirical studies, involving some of our prototypes, show some improvements in the way distributed users coordinate and manage their work. Users are not only better able to organize and understand their respective tasks, but to also self-coordinate their activities to avoid situations in which their work threatens to obstruct or interfere with the activities of others. Moreover, because many of the same methods and tools are applied in both collocated and globally distributed scenarios, the benefits of the Continuous Coordination paradigm are not limited to GSD contexts and can be applied in collocated large organizations.

The rest of this paper is organized as follows: The next section presents an extended analysis of current issues in GSD scenarios, in particular those related to coordination. Section 3 introduces the Continuous Coordination paradigm as one possible remedy to the issues discussed previously. Section 4 presents an overview of current prototype tools we implemented according to the Continuous Coordination paradigm as well as practical experiences with those tools. This paper concludes with a summary of findings and an outlook on future research.

2. Current Issues in Global Software Development

Herbsleb and Moitra [HeMo01] classify the dimensions of the most frequent problems encountered in GSD as follows: (a) strategic issues, (b) cultural issues, (c) inadequate communication, (d) knowledge management, (e) project and process management, and (f) technical issues (see also [CaAg01, HeMo03]). Each one of these problem dimensions demands a different approach and tools. Our primary concern in this paper is addressing **coordination** issues, particularly those related to communication, knowledge, and process management. In doing so, we also address some tool integration issues.

2.1 Coordination Issues

Existing software engineering tools have been used to support GSD, but they also raise several issues that hinder their use in such settings. Among the most distinctive issues are the lack of flexibility and integration with other tools, poor role support, inadequate workplace awareness [LaDO03], and the tension between formal and informal work [CuKI88, HeGr99, HeMo03, SoHR07]. These issues and their impact are discussed in greater detail below.

1st issue – Lack of flexibility and integration: Globally distributed sites normally employ software processes and development tools (editors, compilers, configuration management repositories, and so on). Problems often arise when developer attempt to integrate the artifacts produced by these tools. Developer find that the development tools have typically evolved asynchronously (different versions of the same compiler, for example) and are specific to the sites in which they are utilized. Such situations are common, and often originate from acquisitions and mergers of companies [HMFG00].

2nd Issue – Poor role support: The lack of communication and organizational awareness hinders proper coordination in a global organization. For example, developers are not always aware of overseas team structure, policies, responsibility for certain pieces of code, expertise in some API issues, and so on, which create communication barriers. Therefore, a system supporting collaboration in GSD must be able to direct particular information to particular people based on their roles [StJP99]. For example, the same bit of information that is highly important for one user is often completely uninteresting for another user in the same situation.

3rd Issue – Lack of informal communication and workplace awareness: Empirical studies suggest that informal communication is a very important factor that allow teams to cope with the uncertainty of tasks such as those involved in software development in general [CuKI88, KrSt95] and in GSD in particular [HMFG00]. The physical distance between sites makes it more difficult for distributed team members to spontaneously and informally communicate with one another. So-called serendipity encounters are an integral part of collocated team communication (e.g., when handling exceptions, correcting mistakes, adjusting predictions, and managing the effects of changes [HeGr99]). In addition, these informal communications also raise mutual awareness in collocated teams, so developers in GSD settings find it difficult to discern their colleagues' current activity [DCAC03] and whether it is appropriate to interrupt them at a certain time [BeBl96].

The ***4th*** and final issue software engineers generally face (that our research is concerned with) is the ***constraint that formal communication imposes during GSD***. This issue is discussed in the oncoming section within the context of current approaches to support collaboration and coordination in GSD.

2.2 Current Approaches

Several software engineering tools are designed and developed to support the coordination and collaboration issues discussed in the previous section. These tools generally either (1) rely

on formal (process-oriented) approaches, such as locks in configuration management or prescribed processes in workflow management systems, or (2) provide informal communication channels, as in the case of e-mail, IM, and the Web in general.

Formal approaches [BaAn02] follow specific process models or policies, either implicitly or explicitly defined by software tools. They promote the separation of work into multiple, independent tasks that are periodically resynchronized. These approaches are illustrated in row 1 of Table 1. The canonical example is a configuration management system by which a developer checking out artifacts becomes insulated from other (parallel) activities in the shared repository, whereas a developer checking in any modified artifacts resynchronizes his or her work with the work of the group.

Even though essential for the coordination of GSD teams, this approach suffers from two significant problems: First, formal processes can describe only parts of the activities of software development. No matter how formal and well-defined a process may seem, there is always a set of informal practices by which individuals monitor and maintain the process, keep it on track, recognize opportunities for action, and acknowledge the necessity for intervention or deviation [GeSt86]. Second, even when a process description attains a relatively high degree of detail and accuracy, the periodic re-synchronization of activities remains a difficult and error-prone task. In fact, the more parties are involved, the more conflicts arise and the more faults are introduced in the software at hand [PeSV01]. These problems are inherent in any tool that relies upon a formal encoding of collaborative work, because formal processes are inevitably surrounded by a set of informal practices by which the formal conditions are negotiated and evaluated. Moreover, tools designed for a specific process can prove to be less effective when implemented within the context of informal practices, introducing the challenge of overcoming heterogeneity [HMFG00].

Informal approaches usually rely on the notion of awareness – a concept that has become a central element of Computer-Supported Cooperative Work (CSCW) research, especially impacting the design of different collaborative systems [HeLu92, Schm02]. Awareness is an informal understanding of the activities of others that provides a context for monitoring and assessing group and individual activity [DoBe92, GiLT99]. An example is the mutual awareness of activities that arises in shared physical environments, where we can see and hear each other and "keep an eye out" for interesting or consequential events. Informal coordination therefore needs to provide continual visibility, that is, awareness of concurrent actions in order to foster self-coordination. The canonical example is the multi-user editor: by continuously displaying the ongoing activities of others, users typically self-coordinate by avoiding areas of the document in which others are currently working (see row 2 of Table 1)[ElGi91].

As with the formal, process-based approach, discussed in the previous section, the informal, awareness-based approach suffers from a significant problem that makes it a less-than-effective solution when it comes to coordination and collaboration. In particular, implementations of awareness-based approaches scale poorly; they are largely of value for small groups only. This is primarily caused by two factors: the users' cognitive limitations and the lack of process support. Although some mechanisms have been proposed for "asynchronous awareness" that can more easily support large-group collaboration [HHWM92], existing awareness technologies seem to work well for small groups, but break down for large groups. Consequently, the developers are faced with either formal or informal communications each of which presents its own limitations and advantages. The proposed paradigm, Continuous Coordination, endeavors to provide an alternative means of communication which combines the advantages of both formal and informal communication (Table 1). Continuous Coordination is detailed in the oncoming section.

3. Continuous Coordination

The formal and informal approaches discussed in the previous section have thus far always been treated as opposites. Developers have either looked toward formal processes or informal awareness to support coordination. Our research moves beyond this long-standing dichotomy and proposes an integrated paradigm to supporting collaborative work that combines formal and informal coordination strategies to provide both the tools and the information for users to increase self-coordination. Specifically, Continuous Coordination is a paradigm for the design of systems that provides mechanisms to support awareness sufficient to mediate coordination between otherwise isolated synchronization points (see Table 1). Continuous Coordination aims to combine the strengths of the formal and informal approaches while overcoming the current shortcomings of either one. It retains the checkpoints and measures of the formal approach to coordination (represented by circles in Table 1), but provides developers with a view of each other's relevant activities between these formal checkpoints (represented as arrows in Table 1). In doing so, it provides developers with ways to understand the potential relationships between their own work and the work of their colleagues. This is not a way to step outside the bounds of formal coordination; rather, it allows developers to better judge both the timing and the impact of formal coordination actions. We consider the occurrence of conflicts and other hindrances a normal part of any process and believe that any approach must integrally address them in a combined formal and informal way.

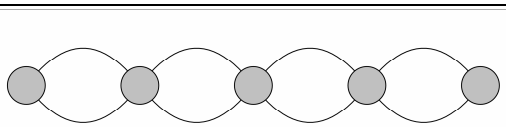
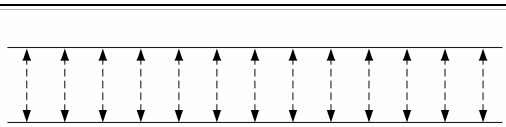
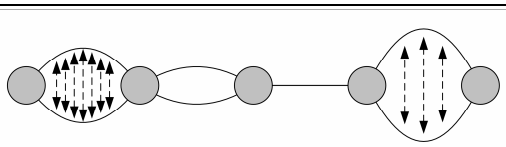
	Conceptual Visualization	*Strengths*	*Weaknesses*
Formal *Coordination*	 Prescribed synchronization points	Scalable; Insulation from other activities; Control; Group-centric	Reconciliation problems; Insulation becomes isolation
Informal *Coordination*	 Constant awareness	Flexible; Promotes synergy; Raises awareness; User-centric	Not scalable; must be initiated and managed manually
Continuous *Coordination*	 Awareness sufficient to mediate coordination between synchronization points	*Combines the strengths of both formal and informal coordination*	*Some applications specifically require isolation of developers (e.g., for security reasons)*

Table 1 An abstract summary of different coordination paradigms and their visualizations, strengths, and weaknesses. Arrows indicate informal communication whereas circles indicate coordination points for synchronization.

In terms of tool requirements, Continuous Coordination implies the integration of formal tools such as configuration management, workflow, and other kinds of process-oriented tools with awareness mechanisms, visualizations, and socio-technical aspects in order to improve awareness and the coordination of both distributed and collocated teams. In particular, we apply a combined set of design principles in the development of software tools that aim at addressing the main issues in GSD (see sections 2.1 and 2.2). From a systems perspective, we propose the use of multiple perspectives and tool integration; whereas from the collaborative perspective, we employ the integration of social and technical relations as well as support for formal and informal coordination. Those principles have been successfully applied separately

in specific context in other domains (as discussed in section 2.2 and by [Kruc95] and [TM81], among others), and now are being combined in the design of Continuous Coordination tools. While several principles are reported, we adopted only those that can potentially address the GSD issues that are the focus of our research. These principles are summarized below together with the GSD issues they address:

1. ***Multiple perspectives***: Rather than attempting to create a single, "one-size-fits-all" view of either data or process, we continually attempt to represent information from multiple perspectives. Examples include: (1) different perspectives on activity encoded by multiple simultaneous process descriptions, (2) different perspectives on an information space reflecting both the formal (checked-in, stable) and the informal (ongoing, active) states of activity; and (3) different perspectives on current activity from the viewpoints of two different members of a project team, to see their work in terms of each other's current state. Providing these multiple perspectives can increase the likelihood of developers sharing common views of the project, which has been stated as a need in GSD [CaAg01] and discussed in section 2.1 (issues 1 and 2).

2. ***Non-obtrusive integration***: Through the use of event-based integration and Web-based Collaborative Software Development Platforms (CSDP, [Robb05]), we sought to integrate different tools and information sources either through synchronous messages (event-based integration) or through the representation of links between different sites and artifacts (hypermedia integration). Adopting this Continuous Coordination principle within a GSD domain will enable traceability and transparency, which have previously been highlighted as important aspects of GSD [HPB05] and discussed in section 2.1 (issue 1).

3. ***Combination of social and technical factors****:* By the explicit integration and representation of socio-technical relations such as relations between artifacts (source code, documentation, change requests) and authorship (developers, managers, users), distributed developers can infer important context information that supports activities such as expertise location, intra-group communication, issue resolution, and organizational role support, among other issues discussed in section 2.1 (issues 2 and 3). Implementing this design principle in a Continuous Coordination prototype can also provide a means to detect and accurately document the socio-technical network to support GSD as it evolves [HeMo03].

4. ***Integrated formal and informal coordination approaches***: In keeping with our paradigm to integrate rather than separate multiple coordination approaches, we combine configuration management (for formal coordination) and change notification (for informal coordination) through the use of visualizations and integrated software development environments. For example, a shared change management system in GSD can enable access to information detailing the number and the nature of outstanding problems and necessary changes [HPB05]. Thus addressing issues identified in section 2.1 (issue 3) and section 2.2 (issue 4).

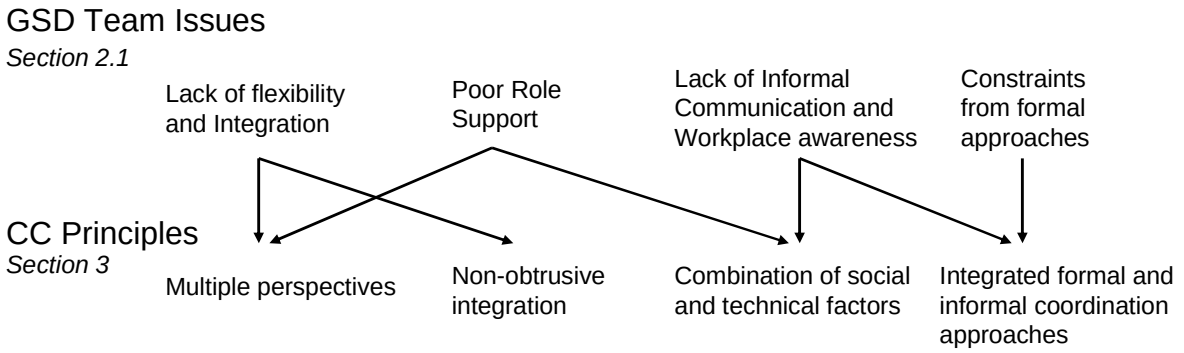


Figure 1 A summary of the relationship among the GSD team issues and Continuous Coordination (CC) principles.

Figure 1 provides a Summary of the GSD Team Issues we address and the Continuous Coordination (CC) design principles adopted in our paradigm. Those principles were used in the implementation of different tools as described in the next section.

4. Continuous Coordination Implementations

The Continuous Coordination paradigm was implemented through a series of tools rather than a single tool or single integrated environment to permit easier exploration of individual aspects of Continuous Coordination. Having multiple tools also separates tool usage from particular processes, affording developers the flexibility to integrate the tools into existing practices rather than introducing new ones. Despite the apparent diversity of our tools, they do share common characteristics (see Figure 1). Note that the different prototypes presented here are currently at varying stages of maturity. Each tool provides a different level of abstraction in terms of granularity and type of information that can be useful to specific developer needs. The prototypes are presented in an order according to their coverage of GSD activities, namely: YANCEES, Palantir, Ariadne, TraVis, and finally WorldView. Each prototype's concise set of features and its relation to GSD issues and Continuous Coordination principles, as well as present application experiences, are outlined in the following sections.

4.1 YANCEES Notification Service

The integration of information from different sources and their meaningful representation in different visualizations and tools require a communication infrastructure (or middleware) that supports heterogeneity – different data formats and network characteristics. In the Continuous Coordination paradigm, event-based integration [BaCT96], provided by notification servers, is used as the main communication and integration infrastructure that supports the implementation of different awareness strategies of the applications previously described.

Notification servers are brokers for system events, generally following a publisher-subscriber pattern [DGJN98]. In this communication style, different applications (information producers) publish information in the form of messages or events to a logically centralized service. Information consumers express interest in those events by means of subscriptions. This loosely coupled integration approach increases the flexibility of a distributed system by separating producers and consumers of information, thus enabling a "plug-and-play" approach that allows the introduction of new producers or consumers in the system.

In the GSD context, end users (e.g., designers, programmers, testers, and others) use different software tools. The end users' interactions with those tools generate notifications (for example, the check-in or check-out of artifacts to repositories, the implementation of a new method in a class, the start of a chat session and so on). Those notifications are first captured (by the monitoring of those tools) and then published to an event notification server.

Visualizations subscribe to those notifications and present this information to distributed developers or teams interested in that specific kind of information. For example, in the case of Palantír (discussed in the following section), the user's Integrated Development Environment (IDE) is constantly being monitored for events such as the modification of a source code file or check-in and out of artifacts in a Configuration Management repository. Those events are then propagated to other IDEs that use this same information to inform users about parallel artifact changes and activities. In WorldView (discussed in section 4.5), activities from distributed sites are kept current by the continuous notification of organizational or artifact changes.

The above description makes it seem straightforward that notification servers provide an infrastructure for keeping users aware of events of interests. However, some issues arise in practice: the generalization versus specialization dilemma: to design a one-size-fits-all infrastructure that supports a large set of application domains, resulting in complex implementations; or to develop an application-specific infrastructure, much simpler and more efficient, but with a limited scope. Both approaches usually provide weak support for customization; and poor support for extensibility (e.g., inability to support different notification policies [SBR02]). To cope with the heterogeneous set of requirements from different visualizations and the differences in events being produced by many information sources, without over simplifying or complicating its implementation, a notification service needs to be flexible. In other words, it needs to contract and expand its capabilities according to the needs of the applications at hand. For example, it needs to support persistence of events and pull notifications; to allow the retrieval of past event history in Palantír; and also to support the integration of different event sources, and push notification as the case of WorldView.

Such flexibility was one of the main challenges in our design of YANCEES, a publish/subscribe infrastructure designed to fulfill this role, providing different extension points around a common publish/subscribe model. YANCEES has been used to support different applications [SiRe05]. Through the use of an event-based infrastructure, many of the flexibility and integration issues demanded by our tools can be addressed, facilitating the integration of configuration management and change notification, as well as synchronous awareness mechanisms. In the context of Table 1, notification servers provide means by which asynchronous notifications (dotted arrows in the diagram) are performed.

4.2 Palantír

Palantír is a workspace awareness tool, which provides developers with insight into ongoing development activities in remote workspaces, providing them with context, and helping in their coordination – an important aspect of collocated software development that is missing in distributed development [SaNH03]. Palantír analyzes ongoing changes in artifacts from different developer workspaces. Those changes include local editions to source code files, Configuration Management operations such as check-ins and check-outs and synchronizations). Notifications about those changes are propagated to distributed Palantír peers through the use of notification servers (e.g. YANCEES) that collect and route information from and to interested parties. Palantír also calculates a measure of magnitude and the impact of those changes, and graphically displays this information in a configurable and non-obtrusive manner (see Figure 2). Palantír thus breaks the isolation in workspaces by continuously sharing information across GSD workspaces. The provision of information on parallel activities and potential conflicts in the project enables the early detection of parallel changes. Thus, even though the developers are notified of changes, the notification does not interrupt their work or force them to take immediate action. This enforces the concept of continues coordination (see Table 1, Row 3) where formal checkpoints (check-ins and check-

outs of files in our case) are retained, but are enhanced with awareness information (dotted lines in Table 1, Row 3); with the amount of information that is provided configurable by the users.

Figure 2 represents how parallel change notifications are presented within the Eclipse development environment in a way that minimizes users' context switches. For example, in the example of Figure 2, an artifact's icon (*Address.java)* is annotated with a blue decorator on its left, implying that it is being changed in parallel in another workspace. A text annotation, "S:24", is also provided, denoting the magnitude of the change. In this example, 24 lines have been changed in the artifact. The *Address.java* icon also has a red decorator on its right, along with a text annotation of "I>>", which implies that changes in *Address.java* are affecting other artifacts in the workspace. On a similar note, red decorators and "I<<" annotations on *CreditCard.java* indicate incoming conflicts in those artifacts (i.e., other users made changes in that file in their workspaces, and those changes were not yet commented to the repository). A log view (as seen on the bottom of the figure) presents which artifacts cause conflicts in other artifacts, as well as the reasons for the conflicts.

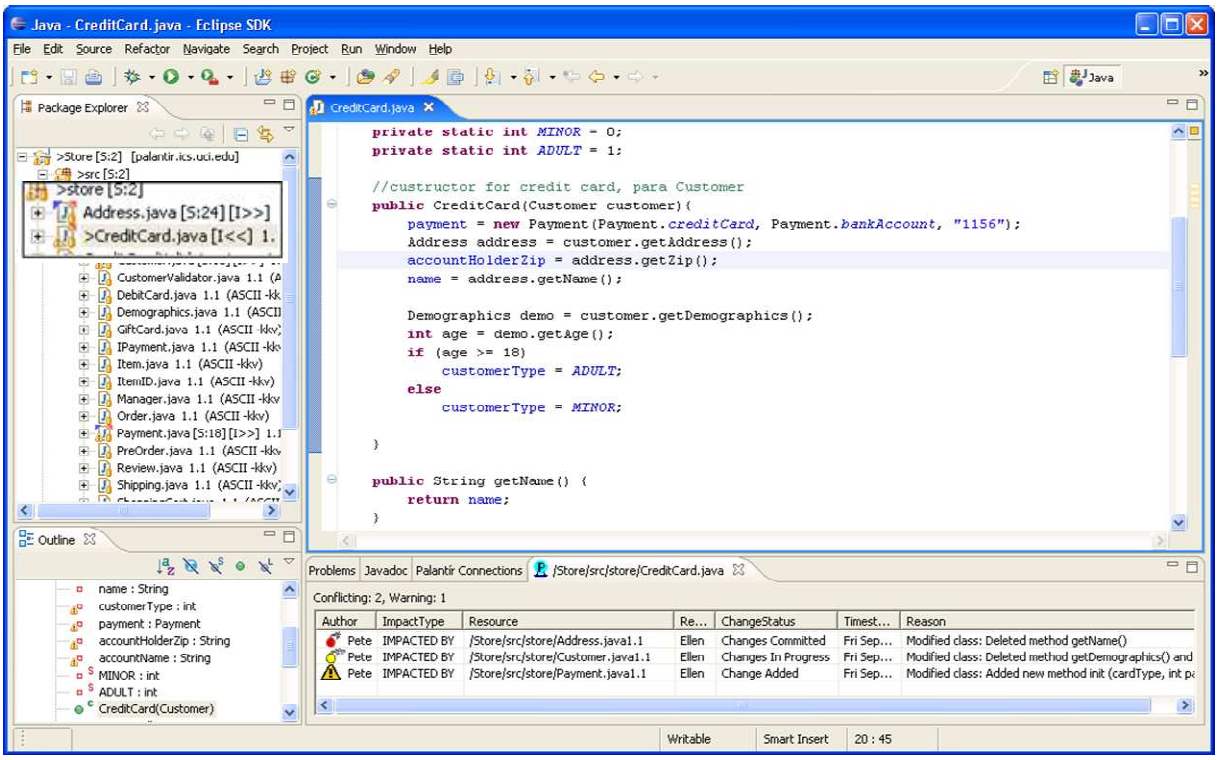


Figure 2 **Palantír screen shot: blue and red decorators indicate concurrent changes in documents in different severity levels.**

Palantír thus integrates multiple perspectives while it augments current configuration management tools with change notifications, which allows users to better coordinate their work during parallel software development. This approach prevents GSD conflicts due to merges that are a consequence of the insulation prescribed by many configuration management tool protocols.

Palantír has been evaluated through controlled laboratory experiments in which subjects completed a set of tasks in Java in a limited amount of time and within a GSD scenario [HMPR04]. As a result, the performance of the experimental group was found to be better than the control group's with respect to the amount of time taken per task and the number of conflicts detected and resolved. Subjects in the experimental group consistently detected

conflicts early on, as they were introduced, and coordinated with their team members to either avoid or resolve them.

4.3 Ariadne

In another prototype, we exploit the use of visualizations that express the socio-technical relation between artifacts. Source code is one of the most important artifacts in any software development effort. Ariadne [SDRQ04] is a collaborative software development tool that enhances developers' awareness of the social dependencies present in their work by seamlessly integrating such information with development activities in Eclipse. Ariadne analyzes software development projects for source code dependencies (e.g., data and control dependencies) and collects authorship information for the source code from a configuration management repository. The tool then links the source code dependencies and authorship information to create a social network of software developers. This social network describes the dependencies among software developers that arise out of the dependencies in their code. Through this novel approach, Ariadne allows the analysis and visualization of social dependencies that surround the source code, which is crucial for informal coordination, expertise location, role awareness, and change impact analysis.

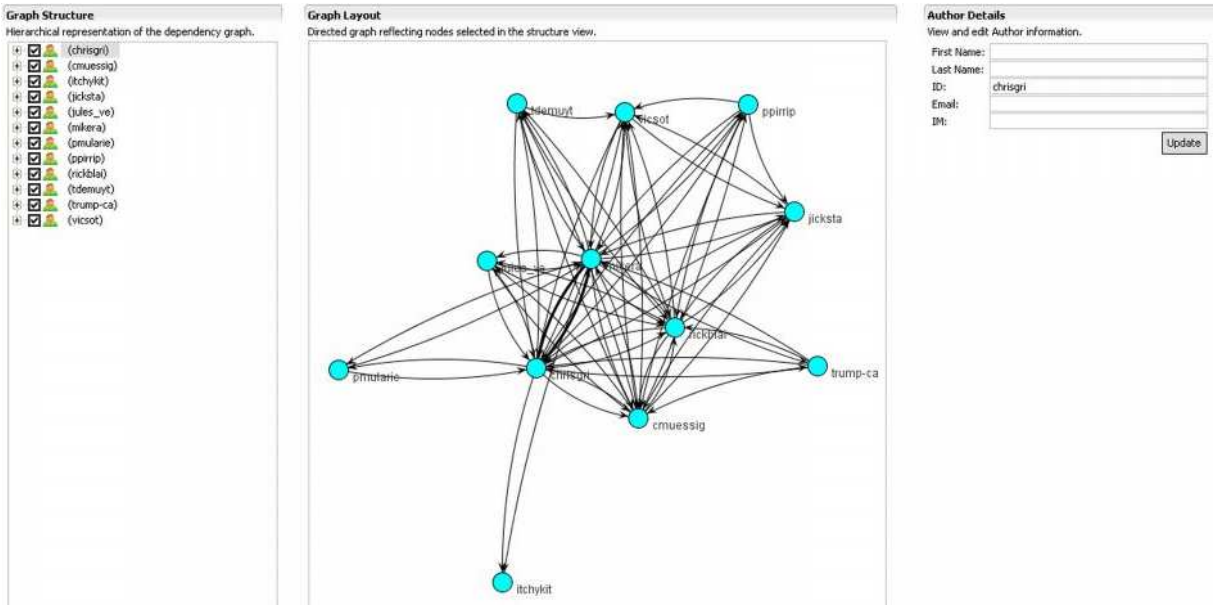


Figure 3 Example of graphical representations of social dependencies in large GSD projects produced by Ariadne

Figure 3 shows a social dependency graph for the open-source Java project "Tyrant," available on Sourceforge.net. Cyan-colored nodes represent developers in the project and directed arrows from one developer to another indicate that the former depends on the latter. Thicker edges between nodes represent stronger social dependencies, indicating the presence of more technical dependencies between developers. Developers can selectively display authors in the visualization by using the check boxes in the left column, and can view author contact information in the right column by selecting a developer node in the graph. By analyzing the social dependency graph, developers can see that dependencies in their source code create social dependencies between them and their colleagues. Depending on the strength of the social dependency, two developers may feel more compelled to begin coordinating their work.

In general, Ariadne combines the Continuous Coordination principles of integrating socio-technical and social network visualizations that combine developers, artifacts, and the source code dependencies. The developers can access these graphical representations of social dependencies and actionable information at a time suitable to them, thus improving their overall awareness about the social and technical dependencies of a distributed software project.

We conducted semi-structured interviews with four software developers at a large software development company (field study-based evaluation [HMPR04]). Questions in the interviews explored the usefulness of Ariadne's dependency information and the usefulness of the information present in the social network visualization. The core members of the project benefited less directly from Ariadne since they were located in the same building; however, they acknowledged that Ariadne would be useful in their organization because of the larger integration effort currently taking place that would lead several distributed and formerly independent teams to work together.

4.4 TraVis

Collaborative software development platforms (CSDPs) are suites of tools that comprise and unify multiple software development and knowledge management tools. These include source code management, issue trackers, Wikis, and discussion forums – tools that often have been successfully used in distributed open source software development projects [AuBS02, Robb05].

The TraVis (Trace Visualization) tool goes beyond Ariadne by leveraging the use of dependencies among different artifacts and their users (instead of source code authorship only). For instance, a CSDP associates tracker items, which represent development tasks, with different documents and their respective authors (see Figure 4, TSK-1195). This creates a rich semantic network, the "traceability network", with task-related entities, artifacts, and users [LiSa96]. Also including code artifacts, CSDP provide a broader spectrum of artifact dependencies and TraVis enables the visualization and analysis of the different relations that might occur. Traceability and rationale information from CSDP is captured as users in GSD projects develop and document their processes. Artifacts are annotated and connected with their respective descriptions, discussions, and tasks represented as tracker items form a heterogeneous network of information. Thus TraVis lays hold of the principles of non-obtrusive integration, multiple perspectives, and the integration of socio-technical issues in a CSDP, i.e. user-artifact relations (cp. section 4.3). Managing all this information on one single CSDP allows creating the links necessary to establishing the actual traceability network [LiSa96]. The networked project information facilitates advanced contextual analyses of different kinds.

TraVis provides several filters for displaying certain aspects of the traceability network (e.g., particular artifact types, process categories, or user groups; see checkboxes on the left side in Figure 4). Thus, different role-based views (for developers, designers, project managers, and so forth), and different projections of the network, can be defined. Moreover, TraVis visualizes networks originating from particular artifacts, activities, and users: For example, Figure 4 shows the adjacency graph of task 1195 (i.e., all other related artifacts and users), which allows distributed developers in the process of identifying group members and understanding the relations and dependencies between artifacts, thus establishing context. Therefore, TraVis also provides increased awareness within GSD projects based on a broad range of information from CSDP, including source code. Through real-time graphical representations and analysis, collaboration among stakeholders can improve, for example, by facilitating impact analysis, e.g. when requirement change late in the project and the

propagation of this change has to estimated, as well as informal team communication. The increased process transparency also facilitates project management tasks. In order to provide a better overview in complex GSD projects with hundreds of related artifacts, TraVis evaluates artifacts according to their importance; this is represented by different vertex sizes in Figure 4 (for example REQ-1316 and TSK-1175). This importance metric is based on the value assigned to the initial set of requirements as well as the task priorities set by the project manager [GeHi06].

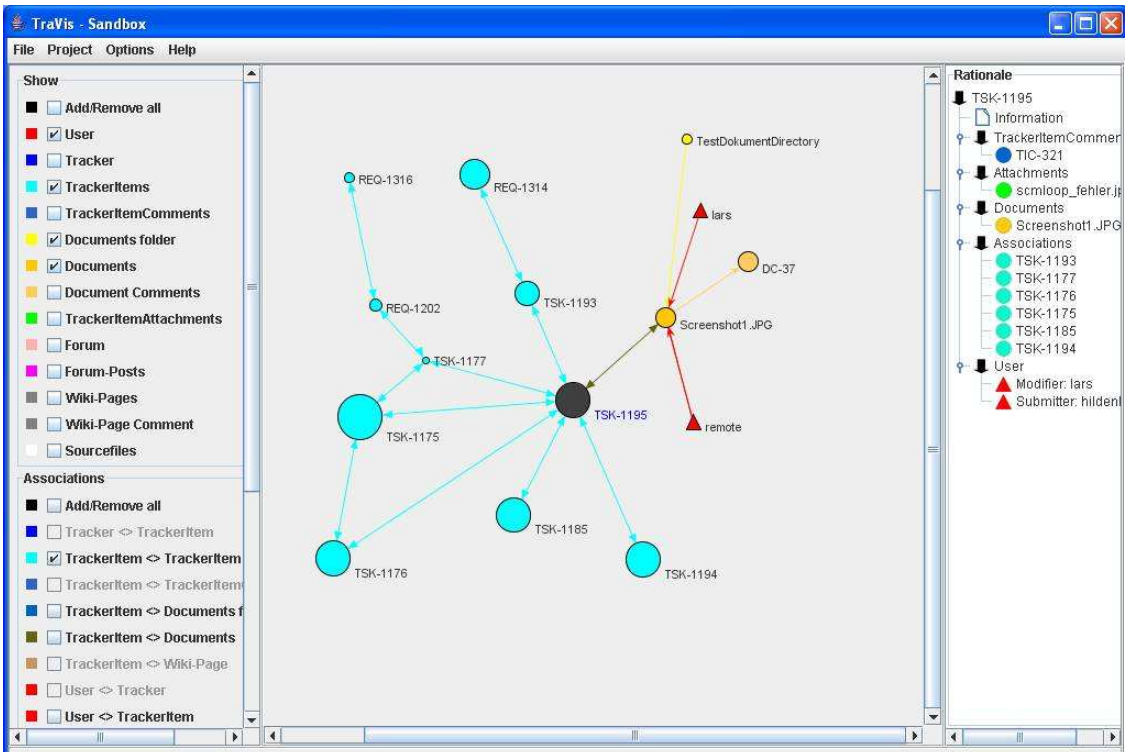


Figure 4 Traceability relations from a GSD project extracted by TraVis

To this point, TraVis has been evaluated by several CSDP vendors and their customers as well as by open source software developers, i.e. domain experts. As with Ariadne, subjects were asked about the usefulness and value the tool provides compared to off the shelf hypermedia-based CSDP. Thus, the tool was evaluated through an empirical field study [HMPR04]. TraVis' core features are based on observations from CSDP vendors' change request lists, that is, features customers were missing in their daily business. These include more purposeful change propagation as well as multiple linking mechanisms and traceability [SoHR07]. After implementing all projected features, TraVis will be systematically evaluated in controlled GSD-like scenarios, such as globally distributed student programming projects [BGHR07].

4.5 WorldView

WorldView provides different visualizations in several levels of abstraction, which extends from low-level artifact representation (design models) to high-level representation of team interactions. This allows a comprehensive view of the team dynamics within a project, geographical location of teams, time zones, and the interdependencies among teams [SaHo06]. The ability to navigate through different levels of abstraction enables the support of developers within different organizational roles. Similar to Ariadne and TraVis, the visualization of the highest level of abstraction of development artifacts can help developers (e.g. designers and managers) to identify global and local team members, interactions between

subgroups, and other vital information such as how and when to contact global members of teams. The world map view of the project context is particularly designed for GSD scenarios.

Figure 5 A representation of team structure, development tasks, and other essential information about GSD projects in WorldView

In the example shown in Figure 5, one possible level of abstraction is presented. In this figure, the teams are represented as “red balloons” on a world map, and interdependencies among teams are represented as “lines” connecting them. Interdependencies among teams are determined based on the number of shared artifacts, which are identified through program analysis of their common code base. The thickness of the lines represents the extent of sharing: the thicker the lines, the larger the number of shared artifacts. Through this view, developers can discern, at-a-glance, which teams are tightly coupled through which artifacts. It also supports organizational role awareness through the visualization of users and their organizational roles (displayed by “mousing over” a site). By allowing team members to be aware of interdependencies and by integrating organizational aspects in the same visualization, this tool demonstrates the implementation of the Continuous Coordination principles of non-obtrusive integration, socio-technical issues, and multiple views.

This tool is still in the exploratory phase of development, with descriptive evaluation plans in place to develop supportive arguments for its utility. Future plans also include the realization of black box and white box testing [HMPR04]. Finally, we intend to conduct an experimental study of the tool within a simulated environment for evaluation [HMPR04].

5. Conclusions and Future Research

Software development performed by traditional collocated teams is an intrinsically difficult task. It is difficult because of the complex nature of the activities being executed requiring strong coordination and communication among developers. When performed in a globally distributed setting, software development becomes even more challenging. In GSD, several problems arise due to the physical, social, and cultural difference between the developers. In particular, less informal communication among the team members results in a lack of mutual awareness and difficulty in establishing trust.

The Continuous Coordination paradigm strives to address those issues by integrating and enhancing existing formal and informal coordination strategies, implemented in current

software engineering tools, as a way to improve the coordination in distributed settings. This is achieved by the use of multiple perspectives, non-obtrusive integration, the combination of social and technical factors, and the combination of configuration management and change notification as design principles for corresponding tools.

Figure 6 summarizes the interrelationships of GSD team needs, the Continuous Coordination design principles that strive to meet those needs, the prototypes that have been developed based on those principles, and the infrastructure they use. The prototype tools we developed allow us to explore our theory and open the door for further theoretical development and evaluation.

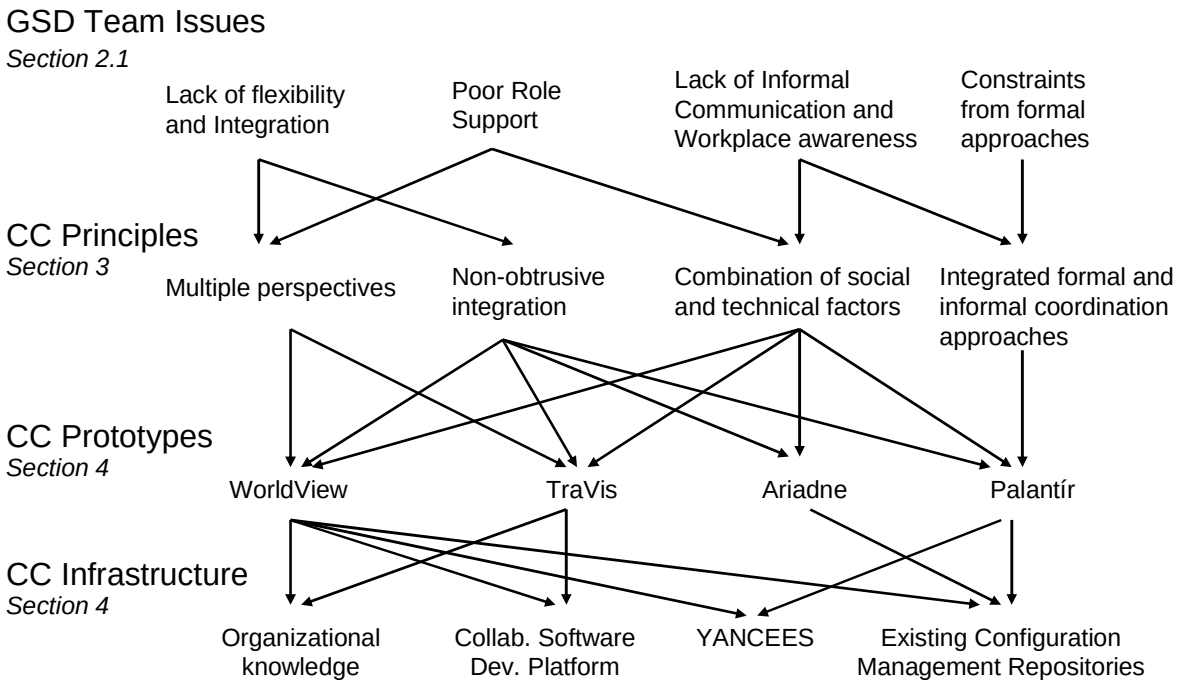


Figure 6 A summary of the relationship between Continuous Coordination principles, prototypes and strategies

WorldView supports the Continuous Coordination principles of multiple perspectives, non-obtrusive integration, and the combination of social and technical factors in a shared visualization of different organizational, technical-social aspects of GSD. It relies both on event-based integration (**YANCEES**), as well as organizational knowledge and information from CSDP. WorldView is current being extended to include alternate forms of team interactions. It will display the impact of a specific change on teams. This aspect of the tool will embody the social and technical factors of GSD by explicitly integrating and representing of socio-technical relations (changes, impact, and authorship).

TraVis supports the Continuous Coordination principle of multiple perspectives through its role-based view on the traceability network. Moreover, the tool provides transparency, traceability, and awareness in a non-obtrusive way by using real-time data from CSDP used in GSD projects. This allows, for instance, developers to establish adequate communication based on the project's context, e.g. when questions about dependent artifacts arise later in the process.

Ariadne combines social (authorship) and technical (source code) dependencies in analysis and visualization of social dependencies that support developers in their informal

coordination, expertise location, role awareness, and change impact assessment. Ariadne is being augmented to support different social network metrics that will provide developers with insight into how their work affects other developers' work and how the work of other developers affects their own.

Palantír supports workspace awareness in parallel development scenarios by integrating configuration management and change notification. It is built upon notification servers such as **YANCEES** and existing CM repositories. So far, no major changes are scheduled and the current functionality has already been evaluated through controlled laboratory experiments.

The tools demonstrate how the Continuous Coordination paradigm seeks to promote self-coordination through continuous provision of information generated as a result of development activities. The tools enable the developers to share an awareness of activities carried out during development in such a manner that developers are not constrained by the lack of information, nor is their work blurred by a high volume of information. We sought to achieve a balance by enabling access to necessary information only at a time suitable to the developers. Practical experiences within our research group and evaluations of particular tools (Palantír, Ariadne, TraVis) in either experimental or field environments indicated the usefulness of Continuous Coordination-based tools in distributed and GSD settings.

Continuous Coordination can be seen as a solution to several GSD issues (i.e., the lack of informal communication and mutual awareness). Furthermore, because methods and tools initially designed for distributed development teams are often adopted in "collocated" sites, Continuous Coordination tools are useful in these contexts as well.

Acknowledgments

This research was supported by the U.S. National Science Foundation under grants 0534775, 0326105, 0093489, and 0205724, by the Intel Corporation, by two IBM Eclipse Technology Exchange grants, an IBM Technology Fellowship, and by the Brazilian Government under CAPES Grant BEX 1312/99-5. Part of this work is also a result of the project CollaBaWue supported by the German state of Baden-Wuerttemberg. CollaBaWue is part of the research association PRIMIUM. We thank the anonymous reviewers for their comments on an earlier draft. We also thank our colleague Steve Abrams for helpful criticisms. Finally, we thank our technical editor, Sandra Rush, who greatly improves the readability of our work for others.

References

[AuBS02] *Augustin, L.; Bressler, D.; Smith, G.*: Accelerating Software Development through Collaboration. In: Proceedings of the International Conference on Software Engineering 2002 (ICSE'02). Orlando, FL, USA, 2002, pp. 559-563.

[BaAn02] *Barthelmess, P.; Anderson, K.M.*: A View of Software Development Environments Based on Activity Theory. In: Computer Supported Cooperative Work, Kluwer Academic Publishers, 11 (2002) 1-2, pp. 13-37

[BaCT96] *Barrett, D.J.; Clarke, L.A.; Tarr, P.L.; Wise, A.E.*: A Framework for Event-based Software Integration. In: ACM Transactions on Software Engineering and Methodology, 1996, 5, pp. 378-421.

[BeBl96] *Bellotti, V.; Bly, S.*: Walking Away from the Desktop Computer: Distributed Collaboration and Mobility in a Product Design Team. In: Proceedings of the 1996 ACM Conference on Computer Supported Cooperative Work (CSCW '96), ACM Press, 1996, pp. 209-218

[BGHR07] *Berkling, K.; Geisser, M.; Hildenbrand, T.; Rothlauf, F.*: Offshore Software Development: Transferring Research Findings into the Classroom. In: Proceedings of the First International Conference on Software Engineering Approaches For Offshore and Outsourced Development (SEAFOOD), Zuerich, Switzerland, 2007 (accepted).

[CaAg01] *Carmel, E.; Agarwal, R.*: Tactical Approaches for Alleviating Distance in Global Software Development. In: IEEE Software, 18 (2001) 2, pp. 22-29.

[CuKI88] *Curtis, B.; Krasner, H.; Iscoe, N.*: A Field Study of the Software Design Process for Large Systems. In: Communications of the ACM, 31 (1988) 11, pp. 1268-1287.

[DCAC03] *Damian, D.; Chisan, J.; Allen, P.; Corrie, B.*: Awareness Meets Requirements Management: Awareness Needs in Global Software Development. In: Proceedings of the International Workshop on Global Software Development, Portland, Oregon, 2003.

[DGJN98] *Dingel, J.; Garlan, D.; Jha, S.; Notkin, D.:* Reasoning about Implicit Invocation. In: 6th International Symposium on the Foundations of Software Engineering (FSE-6), Lake Buena Vista, FL, USA, 1998.

[DoBe92] *Dourish, P.; Bellotti, V.*: Awareness and Coordination in Shared Workspaces. In: Proceedings of the 1992 ACM Conference on Computer-Supported Cooperative Work (CSCW '92), ACM Press, 1992, pp. 107-114.

[ElGi91] *Ellis, C.A.; Gibbs, S.J.;Rein, G.:* Groupware: Some Issues and Experiences. In: Communications of the ACM, 34(1991) 1, pp. 39-58.

[GeHi06] *Geisser, M.; Hildenbrand, T.*: Agiles, verteiltes Requirements-Engineering mit Wikis und einer kollaborativen Softwareentwicklungsplattform. In: OBJEKTSpektrum, 2006 (6), pp. 37-42.

[GeSt86] *Gerson, E.M.; Star, S.L.:* Analyzing Due Process in the Workplace. In: ACM Transactions on Office Information Systems 4 (1986) 3, pp. 257-270.

[GiLT99] *A. Girgensohn; A. Lee; T. Turner*: Being in Public and Reciprocity: Design for Portholes and User Preference. In: Proceedings of the Seventh IFIP Conference on Human-Computer Interaction (INTERACT '99), IOS Press, pp. 458-465, 1999.

[Grud94] *Grudin, J.*: Computer-Supported Cooperative Work: History and Focus. In: IEEE Computer, 27 (1994) 5, pp. 19-26.

[HeGr99] *Herbsleb, J.D.; Grinter, R.E.*: Architectures, Coordination, and Distance: Conway's Law and Beyond. In: IEEE Software, 16 (1999) 5, pp. 63-70.

[HeLu92] *Heath, C.; Luff, P.*: Collaboration and Control: Crisis Management and Multimedia Technology in London Underground Control Rooms. In: Computer Supported Cooperative Work, 1 (1992) 1-2, pp. 69-94.

[HeMo01] *Herbsleb, J.D.; Moitra, D.*: Global Software Development. In: IEEE Software 18 (2001) 2, pp. 16-20.

[HeMo03] *Herbsleb, J.D.; Mockus, A.*: An Empirical Study of Speed and Communication in Globally-Distributed Software Development. In: IEEE Transactions on Software Engineering 29 (2001) 6, pp. 481-494.

[HHWM92] *Hill, W.C.; Hollan, J.D.; Wroblewski, D.; McCandless, T.*: Edit Wear and Read Wear. In: Proceedings of the SIGCHI Conference on Human Factors in Computing Systems (CHI '92), ACM Press, 1992, pp. 3-9.

[HMFG00] *Herbsleb, J.D.; Mockus, A.; Finholt, T.A.; Grinter, R.E.*: Distance, dependencies, and Delay in a Global Collaboration. In: Proceedings of the 2000 ACM Conference on Computer Supported Cooperative Work (CSCW '00), ACM Press, 2000, pp. 319-328.

[HMPR04] *Hevner, A.R.; March, S.T.; Park, J.; Ram, S.*: Design Science Information Systems Research. In: MIS Quarterly, 28 (2004) 1, pp. 75-105.

[HPB05] *Herbsleb, J.D.; Paulish, D.J.; Bass, M.*: Global Software Development at Siemens: Experience from Nine Projects. In: Proceedings of the International Conference on Software Engineering (ICSE 05), St. Louis, MO, USA, May 15-21, 2005.

[JaLe99] *Jarvenpaa, S.L.; Leidner, D.E.*: Communication and Trust in Global Virtual Teams. In: Organization Science, 10 (1999) 6, pp. 791-815.

[KrSt95] *Kraut, R.; Streeter, L.A.*: Coordination in Software Development. In: Communications of the ACM, 38 (1995) 3, pp. 69-81.

[Kruc95] *Kruchten, P. B.*: The 4+1 View Model of architecture. IEEE Software, 1995, 12(6): 42-50.

[LaDO03] *Lanubile, F.; Damian, D.; Oppenheimer, H.L.*: Global Software Development: Technical, Organizational, and Social Challenges. In: SIGSOFT Software Engineering Notes, 28 (2003) 6, pp. 2-5.

[LiSa96] *Lindvall, M.; Sandahl, K.*: Practical Implications of Traceability. In: Software – Practice & Expererience, 26 (1996) 10, 1161-1180.

[OlOl00] *Olson, G.; Olson, J.*: Distance Matters, In: Journal of Human-Computer Interaction, 15 (2000) 2-3, pp. 139-178.

[PeSV01] *Perry, D.E.; Siy, H.P.; Votta, L.G.*: Parallel Changes in Large-Scale Software Development: An Observational Case Study. In: ACM Transactions on Software Engineering and Methodology, 10 (2001) 3, pp. 308-337.

[Robb05] *Robbins, J.*: Adopting Open Source Software Engineering (OSSE) Practices by Adopting OSSE Tools. In: Feller, J.; Fitzgerald, B.; Hissam, S.A.; Lakhani, K.R. (Hrsg.): Free/Open Source Processes and Tools, MIT Press, Cambridge, MA, USA, 2005, pp. 245-264.

[SaHo06] *Sarma, A.; van der Hoek, A.*: Towards Awareness in the Large. In: First International Conference on Global Software Engineering, Costão do Santinho, Florianópolis, Brazil, October 2006.

[SaNH03] *Sarma, A.; Noroozi, Z.; van der Hoek, A.*: Palantír: Raising Awareness among Configuration Management Workspaces, In: Proceedings of the Twenty-Fifth International Conference on Software Engineering, Portland, Oregon, USA, 2003, pp. 444-453.

[SBR02] *Souza, C. R. B. d.; Basaveswara, S.D.; Redmiles, D.F.*: Using Event Notification Servers to Support Application Awareness. In: IASTED International Conference on Software Engineering and Applications, Cambridge, MA, USA, 2002, pp. 691-697.

[Schm02] *Schmidt, K.*: The Problem with "Awareness" – Introductory Remarks on "Awareness in CSCW." In: Journal of Computer Supported Cooperative Work, 11 (2002) 3-4, pp. 285-298.

[SDRQ04] *de Souza, C.; Dourish, P.; Redmiles, D.; Quirk, S.; Trainer, E.*: From Technical Dependencies to Social Dependencies. In: Proceedings of the Social Networks Workshop at the 2004 International Conference on CSCW, Chicago, IL, USA, 2004.

[SoHR07] *de Souza, C.; Hildenbrand, T.; Redmiles, D.*: Towards Visualization and Analysis of Traceability Relationships in Distributed and Offshore Software Development Projects. In: Proceedings of the First International Conference on Software Engineering Approaches For Offshore and Outsourced Development (SEAFOOD), Zuerich, Switzerland, 2007 (to appear).

[SiRe05] *Silva Filho, R.S.; Redmiles, D.*: Striving for Versatility in Publish/Subscribe Infrastructures. In 5th International Workshop on Software Engineering and Middleware (SEM'2005), ACM Press, 2005, pp. 17-24.

[StJP99] *Steinfield, C.; Jang, C.; Pfaff, B.*: Supporting Virtual Team Collaboration: The TeamSCOPE System. In: Proceedings of the International ACM SIGGROUP Conference on Supporting Group Work (GROUP '99), ACM Press, 1999, pp. 81-90.

[TM81] *Teitelman, W. and Manister, L.*: The Interlisp Programming Environment. IEEE Computer., 1981, 14: 25-33.