A Research Agenda for Distributed Software Development

Bikram Sengupta
IBM Research
Block-1, Indian Institute of Technology,
Hauz Khas, New Delhi 110016, India
bsengupt@in.ibm.com

Satish Chandra
IBM Research
Hawthorne, NY, USA
satishchandra@us.ibm.com

Vibha Sinha
IBM Research
Block-1, Indian Institute of Technology
Hauz Khas, New Delhi 110016, India
vibsinha@in.ibm.com

ABSTRACT

In recent years, a number of business reasons have caused software development to become increasingly distributed. Remote development of software offers several advantages, but it is also fraught with challenges. In this paper, we report on our study of distributed software development that helped shape a research agenda for this field. Our study has identified four areas where important research questions need to be addressed to make distributed development more effective. These areas are: collaborative software tools, knowledge acquisition and management, testing in a distributed set-up and process and metrics issues. We present a brief summary of related research in each of these areas, and also outline open research issues.

Categories and Subject Descriptors

D.2.0 [Software Engineering – General]

General Terms

Management, Measurement, Performance

Keywords

Distributed software development, research agenda, collaboration, knowledge management, testing, process, metrics

1. Introduction

During the last two decades, the management, development and maintenance of software have evolved from being concentrated at a single site to being geographically distributed across the globe. This phenomenon is variously referred to as "global", "distributed" or "multi-site" software development. A number of business reasons have contributed to this trend. To start with, the global demand for software products and services beginning in the late 1980s led to a flood of mergers and acquisitions, as IT firms strived to penetrate new markets and complement their product lines. At the same time, companies increasingly chose to focus on core competencies and hand-off or "outsource" some of the other necessary activities to firms specializing in those areas. "Offshoring" brought in further benefits - availability of a large pool of skilled labor, the prospect of being able to do round-the-clock development, and most importantly, huge savings that could be accrued through low labor cost in developing countries. For example, a study by McKinsey [1] reports that the software development costs in India are 4 times less than that in US and for the period between 2003 to 2008 US savings from offshoring would grow from $6.7 billion to $20.0 billion. Of course, the process has also been aided by significant technological advances; in particular, the explosive growth of the Internet, which often makes distances irrelevant, and has made remote collaboration increasingly practical. Little wonder then, that a study in 2000 [2] revealed that 70% of US firms have outsourced some kind of business process, and 203 of US Fortune 500 companies engage in offshore outsourcing; or, that according to a Gartner Inc. estimate [3] in 2004, up to 10% of the workforce in US tech companies would be located in emerging markets by the end of the year.

1.1 Challenges in Distributed Development

The perceived benefits notwithstanding, distributed development of software is fraught with challenges. Previous literature (e.g. [4, 5]) has identified a number of these difficulties, which we briefly discuss here. Many of the challenges that arise in practice can be traced back to *inadequate communication* (particularly informal communication) between team members separated by distance and time-zone differences. In collocated projects, such communication helps easy dissemination of project knowledge, familiarizes individuals with the working styles of others, and fosters greater understanding between team members. There is very convincing evidence, however, that the frequency of communication generally drops off sharply with physical separation among coworkers' offices [18, 35] and in a multi-site environment such communication can be virtually non-existent. Time-zone differences further worsen the situation, in many cases significantly reducing the time-window for effective synchronous communication between remote teams. Consequently, in distributed projects, information flows are often irregular, resulting in frequent misalignment and re-work [4]. Apart from geographic separation, *cultural differences* [20] across sites also impede easy communication. The primary spoken language may vary from one site to another, or a common spoken language may have subtle differences in meaning. Moreover, two sites may also follow different corporate cultures [6] e.g. some companies have well-defined hierarchies and associated protocols, while others

ICSE'06, May 20–28, 2006, Shanghai, China.

have a relatively flat organizational structure. Studies indicate that distributed teams that are culturally divided may not be as cohesive as local ones, and this may lead to less trust [24], poor cooperation and ultimately, conflicts. These difficulties are particularly telling for those software development activities that are communication-intensive. For example, [5] reports on the negative impacts of remote communication, cultural diversity and time differences on the requirements analysis phase of the software lifecycle.

Inadequate informal communication, lack of trust and cultural differences are not the only challenges in distributed software development. There are *strategic issues*, pertaining to the distribution of work across sites; *process differences* that can lead to problems in synchronization and system integration; *knowledge management* challenges that impede timely sharing of knowledge and reduce opportunities for re-use; and finally, *technical issue*s e.g. poor bandwidth, connectivity problems etc. that can have a severe impact on the productivity of remote teams.

These challenges in distributed development often carry a heavy penalty in terms of slowing down multi-site work e.g. [18] reports that distributed work items take about two and one-half times as long to complete as similar items where all the work is collocated. Moreover, while the production costs might be low in distributed software development, there are increased coordination costs involved. The cost-benefit trade-offs in distributed development have been a topic of interest to both researchers and practitioners and a number of studies have been published on the same - studying coordination in distributed software teams [8, 9, 10] and geographically dispersed teams in general [11, 12, 13, 14].

1.2 Outline of this paper

As the above discussion - and also other references throughout this paper - will indicate, there have been a number of academic efforts relevant to different aspects of distributed software development. These include studies of outsourcing trends in specific geographies [2, 15] and economic implications of this trend [16], to experience reports on distributed development projects [17, 19], and summary of challenges observed [4], to tools and methodologies that address specific remote development challenges (e.g. [21, 28]).

Like many efforts in this area, our starting point was also a field study that enabled us to gain rich feedback from distributed development practitioners regarding the challenges they face, and some of the best practices they have formulated to deal with these challenges. Unlike previous efforts, however, we went a step further to identify the core research issues behind the challenges; thus instead of merely confirming previous reports about the difficulties of cross-site communication, we wanted to consider its implications on the next-generation software engineering tools. As another example, while we wanted to report on the knowledge management challenges we observed in outsourcing projects, we also sought to explore research opportunities in combining the information flow capabilities of structured and unstructured project artifacts to aid knowledge migration.

The main contribution of this paper is thus not the findings of our field study per se (which resonate with many of the previous studies in this area), but rather a research agenda for distributed software development that was synthesized from it. As part of this agenda, we will summarize research efforts in each of the areas identified but more importantly, we will also highlight some of the open questions that warrant further investigation. To the best of our knowledge, there has been no comparable effort towards composing a broad research agenda for distributed development based on field observations. We believe that such an agenda will help focus the efforts of software engineering research community to ensure that distributed development of software remains a viable option in the long run.

The paper is structured as follows: in the rest of Section 1, we report on our study of distributed software development in IBM, introduce the research topics that emerged from the study, and set its scope. Sections 2-5 then consider these topics in more detail. Specifically, in Section 2, we explore research in the area of collaboration and sharing tools for software development. Section 3 addresses issues related to application knowledge migration and management. Research directions that can facilitate testing in a distributed environment are addressed in Section 4. Section 5 investigates process and metrics issues in distributed development. In Section 6 we summarize the main research items, and conclude the paper.

1.3 Initial Study

At IBM Research, for more than a year we have been investigating the challenges in distributed software development through extensive interactions with global development practitioners. Our proximity to development teams in India, who are heavily involved in offshore development, placed us in a favorable position to conduct such a study. A very typical setup in a distributed project that we came across is shown in Fig. 1. This involves a customer-facing team (comprising managers, business analysts and senior architects) located somewhere in the US or Europe (shown in the top left of the figure), and multiple development teams (comprising system engineers, designers, programmers and testers) in remote locations like India, China, Brazil etc. (depicted by the remaining teams in Fig. 1). These teams can belong to the same or different organizations. In projects involving maintenance of legacy systems (a major share of outsourcing projects), usually a team from the remote development center visits the customer premises at the start of the engagement, to absorb knowledge about the application. At the end of knowledge acquisition, these "Onsite Trainees" carry back application knowledge to the remote centers, as shown in Fig.1. Project artifacts (e.g. code, documentation, test data etc.) are shared with the remote teams. Subsequently, it is the responsibility of the customer-facing team to closely interact with the customer and elicit high-level business requirements. The analysts then work with the system engineers in remote locations to create concrete system requirements that would meet the business needs. In addition, remote centers need to collaborate to draw up "Interface Agreements" that specify how their modules will interact. Development then proceeds across the different teams, and periodically, work products from different sites are integrated and validated.

During the study, we spoke to around 30 practitioners in different roles across 14 projects. Both onsite (US and Netherlands) and remote team members (based in India) were approached. The discussions with the India team members occurred through face-

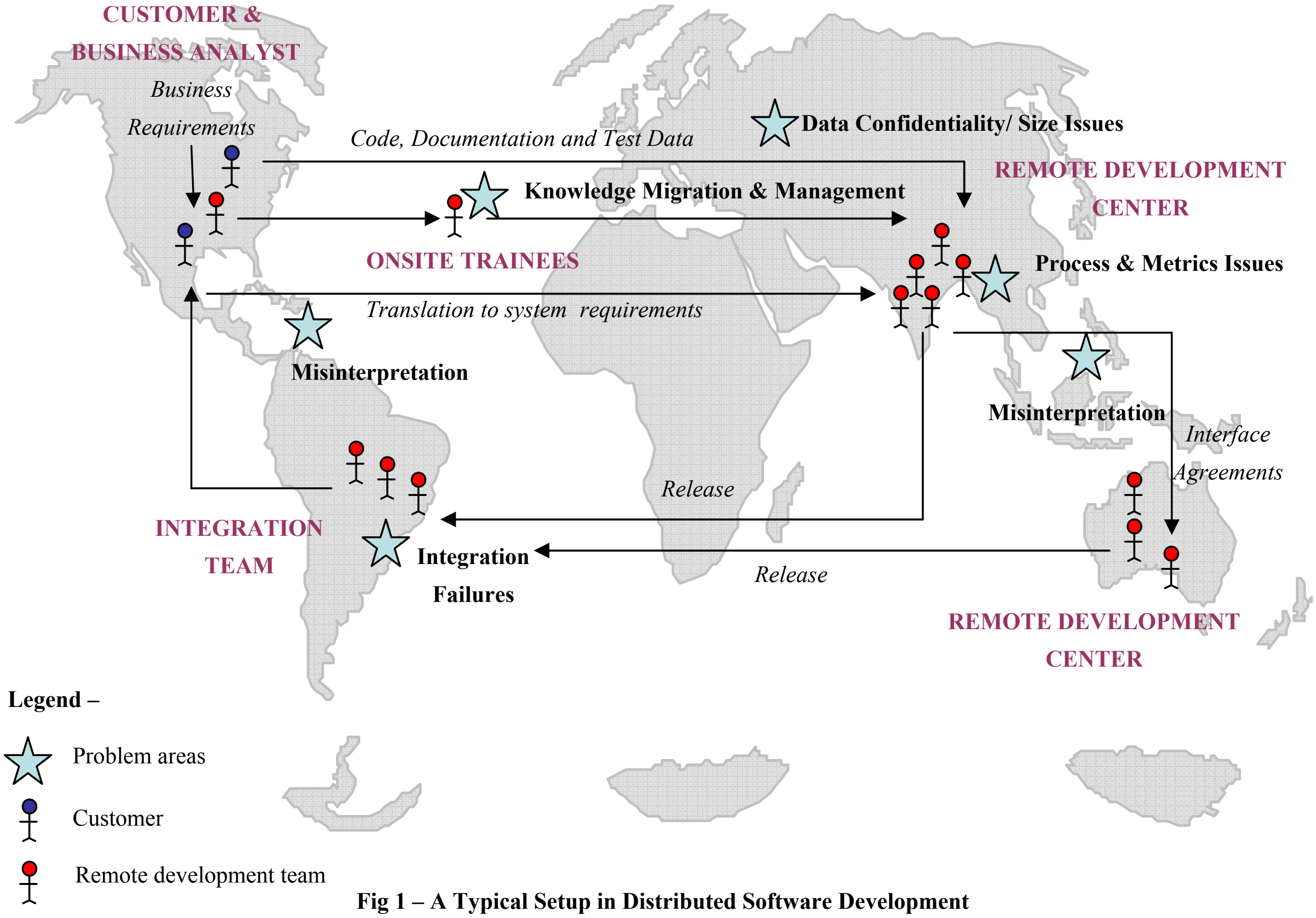

Fig 1 – A Typical Setup in Distributed Software Development

to-face meetings and phone calls. For the onsite teams, teleconferences and follow-up e-mails were used. The primary data collection technique was semi-structured interviews, guided by questions like: "What are the different challenges you face in your work due to the geographic distribution of your project?", "Which of these challenges have the greatest impact on your work?" and "What tools and methodologies do you use to address these challenges?"

Best Practices: We found that the teams have developed several "best practices" to help address some of the well-known challenges of distributed development. Since people aspects play a very important role in distributed projects, managers usually adopt early team-building measures (e.g. face-to-face meetings) to create strong rapport between team members across sites. In fact, successful multi-site projects had as much as 8 weeks of face-to-face contact for training of key team members. In some cases, when team members return to their own sites, they act as contact people or liaisons, a practice that has also been reported elsewhere [22]. While the liaisons may also help bridge cultural differences, as a general rule, new team members are given some kind of informal cultural training according to the requirements of the project. Teams stressed on the need for consistent processes to be deployed and examples of practices they found useful included creating and maintaining a glossary of common terms, performing code releases early and often, and using short cycle times to ensure focus. We found that some teams have adopted interesting optimization techniques e.g. use of an asset management tool to help maximum re-use of assets (e.g. software licenses) through inter-site sharing, particularly when business hours at two sites are non-overlapping. Some coarse indicators of project status and teamwork are also collected, e.g. using frequency of contributions and interactions as a measure of overall project progress and team morale, using interviews and surveys that request numeric answers to questions on personal work, teamwork etc. Finally, there is extensive use of tools to aid in communication (e-mails, chat, phone calls etc.), document sharing (e.g. through Lotus Notes teamrooms), configuration management (e.g. Rational ClearCase [23]) and defect management (e.g. Rational ClearQuest, [23]).

Challenges: The best practices notwithstanding, practitioners also reported on a wide variety of challenges. Some of the main challenges are indicated in Fig.1 using the legend "Problem areas". A basic difficulty is the inability to communicate effectively across distances, cultures and time-zones, as has also been reported elsewhere (Section 1.1). We discovered that these problems were particularly acute in distributed requirements management, since it is one of the most collaboration-intensive activities in software development. Several study participants reported difficulties in gaining shared understanding of requirements, and in propagating and managing requirement changes. Similar issues arose with interface agreements. A related concern was integration testing; since requirements and interface agreements are frequently misinterpreted, developers at one site often make incorrect assumptions about sub-systems being developed at other sites. These discrepancies remain hidden during unit testing and surface only during integration when they are very expensive to fix. Another major challenge that we came across is the acquisition of application knowledge from the customer/parent site and its subsequent management at a remote site through the life of a project. Again, software process and metrics issues, well understood for collocated development, did not seem to scale to distributed projects. In addition, there were networking and other infrastructure issues which lead to challenges in data replication and remote builds, as well as data privacy concerns that discourage customers from sharing much-needed production data to development teams in other organizations.

Note that some of these challenges e.g. inaccurate requirements capture, integration errors etc. arise even in collocated software development. However, the rich anecdotal evidence that we gathered in course of our study suggest that distributed development, by adding factors like geographic dispersion, time-zone issues and cultural/organizational differences, has accentuated some of the existing difficulties, and also added new ones to the development process.

Based on the study, we have synthesized a research agenda for distributed software development. In particular, we propose the following areas as part of a broad research program:

- Collaborative software development tools, to strengthen collaboration in distributed development
- Application knowledge management, to ease the task of migrating application knowledge from the parent site to remote development centers, and preserve and manage the same
- Testing in a distributed set-up, in particular a) how to do effective unit-testing at a remote site where test data cannot be directly accessed in entirety due to data privacy and replication issues and b) how to smoothen the integration of modules developed at separate locations and reduce integration errors
- Process and metrics issues, that determine what development methodologies to follow, what data and metrics to collect when development is distributed

We would like to emphasize that these areas do not constitute an exhaustive list of possible research directions in distributed development. For example, a multi-site project presents rich opportunities for behavioral studies of software practitioners across different sites, but this has not been included in the above list. Again, there has been interesting research on how to build trust between remote teams [24], which is also beyond the scope of this paper. Our discussion focuses primarily on technical challenges we identified where software engineering research can add value and while it covers a broad spectrum of issues in multi-site development, there are indeed research opportunities beyond its scope.

We will now elaborate on the above items, discuss ongoing efforts to meet these challenges, and explore opportunities for further investigation.

2. Collaborative Software Tools

Software development is an inherently collaborative activity. To start with, business analysts have to interact with customers and elicit high-level needs, and then work with system engineers and architects who refine them into concrete technical requirements and work out the system design. These form the basis for development, which again, cannot be accomplished in isolation: in any significant software development effort, programmers work together on the same piece of code, use each other's code, and rope in testers to validate their code. Thus a high-bandwidth mode of collaboration has to be established within the team to ensure that there is shared understanding of the development process and the delivered system meets the customer needs.

Unfortunately, in distributed development, remoteness and time-zone differences put severe strain on cross-site collaboration. For example, during our study, we discovered that remote practitioners are unable to hold effective discussions on requirements. Since existing requirements management tools do not provide rich support for collaboration, teams typically use these tools only as a shared requirements repository, and hold all discussions *outside* of the tool in e-mails, chats or phone calls. This involves a significant amount of *context-switch* (as users have to continually move back and forth between the requirements and communication environments) and moreover, it becomes difficult to track and preserve discussions on requirements that are spread across several media. Again, when requirements change, the information is often not propagated to remote teams in a timely/effective manner, and gaps in understanding creep in over time. Note that other software development activities like design, coding, project management etc. are also impacted when the stakeholders are distributed, as software engineering tools traditionally used for these provide little or no support for the collaboration needs of these activities.

Distributed development thus presents a compelling case to make software development tools and environments more collaborative. The broad aim of this line of research would be to explore ways to weave-in collaboration features in support of common software tasks into the software development tools themselves. In a seminal work [25] Booch and Brown present a vision for a "Collaborative Development Environment" (CDE) tailored to the needs of software practitioners, where a CDE is defined as "a virtual space wherein all the stakeholders of the project – even if distributed by time or distance – may negotiate, brainstorm, discuss, share knowledge, and generally labor together to carry out some task, most often to create an executable deliverable and its supporting artifacts." A number of research projects as well as open source efforts and commercial products are now bringing elements of collaboration into software development activities. Commercially, *collab.net* [26] is one provider of such CDEs; its public face is *SourceForge* [26], an open-source CDE, which offers facilities for configuration management, bug tracking, task management and discussions. *Stellation* [27] is an open source effort that introduces "activity"-oriented fine-grained source control, to simplify collaboration and provide awareness of changes. *Coven* [28] uses a soft-locking mechanism to warn the new committing user of a potential conflict. *Sangam* [29] features a shared editor and chat for pair programming. *Jazz* [30] supports rich synchronous communication, and promotes mutual awareness of coding activities within a development team.

As distributed development becomes increasingly popular, research in collaborative environments will continue to gain momentum. However, there is a need to expand the scope of this line of research. Most of the work in the CDE domain till date has focused on collaborative coding (configuration management, conflict detection etc.) and relatively little attention has been paid to other software development activities like requirements management, project management, design etc., even though many of these activities are highly collaborative in nature (and are thus disrupted by geographic separation). There have been a few exceptions of course, (e.g. [31, 32, 33]) but by and large, the success of coding-specific CDEs is yet to be realized in practice in other domains. To bring about a paradigm shift in distributed development, the situation needs to change. In other words, the scope of CDEs has to be broadened to include all common software development activities within its fold.

There are several important challenges that have to be addressed by researchers for this to succeed. First, a deep understanding of the information flows and models of collaboration in different distributed software development activities have to be developed. Rigorous empirical studies need to be conducted across various kinds of engagements to build and refine such models. Next, a judicious mix of collaboration services have to be deployed in a CDE to facilitate these information flows. In particular, support for highly synchronous activities like software design has to be significantly enhanced. For example, shared virtual whiteboards currently available for synchronous design work do not provide adequate support for concurrency and conflict resolution; highlighting and managing changes also become difficult when non-textual artifacts (e.g. UML models) are involved. A shared whiteboard for collaborative elaboration of UML models has to address these challenges to make virtual collaboration around design artifacts truly effective.

Distributed software development thus presents a fertile ground for researchers in Computer Supported Cooperative Work (CSCW) and Human Computer Interaction (HCI) to help transition traditional CASE tools for requirements, design, project management etc. to the next level, which focuses on the collaborative needs of the extended team and imposes no restriction on its geographic proximity.

3. Application Knowledge Migration and Management

Several distributed development projects we surveyed involved maintenance and enhancement of legacy systems. We discovered that this outsourcing of application maintenance to IT service companies presents a significant information flow challenge. A team from the offshore vendor generally visits the client premises for a limited period of time (during our study, we found this to be typically in the range of 3-4 months) to acquire application knowledge, and carry it back to the remote development center in preparation of subsequent maintenance requests. The speed and accuracy of this knowledge transfer and its subsequent management is a major differentiating factor in the cost and performance of outsourcing. Some of the managers we interviewed reported that the key difficulty in remote maintenance of software lay in the following questions: what is the optimal way for a service team to absorb knowledge of a legacy application from a client team during a brief onsite visit, and how does it retain this knowledge in a person-independent manner? The latter question assumes significance because we found outsourcing teams to grow and shrink quite frequently depending on the funding available from the customer; when a team member has to leave a project, important domain and application knowledge the member acquired often leaves with him or her. When new members join the team, they often need to spend significant time acquiring this knowledge afresh, before they can start to become productive. This churn cuts into the cost advantages of remote outsourcing.

In general, there are two sources of application knowledge: first, the software development artifacts such as requirements, architecture diagrams, interface specifications, code, test cases etc. – the "formal" artifacts (here, "formal" does not imply mathematically precise); and second, "informal" sources which include human agents like original developers and users of the system, and crucially, informal artifacts that include (but are not limited to) ad-hoc documents about the application, notes gathered during the on-site application hand-off meetings, and the information persisted from collaboration tools.

A significant amount of knowledge may be extracted from the formal artifacts through a combination of code and specification inspection, and execution of the application followed by analysis of trace files and error logs. While there is a large body of tools that can aid in this process (e.g. debugging tools, tools for tracing and visualizing the execution of different scenarios, reverse engineering tools such as Doxygen [37] that can parse code to generate interaction and class diagrams), there is still a gap when it comes to integrating the tools that a team member (e.g. a visiting offshore system engineer) may use during knowledge acquisition, with appropriate metrics that can quantify the knowledge captured, and provide feedback on missing links in the acquisition process. A highly desirable end goal would be to bring existing legacy applications into the domain of model-driven development, by linking code to existing or re-constructed UML-style models. This is an exciting area of research, with potentially far-reaching benefits.

On the other hand, informal information sources are often unique sources of generally untapped information; examples include experience reports on system usage, e-mails explaining a particular design rationale, meeting notes that document the resolution of an important issue etc. However, neither formal nor informal sources alone suffice for all the knowledge acquisition needs. It is difficult to rely on the formal artifacts alone for several reasons: one may not know which artifacts contains the information one needs, there may be ambiguities in an artifact, it may be very time-consuming to get the right information out of an artifact or the artifact itself may be obsolete with respect to the intent. There is always some "uncommitted" information in a project that is just not recoverable by examining the formal artifacts, however complete. Over-reliance on informal sources similarly, is unrealistic: human information sources may simply not be there (e.g. the system developers have moved to other projects/organizations), and although ad-hoc documents may provide useful insight into the project, they are unlikely to be detailed enough to explain all system technicalities.

We believe that this challenge presents an interesting possibility of combining the information flow capabilities of formal and informal artifacts. Techniques from unstructured information management area may be used to analyze informal artifacts and organize the recovered knowledge for easy retrieval. A research problem here is how to automatically create useful linkages between formal artifacts and the knowledge recovered from informal sources. Also, since the corpus of knowledge continues to grow over time, the recovered knowledge has to be managed on an ongoing basis.

Apart from knowledge acquisition, *knowledge sharing* between remote team members also gains significance in distributed development, particularly when a new site joins the development effort. Very often, human sources of project-specific information are available, but they may not be known to colleagues at remote sites. This leads to substantial delays in the resolution of even minor issues. Research can investigate ways in which the expertise of different individuals and teams may be "learnt" as development proceeds, depending on how they contribute to the application. This kind of knowledge may be acquired (e.g. [21]) using change management systems, concurrent versioning systems, modification request logs etc. that typically document the persons involved in raising requests, making code commits and so on. There are also tools that help users identify artifacts pertinent to a given task e.g. Hipikat [38] recommends relevant software development artifacts (by searching code repositories, newsgroups, bug-reports etc.) based on the context in which a developer requests help. In addition, learning databases could be maintained permitting team members to publish solutions to problems encountered ("debugging diaries"), tutorials on installing software ("cheat sheets") etc. A research opportunity here would be to see how all such knowledge sources may be integrated to support some kind of "virtual assistant" that may be embedded within the collaborative tools used by distributed teams. For example, the assistant can accept queries entered by the user, search the knowledge-base and find relevant results; if the number of results falls below a certain threshold, the assistant may automatically identify an appropriate team member to help with the query, route it to him/her, and keep track of whether the query has been satisfactorily answered or not. Such assistance can be of considerable help to new team members trying to come up to speed.

4. Testing in a Distributed Environment

In distributed projects, modules are implemented and unit tested at remote development centers, and then integrated periodically at a central location (e.g. customer premises). While actual unit/integration testing procedures in distributed projects are no different from collocated ones, we found that new challenges arise due to privacy of test data, size of the production database and imprecision of interface documents. We consider these issues below.

Data Privacy: In maintenance projects, data available in production databases of the live system have traditionally been used for testing. However, with increasing security concerns, many customers are now unwilling to share this data across organizational/geographic boundaries. During our study of distributed development, we discovered that the unavailability of real-life data makes it difficult for remote development teams to conduct comprehensive unit testing of their modules. The practitioners we interviewed reported two challenges they faced: First, a substantial overhead is imposed on them if they have to generate mock databases themselves to thoroughly test the code. Secondly, mock databases may not sufficiently reflect the complexities and intricacies of real-life data, and successfully unit-tested code based on fabricated data often reveals errors when tested later with actual data.

This calls for techniques to de-sensitize production data before use. Research in the area of privacy-preserving databases is thus relevant to remote software development teams looking to test their code. Wiederhold et al [39] have developed a mechanism to prevent release of confidential information by introducing a security filter between the production databases and the applications being tested. The filter is responsible for storing the security constraint policies and distorting sensitive information being returned to applications based on these pre-defined policies. In [40] Wu et al propose a technique for generating intelligent test databases by first extracting rules and statistical data from the production database and then synthesizing random data into the test database according to the extracted rules. The field of statistical databases has developed methods to prevent the disclosure of confidential individual data while satisfying requests for aggregate information (sum, count, average etc) [41, 43, 44]. Also relevant is research in the area of privacy preserving data mining [42, 45], where the objective is to prevent the disclosure of confidential individual values while preserving general patterns and rules. Future research can explore ways in which privacy policies and rules may be efficiently updated to account for changes in the live production database. It may also be interesting to conduct empirical studies to verify the effectiveness of test data generated using these methods (as compared to the live data) in uncovering defects in the software modules being tested.

Size of Production Database: Even if production data is not considered confidential, the database may be too large to transfer and maintain at a remote development site. For example, the India-based teams we interviewed noted that bandwidth available to them is insufficient for downloading large production databases from the customer location. As a workaround, they often take "slices" of the production data for unit testing at the different sites. This lessens data transfer costs and in-transit security issues, but leads to new challenges in maintaining referential constraints between records in different tables. Consider a simple example: A and B are two tables in a database with attributes a1, a2, a3 and b1, b2, b3 respectively, where a1 is the primary key of table A and foreign key for table B (related to b1). A slice of such a database cannot be taken randomly; rather records should be selected in such a way that for all values of b1 in records of slice of B, there is a corresponding record in slice of A where a1= b1. This problem gets further aggravated when some constraints are embedded in code and SQL query statements rather than database schema. A second challenge related to slicing is similar to the one for creation of mock databases – how to get a truly representative slice which contains values for different possible ranges/types of data.

Integration Testing: A large number of defects in a distributed project show up only during integration, when modules developed and unit tested at different sites are actually put together and executed. In fact integration testing was one of the early challenges of distributed development reported in the literature [19]. The root cause of most integration problems is inadequate documentation/understanding of interface requirements. As [19] noted in a study of integration difficulties in a distributed project, "interface specifications lacked essential details such as message type, return types and assumptions about performance." To this list, we may add semantic considerations like permitted method invocation sequence and constraints on input parameters and return values, which are seldom well-documented.

In collocated development, ambiguous specifications are far less of a problem, since rich informal communication among participants effectively compensates for any gaps in understanding. This is one of the reasons why traditionally software engineers have not considered it worth their effort to make specifications extremely precise descriptions of their intent. The key difference in multi-site development is that high-bandwidth informal communication is much harder to achieve [4]. Thus integration or "recomposition" [48] challenges in such projects, as described above, present a compelling case for leveraging research in the area of formal specification languages [49]. In other words, in distributed software development, it is indeed worth the extra effort in making interface requirements and other specification documents convey the precise and full information needed for product integration. Previous (albeit dated) work [50] in this regard has advocated a "module interconnection language" to create artifacts that carry interface knowledge. With various modeling and constraint languages [51, 52] gaining mainstream acceptance, there is an opportunity for formal methods research to address an important and practical problem in distributed development.

There is another benefit of having formal interface descriptions: it opens up the possibility of automatically generating *smart* simulators of remote modules based on their specification. Such simulators (stubs/drivers) can be used for conducting "pre-integration" checks during unit testing, leading to early discovery of potential interfacing problems. This can significantly enhance the quality of unit testing at any site, and also reduce the gap between unit and subsequent integration testing.

5. Process and Metrics Issues

The dynamics of collocated software development are well understood by now. Popular software process frameworks like Software Engineering Institute's Capability Maturity Model (CMM) and ISO 9001, detail key process areas (KPAs) for software development, and have been well-tested over the years in collocated projects. However, these frameworks were not designed keeping global development in mind and lack KPAs that address capabilities for managing distributed software projects [53]. This deficiency in well-known process frameworks is becoming increasingly critical in practice since quality of software (as opposed to cheap labor) is emerging as the primary differentiator between numerous low-cost vendors competing for outsourcing contracts worldwide. As our interviews of distributed development practitioners -- as well as a survey of existing literature – reveal, there has been relatively little investigation of process and metrics issues in distributed development, making this a fertile ground for research.

Several studies have suggested that there are differences in software development practices and performance levels around the world. For example, in 1990, a survey of forty projects in the United States and Japan reported that in Japanese projects, more time was spent on product design while American teams spent more time on actual coding [56]. A more recent survey [57] of 104 firms in India, Japan, USA and Europe reveals that Indian and Japanese firms invested much more in detailed design specifications compared to others; more Indian and European firms broke down projects into subcycles than their US and Japanese counterparts; again, in terms of defect levels, Indian and US projects were quite similar to each other, but were considerably higher than the Japanese, while European firms showed the highest defect levels. These results are particularly relevant to firms that are considering greater outsourcing of their software development activities, and to managers of distributed projects spanning several regions/organizations. It indicates that metrics and processes that can efficiently keep a collocated project on track may not scale to distributed projects with inherent differences in process and performance across sites. The feedback we collected during our study of distributed development in IBM resonates with this hypothesis.

An early step towards enriching the CMM framework with KPAs for distributed development has been presented in [53]. It identifies 24 new KPAs that address the wide-ranging capabilities needed for managing global projects including the setting of shared business goals among participating development centers, identification, sharing and standardization of best practices in distributed development across global organizations along with policies for common knowledge transfer and the enabling of cross-site informal communication through continuous infrastructure improvement.

There is a need to expand and refine these models by considering other potential KPAs in distributed development e.g. the management of risk. There are inherent risks in any software development effort, but risks in distributed projects tend to be less visible [58], and therefore more difficult to deal with. As such, risk management in distributed projects must begin early; research has to identify the different levels (e.g. strategic level, operational level [59] etc.) at which risk analysis must be performed, and the methods for doing the same. Another important process area concerns the distribution of work across sites. Previous work has reported on a number of models that may be used for organizing work based e.g. on functional areas of expertise, product architecture, process steps, release plans and so on [9, 36]. As [9] reports, however, no model resolves all coordination issues that arise from distribution, so an important research issue is to choose the right model depending on the dominant coordination problem. It would also be interesting to study if hybrid models can be deployed to support coordination in very large development efforts. Such models may then be progressively refined through appropriate post-mortem activities e.g. analysis of defect data that may indicate the effectiveness of geographic distribution [54].

As KPAs for distributed development are identified and incorporated into process frameworks, a related research direction would be the formulation of appropriate metrics to help quantify these areas. For example, what metrics can accurately measure the risks and benefits of project dispersion? How to compose measures from different sites to gain insight into the overall project status? In addition, we propose the following directions for a metrics-based management of distributed projects:

- Identify metrics to represent the effort of each partner involved in the project and measure that the assigned responsibilities are met. In a distributed project, it is important to define the responsibilities of the different organizations/ teams at a sufficiently detailed level so that there is no misunderstanding as to what is going to be performed by whom. Some possibilities here include the detailing of responsibilities of teams using ODC's [55] signature metric and definition of function exit metrics to measure the quality of their deliverables.

- Develop indicators that provide information about client expectations and their impact on the *project value chain*, which is the process by which a series of activities are linked together for the purpose of creating value for the client. This would allow participating organizations to adopt a broad perspective of the project and measure how their input generates value for client.
- Use information from existing projects as well as history of past projects to develop metrics which help provide estimates for future requests such as contract pricing [60], software insurance and distribution of work between teams. The broad aim of these predictive metrics would be to support business decision making processes.

6. Summary and Conclusion

In this paper, we reported on our study of distributed software development, which helped us synthesize a research agenda for this field. Collaborative software tools, knowledge acquisition and management, testing in a distributed set-up, and process and metrics issues were identified as areas where important research questions need to be addressed to make distributed development more effective. We presented a brief summary of related research in each of these areas, and also outlined open research issues. To re-iterate, here are some of the most important (in our opinion) research areas in software engineering that would have a big impact on distributed development:

1. Development of collaborative environments encompassing all phases of software development (requirements, design, coding, testing).
2. Reverse-engineering tools to recover knowledge from existing applications
3. Integrating knowledge extracted from formal and informal artifacts
4. Maintenance of informal knowledge in a human-independent way
5. Development of techniques to model production data for testing to preserve privacy of the real data, and to work effectively with a representative subset of data
6. Use of formal specifications to ease integration of modules developed by different teams.
7. Enhancement of software development process frameworks by identifying and incorporating key process areas relevant to distributed development
8. Development of a set of metrics to quantify these process areas in support of project management and decision-making in a distributed environment

We hope that the agenda presented in this paper will help focus research efforts on areas of practical importance in distributed software development and thereby help advance the state-of-practice in this domain.

7. Acknowledgements

We would like to thank John Patterson for extensive discussions on collaborative development environments. Acknowledgements are also due to James Herbsleb and Peter Santhanam for their inputs on process and metrics issues in distributed development.

8. REFERENCES

[1] Nasscom - Mckinsey report 2002
http://www.nasscom.org/artdisplay.asp?Art_id=1225

[2] Carmel, E. and Agarwal, R. Offshore Sourcing of Information Technology Work by America's Largest Firms. Technical Report, Kogod School, American University, Washington D.C., November 2000.

[3] Business Week Online, March 1, 2004.
http://www.businessweek.com/magazine/content/04_09/b3872001_mz001.htm

[4] Herbsleb, J.D and Moitra, D.: Global software development : IEEE Software, March-April 2001, pages 16-20

[5] Damian, D. and Zowghi, D. Requirements Engineering challenges in multi-site software development organizations. Requirements Engineering Journal 8, 2003, pages 149-160

[6] Heeks, R., Krishna, S., Nicholson, B. And Sahay, S. Synching or Sinking: Global Software Outsourcing Relationships, IEEE Software, March-April, 2001, pages 54-60

[7] Herbsleb, J.D, Mockus, A, Finholt, T.A., and Grinter,R.E . Distance, Dependencies and Delay in a Global Collaboration. ACM Conference on Computer Supported Cooperative Work (CSCW) 2000, pages: 319-328.

[8] Carmel, E. Global Software Teams. Prentice Hall, 1999.

[9] Grinter, R., Herbsleb, J. and Perry, D. The geography of coordination: Dealing with distance in R&D work. International ACM SIGGROUP Conference on Supporting Group Work, 1999, pages 306-315

[10] Espinosa, J., Kraut, R.E., Lerch, F.J., Slaughter, S.A., Herbsleb, J. and Mockus, A. Shared Mental Models and Coordination in Large-Scale, Distributed Software Development. International Conference in Information Systems 2001

[11] Van den Bulte, C., Moenaert, R. The effects of R&D team co-location on communication patterns among R&D, marketing, and manufacturing. Management Science, 1998

[12] Olson, G.M, and Olson, J.S. Distance matters. Human-Computer Interaction, 2000

[13] McDonough, E. F, Kahn, K., and Barczak, G. An investigation of the use of global, virtual, and collocated new product development teams. Journal of Product Innovation Management 2001

[14] Kiesler, S., Cummings, J.N. What do we know about proximity in work groups? A legacy of research on physical distance. In Distributed Work, Hinds, P., Kiesler, S. (ed.) 2002

[15] Kobitzsch, W., Rombach, D. and Feldman, R. Outsourcing in India. IEEE Software, March-April, 2001

[16] Arora, A., and Gambardella, A. The Globalization of the Software Industry: Perspectives and Opportunities for Developed and Developing Countries. NBER Working Paper No. 10538, June 2004. http://www.nber.org/papers/W10538

[17] Ebert, C. and De Neve, P. Surviving Global Software Development. IEEE Software, March-April, 2001.

[18] Herbsleb, J.D., and Mockus, A. An Empirical Study of Speed and Communication in Globally-Distributed Software Development. IEEE Transactions on Software Engineering, 29(3), 2003

[19] Herbsleb, J. D. & Grinter, R. E. Architectures, Coordination, and Distance: Conway's Law and Beyond. IEEE Software, Sept/Oct 1999, pages 63-70.

[20] Krishna, S., Sahay, S. and and Walsham, G. Managing Cross-Cultural Issues in Global Software Outsourcing. Communications of the ACM. Volume 47, Number 4, April 2004.

[21] Mockus, A. and Herbsleb, J. Expertise Browser: A Quantitative Approach to Identifying Expertise. International Conference on Software Engineering, 2002, pages 503-512

[22] Carmel, E. and Agarwal, R. Tactical Approaches for Alleviating Distance in Global Software Development. IEEE Software, March-April, 2001

[23] http://www-306.ibm.com/software/awdtools/clearcase/, http://www-306.ibm.com/software/awdtools/clearquest/

[24] Jarvenpaa, S. and Leidner, D. Communication and Trust in Global Virtual Teams. Journal of Computer Mediated Communication 3(4), June, 1998.

[25] Booch, G. and Brown, A. Collaborative Development Environments. Advances in Computers Vol. 59, Academic Press, August 2003.

[26] http://www.collab.net, http://sourceforge.net

[27] http://www.eclipse.org/stellation

[28] Carroll, M. and Sprenkle, S. Coven: Brewing Better Collaboration through Software Configuration Management. ACM SIGSOFT Foundations of Software Engineering, 2000, pages 88-97

[29] http://sangam.sourceforge.net

[30] Cheng, L., DeSouza, C., Hupfer, S., Patterson, J. and Ross, S. Building Collaboration into IDEs. ACM Queue vol.1 no.9, 2004

[31] Maurer, M., Succi, G., Holz, H. et.al. Software Process Support over the Internet. International Conference on Software Engineering (ICSE), 1999, pages 642-645

[32] Sinha, V., Sengupta, B., and Chandra, S. EGRET: A Collaborative Tool for Distributed Requirements Management. IBM Research Technical Report, RI06001, 2005.

[33] Graham, T.C.N. et al. A World-Wide-Web Architecture for Collaborative Software Design. Software Technology and Engineering Practice (STEP'99). 1999: IEEE Press.

[34] Mehra, A., Grundy, J.C. and Hosking, J.G. Supporting Collaborative Software Design with a Plug-in, Web Services-based. Workshop on Directions in Software Engineering Environments at ICSE 2004.

[35] Allen, T.J. Managing the Flow of Technology. MIT Press, 1977

[36] Allen, T.J. and Hauptman, O. The Influence of Communication Technologies on Organizational Structure: A Conceptual Model for Future Research. Communication Research 14(5), 1987, pages 575-587.

[37] http://www.doxygen.org

[38] Cubranic, D., Murphy, G.. Hipikat: Recommending Pertinent Software Development Artifacts, International Conference on Software Engineering 2003, pages 403-418

[39] Wiederhold, G., and Bilello, M. Protecting Inappropriate Release of Data from Realistic Databases. Ninth International Workshop on Database and Expert Systems Applications, Vienna, Austria, 1998.

[40] Wu, X., Wang, Y., and Zheng, Y. Privacy Preserving Database Application Testing. ACM Workshop on Privacy in Electronic Society (WPES), 2003, pages 118-128

[41] Adam, N.R., and Wortman, J. C. Security-control methods for statistical databases. ACM Computing Surveys, 21(4), 1989, pages 515-556

[42] Agrawal, R. and Srikant, R. Privacy-preserving Data Mining. Proceedings of ACM SIGMOD Conference on Management of Data, pages 439-450, Dallas, Texas, May 2000.

[43] Domingo-Ferrer, J.Current Directions in Statistical Data Protection. In Proceeding of Statistical Data Protection, 1998.

[44] Dinur, I., and Nissim, K. Revealing Information while Preserving Privacy. Proceedings of 22nd ACM Symposium on Principles of Database Systems, ACM Press, 2003, pages 202-210

[45] Brankovich, L., and Estivill-Castro, V. Data Swapping: Balancing Privacy Against Precision in Mining Logical Rules. Proceedings of International Conference of Data Warehousing and Knowledge Discovery, 1999.

[46] D. Chays, S. Dan, P. Frankl, F. Vokolos, E. Weyuker. A Framework for Testing Database Applications. Symposium on Software Testing and Analysis, 2000

[47] Mockus, A., Fielding, R. and Herbsleb, J. Two Cases of Open Source Software Development: Apache and Mozila. ACM Transactions on Software Engineering and Methodology (TOSEM), 11(3), pages 309-346

[48] Grinter, R. Recomposition: Putting it All Back Together Again. ACM Conference on Computer Supported Cooperative Work (CSCW), 1998, pages 393-402.

[49] The World Wide Web Virtual Library: Formal Methods. http://vl.fmnet.info

[50] DeRemer, F., and Kron, H. Programming in the Large vs. Programming in the Small. Proceedings of International Conference on Reliable Software, pages 114-121, 1975.

[51] Booch, G., Rumbaugh, J., and Jacobson, I. The Unified Modeling Language User Guide. Addison Wesley, 1998

[52] Warmer, J., and Kleppe, A. The Object Constraint Language Precise Modeling with UML. Addison Wesley, 1999

[53] Ramasubbu, N., Krishnan, M.S., and Kompalli, P. Leveraging Global Resources: A Process Maturity Framework for Managing Distributed Development. IEEE Software, Volume 22, Issue 3, pages 80-86, May 2005

[54] Bassin, K., and Santhanam, P. Managing the Maintenance of Ported, Outsourced, and Legacy software via Orthogonal Defect Classification. International Conference on Software Maintenance, 2001

[55] Chillarege, R., Bhandari, I., Chaar, J. et. al, Orthogonal Defect Classification-A Concept for In-Process Measurements. IEEE Transactions on Software Engineering. Volume 18,Issue 11, November, 1992.

[56] Cusumano, M., and Kemerer, C.F. A Quantitative Analysis of US and Japanese Practice and Performance in Software Development. Management Science, volume 36, no. 11, pages 1384-1406, November, 1990

[57] Cusumano, M., MacCormack, A., Kemerer, C., and Crandall, W. Software Development Worldwide: the State of the Practice. IEEE Software, 20(6), November/December 2003, pages 28-34

[58] Karolak, D.W. Global Software Development – Managing Virtual Teams and Environments. Los Alamitos, IEEE Computer Society, USA, 1998

[59] Prikladnicki, R., Yamaguti, M. H., Antunes, D. C. Risk Management in Distributed Software Development: A Process Integration Proposal. 5th IFIP Working Conference on Virtual Enterprises at 18th IFIP World Computer Congress, 2004

[60] Li, P., Shaw, M., Herbsleb, J., Ray, B., Santhanam, P. Empirical Evaluation of Defect Projection Models for Widely-deployed Production Software Systems. ACM SIGSOFT Software Engineering Notes, Volume 29, Issue 6, November 2004.