2 July 2004

Effect of Quality Management Practices in Distributed Offshore Software Development

Narayan Ramasubbu, University of Michigan
Sunil Mithas, University of Michigan
M.S.Krishnan, University of Michigan
Chris F. Kemerer, University of Pittsburgh

Abstract

While IT vendors have made significant improvements in the process maturity of software development, growth of information technology outsourcing and distributed software development have posed new challenges in managing software projects. In this paper we develop and test empirical models of productivity and conformance quality in the context of distributed software development. By analyzing data collected on more than forty large globally distributed commercial projects from a leading software vendor operating at the highest software development process maturity level (CMM-5), we find that, while prevention and appraisal-based quality approaches are effective in improving distributed development productivity, failure-based quality approaches are effective in improving quality. These results indicate that investments in quality management practices have varying effects on project performance parameters. Our results suggest that the choice of appropriate quality management approach should be guided by the development context and project goals.

Keywords: Distributed development, Virtual Teams, IT Outsourcing, Software Quality, Process maturity, Capability Maturity Model (CMM), Indian software industry, Global software development, Software Engineering.

Acknowledgements: We thank Satyendra Kumar, Anoop Kumar and V. R. Janardhanan for their support and help in obtaining the necessary data for this research. We acknowledge helpful comments received from Prof. Donald E. Harter and Prof. Judy Olson on an earlier version of this paper. Financial support for this study was provided in part by the Michael R. and Mary Kay Hallman fellowship at Michigan Business School.

Empirical Analysis of Quality Management Practices in Distributed Custom Software Development

1. INTRODUCTION

The need for servicing global customers by leveraging the wide talent pool available across national boundaries has made it imperative to disperse software development across teams at remote locations. Globally distributed software development is fast becoming a popular business model for software service organizations (Amoribieta, et al. 2001, Battin, et al. 2001). India has emerged as an attractive location for software development in the context of increasingly competitive global economy and the need to develop software applications rapidly in a cost effective manner without compromising on quality. Out of the 140-odd companies with SEI-CMM level 5 certification world-wide 42 are from India, the largest pool of high maturity software organizations from any country (Paulk 2003). According to a recent survey, custom software applications worth $ 3.23 billion were jointly developed by software service companies in India and United States. Although in the beginning U.S. software firms took the lead by setting up software factories in India, of late Indian firms have started setting up development centers in the U.S. to provide seamless services to their clients. During 2002-2003 as many as 270 Indian companies set up offices, subsidiaries and alliances in the United States (Nasscom-McKinsey 2002).

Globally distributed software development achieves division of labor by dispersing software development tasks among several remotely located development centers. Typically a software vendor performs major software development at its primary development center that is geographically distant from a secondary development center located at the customer's premises or at a subsidiary office near to the customer. This

mode of distributed software development raises several important issues that have not been widely addressed in previous software engineering research. For example, previous research suggests that work dispersion among distributed teams is associated with increased occurrences of communication, coordination and administration problems that cause difficulty in achieving mutual understanding and a common frame of reference (Cramton 2001, Jarvenpaa and Leidner 1999, Maznevski and Chudoba 2000, Sproull and Kiesler 1986). Although quality management practices such as CMM and ISO 9001 have been developed to improve software quality in general, the efficacy of such quality management systems in the context of distributed software development is an open empirical question. Additionally, while vendors have adopted quality management approaches with varying degrees of emphasis on prevention, appraisal and failure-based approaches; there is a dearth of academic research to guide managerial decision-making for resource allocation among these quality management approaches to achieve desired project goals (Fitzgerald, et al. 2003).

Against this backdrop, in this paper we develop and test models of software productivity and quality to analyze the impact of work dispersion and quality management approaches on software project performance in a distributed environment. The main contributions of this study are as follows. First, we analyze the effect of work dispersion on productivity and quality for projects executed in the highest CMM maturity regime (level 5). Availability of fine-grained and audited quality data at our CMM level 5 research site makes it possible to address the above research questions for the first time. Second, we extend previous software project performance models primarily developed in co-located software development context to the distributed development context to

inform the debate on the benefits and implications of offshore software development (The Economist 2003). Finally, unlike past studies, we develop models of software development productivity and quality that explicitly take into account the managerial allocation of resources among prevention, appraisal or failure-based quality management approaches.

The rest of the paper is organized as follows. In the next section we provide a theoretical basis for our hypothesis. In section 3, we define our variables, develop our empirical models and describe our research site and data collection method. Section 4 discusses our estimation procedures and data analysis results. Finally, we present our conclusions and provide further research direction in section 5.

2. THEORETICAL FRAMEWORK

The modeling framework for this study is developed based on the economic view of software development (Banker, et al. 1991, Banker, et al. 1998, Boehm 1981). Researchers using this framework treat software development as a production process and model software performance indicators such as productivity and quality based on personnel related factors and software methodology related factors. Software engineering literature offers several research studies that have used this framework to analyze the drivers of project performance indicators such as productivity and quality (Banker and Slaughter 2000, Harter, et al. 2000, Harter and Slaughter 2003, Krishnan, et al. 2000). In this study we extend this framework to address distributed development by including work dispersion factors and nuanced quality management factors. We draw on organization theory and quality management literature to develop the conceptual model for this research.

2.1 Work Dispersion and Software Project Performance

Work dispersion has been noted as having influence on software project performance. Past research examining distributed software maintenance tasks among multiple development centers provides preliminary evidence for negative association of dispersion with project performance. For example, Mockus and Weiss (2001) report that distributing product maintenance work across development centers significantly affects the cycle time of a project. Likewise, Herbsleb et al. (2001) report that, compared to same-site work, cross-site work takes a much longer time and requires more people for work of similar size and complexity. Recently, a field study by Teasley et al (2002) found that collocation of team members in software development led to higher productivity and shorter cycle time. However, there is little understanding of the causal mechanisms that apparently underlie the negative effect of work dispersion on software project performance.

Based on our literature review and field-work we uncover and propose three underlying mechanisms to explain the effect of work dispersion on project performance: mutual knowledge to accomplish interdependent tasks, coordination difficulties and human resource issues. We next discuss the role of these three mechanisms in explaining the effect of work dispersion on software project performance.

2.1.1. Dispersion and interdependent tasks

Distributed software development involves remote teams who interact through interdependent tasks guided by a common goal and work across space, time and organizational boundaries (Lipnack and Stamps 1997). In globally distributed software

development settings the common goal for remote teams is to deliver software product to the customer, and this involves numerous interdependent tasks throughout the software development cycle. Interdependence in tasks could be either through the structure of the task (Thomson 1967), or through the way in which members behave in executing their task (Shea and Guzzo 1989). Interdependence through the structure of the task occurs when each member needs to complete their actions for other members to do their part of their work, or when members with different expertise have to work together on a task. For example, in software development a testing team could prepare test cases but cannot complete their tasks until the design and coding teams finish their individual work. Also the task of integrating different software modules to form a whole application is inherently interdependent. Interdependence through the way members execute a task occurs when individual tasks could be completed independently, but are controlled and approved by a different set of individuals. This type of interdependence occurs in software development when control and supervision processes require the individual work of developers to be reviewed by peers and approved by the quality team before implementation.

Managerial theorists have long identified managing task interdependence as an important challenge in a distributed work environment (March and Simon 1958, McCann and Galbraith 1981, Thomson 1967). From an organization sociology perspective, interdependence in performing tasks leads to task uncertainty, fluctuations in resource availability and goal conflicts between interdependent units. In turn, task uncertainty, fluctuations in resource availability and goal conflicts are negatively associated with group performance (Victor 1990). In addition, successful completion of interdependent

tasks across dispersed groups requires common understanding or 'mutual knowledge' about the tasks and other work routines. Cramton (2001) reports that physical dispersion of collaborators negatively affects the means by which mutual knowledge is established. This may be because of unevenly distributed information, failure to communicate and retain contextual information, differences in the salience of information and relative differences in the speed of access to information. One of the software developers at our research site narrated his experience in explaining the difficulty of achieving mutual knowledge in distributed development in the following words:

"…..I had worked on the customer's system from offshore for more than a year. But I never fully understand why my onsite coordinator initiated so many change requests for my work even after we had discussed its design before hand. Only when I visited onsite, I fully understood the production mechanism and end users' expectation. That's when I realized how different our views of the system where. I wish I can pass on this experience to my offshore colleagues; but it is not easy…." [Interview Transcript, Development Programmer, July 7, 2003]

Thus, based on above discussion, we posit that difficulty in achieving mutual knowledge about tasks might lead to task uncertainty and rework, and thus negatively affect the productivity and quality of distributed software projects.

2.1.2. Dispersion & coordination

Software projects involve knowledge intensive work and require intense coordination of resource. Nidumolu (1995), utilizing both a structural contingency perspective and a risk management perspective, shows how effective coordination in software projects could reduce uncertainty and improve performance. Likewise, Koushik & Mookerjee (1995) show that resources spent on coordination efforts of interdependent tasks are critical to project performance. More recently, Faraj & Sproull (2000) highlight

the need for coordinating expertise and managing knowledge interdependencies to achieve high quality work in software projects.

A large number of coordination activities in dispersed teams occur primarily through electronic media. It is important to analyze the extent to which electronic media helps in gaining a better understanding of dispersion on project performance. Based on their review of a decade of research on the impact of electronic communication tools on non-collocated team work, Olson and Olson (2000) conclude that:

"…reports of distance's death are exaggerated. Even with all our emerging information and communications technologies, distance and its associated attributes of culture, time zone, geography and language affect how humans interact with each other. There are characteristics of face-to-face interactions, particularly the space/time contexts in which such interactions take place, which the emerging technologies are either pragmatically or logically incapable of replicating….Distance is not only alive and well, it is in several essential respects immortal." [p. 140-141]

Other researchers examining the effect of distance on communication and coordination activities have come to similar conclusions. For instance, Kraut et al. (1990) report that frequency of communication among collaborators decreases with distance even with the availability of communication tools. Similarly, Herbsleb and Grinter (1999) in their case study of a multi-site software organization, find that distributed coordination activities were often vulnerable to imperfect foresight and uncertainties that required substantial renegotiation of commitments among distributed teams. Thus, coordinating interdependent tasks through electronic communication mechanisms may not be as effective as face-face interactive coordination. Difficulty in coordinating tasks in dispersed teams might lead to increased occurrences of rework and lapses. Thus,

dispersion in software development might be negatively associated with productivity and software quality because of increased co-ordination difficulties.

2.1.3. Dispersion & human resource management

Software development is a personnel-intensive activity and human resources are of primary importance in a software project. One of the important issues in managing a dispersed work force is deciding on the reporting structures amidst temporary relocations. Temporary relocations occur several times in distributed software projects in connection with imparting training, requirements definition and project installation and support. Temporary relocations disrupt normal reporting structures and often lead to the problem of ambiguous authority (Goodman 1967). Often, a matrix form of reporting structure is deployed for such temporarily relocated employees and they end up reporting to multiple managers. Monitoring and appraising such resources is difficult and consumes significant administration effort. Unsatisfactory supervision of these employees might lower motivation and affect employee morale, leading to lowered job performance. An employee at our research site offered this comment:

"……when I travel to my onsite and stay for few months it is not clear to me if my onsite manager passes on all information about my achievements to my offshore manager. Though my onsite manger monitors my task, it is my offshore manager who decides my annual bonus. Some times I feel I am at a disadvantage as compared to my offshore colleagues…."[Interview Transcript, Development Programmer, July 7, 2003]

From a resource allocation perspective too, managers face a significant challenge in ensuring allocation of stable and uniform workload for distributed employees. Employees face severe fluctuations in workload because it is difficult to predict in

advance the daily amount of work for an employee in an interdependent task scenario. A senior software engineer at our research site noted:

"…..my contract says my office time is flexible. Sometimes I spend all my time at office browsing the internet, chatting with friends and doing personal stuff because my colleagues at onsite have not responded. Other times I go home late continuously for a more than a week and I do not get time to spend with my family; it is frustrating…." [Interview Transcript, Senior Software Engineer, July 10, 2003]

Employees encountering distributed nature of the tasks might also face additional process overheads that they might find stressful. The following comment from a software developer illustrates this issue:

"…..I wish I could do my job independently. But I do not have enough permission in the production system. I have to explain each and every change I have done to my onsite colleague and additionally write minutes of meeting, documents for future knowledge transfer. Sometime this takes more time than the task itself and is stressful." [Interview Transcript, Development Programmer, July 7, 2003]

The above observation is consistent with findings of Parasuraman and Alutto (1981) that interdependence and lack of autonomy are significant antecedents of organizational stress. Thus, managing distributed human resources encounters additional administrative and process overhead effort that might hinder superior project performance.

Based on these discussions, we posit our first set of testable hypotheses as,

H1a) Higher dispersion in software development work is associated with lowered development productivity.
H1b) Higher dispersion in software development work is associated with lowered conformance quality.

2.2. Dispersed development, quality management and performance

In this section we develop a theoretical understanding on the impact of quality management practices on software project performance at an operational level.

Although quality has multiple dimensions, consistent with previous research in software engineering, in this study the term quality refers to conformance quality or the degree to which software conforms to customer's specification (Garvin 1987). Past researchers (Harter, et al. 2000, Herbsleb, et al. 1997, Krishnan, et al. 2000) who studied the impact of software quality management practices in a co-located context analyzed the benefits of quality investments at an aggregate level using overall investments in quality management. Our goal in this study is to understand the effectiveness of individual quality management approaches to develop guidelines for the distributed software development context. Building on the categorization used in manufacturing quality research in the past (Mendez and Narasimhan 2002, Nandakumar, et al. 1993), we study the effect of prevention, appraisal and failure-based (testing and correction) quality management practices on software project performance.

Prevention-based quality management practices in software development involve activities such as training that are primarily done to avoid the occurrence of bugs. Appraisal activities involve proactively assessing progress, performance and intermediate artifact quality at various stages in the development. Failure-based quality approaches include testing the adherence of applications to customer specifications and subsequent defect correction activities. Table 1 summarizes the categorization of individual quality management practices at our research site to the broad categories of prevention, appraisal and failure-based quality management approaches.

Table 1. Categorization of individual quality management practices

Prevention	Appraisal	Failure
Programming training Business domain training Process training Configuration management Planning & Scheduling	Requirement, Specification & Design reviews Code inspection Status reviews	Unit testing System testing Error tracking & correction Prototyping Intermediate builds

The quality management literature in the operations domain provides insights for analyzing the effectiveness of individual quality management practices in improving project performance in the distributed development context. The classic economics of quality model (Lundvall and Juran 1974) posits that there exists a cost minimizing conformance quality level resulting from the tradeoffs between the costs of attaining higher quality (prevention & appraisal) and costs of having produced poor quality (failure costs). However, an alternate school of thought (Crosby 1979, Fine 1986, Walton and Deming 1988) believes that producing high quality products is always less costly and posits zero defects as the optimal conformance quality level. There is some agreement that there may be some tradeoffs between productivity and quality, two key indicators of software project performance. Given the difference in the nature of manufacturing goods and software one might doubt the existence of such tradeoffs in software production. However recent software studies verify that these tradeoffs exist even in software production (Krishnan, et al. 2000, MacCormack, et al. 2003). To gain a nuanced understanding of such tradeoffs in distributed development we next theorize the influence of dispersion on the effects of prevention, appraisal and failure-based quality activities on project performance.

Prevention-based quality practices attempt to improve employee skills through training, planning, scheduling and configuration management. Prevention-based activities thus build a suitable environment and infrastructure for software development. Several authors have documented evidence for the benefits of investments in prevention-based quality approaches in the co-located context (Smith, et al. 2001, Zantek, et al. 2002). We expect that the beneficial effects of preventive based quality approaches in the co-located context to prevail in the distributed development context also. Thus our next set of hypothesis is:

H2a) Investments in preventive based quality management practices are positively associated with distributed development productivity.
H2b) Investments in preventive based quality management practices are positively associated with distributed development conformance quality.

Quality appraisal activities attempt to identify potential errors and understand performance lapses by cognitively inspecting specifications, design structures and preliminary code. Unlike failure-based quality practices, appraisal-based activities do not require a functioning application and hence can be performed even during the initial stages of the development lifecycle. This enables developers to identify potential lapses early in the development stages and avoid the cascading flow of errors to subsequent development stages and other interdependent tasks. Also, appraisal-based quality management approach requires peer reviews of accomplished tasks, system performance, and overall status throughout the development life cycle. These reviews facilitate distributed resources to discuss and debate their individual perspectives and help development of a common understanding of goals, tasks and processes and facilitate a forum for conflict resolution. We posit that this mutual understanding gained through appraisal activities help dispersed team members to be more efficient. Another benefit of

investments in appraisal-based activities is that an assessment of project progress is available to the manager for possible re-estimation and rescheduling, thus averting the detrimental effects of last minute schedule pressure. Averting a last minute schedule pressure might create conducive environment for superior performance of employees. Thus we hypothesize that,

H3a) Investments in appraisal-based quality practices are positively associated with distributed development productivity.

H3b) Investments in appraisal-based quality practices are positively associated with conformance quality.

Failure-based quality management activities involve actual execution of the newly built software application in a simulated production environment and observing the behavior of the software code. Detailed test plans and use cases are built that are used to simulate real time situations with the software application. Specific tests, such as module integration tests and system integration tests, examine the reliability of interfaces. It is well known that critical software errors occur at the interfaces between interdependent modules and failure-based techniques identify such errors before actual implementation. Thus, investments in failure-based quality management practices are designed to help achieve better levels of conformance quality.

However, failure-based software quality management practices entail several iterations of build-test-analyze-rebuild cycles. Any deviance from the expected behavior observed during testing needs a root cause analysis and error tracking follow-up activities. In a distributed development setting knowledge for such analysis may not be present locally for easy reference. Also, identifying errors after the system is built leads to significant rework. In the presence of interdependencies rework of one task cascades to

several levels. Thus, increased levels of rework resulting from failure-based approaches tend to increase the development effort required for a project.

These lead to our next set of hypotheses,

H4a) Investments in failure-based quality practices are negatively associated with distributed development productivity.

H4b) Investments in failure-based practices are positively associated with conformance quality.

We next compare appraisal and failure-based quality management practices in the distributed development context. Fine (1986) proposes a learning theory perspective to analyze the relative efficacy of appraisal and failure-based quality practices. He analytically shows that learning or gaining knowledge is one of the mechanisms through which relationships between quality investments and project performance can be understood. Specifically, learning theory analyzes performance using the concepts of *'learning-by-doing'* and *'learning-before-doing'*. The notion of '*learning-by-doing* (Hatch and Mowery 1998, Hippel and Tyre 1995) posits that critical problems that cause inefficiencies can be discovered only through an actual production process. In other words, achieving potential performance is an iterative process. In contrast, the notion of *"learning-before-doing"* (Pisano 1994) posits that, when process knowledge is sufficiently strong, effective learning might take place outside the final production environment and can be productively harnessed for better performance.

In software development, investments in appraisal-based quality management activities such as peer reviews assist better understanding by identifying potential problems through cognitive analysis *before* a functional system is built. Thus these investments constitute '*learning-before-doing'*. In contrast, failure-based approaches rely on activities that are conducted only *after* a functional system is built and thus

investments in failure-based quality management activities constitute investments to achieve '*learning-by-doing'*.

In a distributed development environment integration across functional boundaries and system components are critical to development performance (Iansiti 1997). Distribution of resources across locations results in partitioning of knowledge (Takeishi 2002) and hence integration and learning capabilities rest on the mechanisms to generate and facilitate quick feedback (Pisano 1994). Appraisal based quality management practices, emphasizing '*learning-before-doing*' facilitate peer reviews at the specification and design stages before embarking on building the entire application. Thus, project participants can learn different perspectives about their interdependent tasks and agree during integration conventions. Learning about interfaces even before building the application helps developers to achieve a certain level of autonomy in their work and reduces uncertainty in the project. Also, potential mismatches in the component interfaces can be resolved even before the actual code is developed.

In contrast to appraisal-based activities, failure-based activities emphasize 'learning-by-doing' and solve interdependency problems after the application has been built. A failure-based approach results in efficient problem identification and experienced based knowledge gaining. Knowledge and experience gained through *'learning-by-doing'* failures is important and pays off in the long run. However, in the short term, significant rework needed to correct the problems identified might cause an increase in the effort needed to complete the project. Thus we posit that in a distributed development environment, *'learning-before-doing'* activities are more effective in improving productivity than *'learning-by-doing'* activities:

H5a) Investments in appraisal-based quality management practices are more effective than investments in failure-based quality management practices in improving distributed development productivity.

We hypothesized that both appraisal-based quality practices and failure-based quality practices are positively associated with conformance quality. However, these practices do not have same degree of influence. Appraisal based approaches use cognitive analysis for error detection. While cognitive analysis is effective in identification of *logical* lapses, it is likely to be less effective in identifying *data related* errors. On the other hand, failure-based approaches detect deviance from conformance by executing code and by employing modern testing tools. Thus, failure-based approaches can more easily identify both *logical* and *data related* errors. Also failure-based approaches test functionality in a simulated production environment and hence will be able to identify lapses due to configuration and customization activities which do not come under the purview of code inspection and other appraisal activities. Further, warnings and failure signs identified during quality appraisals have chances of being overlooked in a distributed development context because they are often conveyed through peer meetings or computer mediated channels. The geographically distributed nature of resources hinders immediate feedback and follow-up. However, in failure-based quality approaches there is a common working application for reference across geographical boundaries and any failure in the system due to activities like testing attracts attention of all concerned resources. Thus we propose our final hypothesis as,

H5b) Investments in failure-based quality management practices are more effective than investments in prevention and appraisal-based practices in improving distributed development conformance quality.

2.3 Control variables

We control for the effect of software reuse, code size, team size, code coverage and project duration while analyzing the effect of work dispersion and quality management approaches on software quality and productivity.

Reuse in software enables developers to use standardized routines that have been stored in repositories to accomplish certain functionality. Modern integrated development environments used for programming also provide standard functionalities to accomplish basic functions such as date conversion. Also, with the growth of open source, several functions are available on specialized sources on the internet. Usage of such standardized functions help developers to avoid reinventing the wheel and focus on customer specific needs. The impact of reuse on productivity has been well documented in software research (Banker and Kauffman 1991, Mili, et al. 1995). Thus reuse of software code is our primary control variable in the productivity model.

Code size has been an important control variable for software quality models in the past. Software size captures both the magnitude of the project and much of the complexity involved in developing the application. Code coverage, or the extent to which software code has been verified through the quality process, has been correlated with software reliability by past researchers (Chen, et al. 2001, Frate, et al. 1995).

Team size is a good indicator of the coordination difficulties that could occur within the software project team. An increased team size poses difficulties in both administrative coordination and expert coordination. Project duration can signify the intensity of the project. Researchers have reported that project intensity is a critical factor

in determining project performance (Smith, et al. 2001). Thus, we control for these factors while determining the effect of dispersion and quality management practices.

Figure 1 depicts our complete research model.

---Insert Figure 1 about here---

3. RESEARCH DESIGN & METHODOLOGY

In this section we provide a brief description of the research setting, data collection procedure, define the variables used in this study and present our empirical models.

3.1 Constructs & Variables

3.1.1 Endogenous Variables

Development productivity (PROD): Development productivity is defined as the ratio of software size (CODE-SIZE) in function points to the total development effort (DEV-EFFORT) in person-hours. Measuring software size in function points has been popular among researchers when the projects involve different programming languages. The advantage of function point measurement is that it incorporates measures of customer perceived software functionality as well as complexity (IFPUG 1999). The development effort includes effort incurred in all stages of development until the customer is satisfied and signs off the project after the completion of error fixes identified during acceptance testing.

Conformance Quality (QUALITY): Our quality measure captures the number of unique problems reported by customers during the acceptance tests and production trials before the project signoff. Similar to prior work by Subramanyam and Krishnan (2003), we calculate quality variable as conformance quality = [1/ (defects+1)].

3.1.2 Exogenous variables

Work Dispersion: A simple way to operationalize division of labor between development centers is to calculate the proportion of work done at these centers **(WRK-SHR)**. While this measure captures the division of labor for distributed tasks at an aggregate level, it does not adequately account for the interdependencies that arise in distributed development. Higher work dispersion involves greater effort to integrate modules from tasks accomplished at remote centers. Thus, effort spent on integration processes, such as system integration, synchronizing versions of code across the development centers, effort spent in creating application programming interfaces, interface specifications for colleagues in remote development centers and effort spent on administering temporarily relocated resources are a more reliable measures of work dispersion. Our models include percentage of effort spent on such integration activities of the total development effort (**WRK-INTGRN**) as the primary measure of work dispersion, while accounting for proportion of work **(WRK-SHR)** accomplished at individual development centers as an aggregate control measure.

Defect Prevention (PREV): We calculate the investments in defect prevention activities as the percentage of total development effort spent on training, planning and configuration management activities.

Quality Appraisal (APPRSL): We calculate the investments in appraisal-based management activities as the percentage of total development effort spent on peer reviews of requirement, design, status reviews and code inspection.

Failure (FAILURE): We calculate the investments in failure-based management activities as the percentage of total development effort spent on unit tests, module integration tests, system tests, and the resulting rework for correcting errors.

Team Size (TEAM-SIZE): Team size is the head count of number of persons involved in the project.

Project duration (MONTHS): Project duration is measured in terms of the calendar months elapsed since the start of the project to delivery and signoff of project.

Reuse (REUSE): Reuse in this study is measured as the percentage of lines of code that have been utilized from the generic code libraries maintained centrally in the knowledge database at our research site. All reused modules and objects in the projects we studied were maintained with unique tags for readability and hence could be easily identified.

Code Coverage (CODCOV): Code coverage is defined as the percentage of code that has been tested or covered before implementation in the production environment.

Summary statistics for these variables are presented in table 2. Correlations among variables are shown in table 3.

---Insert Tables 2 and 3 about here---

3.2 Empirical Models

Based on our discussion in section 2, our empirical models of productivity and quality are shown in equations (1) and (2).

Development productivity = f {conformance quality, work dispersion, defect prevention, task appraisal, failure costs, reuse, team size, project duration} *.....(Eq. 1)*

Conformance quality = f {development productivity, work dispersion, defect prevention, task appraisal, failure costs, code coverage, code size, team size, project duration}

..... (Eq. 2)

Past research in software development cost and quality has shown that the effects of size, effort, cycle time, defect density on quality and productivity are not linear and scale economies exist in software development (Banker and Kemerer 1989). We posit that the effect of dispersion and quality management practices on both conformance quality and development productivity are not linear and use a generic multiplicative specification. These empirical models are given in equations (3) and (4):

$$ln(development\ productivity) = \beta_0 + \beta_2 * ln(conformance\ quality) + \beta_3 * ln(work\ dispersion) + \beta_4 * ln(defect\ prevention) + \beta_5 * ln(task\ appraisal) + \beta_6 * ln(failure) + \beta_7 * ln(reuse) + \beta_8 * ln(team\ size) + \beta_9 * ln(project\ duration) + \varepsilon_1 \quad \text{..... (Eq. 3)}$$

$$ln(conformance\ quality) = \beta_0 + \beta_1 * ln(development\ productivity) + \beta_3 * ln(work\ dispersion) + \beta_4 * ln(defect\ prevention) + \beta_5 * ln(task\ appraisal) + \beta_6 * ln(failure) + \beta_8 * ln(team\ size) + \beta_9 * ln(project\ duration) + \beta_{10} * ln(Code\ size) + \beta_{11} * ln(Code\ coverage) + \varepsilon_1 \quad \text{..... (Eq. 4)}$$

3.3 Research site and Data collection

We collected detailed distributed development data from a leading software service company that employs over 19,000 people in 17 countries world-wide and has annual revenue of about a billion dollars. This firm provides an ideal research setting to study software productivity and quality in a distributed development scenario because the firm has adopted a global delivery model for its services and employs high maturity processes. Our research site was independently assessed and certified to operate at the highest maturity level (level 5) of the software Capability Maturity Model instituted by the Software Engineering Institute at Carnegie Mellon University. High maturity operations at our research site help us to gather reliable and fine-grained data to test our research hypotheses.

Our data collection involved gathering information on forty-two completed projects in the time period of August 2001 until September 2003. We retrieved data on project allocation between remote development locations, development effort, effort spent on project management, defect data and data on investments in individual quality practices followed in the projects from a central process database maintained by the quality division of the organization. We also conducted interviews with the two senior business development managers responsible for the projects and with ten randomly[1] picked team members involved in the projects. All the quantitative data we gathered had been previously audited by an independent assessment auditor for CMM and ISO 9000 conformance certifications.

All the projects we studied involved development of commercial business applications and involved development in high level programming languages of C, C++ , COBOL and also involved the usage of relational databases DB2 and Oracle. Although individual project managers had a high level of autonomy in deploying resources and following project specific processes, they were strictly bound by company-wide regulations and quality service contracts with customers regarding key processes followed at CMM level-5 maturity. Hence in our study we found no considerable variance with respect to adherence to key process areas of the CMM models among different projects.

All the forty-two projects that we studied were executed using two software development centers located in India and the United States. The development center in India was the primary development center for all the projects and employed more head

[1] A spread sheet random number generator was used to pick ten employees from a pool of hundred available employees based on their employee id numbers. Employee id numbers were obtained from the human resources department based on project identification codes.

count than the secondary development center in United States. To assess if there were significant differences in human resource practices among the different units we analyzed employee performance appraisal templates, code of conduct guidelines and incentive structures from different units. We found that the firm had a uniform human resource policy through out the world and there were no significant differences across project units. Also our research site was recently assessed for maturity in human resources management practices and was certified at level 5 as per People Capability Maturity Model standards (PCMM).

4. Estimation, Results & Discussion

Our theory and empirical models indicate the presence of endogeneity between conformance quality and development productivity. We tested for the same in our dataset using Hausman's endogeneity test and could reject the null hypothesis that regressors are exogenous at 1% level. Because of the presence of endogenous variables in our models, OLS estimates are biased and inconsistent. Thus, we use instrumental variable regression (2SLS) method to estimate the parameters of equations (3) and (4). We tested for identification problems in both of our models using Sargan's statistic (Sargan 1958) and did not identify any problems. The results of this estimation are presented in table 4.

---Insert Table 4 about here---

Our first set of hypotheses predicted a negative association between dispersion in software development with development productivity and conformance quality. We find support for this in our analysis. We find that integration activities that result from distribution of work across geographically distributed centers are negatively associated with both development productivity and conformance quality. Specifically we find that,

1% increase in integration activities resulting from work dispersion decreases development productivity by about 0.2% (refer to table 4 model 1, β_3=-0.196, p=0.004) and decreases conformance quality by about 0.4% (refer to table 4 model 2, β_3=-0.404, p=0.013).

These results suggest a careful consideration of work allocation and resource deployment across development locations. Software organizations adopt distributed development with a strategic intent to leverage talent across the globe, to have a better connection with the customer or to have a 'follow the sun' twenty-four hour work force. However, such benefits of a distributed work force come with the challenges of managing interdependent tasks that are predominant in software development. Our results show that even at high process maturity, distributing interdependent software development tasks decreases project productivity and quality. Software organizations should consider these while they assess the cost-benefits of work dispersion.

Our next sets of results pertain to the effects of quality management practices in the distributed development context. We had hypothesized that prevention and appraisal-based approaches aid improvement in distributed development productivity and quality. Our analysis shows that one unit investment in prevention-based quality management practices improves project productivity by about 2% (refer to table 4, model 1 β_4=0.177, p=0.075). We do not find significant results for prevention-based quality management practices in the quality model. Results for quality-based approaches show the same pattern: a 1% increase in investment in appraisal-based quality management approach improves development productivity by about 0.4% (refer to table 4, model 1 β_5=0.404,

p=0.006). However, we do not see a statistically significant association between appraisal-based quality management schemes and conformance quality in our data set.

One reason why we may not see quality improvement benefits through preventive investments could be because these investments aim at overall understanding of technology paradigm and development processes, rather than addressing project specific issues. However, appraisal activities emphasizing "learning-before-doing" activities aim to develop such project-specific knowledge. In order to analyze why such knowledge gained is effective in improving productivity, but not quality, we need a fine grained analysis of appraisal activities with different software artifacts. While such an analysis is beyond the scope our data set, studies in the software inspection literature offer interesting insights (Fagan 1986, Porter, et al. 1997).

With the advent of sophisticated code testing tools practitioners tend to discount the importance of appraisal-based quality management tools (Porter, et al. 1997). But positive and significant association between appraisal activities and dispersed development productivity in our results highlight the importance of investments in an appraisal-based quality management approach. Appraisals at different stages of software development before proceeding to the subsequent stages helps good dissemination of otherwise partitioned knowledge in a distributed environment. Also, appraisal-based quality management practices provide useful 'learning-before-doing' experience and early indications of any performance lapses and contribute to significant improvement in project performance.

The results for failure-based quality management show mixed support for our hypothesis. Results indicate that a unit increase of investments in failure-based quality

management practices yields about 0.7% increase in conformance quality (refer to table 4, model 2 β_6=0.746, p =0.072) but is insignificant in the productivity model. While quality results for failure-based approaches are as expected, productivity results are not. This may offer some clues about potential positive effects of failure-based approaches on improving productivity. We discussed this result with mangers at our research site and a senior certified quality associate offered this insight:

"…..getting it right the first time with the aid of training and monitoring is no doubt cost effective and productive. But, when complexity exceeds certain limits and when our knowledge is uncertain it helps to go for quick builds, test and improve on subsequent builds. Building iteratively and repeated testing for conformance reduces uncertainty, gives immediate feedback and a consistent picture of the entire system to resources spread in remote locations…." [Interview Transcript, Quality Officer, November 5, 2003]

Thus, building software applications iteratively with judicious investments in failure-based quality management helps distributed teams to develop common knowledge about interdependencies and problems at interfaces. However, we should note that to realize the benefits of 'learning-by-doing', a complementary high maturity processes, such as diligent error tracking and follow up, documentation and change management are essential. This result also reveals a new perspective for the choice of a distributed development model. While the waterfall model of development is still widely prevalent, our result indicates that other rapid development schemes such as extreme programming and prototyping that emphasize iterative development could improve development productivity, though in the background of an established high maturity basic development processes.

Comparing the effects of quality management and failure-based management approaches is useful to infer the relative efficiencies each of these approaches. Our results

in this study show that appraisal-based quality management practices are significant in improving productivity whereas failure-based quality management practices are significant in improving conformance quality. This indicates the presence of tradeoffs among investments in different quality practices. This result emphasizes that managers should invest in appropriate quality management practices according to the development context and project goals, rather than blindly follow all operational tasks proposed by generic quality frameworks. Fitzergald (2003) endorses this view and notes:

"….when (generic) methods are used, the problem of means-end inversion often arises; that is rather than focusing on the end (the development of software), developers become pre occupied with the means and lapse into blind adherence to low-level methods rather than focusing on higher-level principles that make sense in the development context" [p. 65]

To test the relative efficiencies of appraisal-based and failure-based quality practices, we performed several joint tests. While our tests indicated that appraisal-based and failure-based quality practices are jointly significant in impacting productivity and quality, we could not statistically infer on their relative efficiencies because we could not reject the null hypotheses (for example, H0: $\beta_5 - \beta_6 = 0$).

Referring to Table 4 models 3 and 4 we can see that our results are robust and consistent even after we control for the aggregate level of work dispersion in terms of the proportion of work accomplished at individual development centers. This indicates that sharing of work between development centers can be used to achieve strategic goals if firms manage to disperse work with minimal interdependent tasks. Since interdependent tasks entail cumbersome integration activities which affect both development productivity and quality negatively, it is important to find ways to minimize interdependencies between remote development centers.

Results for the control variables are as expected. Reuse improves productivity and, as software size increases, quality deteriorates. Team size and project duration show similar patterns and an increase in each of these factors increase conformance quality, but decrease development productivity.

We would like to acknowledge the limitations of this study that future researchers may address. First, we studied the effect of work dispersion and quality management approaches in the context of custom software development only. Second, we did not examine the impact of cultural differences that come into play in globally distributed teams. Exploration of effect of these cultural differences on coordination and software project performance will further enhance our understanding of the challenges involved in distributed software development. Third, while we analyzed the impact of quality investment decisions in a distributed development context, we have not analyzed other factors in the distributed software development context that influence such investment decisions. Exploring the organizational structures of distributed work and the context of managerial decisions could help researchers gain a more nuanced understanding of the processes needed for functioning of distributed teams. Finally, future research on distributed software development could explore the impact of software architectures on the ability to disperse work and achieve efficient division of labor with minimal interdependencies.

5. CONCLUSION

Distributed development is becoming pervasive in the software industry and firms are adopting global delivery business models. Several researchers have reported evidence

for communication and coordination difficulties and problems of achieving common ground in distributed teams that might impede personnel performance. In this context it is important to analyze if the successful quality management practices in co-located software development extend equally well to the distributed software development context.

In this study we explored the effect of work dispersion on software project performance by analyzing the projects operating in a high process maturity environment. We also analyzed the effectiveness of prevention, appraisal and failure-based quality management approaches in improving productivity and quality. Our results suggest that even in a very high process maturity environment, work dispersion has a negative effect on productivity and quality. We find that the primary cause for such an effect is due to the additional integration activities that happen between remote locations. This finding suggests the need to design work dispersion mechanisms that minimize interdependencies between remote locations. Also, we find that an appraisal-based quality management approach to be effective in improving distributed development productivity and a failure-based approach to be effective in improving conformance quality. This highlights the presence of inherent tradeoffs in quality investments, and calls for quality investments be guided by development context and project specific goals.

Four important managerial implications arise from our analysis. First, our results indicate that the effect of distributing interdependent tasks in software development may persist even when teams follow highly disciplined processes. Thus, distributed development managers should not discount the effect of dispersion while estimating potential performance and returns from their projects. Potential cost savings from

distributing work to offshore centers have to be weighed against possible loss of overall development productivity and the challenge of tackling increased number of defects. Secondly, to reap the benefits of popular quality models such as CMM, judicious tailoring of processes specific to the development context may be essential. Inherent tradeoffs exist between development productivity and conformance quality and investments in prevention, appraisal and failure-based quality schemes have differing effects on project performance. Thus the onus is on the development manager to allocate appropriate resources according to the goals of the project. Third, investments in failure-based quality approaches and iterative development might yield benefits in the distributed development scenario. This might explain the increasing interest among software practitioners towards modern development paradigms, such as rapid prototyping and extreme programming. However, it is important to stress the need for establishing high maturity basic development processes as a backbone on which such processes that emphasize learning-by-doing can be implemented. Fourth, work sharing mechanisms between remote locations have to be designed in a way to minimize interdependencies to achieve superior project performance.

Figure1. Research Model

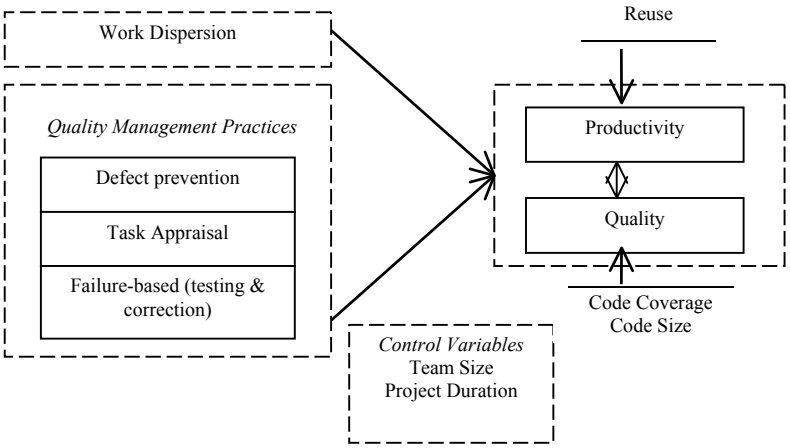

Table 2 Summary Statistics

Variable	Mean	Std. Dev.	Min	Max
PROD	0.26	0.15	0.02	0.64
QUALITY	0.28	0.35	0.01	1.00
PREV	4.57	4.56	1.00	25.67
APPRSL	2021.26	2342.79	102.00	11581.00
FAILURE	1573.41	1356.28	172.00	5533.00
WORKDISP	36.38	29.00	1.00	142.00
CODE-SIZE	2191.83	2927.72	33.00	18247.00
TEAM-SIZE	11.19	6.42	2.00	30.00
MONTHS	5.76	4.32	1.00	18.00
CODECOV	42.38	21.27	10.00	100.00
RESUE	5.26	3.25	1.00	20.00

Table 3 Correlation among Variables (log transformed)

	PROD	QUALITY	PREV	APPRSL	FAILR	WORKDISP	CODE-SIZE	TEAM-SIZE	MONTHS	CODECOV	REUSE
PROD	1										
QUALITY	-0.1403	1									
PREV	-0.0366	0.2473	1								
APPRSL	0.2319	-0.0641	-0.162	1							
FAILURE	0.008	0.2438	0.2499	0.0369	1						
WORKDISP	-0.2112	-0.1314	0.1486	0.1319	0.0864	1					
CODE-SIZE	0.6638	-0.5462	-0.0796	0.3307	-0.028	-0.1068	1				
TEAM-SIZE	0.0568	-0.3362	0.0496	0.1391	0.1469	-0.0641	0.6235	1			
MONTHS	-0.2764	-0.1993	0.0274	0.1034	-0.212	0.3792	0.0379	-0.1591	1		
CODECOV	-0.0559	0.1363	0.1034	-0.1665	0.5296	-0.1971	-0.1586	-0.1487	0.009	1	
REUSE	0.8085	-0.0233	0.0458	0.0903	0.0009	0.0205	0.4999	0.0226	-0.0567	-0.0469	1

Table 4. Estimation Results *(p values are in parentheses)*

Models→	Parameter	(1)	(2)	(3)	(4)
Dependent Variable→		Development Productivity	Conformance Quality	Development Productivity	Conformance Quality
Development Productivity	β_1	----	2.03*** (0.002)	----	2.02*** (0.002)
Conformance Quality	β_2	- 0.374*** (0.000)	----	- 0.382*** (0.000)	-----
Integration effort due to work dispersion	β_3	- 0.196*** (0.004)	- 0.404*** (0.013)	-0.152* (0.066)	-0.515** (0.018)
Aggregate Work Sharing (% onsite / total)	β_3	----	----	- 0.243 (0.332)	0.469 (0.449)
Defect Prevention	β_4	0.177* (0.075)	0.235 (0.259)	0.153 (0.134)	0.283 (0.183)
Task Appraisal	β5	0.404*** (0.006)	0.496 (0.147)	0.380** 0.011	0.522 (0.126)
Failure	β_6	0.2 (0.192)	0.746* (0.072)	0.256 (0.118)	0.744* (0.071)
Reuse	β_7	0.942*** (0.000)	--	0.942*** (0.000)	---
Team Size	β_8	- 0.443*** (0.004)	1.652** (0.022)	-0.429*** (0.005)	1.58** (0.025)
Project duration	β_9	- 0.344*** (0.002)	0.852** (0.031)	- 0.317*** (0.006)	0.824** (0.034)
Code Size	β_{10}	----	-2.36*** (0.000)	----	-2.36*** (0.00)
Code Coverage	β_{11}	----	- 0.463 (0.210)	----	-0.599 (0.142)
Constant	β_0	-3.962*** (0.000)	11.312** * (0.002)	-3.43*** (0.000)	10.56*** (0.003)
Observations		42	42	42	42
Centered R-squared		0.663	0.564	0.66	0.57
F		11.23*** (0.00)	4.79*** (0.000)	9.71*** (0.000)	4.24*** (0.000)

* significant at 10%; ** significant at 5%; *** significant at 1% (two-tailed tests)

References

Amoribieta, I., K. Bhaumik, K. Kanakamedala and A. D. Parkhe. 2001. Programmers abroad: A primer on offshore software development. *The McKinsey Quarterly.* 2001(2)
Banker, R. D., S. M. Datar and C. F. Kemerer. 1991. A model to evaluate variables impacting the productivity of software maintenance projects. *Management Science.* 37(1) 1-18.

Banker, R. D., G. B. Davis and S. A. Slaughter. 1998. Software development practices, software complexity, and software maintenance: A field study. *Management Science.* 44(4) 433-450.

Banker, R. D. and R. J. Kauffman. 1991. Reuse and productivity in integrated computer-aided software engineering: An Empirical Study. *MIS Quarterly.* 15(3) 375-401.

Banker, R. D. and C. F. Kemerer. 1989. Scale economies in new software development. *IEEE Transactions on software engineering.* 15(10) 1199-1205.

Banker, R. D. and S. A. Slaughter. 2000. The moderating effects of structure and volatility and complexity in software environment. *Information Systems Research.* 11(3) 219-240.

Battin, R. D., R. Crocker, J. Kreidler and K. Subramanian. 2001. Leveraging resources in global software development. *IEEE Software.* 18(2) 70-77.

Boehm, B. W. 1981. *Software engineering economics.* Prentice Hall, Upper Saddle River, NJ

Chen, M.-H., M. R. Lyu and E. Wong. 2001. Effect of code coverage on software reliability measurement. *IEEE Transactions on reliability.* 50(2) 165-170.

Cramton, C. D. 2001. The mutual knowledge problem and its consequences fro dispersed collaboration. *Organization Science.* 12(3) 346-371.

Crosby, P. B. 1979. *Quality is Free: The art of making quality certain.* McGraw-Hill, New York

Fagan, M. E. 1986. Advances in Inspections. *IEEE Transactions on software engineering.* SE-12(2) 744-751.

Faraj, S. and L. Sproull. 2000. Coordinating expertise in software development teams. *Management Science.* 46(12) 1554-1568.

Fine, C. H. 1986. Quality Improvement and Learning in productive systems. *Management Science.* 32(10) 1301-1315.

Fitzgerald, B., N. L.Russo and T. O'Kane. 2003. Software development method tailoring at Motorola. *Communications of the ACM.* 46(4) 65-70.

Frate, F. D., P. Garg, A. P. Mathur and A. Pasquini. 1995. On the correlation between code coverage and software reliability. *Proceedings of the Sixth International Symposium on Software Reliability Engineering.* 124-132.

Garvin, D. 1987. Competing on the eight dimensions of quality. *Harvard Business Review.* 65(6) 101-109.

Goodman, R. A. 1967. Ambiguous authority definition in project management. *The Academy of Management Journal.* 10(4) 395-407.

Harter, D. E., M. S. Krishnan and S. A. Slaughter. 2000. Effects of process maturity on quality, cycle time, and effort in software product development. *Management Science.* 46(4) 451-466.

Harter, D. E. and S. A. Slaughter. 2003. Quality Improvement and Infrastructure Activity Costs in Software Development: A Longitudinal Analysis. *Management Science.* 49(6) 784-800.

Hatch, N. W. and D. c. Mowery. 1998. Process Innovation and Learning by doing in seminconductor manufacturing. *Management Science.* 44(11) 1461-1477.

Herbsleb, J., D. Zubrow, D. Goldenson, W. Hayes and M. Paulk. 1997. Software quality and the capability maturity model. *Communications of the ACM.* 40(6) 30-40.

Herbsleb, J. D. and R. E. Grinter. 1999. Splitting the organization and integrating the code: Conway's law revisited. *Proceedings of the 21st International conference on software engineering.* Los Angeles, CA. 85-95.

Herbsleb, J. D., A. Mockus, T. A. Finholt and R. E. Grinter. 2001. An empirical study of global software development: distance and speed. *Proceedings of the 23rd international conference on Software engineering.* Toronto, Canada. 81-90.

Hippel, E. v. and M. J. Tyre. 1995. How learning by doing is done: problem identification in novel process equipment. *Research Policy.* 24(1) 1-12.

Iansiti, M. 1997. *Technology Integration: Making critical choices in a dynamic world.* Harvard Business School Press, Boston, Ma

IFPUG. 1999. *FP counting manual.* International Function Point Users group,
Jarvenpaa, S. L. and D. E. Leidner. 1999. Communication and Trust in Global Virtual Teams. *Organization Science.* 10(6) 791-815.

Koushik, M. V. and V. S. Mookerjie. 1995. Modeling coordination in software construction: An analytical approach. *Information Systems Research.* 6(3) 220-254.

Kraut, R. E., C. Egido and J. R. Galegher. 1990. Patterns of contact and communication in scientific research collaborations. In *Intellectual Teamwork: Social and technological foundations of cooperative work.* C. Egido (Ed.). Lawrence Erlbaum Assoc. Hillsdale, New Jersey. 149-172.

Krishnan, M. S., C. H. Kriebel, S. Kekre and T. Mukhopadhyay. 2000. An empirical analysis of productivity and quality in software products. *Management Science.* 46(6) 745-759.

Lipnack, J. and J. Stamps. 1997. *Virtual Teams: Reaching across space, time and organizations with technology.* John Wiley & Sons, New York, NY

Lundvall, D. M. and J. M. Juran. 1974. Quality Costs. In *Quality control handbook.* J. M. Juran (Ed.). McGraw-hill. San Francisco, CA.
MacCormack, A., C. F. Kemerer, M. Cusumano and B. Crandall. 2003. Trade-offs between productivity and quality in selecting software development practices. *IEEE Software.* 20(5) 78-85.

March, J. G. and H. A. Simon. 1958. *Organizations.* John Wiley & Sons, New York

Maznevski, M. L. and K. M. Chudoba. 2000. Bridging Space over time: global virtual team dynamic and effectiveness. *Organization Science.* 11(5) 473-492.

McCann, J. E. and J. R. Galbraith. 1981. Interdepartmental relations. In *Handbook of organization design.* W. Starbuck (Ed.). 60-84. Oxford University Press. New Jersey.
Mendez, D. and R. Narasimhan. 2002. Examining market oriented aspects of cost of quality. *IEEE Transactions on engineering management.* 49(2) 131-139.

Mili, H., F. Mili and A. Mili. 1995. Reusing software: Issues and research directions. *IEEE Transactions on software engineering.* 21(6) 528-562.

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

Nandakumar, P., S. M. Datar and R. Akella. 1993. Models for measuring and accounting for cost of conformance quality. *Management Science.* 39(1) 1-16.

Nasscom-McKinsey. 2002. NASSCOM-McKinsey Report 2002. National Association of Software and Service Companies, New Delhi

Nidumolu, S. 1995. The effect of coordination and uncertainty on software project performance: Residual performance risk as an intervening variable. *Information Systems Research.* 6(3) 191-219.

Olson, G. M. and J. S. Olson. 2000. Distance Matters. *Human-computer interaction.* 15(2) 139-178.

Parasuraman, S. and J. A. Alutto. 1981. An examination of the organizational antecedents of stressors at work. *The Academy of Management Journal.* 24(1) 48-67.

Paulk, M. C. 2003. List of Maturity Level 4 and 5 Organizations, http://www.sei.cmu.edu/publications/articles/paulk/high.mat.orgs.html. Software Engineering Institute, Pittsburgh

Pisano, G. P. 1994. Knowledge, integration and the locus of learning: An empirical analysis of process development. *Strategic Management Journal.* 15(Special Issue: Competitive Organizational Behavior) 85-100.

Porter, A. A., H. P. Siy, C. A. Toman and L. G. Votta. 1997. An experiment to assess the cost-benfits of code inspections in large scale software development. *IEEE Transactions on software engineering.* 23(6) 329-346.

Sargan, J. D. 1958. The estimation of economic relationships using instrumental variables. *Econometrica.* 26(393-415.

Shea, G. P. and R. A. Guzzo. 1989. Groups as human resources. In *Research in personnel and human resources management.* K. M. Rowlands (Ed.). JAI Press. Greenwich, CT. 323-356.

Smith, R. K., J. E. Hale and A. S. Parrish. 2001. An empirical study using task assignment patterns to improve the accuracy of software effort estimation. *IEEE Transactions on software engineering.* 27(3) 264-271.

Sproull, L. and S. Kiesler. 1986. Reducing social context cues: electronic mail in organizational communication. *Management Science.* 32(11) 1492-1512.

Subramanyam, R. and M. S. Krishnan. 2003. Empirical analysis of CK metrics for object-oriented design complexity: implication for software defects. *IEEE Transactions on software engineering.* 29(4) 297-310.

Takeishi, A. 2002. Knowledge partitioning in the interfirm division of labor: The case of automotive product development. *Organization Science.* 13(3) 321-338.

Teasley, S. D., L. A. Covi, M. S. Krishnan and J. S. Olson. 2002. Rapid software development through team collocation. *IEEE Transactions on software engineering.* 28(7) 671-683.

The Economist. 2003. Relocating the back office. *The Economist (13 December 2003)* 67-69.

Thomson, J. D. 1967. *Organizations in Action.* McGraw-Hill, New York

Victor, B. 1990. Coordinating work in complex organizations. *Journal of organizational behavior.* 11(3) 187-199.

Walton, M. and W. E. Deming. 1988. *The Deming Management Method.* Perigee, New York

Zantek, P. F., G. P. Wright and R. D. Plante. 2002. Process and product improvement in manufacturing systems with correlated stages. *Management Science.* 48(5) 591-606.