Journal of Information Technology (2007) 22, 161–173
© 2007 JIT Palgrave Macmillan Ltd. All rights reserved 0268-3962/07 $30.00
palgrave-journals.com/jit

Research article

Globally distributed component-based software development: an exploratory study of knowledge management and work division

Julia Kotlarsky[1], Ilan Oshri[2], Jos van Hillegersberg[3], Kuldeep Kumar[4]

[1]Warwick Business School, Warwick University, Warwick, UK;
[2]Rotterdam School of Management, Erasmus University, Rotterdam, The Netherlands;
[3]Department of Information Systems and Change Management, University of Twente, AE Enschede, The Netherlands;
[4]College of Business Administration, Florida International University, Miami, FL, USA

Correspondence: J. Kotlarsky, Warwick Business School, Warwick University, CV4 7AL Coventry, Warwick, UK.
Tel: +44 2476 524692;
Fax: +44 2476 524539;
E-mail: Julia.Kotlarsky@wbs.ac.uk

Abstract

Component-based development (CBD) can be an appealing proposition to globally distributed software development teams because of the almost endless possibilities to recombine and reuse components in new products. In particular, it has been suggested that CBD will improve globally distributed software development practices by allowing each site to take ownership of particular components, resulting in reduced inter-site communication and coordination activities. Such an approach may indeed overcome breakdowns in inter-site coordination efforts; however, it may also lessen opportunities to share knowledge between sites and may hamper opportunities to reuse existing components. A case study approach, exploratory in nature, was adopted to explore knowledge aspects in global component-based software development. Evidence collected at several globally distributed CBD projects suggests that the true potential of CBD, which mainly relates to reusing components, can also be achieved when components are developed in a joint manner (i.e. by several sites) by accessing and utilizing *expertise* regardless of its geographical location. To improve the rate of component reuse, the studied teams developed capabilities in three particular areas: inter-site coordination, communications, and knowledge management. The paper concludes by discussing the links between component reuse, CBD principles and organizational capabilities, and offers managers and engineers some guidelines to consider in their CBD projects.

Journal of Information Technology (2007) **22,** 161–173. doi:10.1057/palgrave.jit.2000084
Published online 2 January 2007
Keywords: component-based development; global software development; component reuse; expertise

Introduction

Global distribution of software development has become widespread over the last 15 years (Herbsleb and Moitra, 2001). Moreover, there are a number of economic and technical drivers that are likely to accelerate the growth of distributed software development. For one to access a larger pool of expertise, often in low-cost geographical locations, many companies are switching to globally distributed software development or offshore outsourcing of software products and services. On the technological side, ongoing innovations in information and communication technologies (ICT) improve the possibilities to cooperate in a distributed mode.

Indeed, geographical distance and the time zone and cultural differences associated with global distribution have caused problems for globally distributed software teams in achieving successful collaboration. A growing number of

studies have reported problems regarding collaboration in distributed work, such as coordination breakdowns (Carmel, 1999; Cheng *et al*., 2004), lack of understanding of a counterpart's context (Cramton, 2001), and different language competencies across remote sites (Sarker and Sahay, 2004). Other studies have argued that globally distributed work may exacerbate the chance of misunderstandings (Battin *et al*., 2001; Olson and Olson, 2004), lack of trust (Jarvenpaa *et al*., 1998; Cramton, 2001), asymmetry in distribution of information among sites (Carmel, 1999), difficulty in collaborating due to different skills and training, and mismatches in information technology (IT) infrastructure (Sarker and Sahay, 2004).

The practices recommended to overcome these difficulties have mainly focused on a division of work that minimizes the need for inter-site coordination and, therefore, communication and synchronization (Ebert and De Neve, 2001; Mockus and Weiss, 2001; Repenning *et al*., 2001; Herbsleb and Mockus, 2003). To achieve this, it was recommended that 'tightly coupled work items that require frequent coordination and synchronization should be performed within one site' (Mockus and Weiss, 2001).

In this regard, past research (e.g. Carmel, 1999) has indicated that the adoption of component-based development (CBD), which aims to reduce the costs involved in developing new products and to improve time to market of new products through the reuse of components across multiple products (Huang *et al*., 2003), may also facilitate globally distributed software development. According to this view, components can be developed independently from remote locations (Repenning *et al*., 2001) and, therefore, mitigate some coordination breakdowns associated with the traditional, non-component-based (CB) globally distributed software development (Carmel, 1999). Put simply, each site should take responsibility to develop a particular component with minimum inter-site communication and coordination activities (Repenning *et al*., 2001). We claim that such an approach may, in fact, result in fewer opportunities to reuse components because remote sites may have limited exposure to the knowledge of components developed by other remote teams and because of the limited knowledge sharing between remote counterparts (Kotlarsky and Oshri, 2005).

In this paper, we report on two companies that followed an approach, in which the development of a component was jointly carried out across several sites. We explore the capabilities that were imperative to achieve component reuse in globally distributed CBD projects. Following this introduction, the next section will provide a historical background of CBD, followed by a review of the literature on CBD in globally distributed software development projects. The research method section then poses the key questions that the studied companies were facing and explain the research approach, exploratory in nature, adopted for this study. The following section, based on the empirical evidence, provides the rationale that guided the companies in pursuing a division of work, based on the expertise available within the dispersed teams regardless of their geographical location (as opposed to the approach in which one site takes ownership of the development of a particular component). Next, the empirical section outlines the capabilities developed by these companies in order to support component reuse in globally distributed CBD projects. Lastly, practical implications are offered to managers and engineers.

Component-based development: background

CBD has its roots in manufacturing. Since the mid-1960s, when the concept of modular production was introduced (Starr, 1965), *modular* (later referred to as CB) product architectures became dominant in manufacturing industries. Indeed, the trend to develop products that have a CB architecture is well established in automotive, electronics, aircraft, and other industries, and it has been successfully used for setting up globally distributed design and production activities. For example, in the computer industry, Dell products include components produced by different vendors in various locations. In the automotive industry, the design of a car and the manufacture of car components both involve designers and component suppliers at various dispersed locations (Olin *et al*., 1999). Even a very large and complex product such as an aircraft could be developed from remote locations, as in the case of Boeing-Rocketdyne (Malhotra *et al*., 2001), Boeing 777 (Yenne, 2002), and Airbus.

In the software development industry, CBD involves (i) the development of software components and (ii) the building of software systems through the integration of pre-existing software components (Bass *et al*., 2000). *Components* are units of independent production, acquisition, and deployment that interact with each other to form a functioning system (Szyperski, 1998). Being self-contained and replaceable units, components can be reused across a number of products, and be replaced by more recent and advanced versions of components in a 'plug-and-play' manner, as long as the interfaces across the components comprising the products are compatible.

Indeed, the idea to develop software components goes back to the early days of computer systems in the form of subroutines and subroutine libraries. In this regard, the transition in software development took place as '[…] programmers observed that they could insert subroutines extracted from their previous programs, or even written by other programmers, and take advantage of the functionality without having to concern themselves with the details of coding. Generally-useful subroutines were collected into libraries, and soon very few people would ever again have to worry about how to implement, for example, a numerically-well-behaved double-precision cosine routine' (Clements, 1995). Nevertheless, until the mid-1990s reusability was limited because module interfaces were programming language specific and even platform specific ones. Also, specification techniques to document module interfaces were insufficiently developed. Finally, the Internet as an effective means to facilitate the development and distribution of software modules was not as widespread as it is today. These limitations have hindered the development of global software component reuse (Traas and van Hillegersberg, 2000).

With the introduction of software component technologies in the mid-1990s, such as Enterprise JavaBeans, Microsoft COM, and CORBA (Peters and Pedrycz, 2000), there have been expectations that CBD will offer additional

avenues to software development and reuse opportunities. As opposed to a monolithic system, a CB system has a number of advantages for production and competitiveness. In particular, a CB system allows changes to be made to isolated functional elements of the product without affecting the design of other components (Ulrich and Eppinger, 2000). Also, a CB system supports the rapid customization of products through the reuse of existing standardized components, resulting in an improved product variation, higher utilization of existing assets, better reliability of products (Vitharana, 2003), shorter time to market (Bass *et al*., 2000; Crnkovic and Larsson, 2002; Kim, 2002; Huang *et al*., 2003; Vitharana, 2003), and lower costs associated with product development (Kunda and Brooks, 2000; Ravichandran and Rothenberger, 2003).

Along with this positive outlook, some possible challenges associated with CBD that co-located software development teams have faced, have been reported in the literature. For example, it has been argued that 'it often took longer to develop a reusable component than to develop a system for a one-off purpose' (Huang *et al*., 2003). Other studies have reported difficulties associated with the management of CBD in co-located projects, such as a lack of stable standards, lack of reusable components, and problems related to the granularity and generality of components (Crnkovic and Larsson, 2002; Vitharana, 2003).

CBD in globally distributed software development contexts

As discussed above, globally distributed teams represent a new organizational form that has emerged in conjunction with the globalization of socio-economic processes. Such teams have replaced the traditional single-site hierarchy and the functional department structure for various reasons. For one, companies in developed nations have outsourced parts of their IT services and business processes to developing nations (Carmel and Agarwal, 2002). Short-cycle development and the launch of new products and software for global markets have required expertise from a range of geographical areas (Desouza and Evaristo, 2004). Indeed, an increasing number of companies are setting up software development in a globally distributed mode and at the same time, are adopting a CBD methodology. At the basis of this approach is the assumption that in globally distributed software development, particular parts of large software systems reappear with sufficient reliability such that common parts can be written once, rather than many times, and that common systems can be built through reuse rather than being rewritten over and over.

Alongside such advances in software development, recent years have witnessed the emergence of open source software (OSS) development. OSS projects develop 'software whose source code is distributed without charge or limitations on possible modifications and distributions by third parties' (Crowston and Scozzi, 2002: 3). In most cases, the development of OSS is entirely global and is voluntarily carried out by individuals sometimes through the application of CBD principles such as the use of JavaBeans. While there are some similarities between OSS and commercially driven globally distributed CBD projects in terms of the application of specific methodologies and development tools, being a commercially driven effort as opposed to the voluntary process associated with OSS (Murugesan, 1999), project managers of globally distributed CBD can consider aspects relating to the way a component will be developed that ensure that the challenges reported above relating to coordination breakdowns across remote sites will be minimized.

In this regard, the adoption of CBD by globally distributed software development teams has indeed created expectations that some coordination breakdowns associated with the traditional, non-CB globally distributed software development would be mitigated (Carmel, 1999) through the organization of distributed work. For example, it has been argued that each site could take ownership of a particular component to develop it independently, with a reduced degree of inter-site communication and coordination (Repenning *et al*., 2001). Indeed, this process would facilitate the globalization of the software industry, as components could be developed remotely with minimum inter-site coordination (Repenning *et al*., 2001; Turnlund, 2004).

From a knowledge processes viewpoint, such an approach, in which each site takes responsibility for the development of a particular component, may result in fewer opportunities for the entire project team to share knowledge about existing components available for reuse. Research has indeed shown that the knowledge embedded in a design can be contextual and specific to a particular business context to the extent that the transfer of any solution, be it a component or a program, from one project team to another often requires intensive communications between members of these teams (Oshri and Newell, 2005). This is because, in most cases, a component will need to be modified before being re-applied in another product. Against past expectations that CBD would result in fewer communication and coordination activities, we claim that in pursuing CBD principles, remote teams are more likely to intensify their communication and coordination activities in an attempt to increase the reuse rate of existing components.

Table 1 summarizes the issues revolving around coordination, communication, and knowledge processes in global traditional (non-CB) software development and in global CBD projects.

To illustrate our claim, evidence from two globally distributed CB software development projects is prezsented, after the research approach section. First, evidence relating to the way components were developed within these projects is outlined. Following this, the capabilities developed by these companies are analysed and discussed.

Research approach

In line with past research (Eisenhardt, 1989; Yin, 1994), a case study method was selected for this research. An in-depth case study of three globally distributed CBD projects – two at TATA Consultancy Services (TCS) and one at LeCroy – was carried out. We adopted a qualitative, interpretive approach. The main goal of the empirical

Table 1 Challenges in global software development projects.

	Global non-CB software development	*Global CB software development*	
	Challenges	*Expectations*	*New challenges*
Coordination	Breakdown of traditional coordination mechanisms (Rafii, 1995; Espinosa and Carmel, 2003; Cheng *et al*., 2004). Lack of complete picture about what is going on at remote locations (Carmel, 1999). Difficulties to work in different time-zones (Karolak, 1999; Kobitzsch *et al*., 2001). Delays in distributed collaborative work processes (Jarvenpaa and Leidner, 1999; Herbsleb *et al*., 2000). Different tools, technologies, ICT infrastructure (Sarker and Sahay, 2004).	Each site could take ownership of particular components and work on them independently without much need for inter-site coordination (Carmel, 1999; Colbert *et al*., 2001; Repenning *et al*., 2001).	Even when remote sites had ownership of particular components, extensive inter-site coordination was required (IBM case in Carmel, 1999). Difficulties to manage a complex integration processes across sites if components are fine-grained (Turnlund, 2004). A concern raised in recent studies that there is a need to coordinate onsite-offshore activities related to component development, modification and integration (Alexandersen *et al*., 2003).
Communication	Loss of communication richness (Carmel, 1999). Language barriers (Sarker and Sahay, 2003). Misunderstandings caused by cultural differences (Battin *et al*., 2001; Olson and Olson, 2004). Lack of informal, inter-personal communications (Herbsleb and Mockus, 2003).	Opportunity to reduce communication needs between sites as each site is working on different components (Colbert *et al*., 2001; Repenning *et al*., 2001).	Remote sites know less about specific and contextual knowledge embedded in solutions. Remote sites know less about the area of specialization of their remote counterparts. Remote sites possess limited knowledge of the 'big picture' i.e. new developments in other sites are less visible.
Knowledge processes	Lack of understanding of counterpart's context (Orlikowski, 2002). Asymmetry in distributing information among sites (Carmel, 1999). Lack of trust and motivation to collaborate (Jarvenpaa and Leidner, 1999).	Intensive knowledge processes within remote sites and fewer across remote sites.	Risk of 're-inventing the wheel' if inter-site coordination and communication activities are reduced. Differences in the specialization domains developed in each site (e.g. site A specializes in the banking sector while site B specializes in the insurance sector) that could hinder the development of complementary solutions and the implementation of agility in design that offers future integration of existing solutions. Differences in the level of technical expertise developed within sites that could hamper the quality of knowledge transfer processes from team A to B when a particular component is reused.

research was to explore:

1. Which division of work approach (e.g. one site owns a particular component, or joint development utilizing expertise regardless of its geographical location) was implemented by TCS and LeCroy, and how this division of work affected inter-site coordination efforts between remote sites.
2. What capabilities were critical in supporting component reuse in the globally distributed CBD environment.

Evidence was gathered through a variety of sources, such as interviews, documentation, and archival records (Eisenhardt, 1989; Yin, 1994). Interviews were conducted at two remote sites per company: in Switzerland, India, and USA for TCS, and Switzerland and USA for LeCroy. Interviewees were chosen to include (1) counterparts working closely from remote locations and (2) diverse roles such as managers and developers. In total, 20 individuals were interviewed (14 at TCS and six at LeCroy: see Appendix A for information about the interviewees). Interviews lasted 1 h and 30 min on average; they were recorded and fully transcribed. A semi-structured interview protocol was applied, to allow the researchers to clarify specific issues and follow up with questions. Data were triangulated through interviews with team counterparts in different locations and through the use of project documents.

Data analysis followed several steps. It relied on iterative reading of the data using the open-coding technique (Strauss and Corbin, 1998), and sorting and refining themes emerging from the data with some degree of diversity (Miles and Huberman, 1994).

Following a short introduction to the projects, we report on the approach taken in terms of the division of work, and discuss the capabilities developed to support component reuse in globally distributed CB project teams at TCS and LeCroy.

TCS globally distributed CBD

The projects studied at TCS concerned the development and implementation of Quartz, an integrated financial platform aimed at providing solutions for financial institutions, such as traditional and internet banks, brokerage/securities houses, and asset managers. Quartz consisted of a collection of architectural and business components that can be integrated with third-party components to provide a solution according to the requirements of a specific customer. The following Quartz implementation projects were investigated: (1) Skandia bank in Zurich and (2) Dresdner bank in San Francisco. Both projects were concerned with the implementation of Quartz; therefore, they were analysed together as one 'embedded' case study (Yin, 1994). The two projects are sub-units of analysis. In terms of global distribution, the Skandia development team was distributed across three geographical locations: two *offshore* teams in Gurgaon and Mumbai (India) and an *onsite* team at the customer location in Zurich (Switzerland). Furthermore, vendors of third-party components were located in several countries (more than 25 vendors in total). The Dresdner development team involved an *offshore* team in Gurgaon and an *onsite* team at the customer site in San Francisco.

LeCroy globally distributed CBD

The project studied at LeCroy, called Maui, involved the development of a software platform for new generations of oscilloscopes and oscilloscope-like instruments based on the Windows operating system. The LeCroy global software team involved two teams located in Geneva (Switzerland) and New York (USA), and a main architect telecommuting from Maine (USA). One key challenge for the Maui project was to switch from embedded software to Microsoft COM technology.

CBD: expertise-based or location-based division of work

Despite the expectation that CBD would enable a division of work based on *geographical location,* in which one site takes responsibility for the development of a particular component, thus reducing the need for inter-site coordination required to complete a particular development (Repenning *et al*., 2001), evidence from both TCS and LeCroy suggests that dividing the work based on *technical or functional expertise* enabled these companies to achieve a high rate of component reuse and product variation. In addition, a division of work based on expertise enabled TCS and LeCroy to utilize the knowledge and expertise of their employees regardless of their geographical location, thus allowing these companies to access the pool of expertise available in offshore locations. Lastly, an expertise-based division of work approach required remote engineers and managers to interact, and consult with remote counterparts in order to solve design issues.

In dividing the work between the sites, some ground rules were followed. For example, at TCS the first activities that required direct customer contact and access to the customer's site, such as user requirements and release management were done *onsite*. Self-contained activities, such as coding and unit testing, were, on the other hand, sent *offshore* to take advantage of the cost, quality, and availability of offshore personnel. As one interviewee explained, the onsite team was sending requirements offshore:

> Because the expertise and major source code are here [offshore, in Gurgaon], and mainly because of the expertise; it is quicker and easier to work here (Pankaj, TCS).

The composition of the onsite team included experts who provided expertise in those areas required at the customer location. One manager described the areas of expertise developed onsite:

> Technical people look at technical and architectural issues, functional people look at the functionality and the development of the functionality. Support people handle the configuration management, the groupware, the other various tools which are being used by the team (Sanjay, TCS).

On the other hand, activities that required involvement of the client and close interactions among the TCS developers were conducted in a mixed onsite – offshore manner.

Overall, the majority of the activities required extensive communications, and onsite and offshore teams conducted them jointly through frequent communications using ICTs and visits to remote sites. The only activities that were done independently at the offshore location were coding and unit testing.

At LeCroy, the division of work was also based on the technical domains (areas of expertise) of the engineers. Anthony, LeCroy's Chief Software Architect, describes the division of work in the following way:

> What happened was, part of it was based on who was free at that time, and part of it was where the expertise was.

The software team in Geneva was responsible for most of the original core code in the oscilloscope: these were the developers who had written the original code for oscilloscopes since the mid1980s. Anthony continued:

> So they were also the natural guys to work on the next generation, or defining the next generation.

Maine was another location where Jon, a system architect, was based and collaborated with the team. Jon had worked in New York for many years and had spent a year working in Geneva. In 1999, he moved to Maine (USA) from where he continued to work for LeCroy by telecommuting.

Originally five people, Jon in Maine and four people in Geneva, developed the basics of the Maui. Later on in the development process, additional people in NY and Geneva started to work on the new platform, jointly developing new features. This approach to development required a high utilization of the Integrated Development Environment and the employment of collaborative technologies. As the manager of Geneva team pointed out:

> We use MSN messenger – every member of the software development group appears on the list. So, for having a chat with someone, wherever they may be in the world in the given time, you just need to double-click on their name and start typing a line (Anthony, LeCroy).

To summarize, both companies, TCS and LeCroy, have chosen to utilize the expertise available within their globally distributed CBD projects regardless of their geographical locations. In doing so, the companies recognize the need to develop capabilities that will allow these teams jointly to develop software components based on CBD principals.

Capabilities supporting globally distributed CB software development

The division of work approach pursued by TCS and LeCroy posed these companies new challenges. In particular, as these companies attempted to reuse components across different projects (for TCS) and products (for LeCroy), and improve product flexibility through the application of CBD principals, three areas had become critical in achieving success: (i) inter-site coordination; (ii) knowledge management; and (iii) communication channels.

Inter-site coordination capabilities

Two areas of capabilities were developed by TCS and LeCroy that supported inter-site coordination activities. The first area, which is generic in nature, involved a high-speed, wide-band ICT infrastructure that ensured connectivity between remote sites. The ICT infrastructure supported rapid access to the network, shared resources (e.g. server and project repository, databases), and Web access to common resources. For example, as Corey (VP of IS, LeCroy) outlined:

> The role of the WAN, server and applications pool, how file shares are set up, conferencing tools, and just plain network speed are of very high importance [...] and no firm trying to execute globally distributed CBD successfully can do so without the right infrastructure.

The second area of capabilities was developed mainly in-house to support joint development activities carried out across remote sites, without increasing the inter-site coordination overhead. These included the following capabilities:

- *Automated management of interdependencies between components and related files*. This capability allows for managing dependencies between components. Managing these interdependencies is not a problem as long as the number of components is small, because in such a case, the dependencies can be modelled and understood visually. However, when the number of components becomes high, visual understanding is no longer an option. To manage inter-dependencies between a high number of components, LeCroy developed an in-house tool that supported rapid update of changes, four times a day, by automatically building components that had changed since the last 'build', thus utilizing time-zone differences. One manager from LeCroy explained:

  > We have a server that builds four times a day all the components, and produces 'release binaries'. So I don't have to build every component locally. If someone changes the hardcopy component and they put it back, it will be rebuilt on the server and then in the morning I can import that component and just use it (Larry, LeCroy).

  This capability allowed developers from NY and Geneva to access the latest versions of the development files when they started their working day.

- *Bug tracking capability*. The complexity involved in developing components from several remote locations created new challenges in terms of tracking bugs and managing versions of components and products. This is because, each single bug being reported required its source to be tracked and traced (i.e. whether it had originated from one of the customers or from an internal development team), and required the location of all the components in which this particular code was reused. In CBD, there is a need to trace bugs on three different levels: first, on the system level, where customers report problems with their specific product; second, on the product level, where errors are detected in a specific

product version; and finally, on the component level, where the fault is located. The need to trace bugs is closely related to version and configuration management, because any modification in the component could lead to an upsurge of new versions of different products, which already exist in several versions (see a similar claim by Crnkovic and Larsson, 2002). Therefore, it is imperative to carefully record links between components, products, and systems so that tracing errors on all levels is possible. This requirement may not be unique to CBD; however, what is specific to the CB approach is the mapping between products and components and the management of error reports on product and component level, which is the most difficult part of the management involved in CBD (Crnkovic and Larsson, 2002). Indeed, for bug tracking TCS used a combination of tools: MS-Access for issues registration and resolution, and PVCS Tracker for defect logging, tracking, and analysis. At LeCroy, for instance, the use of automated tools for bug tracking helped to speed up the development by utilizing time-zone differences. For example, as the manager of the Geneva team explained:

> I would not say that time differences are a disadvantage, and close to a release or big milestone they can be a big advantage. Because problems, bugs fixes, can be passed on from time-zone to time-zone (Anthony, LeCroy).

- *Standardization and centralization of the tools and methods* across locations was perceived as imperative by TCS and LeCroy for the effectiveness of component reuse, mainly because software development activities such as building and testing a code could be carried out on the central server. Some of the technologies implemented to facilitate this capability are: Web access, replicated databases, and a single (integrated) development environment. To facilitate the standardization of processes and learning among newcomers, TCS and LeCroy created a guide that explains how to use tools and methods.

Knowledge management capabilities

Ensuring a high degree of component reuse requires knowledge management capabilities. One key challenge that TCS faced, for example, was that several Quartz implementation projects for different clients were running in parallel, and the people involved in different Quartz implementations did not have a direct exposure to the work that other project teams were engaged in. Therefore, several knowledge management mechanisms were introduced by TCS that facilitated the transfer and reuse of components across product teams. For example, TCS created a new role in the form of a *program manager,* who coordinated all Quartz implementations and was aware of new components being developed for a specific customer, and who facilitated the reuse of components across different Quartz implementations at different geographical locations. This way, TCS exploited customer-specific components by adding them to the Quartz package, so that with each new Quartz implementation, TCS increased the variety of components/functionalities that this product offered to potential clients. For example, after implementing Quartz at Royal Skandia UK, an insurance company, where Quartz was implemented as an investment engine, insurance products were added to the next version of Quartz:

> A lot of changes were made to the basic Quartz system just to be able to integrate it with the insurance business. We had to build in a lot of things that deal with policy administration and policy distribution, which are not particularly bank products. This way typical insurance products were added to Quartz: they were released as the next version of Quartz (Sanjay, TCS).

Furthermore, to facilitate the reuse of knowledge and components in a globally distributed environment, engineers were encouraged to rotate between onsite and offshore locations to bridge knowledge gaps between the two sites.

LeCroy, on the other hand, developed a 'component toolbox', a *repository of components* that implemented functionalities common to the oscilloscopes and oscilloscope-like instruments. These components included (i) hundreds of mathematical functions, one component per functionality; (ii) Graphical User Interface components that provide the user interface; (iii) core components that allow the systems to work together and provide the basic instrument capabilities; and (iv) acquisition board driver components responsible for controlling the acquisition hardware of an oscilloscope.

Based on this knowledge management system, a specific oscilloscope could be built by selecting and integrating these components with oscilloscope-specific acquisition and application systems. Furthermore, LeCroy introduced an integrated development environment on the central server, accessible for all members of the dispersed team. Centralization of tools in one location ensures one single environment for all remote locations. For the LeCroy software team, there are no 'local' tools as such: all tools are located at one central place. For example:

> Version Control System (VCS) – Perforce – exists in Geneva and guys access it here [in NY] the same way over WAN, so the only difference there is: from here it takes a little longer to access it, speed is slower. It doesn't matter where you are in the world, you still can access the same single VCS (Anthony, LeCroy).

Indeed, managing knowledge across sites required both companies to invest in unique repositories, but also to identify individuals from the globally distributed teams who could share the knowledge created in one team with the others. By doing so, TCS for example, was able to develop its Quartz platform rapidly and offer solutions in new or related markets. Enabling knowledge flow between remote sites to increase the possibilities of reuse was also largely supported by the communication capabilities developed by TCS and LeCroy.

Communication channels

In light of the division of work applied by both TCS and LeCroy, the development of communication channels and capabilities that ensure the flow of information with

minimum breakdowns and misunderstandings between remote sites was considered critical in their globally distributed CBD environments. To cope with this challenge, both TCS and LeCroy encouraged *frequent communications* between remote members and introduced design rules, aiming to make communications more effective. For example, these companies encouraged systematic and frequent communications in the form of regular teleconferences between software managers in dispersed locations, and transatlantic videoconferences with the entire team every one or two months. Such communications helped to coordinate expertise across locations and better utilize the dispersed knowledge. For example, as the onsite manager of the Dresdner project explained:

> Between us [offshore] and our onsite team we say 'we'll do this portion of the job because we have more competent people here who can look at this part, and you can look at that portion of the job'. It's mutual communication (Sunil, TCS).

This helped to streamline information flows between dispersed teams. In addition, attention was given to *improving the style and content of communications,* which helped to reduce misunderstandings and confusion induced by different cultural backgrounds. This was particularly important for the LeCroy global team, where people from different cultures collaborated over distance. For example, as one interviewee explained:

> I have a lot of experience working with a lot of foreign cultures. In some cultures if you are on the phone explaining something to somebody and they don't understand it – they still say 'I understand'. So the way I try to ensure that the information was received correctly is through a very detailed process of describing the issue. For example I say, 'open this Web link. What do you see?' So it is very specific, very detailed (Adrian, LeCroy).

Both companies encouraged *working flexibility*, in terms of working conditions, for example, working from home, and flexible working hours, which helped increase the overlap of working hours between dispersed locations so that teams could collaborate in real time.

Reuse at Lecroy and TCS: some evidence

Both companies utilized the benefits offered by CBD, such as the ability to reuse components and to build product families rapidly by developing capabilities in the areas of inter-site coordination, knowledge management and communication. LeCroy's Maui CB architecture (platform) served as a basis for a number of future products, helped to reduce considerably time-to-market (an increased number of products offered to a market per year), and made possible the easy integration of LeCroy products with additional functionalities developed by third parties. After the first product (i.e. WaveMaster) based on the Maui architecture was launched in January 2002, a large number of WaveMaster models and Disk Drive Analysers, which are based on the Maui platform, were released, all as a result of a rapid recombination and reuse of existing components. In January 2003, LeCroy launched the WavePro Oscilloscope series (7000, 7100, and 7300), which is also based on the Maui platform.

TCS Quartz CB architecture also supported the reuse of components across different implementations by adding customer-specific components to the Quartz package. Since the first Quartz implementation project started in 1998, Quartz has been implemented in over 40 clients, and the platform has grown to include three different product families: Quartz Securities, Quartz Payments, and Quartz Financial. Furthermore, in the last 3 years, Quartz has been ranked among top 25 best-selling banking systems by the International Banking Systems Journal.

Discussion and implications

Before discussing the cases, it is important to note that our findings are based on two case studies and therefore, by definition, meet only to a limited extent the criteria of transferability (the extent to which the findings can be replicated across cases). Additional research across multiple case studies is needed in order to verify the insights reported in this paper. With this in mind, we can explore the approach to coordinating work and the capabilities developed by the studied companies, recognizing that not all the practices implemented by TCS and LeCroy would be appropriate in the context of other globally distributed CBD projects.

In developing these capabilities, TCS and LeCroy mainly focused on how to improve the rate of reuse when implementing CBD in globally distributed software development projects. To improve the rate of reuse of components, TCS and LeCroy followed an approach in which the development of components is jointly carried out by several remote sites instead of the more commonly found approach in which each site takes responsibility for a particular component. Managers from LeCroy and TCS indicated that while the approach in which dispersed sites jointly develop components requires a higher degree of coordination between dispersed project teams and intensified communication between remote counterparts, it also offers advantages in the form of knowledge integration (Walz *et al*., 1993) and knowledge sharing (Kotlarsky and Oshri, 2005) across the various sites, processes that are imperative for component reuse. Against past expectations that CBD would mainly focus on developing component knowledge within particular teams (Colbert *et al*., 2001; Repenning *et al*., 2001), this study has illustrated how TCS and LeCroy have invested in developing component knowledge within and *across dispersed teams* to ensure that critical knowledge of component is available at more than one site. Through coordination mechanisms, such as the automated management of interdependencies between components and standardized tools, remote counterparts were able to access work done in a different location easily and to continue the development or debugging of a particular component. This across-site participation in development and debugging activities assisted in expanding the knowledge relating to a particular component beyond the boundaries of the site involved. Furthermore, for remote teams who often work on several projects in parallel, this deeper familiarity with a particular component

meant that they could assess the possibilities of reusing this component in other products or markets. For this reason, putting in place, mechanisms that supported the sharing of component knowledge across various sites was critical to promoting component reuse at TCS and LeCroy.

The communication mechanisms implemented by both TCS and LeCroy further supported knowledge sharing and integration needed for component reuse. In line with the approach that component knowledge should be jointly developed across sites, TCS and LeCroy promoted the use of various communication methods (e.g. videoconferencing, chats, short- and long-term relocations). Communication styles were designed to ensure that remote sites would be able to understand each other and share component knowledge, despite language barriers and cultural diversity. As stated above, past studies anticipated that CBD principles would indeed reduce the need to communicate between sites. Our study suggests that in order to achieve a high rate of reuse, remote sites actually need to maintain a high degree of communication and utilize various means of communication in order to ensure that component knowledge is shared and the possibilities of reuse are improved.

Lastly, the implementation of certain knowledge management practices also contributed to the reuse achieved by these companies. In seeking to reuse components and by utilizing expertise residing in a project team regardless of its geographical location, these companies pursued an approach that promoted the capturing and distribution of explicit component knowledge as well as the sharing of tacit knowledge acquired through participation and involvement (Lave and Wenger, 1991). Indeed, TCS and LeCroy invested in mechanisms that captured both the explicit knowledge (e.g. via component repository) and tacit knowledge (e.g. via program managers) of a particular component, acknowledging the complexity involved in transferring knowledge between remote sites. In this regard, explicit knowledge about various components was captured and shared between remote sites through the codification and storage of design documents and components in a central repository. These documents and components could be retrieved by the entire global team in their search for reusable components. Tacit knowledge was acquired by the program manager through his/her involvement in the design of various components, a process that allowed this person to develop understanding and reuse knowledge about markets, existing in-house solutions, and their suitability for emerging business opportunities, something that can hardly be codified and captured.

The capabilities developed by TCS and LeCroy are summarized in Table 2. Furthermore, we outline the benefits expected from a component reuse viewpoint when implementing these capabilities in globally distributed CBD teams. While the evidence presented here is unique in the context of CBD in globally distributed teams, these capabilities are still firm specific and, therefore may vary from one company to another.

What implications does this study have for research and practice?

From a research viewpoint, against the expectations of past studies, this study illustrates how globally distributed CBD project teams may make the most of component reuse opportunities by pursuing an approach in which remote sites jointly develop components. Indeed, in this study, the costs associated with inter-site coordination and communication activities are likely to be higher; however, the possibilities to reuse components can also be improved should companies invest in capabilities that promote knowledge sharing and knowledge integration activities. We further claim that intensive communication activities between globally distributed CBD teams may lead to the development of a 'virtual shared memory' of the entire team, contributing to component reuse processes. In this regard, we align our findings with the emerging body of studies on *transactive memory* in the context of globally distributed teams. Transactive memory as a concept highlights the need to build a shared memory in which person A knows what person B knows. Through such knowledge activities, often promoted through the utilization of various communication means, team and product performance could be improved (Moreland and Myaskovsky, 2000; Akgun *et al*., 2005), including the sharing and reuse of knowledge (Faraj and Sproull, 2000; Majchrzak and Malhotra, 2004).

In addition, in line with past research (Land and Kennedy-McGregor, 1987; Galliers and Swan, 1997), we have learned that globally distributed teams at TCS and LeCroy have often discovered opportunities to reuse existing components through involvement in development and via informal communication channels, and not always through formal knowledge management systems. Furthermore, the joint development of a particular component by several remote sites may impel a higher degree of standardization in terms of development tools, project methodologies, procedures and documents, which is imperative in itself for CB software development. Indeed, some interviewees indicated that the dispersed and yet highly integrated development process of components resulted in a set of standardized tools, procedures and documents. In turn, a well-developed set of standardized tools, templates, and procedures may further improve the possibilities to reuse component knowledge.

Considering the exploratory nature of this study, we submit that our main contribution revolves around a number of pragmatic guidelines to companies that consider the application of CBD principals in their globally distributed environments. A theory related to CBD in globally distributed environments may emerge, should future research confirm the findings reported in this study by conducting cross-industry surveys, and testing the relationship between the capabilities reported above and the success criteria of globally distributed CB software development projects.

While this case highlights the benefits associated with the joint development of a particular component regardless of the geographical location of the expertise, in practice, we suggest that project managers should first assess whether such an approach is in their interest. Indeed, both TCS and LeCroy sought to improve the rate of reuse of their components and therefore, pursued an approach that advanced knowledge sharing and knowledge integration across remote sites. In this regard, project managers should evaluate the objectives of their project to determine

Table 2 Capabilities developed by TCS & LeCroy supporting globally distributed CBD

Capabilities implemented by TCS and LeCroy	*Characteristics*	*Benefits*
Inter-site coordination capabilities		
Automated management of interdependencies between components and related files	Automatically building components that have changed since the last 'build', four times a day, on the central server.	Rapid update of changes. Members of dispersed teams have access to work done at remote site; they are aware of reusable components being developed elsewhere. Time-zone differences are utilized by programming 'builds' of components to run during night time.
Bug tracking across products and projects on system, product and component levels	Setting up bug tracking tool on the central server, enabling Web access.	All components and products that are affected by the bug can be identified.
Standardization of tools and methods across locations	Using similar tools and methods across dispersed locations, creating a Guide that explains how to use methods and tools.	Ensure compatibility and 'integratability' of files, components and applications developed and used at remote locations. Speeds up the learning among newcomers.
Centralization of tools	Centralizing tools used by dispersed team members on a central server, Web-access, Integrated Development Environment.	Members of dispersed teams always work with updated files and have access to updated project and product documents.
Knowledge management capabilities		
Component repository accessible from all dispersed locations	Creating repository of reusable components that implement functions common to majority of products.	Enable knowledge reuse across products. Reduce time-to-market through reuse of existing components. Increase product variety. Avoid 'not invented here' syndrome.
Program manager	Appointing program manager to have overview of all components being developed for different clients.	Enable knowledge reuse across projects. Increase product variety through reuse of customer-specific components in future products.
Communication channels and capabilities		
Frequent communications	Organizing systematic and frequent communications between managers and developers of dispersed teams using on-line chat, email, teleconferencing and videoconferencing.	Streamline information flows and coordinate activities between dispersed teams. Time-zone differences are utilized by handing over bugs and some small tasks between the teams.
Improving the style and content of communications	Adjusting style and content of communication (e.g. wording and selection of media) to personal and cultural characteristics of remote counterparts.	Improve understanding between remote counterparts. Reduce possibility of misunderstandings and conflicts.
Working flexibility	Supporting flexible working conditions, e.g. equipment to enable working from home and flexible working hours.	Enable remote counterparts to collaborate in real-time by increasing overlap in working hours.

whether achieving cost-efficiencies, often gained by outsourcing development activities to low-cost locations such as India, is more important than the investment in a long-term development of shared knowledge and reuse capabilities. A hybrid approach, in which some development activities will be carried out jointly across the remote sites and other activities will be centralized and accomplished in one location, is another possibility. In terms of the capabilities needed to support a joint development effort of a particular component across several sites, based on the evidence from TCS and LeCroy, Table 2 offers project managers some capabilities in the areas of the inter-site coordination, communications, and knowledge management required for the sharing of knowledge and the reuse of components in CBD environments. Last but not the least, we suggest that managers attempt to appoint project members based on their shared histories of collaboration in their area of expertise. In this way, members of a dispersed team will know each other, and their respective and shared communication and coordination routines, from past projects. Through such staffing strategies, the costs associated with inter-site communication and coordination activities can be reduced, and the challenges associated with inter-site coordination and communication activities can be overcome while knowledge sharing and component reuse can be improved.

References

Akgun, A.E., Byrne, J., Keskin, H., Lynn, G.S. and Imamoglu, S.Z. (2005). Knowledge Networks in New Product Development Projects: A transactive memory perspective, *Information & Management* **42**(8): 1105–1120.

Alexandersen, C., Kumar, K. and Van Hillegersberg, J. (2003). *Bank-in-a-Box: Skandia's a Agile and Customizable Financial Services Platform*, Chicago, USA: Society for Information Management (SIM).

Bass, L., Buhman, C., Comella-Dorda, S., Long, F., Robert, J., Seacord, R. and Wallnau, K. (2000). *Volume I: Market Assessment of Component-Based Software Engineering*, Pittsburg, USA: Carnegie Mellon Software Engineering Institute (SEI).

Battin, R.D., Crocker, R. and Kreidler, J. (2001). Leveraging Resources in Global Software Development, *IEEE Software* **18**(2): 70–77.

Carmel, E. (1999). *Global Software Teams: Collaborating Across Borders and Time Zones*, Upper Saddle River, NJ: Prentice-Hall P T R.

Carmel, E. and Agarwal, R. (2002). The Maturation of Offshore Sourcing of Information Technology Work, *MIS Quarterly Executive* **1**(2): 65–77.

Cheng, L., De Souza, C.R.B., Hupfer, S., Patterson, J. and Ross, S. (2004). Building Collaboration into IDEs, *Queue* **1**(9): 40–50.

Clements, P.C. (1995). From Subroutines to Subsystems: Component-based software development, *The American Programmer* **8**(11): 1–8.

Colbert, R.O., Compton, D.S., Hackbarth, R.L., Herbsleb, J.D., Hoadley, L.A. and Wills, G.J. (2001). Advanced Services: Changing how we communicate, *Bell Labs Technical Journal* **6**(1): 211–228.

Cramton, C.D. (2001). The Mutual Knowledge Problem and Its Concequences for Dispersed Collaboration, *Organization Science* **12**(3): 346–371.

Crnkovic, I. and Larsson, M. (2002). Challenges of Component-Based Development, *The Journal of Systems and Software* **61**(3): 201–212.

Crowston, K. and Scozzi, B. (2002). Open Source Software Projects as Virtual Organizations: Competency rallying for software development, *IEE Proceedings Software* **149**(1): 3–17.

Desouza, K.C. and Evaristo, J.R. (2004). Managing Knowledge in Distributed Projects, *Communications of the ACM* **47**(4): 87–91.

Ebert, C. and De Neve, P. (2001). Surviving Global Software Development, *IEEE Software* **18**(2): 62–69.

Eisenhardt, K.M. (1989). Building Theories from Case Study Research, *Academy of Management Review* **14**(4): 532–550.

Espinosa, A. and Carmel, E. (2003). Modeling Coordination Costs Due to Time Separation in Global Software Teams, in Workshop on Global Software Development, part of the International Conference on Software Engineering (ICSE) (Portland, Oregon, USA, 2003).

Faraj, S. and Sproull, L. (2000). Coordinating Expertise in Software Development Teams, *Management Science* **46**(12): 1554–1568.

Galliers, R.D. and Swan, J.A. (1997). Against Structured Approaches: Information Requirements Analysis as a Socially Mediated Process, in The Thirtieth Annual Hawwaii International Conference on System Sciences (Hawwaii, 1997).

Herbsleb, J.D. and Mockus, A. (2003). An Empirical Study of Speed and Communication in Globally-Distributed Software Development, *IEEE Transactions on Software Engineering* **29**(6): 1–14.

Herbsleb, J.D., Mockus, A., Finholt, T.A. and Grinter, R.E. (2000). Distance, Dependencies, and Delay in Global Collaboration, in Conference on Computer Supported Cooperative Work (Philadelphia, Pennsylvania, USA, 2000).

Herbsleb, J.D. and Moitra, D. (2001). Global Software Development, *IEEE Software* **18**(2): 16–20.

Huang, J.C., Newell, S., Galliers, R.D. and Pan, S.-L. (2003). Dangerous Liaisons? Component-Based Development and Organizational Subcultures, *IEEE Transactions on Engineering Management* **50**(1): 89–99.

Jarvenpaa, S.L., Knoll, K. and Leidner, D.E. (1998). Is anybody out there? Antecedents of trust in global virtual teams, *Journal of MIS* **14**(4): 29–64.

Jarvenpaa, S.L. and Leidner, D.E. (1999). Communication and Trust in Global Virtual Teams, *Organization Science* **10**(5): 791–815.

Karolak, D.W. (1999). Global Software Development: Managing Virtual Teams and Environments. IEEE Computer Society: Los Alamitops, USA.

Kim, S.D. (2002). Lessons Learned from a Nationwide CBD PROMOTION PROJECT, *Communications of the ACM* **45**(10): 83–87.

Kobitzsch, W., Rombach, D. and Feldmann, R.L. (2001). Outsourcing in India, *IEEE Software* **18**(2): 78–86.

Kotlarsky, J. and Oshri, I. (2005). Social Ties, Knowledge Sharing and Successful Collaboration in Globally Distributed System Development Projects, *European Journal of Information Systems* **14**(1): 37–48.

Kunda, D. and Brooks, L. (2000). Assessing Organisational Obstacles to Component-Based Development: A case study approach, *Information and Software Technology* **42**: 715–725.

Land, F.F. and Kennedy-McGregor, M. (1987). Information and Information Systems: Concepts and perspectives, in R.D. Galliers (ed.) *Information Analysis: Selected Readings*, Sydney: Addison-Wesley Publishers Ltd.

Lave, J. and Wenger, E. (1991). *Situated Learning Legitimate Peripheral Participation*, Cambridge: Cambridge University Press.

Majchrzak, A. and Malhotra, A. (2004). Virtual Workspace Technology Use and Knowledge-Sharing Effectiveness in Distributed Teams: The influence of a team's transactive memory, Knowledge Management Knowledge Base. URL:http://knowledgemanagement.ittoolbox.com/documents/document.asp?i = 3164. Date accessed: September 23, 2004.

Malhotra, A., Majchrzak, A., Carman, R. and Lott, V. (2001). Radical Innovation Without Collocation: A case study at Boeing-Rocketdyne, *MIS Quarterly* **25**(2): 229–249.

Miles, M.B. and Huberman, A.M. (1994). *Qualitative Data Analysis: An Expanded Sourcebook*, California: Sage.

Mockus, A. and Weiss, D.M. (2001). Globalization by Chunking: A quantitative approach, *IEEE Software* **18**(2): 30–37.

Moreland, R.L. and Myaskovsky, L. (2000). Exploring the Performance Benefits of Groups Training: Transactive memory or improved communication? *Organizational Behavior and Human Decision Processes* **82**(1): 117–133.

Murugesan, S. (1999). Leverage Global Software Development and Distribution Using the Internet and Web, *Cutter IT Journal* **12**(3): 57–63.

Olin, J.G., Greis, N.P. and Kasarda, J.D. (1999). Knowledge Management Across Multi-tier Enterprise: The promise of intelligent software in the auto industry, *European Management Journal* **17**(4): 335–347.

Olson, J.S. and Olson, G.M. (2004). Culture Surprises in Remote Software Development Teams, *Queue* **1**(9): 52–59.

Orlikowski, W.J. (2002). Knowing in Practice: Enacting a collective capability in distributed organizing, *Organization Science* **13**(3): 249–273.

Oshri, I. and Newell, S. (2005). Component Sharing in Complex Products and Systems: Challenges, solutions and practical implications, *IEEE Transactions on Engineering Management* **52**(4): 509–521.

Peters, J.F. and Pedrycz, W. (2000). *Software Engineering: An Engineering Approach*, New York: John Wiley & Sons, Inc.
Rafii, F. (1995). How Important Is Physical Collocation to Product Development Success? *Business Horizons* **(January–February)**: 78–84.
Ravichandran, T. and Rothenberger, M.A. (2003). Software Reuse Strategies and Component Markets, *Communications of the ACM* **46**(8): 109–114.
Repenning, A., Ioannidou, A., Payton, M., Ye, W. and Roschelle, J. (2001). Using Components for Rapid Distributed Software Development, *IEEE Software* **18**(2): 38–45.
Sarker, S. and Sahay, S. (2003). Understanding Virtual Team Development: An interpretive study, *Journal of the Association for Information Systems* **4**: 1–38.
Sarker, S. and Sahay, S. (2004). Implications of Space and Time for Distributed Work: An interpretive study of US–Norwegian system development teams, *European Journal of Information Systems* **13**(1): 3–20.
Starr, M.K. (1965). Modular Production: A new concept, *Harvard Business Review* **43**(November–December): 131–142.
Strauss, A.L. and Corbin, J.M (1998). *Basics of Qualitative Research*, Thousand Oaks, CA: Sage Publications.
Szyperski, C. (1998). *Component Software Beyond Object-Oriented Programming*, New York: Addison-Wesley.
Traas, V. and van Hillegersberg, J. (2000). The Software Component Market on the Internet: Current status and conditions for growth, *ACM SIGSOFT* **25**(1): 114–117.
Turnlund, M. (2004). Distributed Development Lessons Learned, *Queue* **1**(9): 26–31.
Ulrich, K.T. and Eppinger, S.D. (2000). *Product Design and Development*, New York, USA: McGraw-Hill.
Vitharana, P. (2003). Risks and Challenges of Component-Based Software Development, *Communications of the ACM* **46**(8): 67–72.
Walz, D.B., Elam, J.J. and Curtis, B. (1993). Inside a Software Design Team: Knowledge acquisition, sharing, and integration, *Communications of the ACM* **36**(10): 62–77.
Yenne, B. (2002). *Inside Boeing 777: Building The 777*, St. Paul: MBI Publishing Company.
Yin, R.K. (1994). *Case Study Research: Design and Methods*, Newbury Park, CA: Sage.

About the authors

Dr. Julia Kotlarsky is an Assistant Professor of Information Systems, Warwick Business School, UK. She holds a Ph.D. degree in Management and IS from Rotterdam School of Management Erasmus (The Netherlands). Her main research interests revolve around social and technical aspects involved in the management of globally distributed IS teams, and IT outsourcing. Julia published her work in journals such as *Communications of the ACM, European Journal of Information Systems, Information Systems Journal, IEEE Security & Privacy*, and others.

Dr. Ilan Oshri is an Assistant Professor of Strategic Management, Rotterdam School of Management Erasmus, The Netherlands. Ilan holds a Ph.D. degree in technological innovation from Warwick Business School (UK). His main research interest lies in the area of knowledge management and innovation. Ilan has published widely his work in journals and books which include *IEEE Transactions on Engineering Management, Communications of the ACM, European Journal of Information Systems, Management Learning, Information Systems Journal*, and others.

Dr. Jos van Hillegersberg is a Professor of Design and Implementation of Information Systems and head of the Department of Information Systems and Change Management at the University of Twente (The Netherlands). He has published in *Journal of Product Innovation Management, Communications of the ACM, Information Systems*, *Journal of Information Technology*, and others.

Dr. Kuldeep Kumar is a Visiting Professor of IS at The City University of Hong Kong (China). He is also the Professor of IS at Florida International University, Miami (USA), and Professor of IS Research at RSM Erasmus (The Netherlands). His research has been published in books and journals, such as *MIS Quarterly*, *Communications of the ACM*, *IEEE Transactions on Software Engineering*, *Information Systems*, *Informatie*, and others.

Appendix A

Interviewees' information

TCS – Interviewees' details
Interviews were carried out in March 2002 in Gurgaon and Zurich sites. Roles are correct for 2002.

Name	*Role*	*Location*
Sanjay B[a]	Executive manager for Quartz	Gurgaon
Sanjay S[a]	Delivery manager for Quartz	Gurgaon
Sunil[a]	Offshore project leader for Dresdner	Gurgaon
Sandeep	Project manager and onsite project leader of Dresdner	San Francisco
Bala	Software engineer (team of Sunil)	Gurgaon
Pankaj[a]	Offshore project leader for Skandia	Gurgaon
Sourin	Software engineer (team of Pankaj)	Gurgaon
Nitin[a]	Technical consultant (former technical architect for Skandia)	Gurgaon
N.G.S	Vice President	Gurgaon

Appendix Continued

Name	*Role*	*Location*
Ashvini	Technical architect and team leader for Skandia	Zurich
Tuhin	Software engineer (team of Ashvini)	Zurich
Krishna	Manager of front-end team (head of team in Mumbai)	Zurich
Rik	Executive for TCS-Skandia Relationships (IS)	Zurich
Rajan	Project manager and onsite project leader for Skandia	Zurich

[a]These individuals were interviewed twice.

LeCroy - interviewees' details

Interviews were carried out between November 2001 — January 2003 in Geneva and NY sites. Roles are correct for 2002.

Name	*Role*	*Location*
Larry[a]	Director of software engineering, responsible for the NY team	NY
Anthony[a]	Chief software architect, responsible for the Geneva team	Geneva
Gilles	Software engineer	Geneva
Adrian	Web-master	NY
Corey	Vice President Information Systems	NY
Dave	VP, Chief Technology Officer	NY

[a]These individuals were interviewed twice.