Geographically Distributed Development: Trends, Challenges and Best Practices

By

Yuhong Yin

Master of Computer Science (2000)
Northeastern University, Boston, MA

Submitted to the System Design & Management Program
in Partial Fulfillment of the Requirements for the Degree of

Master of Science in Engineering & Management

at the

Massachusetts Institute of Technology, Cambridge, MA, 02139

[February 2007]
January 2007

Author .....
Yuhong Yin
System Design and Management Program
January, 2007

Certified by.....
Patrick Hale
Thesis Supervisor
Director, System Design& Management Program

This page was intentionally left blank

Geographically Distributed Development: Trends, Challenges and Best Practices

by
Yuhong Yin

SUBMITTED TO THE SYSTEM DESIGN & MANAGEMENT PROGRAM
ON JANUARY 17, 2007, IN PARTIAL FULFILLMENT OF THE
REQUIREMENTS FOR THE DEGREE OF
MASTER OF SCIENCE IN ENGINEERING & MANAGEMENT

Abstract

Economic and market forces as well as technological progress emerging throughout the last decade signifies the Geographically Distributed Development (GDD) or Global Software Development (GSD) as a software industry norm or necessity that is receiving considerable interest from companies all over the world.

This thesis clarifies the terms used in distributed development practice, studies the status, key business drivers and major challenges of GDD or GSD, and then proposes a revised six-force framework to leverage the capabilities of geographically distributed development model. The proposed framework covers both the strategic and tactic aspects of investing and managing a global project team, including Strategic Vision and Management Skills, Organizational Structure and Team Building, Collaborative Technologies, Development Methodology and Software Life-Cycle Management in planning and managing distributed project and leveraging global teams.

This thesis uses a case study to validate the challenges and concerns identified and to conclude that many problems in geographically distributed development can be overcome to enable companies to systematically harness the potential of this development model.

Thesis Advisory: Pat Hale
Title: Director, SDM Fellows Program
Senior Lecturer, LFM-SDM Programs

Table of Contents

List of Figures

List of Tables

Acknowledgements

First, I would like to thank IBM[1] for sponsoring me through the MIT System Design and Management (SDM[2]) program and affording me this opportunity to learn and grow. A special thanks to my fellow IBM Rational co-workers who continuously contribute to the technology, infrastructure, toolsets and best practices in the Geographically Distributed Development (GDD) field. I would like to thank my managers, Beth Benoit, Bob Ryder, Will Goddin and James Felty, for their support; my mentors, Carol Yutkowitz, Shirley Hui, and Steve Pitschke, for their guidance and encouragement; the Rational Green Threat team, for allowing me to participate in their study and analysis of GDD; and Rolf Nelson, who talked to me and provided material on the trend of GDD.

I am grateful to the MIT SDM Program for providing me a fantastic and extremely rewarding learning opportunity that I will always cherish. My thanks go out to the SDM staff, professors and lecturers, my classmates, all of whom I have learned from, laughed with, and admire. Through out the thesis, I have integrated my notes from various SDM core courses and electives, which turned out to be very valuable.

I would like to thank Pat Hale[3], my thesis advisor, for support, for direction and for sharing his own experience with geographically distributed development at Otis Elevator Company, where he developed and implemented Otis' first systems engineering process.

I would like to thank Professor Michael Cusumano[4] for giving a series of great lectures on the business of software and for identifying global software development as a one of the term paper topics. Special thanks to my 15.358 (The Software Business) classmates, Tanya Flores, Noritaka Kawashima, Janice Maloney and Timea Pal. Together, we have done research and analysis on global software development. Our course paper provided some of the materials in my thesis.

---

[1] http://www.ibm.com
[2] http://sdm.mit.edu

[3] http://sdm.mit.edu/?fileName=news_articles/pat_hale/pat_hale.html
[4] http://web.mit.edu/cusumano/www/

My thanks also go out to Lei Li, my dear friend and co-worker, who spared time proofreading my thesis.

I'm especially indebted to my husband, Joey, for his encouragement, support and patience, to my parents, Zhaolun and Shuqing, for introducing me to the field of engineering, and to my daughter, Helen, for her love and accepting the need for Mommy to study.

Note on Proprietary Information

In order to protect proprietary and competitive IBM Rational Software information, the actual project name, release name and numbers used in case study in this thesis are either removed or do not represent actual ones.

Research Methods

The main research methods used in the thesis include literature study, empirical analysis and case studies.

Thesis Contents

My goal in this thesis is to provide an overview of the geographically distributed development, to study the business drivers, challenges, trends, and to provide best practices in managing and leveraging geographically distributed development. This thesis covers both the business and technological aspects relevant to geographically distributed development and its increasing popularity within the industry. The focus, however, is on the technological and operational aspects.

Chapter 1, introduction, starts out by clarifying the definition of geographically distributed development, analyzing open source software as a case of GDD, and describing the overall status of this practice.

Chapter 2, business drivers, focuses on the question on why geographically distributed development becomes an industry norm. Friedman's ten-flattener theory, Carmel's four-factor theory, and some other findings are discussed.

Chapter 3, challenges, probes the challenges and issues of geographically distributed development, which include strategic, cultural, communication and coordination challenge, as well as technical, knowledge management and organizational challenges.

As managers, architects and project leads, understanding the business drivers, challenges and trends of geographically distributed development is prerequisite to engaging in the global development environment.

Chapter 4, best practice, proposes a revised five-force framework to examine how to effectively manage a geographically distributed project and global teams based on

Carmel's framework [1], my research study and my own industry experience. The revised five forces include Strategic Vision and Management Skills, Organizational Structure and Team Building, Collaborative Technologies, Development Methodology and Software Life-Cycle Management.

Chapter 5, case study, of a distributed development by IBM Rational, represents a typical organization-led practice, validating the business drivers and challenges identified in chapter 2 and 3, and best practices proposed in chapter 4.

Chapter 6, conclusions and recommendations, offers final thoughts on the future geographically distributed development and how to capture the full potential and values associated with this practice.

1. Introduction

In today's global economy, many companies are shifting their development strategy to one that is geographically distributed. This thesis studies the main factors and drivers that determine companies to engage in a distributed development model. With this shift, however, many companies have experienced a high degree of project development failure due to several challenges magnified by a distributed staff. The goal of this thesis is to identify the challenges in a distributed development model and to propose a five-force framework that combines the strategic and tactic aspects of managing a distributed development.

The author believes that through best practices in strategies, technologies and operations, it is possible to leverage the full potential and capabilities of distributed software development while maintaining a unified team approach.

1.1 The Definitions

In literature and industry practice, there are three terms, Geographically Distributed Development (GDD), Geographically Dispersed Development (GDD) and Global Software Development (GSD), which are often used to refer to the same distributed development model.

IBM defines geographically distributed development (GDD) as the practice of managing software development projects beyond the traditional bounds of a single building or office structure where the development staff is singularly located. In a GDD model, the development staffing may be distributed across town, across a state or provincial border, or overseas. Some companies are engaged in distributed development within their geographically distributed corporate boundaries, which may involve multiple sites in a single software development project, as well as service suppliers and outsource companies that assume part of the software development lifecycle [2].

Mohagheghi (2004) defines Global Software Development (GSD) as the mode of developing software that allows software projects or products or systems to pass through the boundaries of companies and countries, by coordinating project teams distributed across different geographical regions and time zones. Sahay (2003) also claims that GSD can include work done across global borders through outsourcing, alliances, or subsidiary arrangements.

Despite some of the subtle differences in these definitions of GDD and GSD, they all emphasize the facts that development teams are not co-located and thus there are distance, regions and time zone challenges, and that the development may involve outsource providers and across corporate or country borders and thus there are cultural, political and security concerns.

While geographically distributed (or dispersed) development (GDD) seems to focus more on distance and time zone, and global software development (GSD) implies more on cultural difference and political conflict, the author thinks all the aspects of GDD and GSD should be taken into account. In this thesis, these terms are used interchangeably.

1.2 Open Source Software as a Special Case

Open Source Software (OSS) is software in source-code form that is often created and maintained by a collaborative, virtual community and is usually downloadable and may be used, copied, and distributed with or without modifications for free or at a nominal cost. If the end-user makes any alterations to the software, he can either choose to keep those changes private or return them to the community so that the changes can potentially be added to future releases. An open source license[5] is certified by the Open Source Initiative (OSI), a nonprofit, unincorporated research and educational association with the mission to own and defend the open source trademark and advance the cause of OSS, and is dedicated to promoting and maintaining a formal definition of OSS called the "Open Source

[5] Most of the popular OSS licenses (e.g. GPL, LGPL, CPL, MPL, Apache, BSD, X-11) have been certified "OSD-compliant" by the OSI and are listed on their website: http://www.opensource.org/licenses/.

Definition" (OSD). The open source community consists of individuals or groups of individuals who contribute to a particular open source product or technology. The open source process refers to the approach for developing and maintaining open source products and technologies, including software, computers, devices, technical formats, and computer languages.

Open source software has, over the last 10 years or so, become a hot topic in the press and in the software industry, even though it has actually been in existence since the 1960s and has shown a successful track record to-date. Examples of popular open source products include Emacs, GNU toolset, Apache, and Linux. In 2006, Gartner[6] starts to include open source vendors in the software market and segment analysis.

Before Internet use became affordable, early open source software communities were rather geographically compact. Starting in late 80s and early 90s, many open source software projects rely on the use of a globally free workforce via the cheap of Internet and start with the premise that their contributors are rarely co-located. As a consequence, open source software projects are cases of geographically distributed software development.

Throughout the paper, the author uses open source software's GDD practice as a reference to compare to, and as potential force that is reshaping, the company-led, close source software's geographically distributed development which is the focus of this thesis.

1.3 The Status

Broader economic and political forces as well as technological progress emerging throughout the last decades allow for software development to increasingly become a global undertaking. Traditionally, software development used to constitute an activity conducted by white-collar engineers collocated in large companies. Co-location represented the norm in the software industry as it was practiced by Microsoft as well. Development activities distributed to multiple sites, carried out at a lower scale by IBM,

[6] http://www.gartner.com/

one of the early adventurers of global software development, represented an exception (Carmel, 1999) [1].

Increased communication capabilities, computer networks, and increasing language compatibility brought about a challenge to established models of conducting software development and testing in one location. Software companies are currently following a distributed development model with development offices around the world, and with development as well as testing happening round the clock.

Location sites are presumably determined primarily by the availability of well-trained software professionals and the presence of other sophisticated IT firms [1]. Instead of allowing for labor to come to industrialized nations, organizations are chasing scarce labor. The "excess" software labor is likely to stay put where is currently located. The main factors identified in the literature to drive the transition of software development from its traditional (centralized, co-located) form to a collaborative practice that spans the globe, are numerous and include a wide range of economic and organizational aspects.

As believed by T. Friedman in his 2005 book, *The World Is Flat: A Brief History of the Twenty-first Century*, the world is flat in the sense that the competitive playing fields between industrial and emerging market countries are leveling [3]. Today, large IT companies such as Microsoft, SAP, IBM, Hewlett-Packard and others have come to set up development offices in the United States, Europe, Israel, India, Mexico, Brazil, Russia; global consulting services are increasingly relying on software development offices set up in China, etc (Sikka, 2005). According to an estimation conducted by NASSCOM in 2000, approximately 40% of the Fortune 500 companies used global software development (Mohagheghi, 2004). Some additional indicators cited to estimate the magnitude of the global software development include: 80% of the Irish software industry's output is exported (Cochran, 2001); Gartner Dataquest has projected that IT outsourcing will reach $159 billion by 2005 (Laplante, 2004); 87,000 open source projects are hosted at SourceForge.net on Sept. 2005. The USA has been the most significant user of global

software development, with a predicted increase in spending from $5.5 billion in 2000 to a rather optimistic $17.6 billion by 2005 (Sahay, 2003).

1.4 Summary

Geographically distributed development refers to a development model of which development teams are not co-located and thus there are distance, regions and time zone challenges, and that the development may involve outsource providers and cross cooperate or country borders and thus there are cultural, political and security concerns. Geographically distributed development can be categorized into open source software practice and company-led, close source software practice. The thesis is interested in the latter.

The thesis is focused on studying the factors that drive companies to engage in the distributed development, identify challenges in managing a global team and propose best practices to leveraging the potential of this development model.

2. Business Drivers

The norm of geographically distributed development deserves a study in the business factors and forces that drive companies and development communities to engage in geographically distributed development.

2.1 Friedman's Ten Flatteners

Friedman recognized ten forces as the major flatteners of the worlds to support his theory of why the world is getting flat. The ten forces [3] include:

- The fall of Berlin Wall on November 9, 1989, which signals that the world was left with one system; and the ship of Window 3.0 on May 22, 1989, which allows people across the world to communicate with a unified and friendly computer platform. Software also grows to become a separate business.
- When Netscape went public on August 9, 1995 with a killer application, the web browser, which connects the world together via Internet and starts the waves of building new telecommunication infrastructure enabling the global software development.
- Work Flow Software and web-based standards, such as XML and SOAP, drive software applications to be interoperable. They have become the underlying infrastructure for global software collaboration.
- Open Sourcing, as a notion that company or ad hoc groups are making their source code of the program available to download for any one in the public, is emerging as a competitive advantage of attracting global talents and achieving innovation.
- Outsourcing, which entered the business lexicon in the 1980s[7], originally often refers to the delegation of non-core operations from internal production to an external entity specializing in the management of that operation. "Outsourcing" involves transferring or sharing management control and/or decision-making of a business function to an outside supplier, which involves a degree of two-way information exchange, coordination and trust between the outsourcer and its client. Outsourcing plays an

[7] http://en.wikipedia.org/wiki/Outsourcing

important role in the GDD or GSD development mode and represents unique challenges in managing the project and project teams.

- Offshoring[8], describes the relocation of business processes from one country to another, which includes any business process such as software, service, production, or manufacturing. Offshoring adds dynamics to GDD or GSD practice and often requires that the development mode be adaptive.
- Supply Chaining, as "a method of collaborating horizontally – among suppliers, retailers, and customers" [3], in GDD or GSD context, means that the distributed or global teams form the supply chain in delivering software and services.
- Insourcing, which often refers to the opposite of outsourcing, and should be considered together with outsourcing and offshoring as varieties or dynamic in GDD or GSD practice.
- In – Forming, Google, Yahoo!, MSN Web Search. All these search engines unfold the world's information to one's finger and further empowers individuals.
- The Steroids, digital, mobile, personal and virtual, give a boost to all the other flattener forces and allow them to perform at an even higher level. Eventually all software development will become "virtual" in the sense that mobility is the core of a development team.

2.2 Carmel's Four Factors

The existing literature and studies on globally distributed software development identify a long list of factors presumed to propel this phenomenon with considerable disagreement regarding their relative importance. Carmel groups these factors into four major categories labeled as catalyst factors, sustaining factors, size factors, and vision factors. The mentioning of these factors is essential to set up the context for a discussion on the relative importance of cost considerations.

The catalyst category refers to factors identified by executive managers as the main drivers for launching the practice of business software development teams in the first place. This category includes interest in accessing specialized "programming" talents, reduction in

[8] http://en.wikipedia.org/wiki/Offshoring

development costs, recent acquisitions, strategic interest in global presence, reduction in time-to-market, proximity to customer, etc.

Sustaining factors in Carmel's language refer to the issues that emerged once the catalyst factors spurred the initial move towards software development. They include the tendency towards greater formalism and documentation, spurred creativity and inspiration to the acquired diversity of the global team, distance from the distractions associated with headquarters, gained experience, and even a certain organizational inertia once a new set of executives have established themselves at global software companies.

With the increasing consolidation of the software industry, size factors are becoming increasingly important to consider, with their associated advantages as well as disadvantageous effects on coordination and efficiency. With the size of development teams increasing in the process of industrial concentration, co-located development teams often times become too large and difficult to manage. Under these circumstances, companies increasingly consider the relative benefits of opening up research and development centers at different locations across the globe. Executives also mention the benefits of the synergistic effects of having a larger number of teams networked together as an important factor sustaining the practice of software developments. Remote centers can contribute to increased localized knowledge and provide adequate opportunities for new, unexpected collaborative possibilities to emerge.

Finally, the vision factors refer to the conceptual understandings of executives of the organizational future of the businesses environment in general, and of IT companies in particular. Several executives thus cite their experimentations to accomplish leading organizational visions: transparency and virtuality as important drivers towards global software development.

2.3 Other Theories and Findings

In their 1998 study, Carmel and Tjia identify the scarcity of qualified software professionals as the primary driver of the move towards globally distributed development

teams. The acquisitions made by software companies located in industrialized countries as a result of the spectacular appreciations of their stocks are of secondary importance. The wage differentials existing across the globe and cost-considerations are of only tertiary importance. This is followed by the interest of companies in global presence and reduction in time-to-market.

These findings are based on an empirical survey conducted recently before 1999 with a sample of software companies from the United States. More recent developments emerging within the software industry, and domains relevant to it, seem to indicate a shift in the relative importance of these factors with important implications over future business decisions.

As the software industry has become more mature, so have educational and training programs aimed at preparing next generations of programmers. Hence, global shortages in qualified labor supply have become less of a concern for existing business. In parallel, the increasing trend towards standardization of software development practices and tools also play an important role in further removing the availability of highly qualified workforce as a major constraint on software companies. Finally, a recent look at the types of projects and activities being distributed to geographically distant teams also suggest the relatively more significant reliance of routinized activities that require specialized programming talent to a lesser degree.

While several studies mention the potential advantages of follow-the-sun and round-the-clock development, these practices require considerable coordination efforts and have been increasingly recognized as a drawback, rather than advantage of global software development. Extensive empirical analysis conducted by Herbsleb and Mockus thus indicate that distributed development projects appear to take about two and one-halftimes as long to complete as comparable projects where all the work is co-located (Herbsleb, Mockus, 2003).

With the relative decline in importance of the interest in acquiring specialized labor and the increasing recognition of the delay-mechanism inherent to globally-distributed development projects, cost considerations appear to become increasingly significant in driving the phenomenon, along with the strategic interest of companies to acquire a global presence.

As the prices of software products are declining and existing software companies are increasingly required to shift more towards a hybrid or software as service business model, one could argue that local presence is also becoming a strategic interest for companies in order to ensure the high quality of the services provided to customers located mainly in industrialized countries.

This is consistent with Carmel and Tjia's more recent findings presented in their 2005 study where they argue that decisions to set up globally distributed software development teams are led almost exclusively by cost considerations. With the price of software products declining, they claim that shifting development work to geographically remote locations has in fact become a strategic necessity for software companies to overcome financial constraints. This is becoming increasingly recognized both by managers as well as venture capitalists in the US and Europe. They argue that managers present alternative reasons only as part of a public transcript, intended to make the strategy more politically acceptable.

Finally, they claim that only in later stages of engaging in global software development either via subsidiaries or outsourcing - that only a few, most important companies reach (Carmel, 2005), can businesses benefit from more strategic advantages like increasing speed, accessing new pools of talent, extending knowledge networks, diversifying technologies, deepening localization, generating new revenues.

Figure 2-1 shows some of the important drivers for companies to engage in geographically distributed development using global workforce.

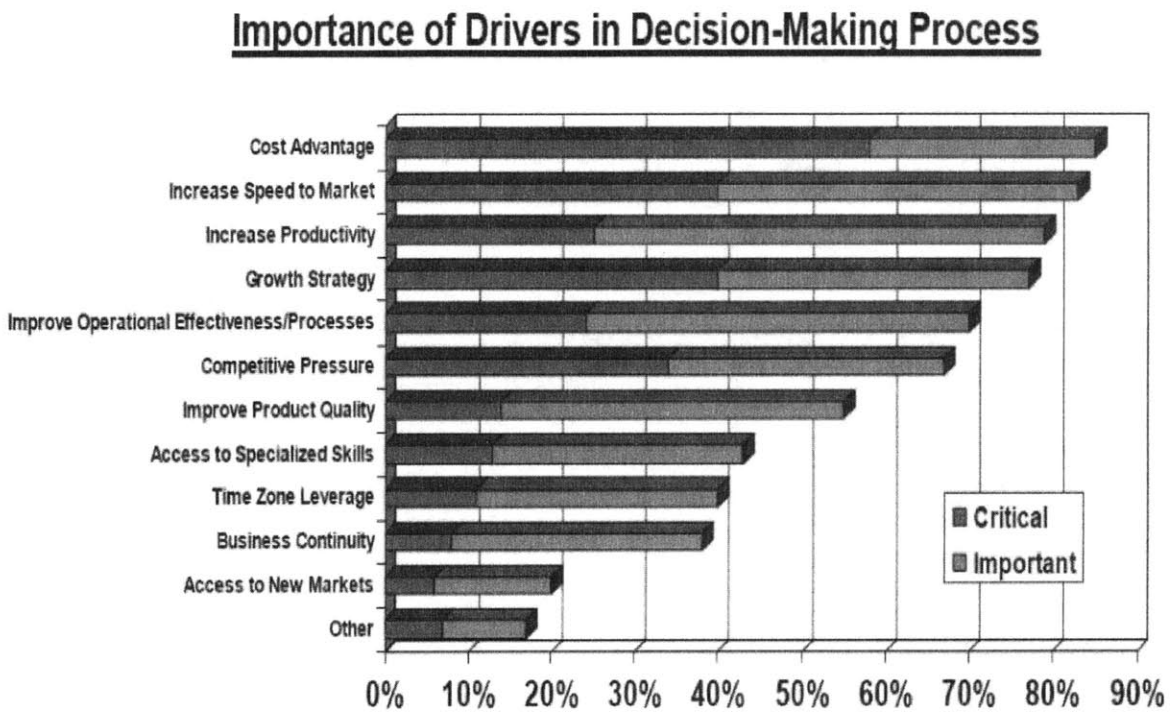

Figure 2-1 Importance of Drivers in Decision-Making Process [9]

One thing worth pointing out is that while tactical and strategic economic factors clearly predominate, setting up globally distributed teams sometimes becomes part of a political transaction. This is the case especially for large, global IT companies who want to gain access to the domestic markets of important developing countries. Russia and China are the classic examples in this respect who are interested in economic growth as well as knowledge externalities that can spur from software development teams located on their territories. Foreign IT firms who want to access the Chinese market are thus often required to invest in software R&D locally (Carmel, Tjia, 2005).

2.4 Summary

Friedman's ten flatteners, Carmel's four factors and the software industry trends analysis all lead to the two thrusts to Software Globalization: the spread of development activities to newly industrialized developing countries, and the transition of software development away from traditional (centralized, co-located) processes to a form where teams collaborate

[9] MIT Enterprise Forum presentation

across national borders – futuristic organizational model made possible by new developments in collaborative technology.

The key business drivers for GDD can be summarized as:

- Manage operational cost in perspective of current organization
  Mergers and acquisitions have yielded development organizations partitioned along the boundaries of the acquired companies.
- Improve competitive advantage by organizational re-focus and cost reductions
  Corporations use GDD to improve service level and to lower labor cost by offshoring or outsourcing. They move and refocus in-house responsibility and resources to drive new competitive market opportunities.
- Grow revenue by market globalization or by presence in the local market
  Corporations address new markets by localization of software products, by compliance to local compliance regulations and enforcement, by response to customer requirements of customization and support at the client's site, and by establishing geographic presence.
- Attract a large pool of global talents
  The scarcity of skills and variations in labor costs have become a function of geographical location.
- "Follow-the-Sun" Development
  Corporations leverage GDD to shorten the "time-to-market" by evolving the "Follow-the-Sun" or "Round-the-Clock' business model and to be more responsive and competitive.

In the next chapter, the author will examine the challenges facing geographically distributed development.

3. Challenges

However, as mentioned in chapter 1, many companies have experienced a high degree of project development failure due to several challenges magnified by a distributed staff. The major challenges include strategy, communication, coordination, as well as infrastructural demands across global sites.

Mohahheghi (2004) summarizes the challenges as strategic issues, communication issues, coordination issues, cultural issues, geographically dispersion, technical issues and knowledge management issues, which will be examined in the following sections.

3.1 Strategic Challenge

The first strategic issue is the uncertainty of accessing when, where, to who and how to setup geographically distributed projects and teams.

In general, corporations lack experience in governing a new remote operation or new remote service provider and are not sure about the performance of an outsourced operation, and constantly worry about whether compliance is preserved. Heterogeneous inherited legacy is hindering acquired businesses to quickly deliver value. Development collaboration infrastructure is not set up to support a new mobility workforce. Geographic relocation of workforce requires manual administration reconfiguration and often results in high Total Cost of Ownership[10] (TCO – refers to the lifetime costs of acquiring, operating, and changing something).

Corporations often also lack the knowledge on strategic analysis of governing separation of core features and localized extensions, the management skills of quickly transforming new international business opportunity to first delivery. They also lack the infrastructure of supporting a flexible development methodology (waterfall, iterative, agile, etc.) and of being secure, scalable, reliable, 24/7, customizable and global.

[10] http://en.wikipedia.org/wiki/Total_cost_of_ownership

3.2 Communication and Coordination Challenge

Due to distance, time zone difference and budget constrains in travel, the global teams have fewer opportunities for incidental contacts and other informal communications that are essential to coordinate development efforts, manage task dependencies, and maintain awareness. While tools are helpful, there is little substitute for the cohesive, intimate working environment that teams tend to require.

Allen describes a communication pattern in that people tend to interact more with local and less with remote co-workers. He claims that the frequency of communication among engineers decreased with distance. Even 30 meters distance in the same building or floor is like miles away[11].

Coordinating and allocating the work of virtual teams is complex. In spite of a trend towards commoditization, the activities involved in software development are inherently different from traditional production processes involved in the manufacturing sector. Breaking up the development process into its chain components still represents significant risks and attempts to do so are often followed by considerable difficulties in coordination. Some activities, such as requirement engineering and architecture engineering, depend more on communication than others, such as testing, if well-written use case documents or functional specifications are available. Coordination across geographical locations is especially difficult when the task is creative. Whether defined as adaptive adjustments or as radical differences from established practices, creative projects pose a challenge to the coordination of distributed teams. Because these types of tasks cannot be broken down into small and well-defined components, and because they often require variations in the needed knowledge base, creative tasks are extremely difficult to coordinate.

[11] Tom Allen, Lecture Notes from course 15.980 "Organizing for Innovative New Product Development", Spring, 2006

3.3 Cultural Challenge

Cultural differences are often the root cause of many of the other issues, such as communications and misunderstandings, and can create significant stress in offshore project team members (R. Bakalov[12], 2004). Because culture is such a complex thing comprised of attitudes, experiences, beliefs and values, it is important that companies undergoing globally distributed projects understand and respect the differences in cultures across sites. The impact of not understanding and respecting cultural differences is severe, as many globally distributed teams have already seen.

Some of the impacts of cultural differences on global software development are explained by Edward T. Hall (1976)'s cultural dimensions – high context vs. low context cultures, poly-chronic vs. mono-chronic cultures [4], and by Geert Hofstede (2002)'s cultural dimensions terminologies include high power distance and the preference of uncertainty avoidance [5]. Hofstede claims that culture is "more often a source of conflict than of synergy" and cultural differences are "a nuisance at best and often a disaster." Within distributed teams, expectations that some individuals have larger amounts of power than others across geographical locations cause hierarchical forms of communication and slow decision making. This lack of efficiency highly hinders the progress of the software development projects. This type of behavior was seen during requirement engineering in a study of the Thai culture in 2000.

Another impact of cultural differences is that of uncertainty avoidance, which can be described as the preference of rules and structured circumstances. This particularly is one of the major causes for the waterfall development model and restrictive change models in distributed development. Other factors of cultural difference include emotional versus neutral, attitude to time, race, class, religion, attitude to governments and specific versus diffuse.

[12] Ernst & Young

All these identified cultural differences contributed to the challenges in a global software team which most companies are still struggling with. Most importantly, cultural difference could not be easily changed, and must be well understood, respected and coped with.

3.4 Technical and Knowledge Management Challenge

Yet another huge challenge that exists within globally distributed teams is that of inadequate infrastructure to support information and artifact sharing, which can hamper communication and collaboration among the distributed project members and even cause project failure. Distributed work of any kind puts a premium on effective and more formalized communication techniques and tools. This is especially true if distributed teams are working on the same body of code.

Due to the challenges of differences in time zones, special emphasis needs to be placed on enabling synchronous infrastructural means rather than asynchronous. For example distributed teams need to have things like phone, video conference, net meeting, e-chat, and instant messaging capabilities. Asynchronous tools such as e-mail, voice-mail, discussion lists, on-line calendars have delayed responses and are sometimes troublesome. Other tools such as document sharing, distributed software configuration management (SCM) systems, file transfer, remote access are also preferred tools. Overcoming the problematic asynchronous communication and time delays for communication and solution turnaround is a challenge that companies choosing to employ globally distributed teams need to reflect on.

3.5 Organizational Challenge

Another important challenge that poses a risk in global software development is organizational differences. Factors such as complicated organizational structures, many escalation levels, and different approaches in responsibility sharing make it difficult for distributed teams to function cohesively. Many organizations involved in distributed projects are not ready to change their internal structure and processes along with new partners and thus run into these organizational challenges.

3.6 Challenges in Open Source Software

Open source software development, as a natural practice of GDD, faces other characteristic challenges, such as lacking of explicit system-level or detailed design, lacking of project plan, schedule or list of deliverable and lacking of infrastructure to support more frequent release and rapid cycle time. These challenges, however, are not the ones this thesis aims to address.

3.7 Summary

Company-led geographically distributed development is facing multiple challenges including uncertainty in strategies, inadaptive in organizational structure, lacking communication and coordination due to distance and cultural differences, inadequate support in knowledge management and information sharing, etc. In the next chapter, the author will propose a framework to addressing these issues.

4. Best Practices to Harness the Full Potential of GDD

In his 1999 "Global Software Teams" book, Carmel describes "centrifugal forces" that drive global teams apart and proposed a six powerful "centripetal forces" for holding together a successful GSD team [1]. These six forces (Figure 4-1) include a stronger telecommunications infrastructure, better collaborative Technologies, a solid development methodology, careful architecture and task allocation, team building, and specialized management techniques.

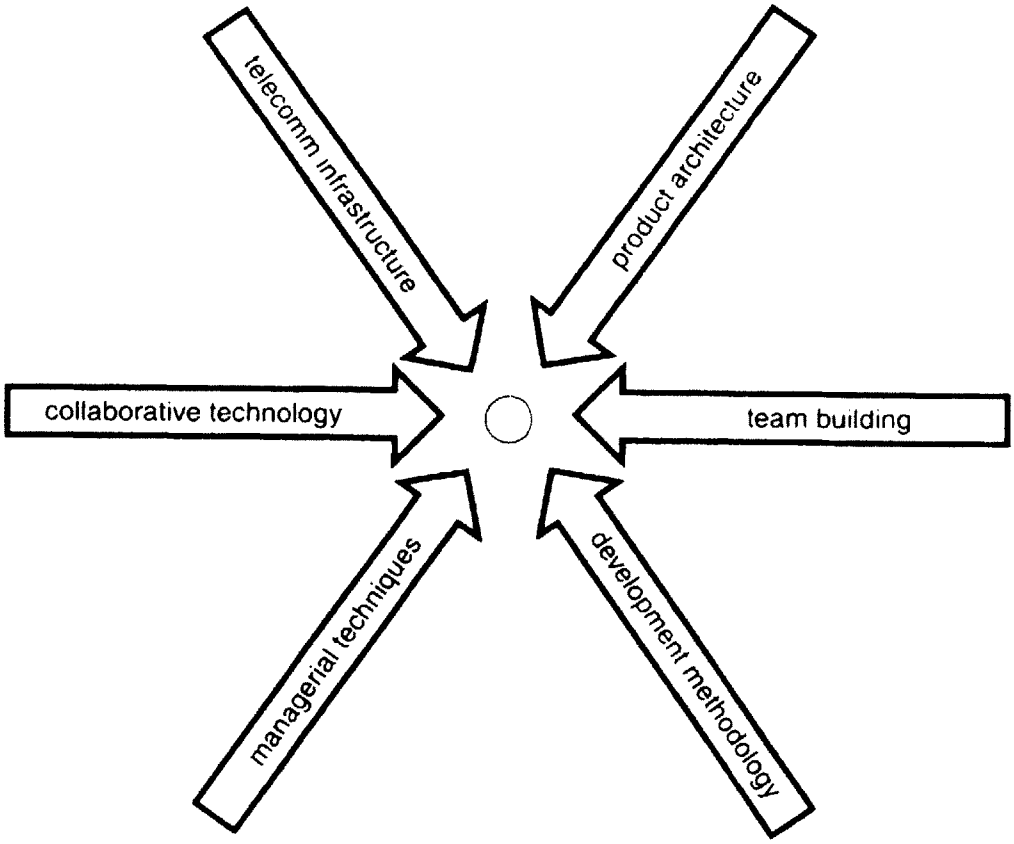

Figure 4-1 Carmel's Six-Force Framework [1]

Though still very useful and helpful as guidelines to managing a global team, the author argues that Carmel's framework has two obvious limitations:

- lacking strategic emphasis with too much focus on the tactical aspects

This could lead to short-term tradeoffs in the GDD investment.

- and the claim that all six forces are equally important
  Telecommunication infrastructure, for example, is now of less concern and should not be treated as a parity to other forces.

Based on Carmel's framework, research study and my own industry experience, I now propose a revised framework to address both the strategic and tactic aspects when dealing with geographically distributed development and global teams. As show in Figure 4-2, the revised five forces include strategic vision and management skills, organizational structure and team building, collaborative technologies, development methodology, and software life-cycle management.

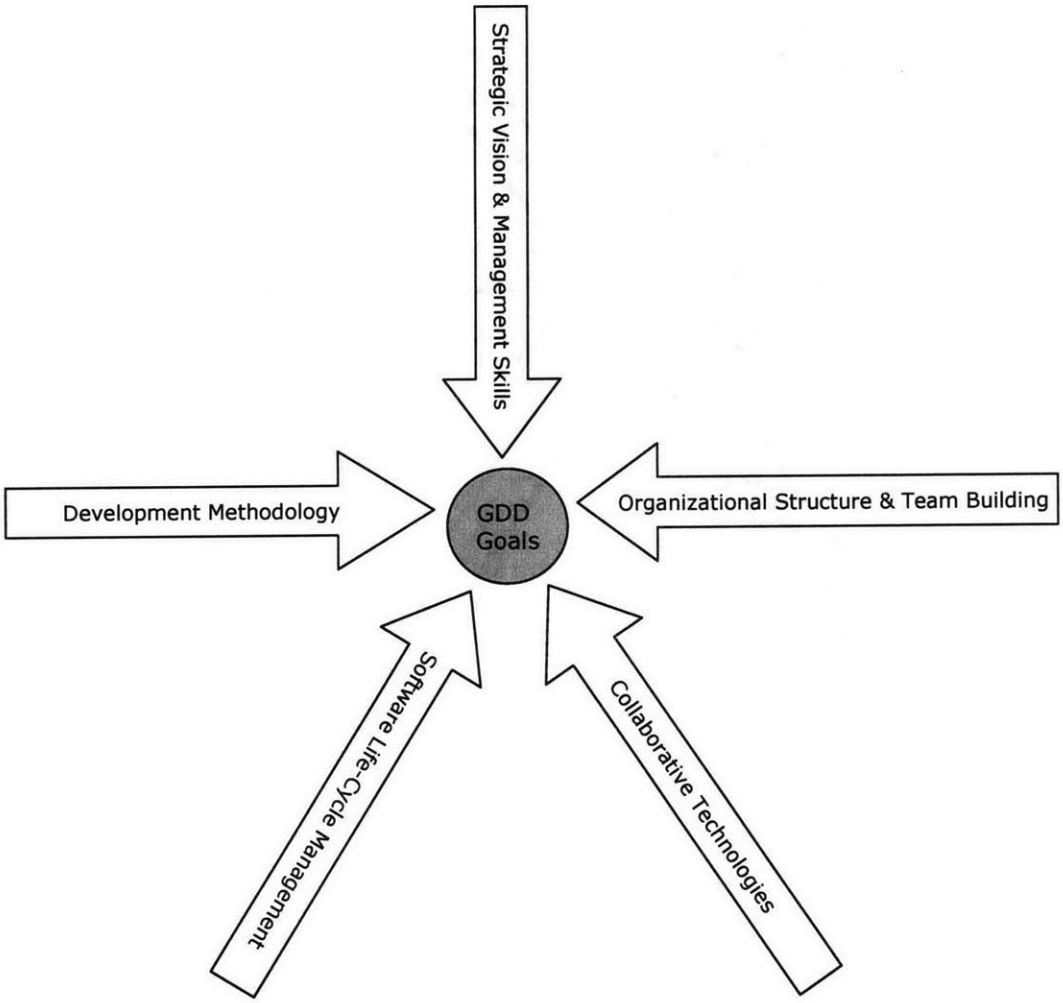

Figure 4-2 Revised Five-Force Framework

The following sections will examine each of the five forces and propose solutions in response to the challenges identified in chapter 3.

4.1 Strategic Vision and Management Skills

The main goals of software development activities are: on-time delivery, reduced development costs, innovative products and services achieved through increased diversity and creativity, and high quality. These goals tend to remain the most important ones, with the associated trade-offs, regardless of whether software development is set up locally or distributed across a wider geographical range. Software companies with globally distributed teams face however different sets of risks in their attempts to achieve these objectives. Evidence suggests that firms do not or are not able to systematically harness the full potential of geographically distributed development, resulting in sub-optimal value capture.

In recent years, company-led software development has been challenged by the open source software mode, a force to drive the commodity of software and to drive the software business model shift from "software as a product" to "software as a service". Companies such as Microsoft, Sun Microsoft and IBM have been changing or adapting their strategy to take into account this unconventional development approach.

This can often be attributed to a tactical approach towards GDD, absence of a realization that strategic vision is crucial and that special management skill is an acquired competency, and to an insufficient understanding of the structure and dynamics of GSD [6]. So the foremost force the author argues about is strategic vision and management skills.

According to the Gartner's road map for the overall global sourcing [7], Global sourcing for software and IT services, internally or externally, is now a mainstream business practice. Gartner estimates that more than 80 percent of Fortune 500 companies currently have some IT services delivered non-domestically. So a global development and delivery model is an integral part of a sourcing decision and understanding the effects of

formulating a sourcing strategy to optimize global resource and optimize revenue growth is critical.

Corporations and enterprises that are successful at global sourcing approach their sourcing decisions as long-term journeys that must be properly planned and executed as part of an overall sourcing strategy. The framework created by Gartner can help organizations understand the potential business value of global sourcing, and the major stages and attributes of adoption (see Figure 4-3). Organizations can use this framework to determine where they are today and where they want to be in the future.

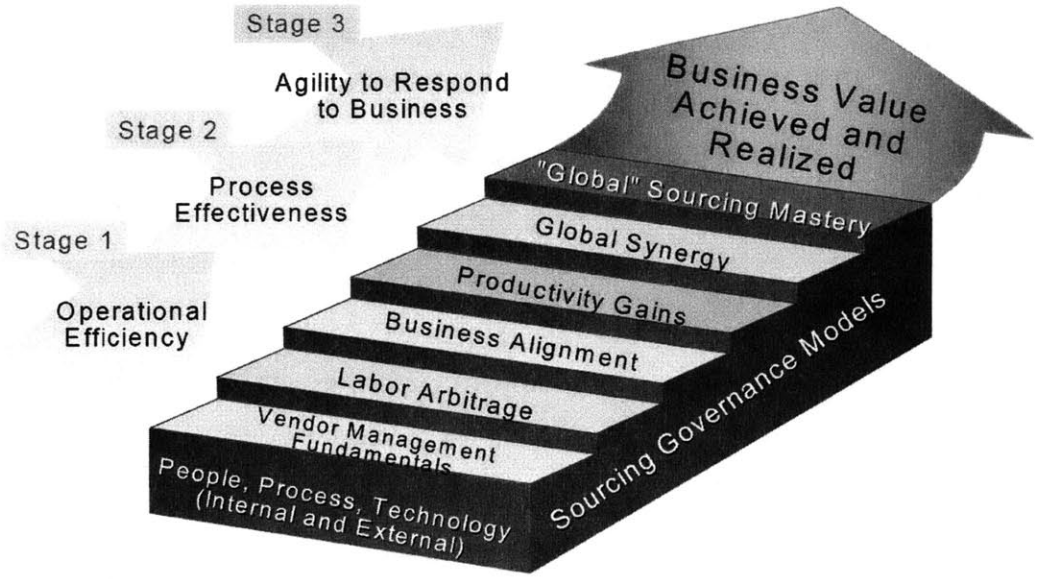

Figure 4-3 Gartner's Global Sourcing Framework: The Journey of Globalization[13]

This framework suggests that to reap all of the benefits of global sourcing, and to optimize the use of global resources as a strategic versus tactical initiative, to achieve the agility at the best prevailing global price point. Companies commit to a multiyear journey that involves three stages [7]:

- Stage 1: Competing Through Operational Efficiency
  At this stage, companies focus on achieving efficiency from accessing cheap labors and take GDD as a tactical initiative.

[13] Source: Garner (October 2006)

- Stage 2: Competing Through Process Efficiency
  At this stage, companies focus on achieving effectiveness from re-engineering / re-architecting in a global sourced model, from automating and re-designing development process, and from optimizing software performance and quality, which result in an increase in software productivity and delivery capability.
- Stage 3: Competing Through Agility to Respond to Business
  This stage brings the first two stages together to take advantage of the synergies of a global sourcing strategy.

A common pitfall in GDD strategy and management is the tactical, cost-focused outsourcing and offshoring mentality, which is particularly evident in US companies' approaches – cost is king, but not necessarily total enterprise and life-cycle cost, and this leads to short-sighted decisions versus long-time plans. Gartner's guideline and industry experience show that success at each level requires organizational commit to evolving key skills and expertise in global sourcing, as well as a concerted effort and planning, and a vision and leadership to realize the optimum benefits of that stage. Adopting the viewpoint of global sourcing as a journey implies a strategic commitment, not tactical actions, to institutionalize global sourcing best practices and lessons learned.

Specialized management techniques should be also considered carefully for successful global software development. There are empirical evidences that a manager with GDD experience (one that not even necessarily has a good record) can make a big difference compared to a manager who is naïve to GDD project management. Once a team is dispersed globally, the traditional simple team work structure of co-located work is not effective. Instead, management should take various and flexible approaches.

Social cohesion and inclusion are positive for business by opening and increasing the potential market and the potential talent pool. A more culturally diverse workforce initiates greater innovation and is necessary to serve an increasingly complex and global market. Developing cross-cultural skills enhances team performance and results in successful

software project. Culturally diverse teams are found to outperform homogenous teams by 10 percent on measures of alternatives generated and range of perspectives (The Ashridge Journal, 2000).

Managers of global teams are required to pay attention to many aspects of management, such as management transparency, coordination, cultural differences, and balance between a centralized and a decentralized structure. Managers can no longer assume tat the person in their team will act in accordance with national norms and the focus becomes more personal with different approaches to different people (Schineider, 1997) [8]. It is not effective for a manager to apply simply one style of management to the GSD team. Following different culture that each team shares with, specialized managers should apply their techniques to each case of different team.

In summary, GDD is strategic only when it is exploited for business transformation. Successfully capitalizing on GSD requires not only effectively dealing with the complexities and challenges that the global distribution of software work throws-up but also employing a well-crafted strategy to realize the full potential of GSD [6]. GSD can be strategically leveraged and tactically managed to harness GSD for business transformation, and for gaining competitiveness and growth. Evidences show that once settled, multicultural teams tend to perform better than monoculture ones in identifying problem perspectives and generating alternatives [8]. Cultural awareness, special skills and flexible approaches are required to manage a diversity team.

4.2 Organizational Structure and Team Building

The global organization of software development is both shaping and being shaped by visions of future organizational structures. By understanding the organizational spectrum (whether or not the GSD model will include outsourced and offshore segments) and discipline alignment (where tasks in the development process will be done and by whom), organizations can focus on specific challenges and requirements introduced at each level.

Virtual Team

Geographically Distributed or Dispersed Team (GDT), also known as a Virtual Team, is a group of individuals who work across time, space, and organizational boundaries with links strengthened by webs of communication technology[14]. They have complementary skills and are committed to a common purpose, have interdependent performance goals, and share an approach to work to which they hold themselves mutually accountable. Geographically dispersed teams allow organizations or development community, such as Open Source Software, to hire and retain the best people regardless of location. Members of virtual teams communicate electronically, so they may never meet face to face. However, most teams will meet at some point in time. Many virtual teams in today's organizations consist of employees both working at home and small groups in the office but in different geographic locations.

Virtual teams bring the benefits of obtaining the best employees located anywhere in the world, dealing with increasing technological sophistication, personal flexibility and productivity, and achieving a "follow-the-sun" development model.

Virtual Organization

Virtual organizations refer to entities organized around a structure resembling a network that has a weak hierarchy and a weak center. Even though geographically distributed development seems particularly consistent with the concept of virtual organization, the configurations existing in reality suggest however that global software teams are neither virtual organizations nor virtual projects.

Research and studies show that in spite of continued globalization of software development, the core activities tend to take place primarily at the headquarters located in the industrialized countries (Carmel, Tjia, 2005). Software firms are distributing some of their development projects, but they are still reluctant to disseminate core competencies and the existing organizations still carry several layers of hierarchy. While it is not unusual to find a network of collaborating teams, the activities associated to the highest levels of competency and authority are still kept within the boundaries of headquarters or at least

[14] http://en.wikipedia.org/wiki/Virtual_team

host-countries, and the geographical range is still relatively limited, including only a few countries. The current stage of global organization of software development thus also maintains some of the distinguishing features of traditional supply chains in a sense that software producers add value as the project is transformed and passed from one location to the other (Carmel, 1999 and 2005).

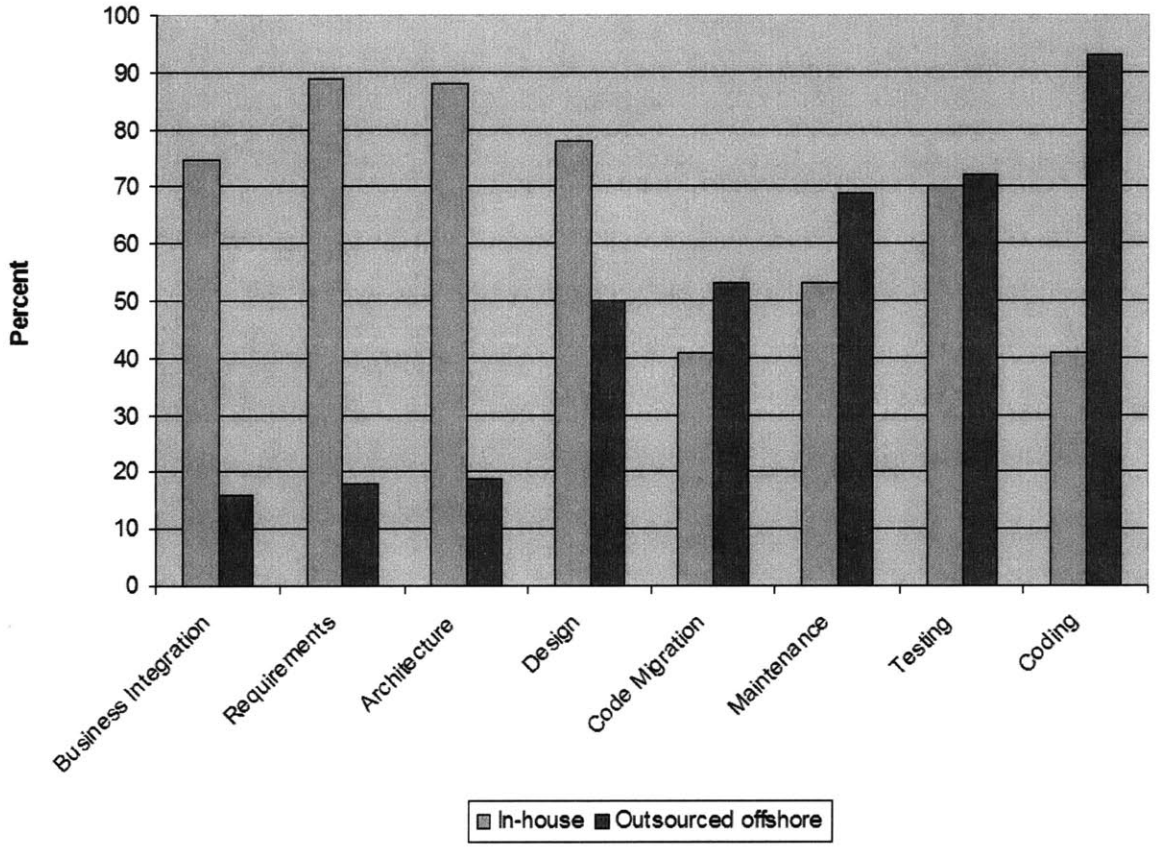

Figure 4-4 In-house vs. Outsourced-Offshore activities[15]

Figure 4-4 thus illustrates the relative importance of in-house versus outsourced, offshored development for different chains of the software development process, based on data acquired through a survey conducted with US software companies in 2004. These results strengthen the validity of the assumption that the activities that tend to be distributed globally are indeed the ones that are more standardized – they can be specified precisely and are less reliant on high-end technical expertise.

[15] Source: Adapted from Carmel, 2005

Intra-organizational and Inter-organizational

Another approach to analyzing the organizational spectrum of global software development is, instead of focusing on the distribution of core versus non-core activities, consider the character of the legal ties connecting the elements of the development community. Global software development in this respect expands over two main forms including intra-organizational or intra-enterprise referring to legally related companies and development teams, and inter-organizational or extra-enterprise referring to a 3rd party service provider. A graphic illustration of these main forms is illustrated on Figure 4-5 below.

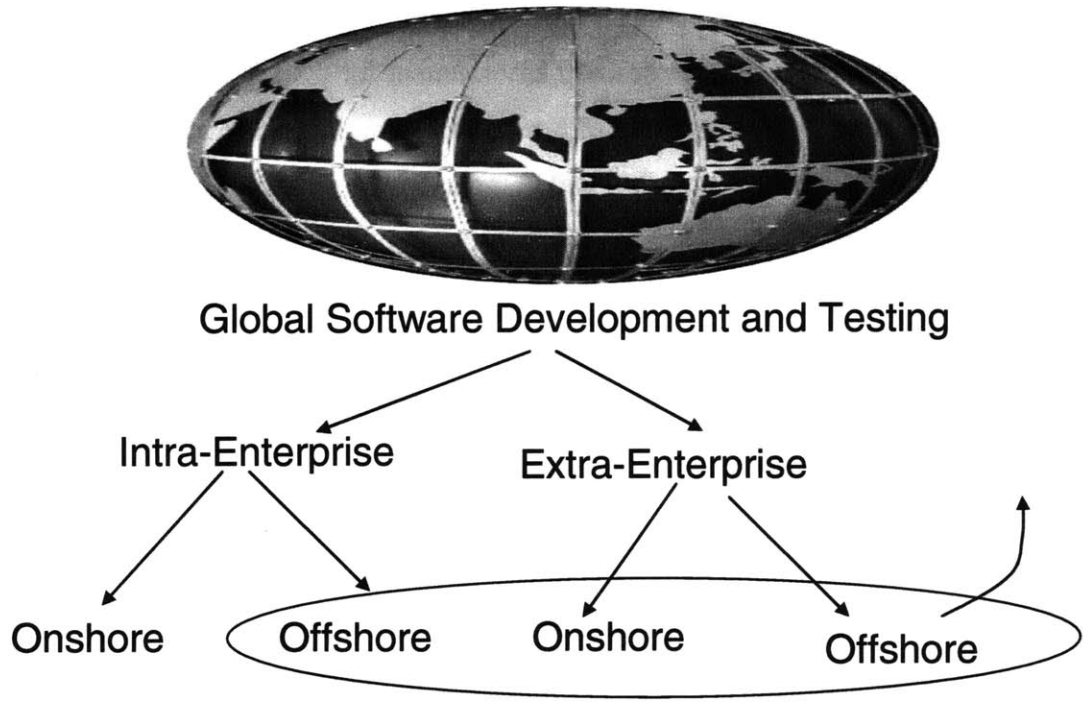

Figure 4-5 the legal organizational spectrum of GSD[16]

Non-organizational Form

With recent developments in the business practices and technological practices shaping the software industry, non-organizational forms, such as open source software development, and services or components provided via the Web, as in the case of Application Service

[16] Adapted from Sikka, 2005

Providers (ASP), are increasingly recognized as possible third and fourth alternative approaches to global software development (Mohagheghi, 2004).

In particular, open source software represents a radically different model of global software development. The loose organization of some open source projects allow for more of a networked organizational structure, where distributed developers encompass the entire development cycle – from high level design through final testing and implementation. Due to the very nature of open source development, this organizational structure can possibly be the movement that brings about a flattening of the organization of software development across the world.

Take the GNOME (GNU Network Object Model Environment) project, an attempt to create a free and therefore open source desktop environment for the UNIX systems, as an example. The GNOME project is composed of three main components: an easy-to-use GUI environment, a collection of tools, libraries, and components to develop this environment, and an "office suite." (German, 2002) During its nine years of development, the engineers who contribute to the GNOME project come from anywhere that has web access. Importantly, there is no identified headquarters for this project. There is no longer the organization or management of getting the least efficient link to complete its task in order to hand off the module to the integrator.

However, even with accusations regarding conventional organizations and conventional-managed software, and the praise of the strengths of the OSS-style development (Raymond, 1999) [10], there were instances of chaos. For example, in the GNOME project, the fact that particular tasks were not done by volunteers had hindered the development of the project, and several companies, such as RedHat, Sun Microsystems, and Ximian, with the hopes of accelerating the development of the project, decided to pay full time employees as contributors who would enable the completion of these tasks. These paid employees are usually responsible for tasks such as project design and coordination, testing, documentation, and bug fixing. Because these tasks are less attractive to volunteers, by taking care of them, the paid employees make sure that the development of GNOME

continues at a steady pace. Because the tasks that these particular contributors take on are usually the less desired, no one company gains power or gives direction over the development of the project (German, 2004) [9].

Team Building

Creating common culture, building personal trusts, and developing common protocols are not easy for global software development teams. In most cases, the team members do not travel from site to site and have never met with team members in other sites. The lack of personal contact among the project members can also lead to troubled trust; confidence achievement is an important objective of project management. Trust in the relationship enables cooperative behavior, reduces conflicts, decreases transaction costs, and promotes effective responses to crises (Rousseau *et al.* 1998). The involved parties may consider each other as 'us' and 'them', but may also perform as a unified team labeled as the 'us'. Therefore, managers and members in a distributed team should pay careful attention on team buildings.

Successful teams tend to embody many aspects of team building to actively build trust across different sites. Lessons show that there is no greater opportunity for building an effective virtual team than at startup. It is recommended to begin a global project with kick-off meetings to introduce teams (for example, using PowerPoint slides with team member photos, background, and pastime hobbies), lay out vision, and explain development methods and communication protocol. It is also helpful to have milestone meetings to articulate accomplishments and lessons learned, and preview next step and roadmap. Other team building methods include providing a 360 view, adopting a flexible communications protocol – lateral or vertical, building personal bridges between sites, and investing in training on language and cultural issues.

In open source software development, the core team (with characteristic of highly active, mainly focused on coding, and not architects of the modules to which they contribute) is composed of self-selected, talented programmers with high motivation and true belief in

the code base, and led by project coordinators who normally have good people and communication skills.

Identifying and keeping a "gatekeeper[17]" in a virtual team, as glue to team building and link to outside technology, is also desired. The "gatekeeper" is normally someone with high technical performance but low in the organizational hierarchy. "Gatekeeper" is not just a communicator, but rather someone who has the capability of managing the flow of messages, knowledge and information, approachable and receptive to people. He/she can help team building. However, "the Gatekeeper" cannot be created by management and he/she may not be so visible in a virtual team as in a co-located team. Raymond describes Linus Torvalds as the "gatekeeper" [10] to the Linux open source software project.

Carmel (1999) explicitly warned that companies guard against giving too much power to the headquarters [1] in terms of assigning leadership roles and allocating tasks. Another big issue regarding virtual software team management is when and how team leaders should delegate authority and responsibility to the team. Delegation means that one has been empowered by one's superior to take responsibility for certain activities which are originally reserved for managers (Bass, 1990) [11]. Traditional literature indicates that the influence of delegation on team outcomes is moderated by a number of factors such as follower maturity, group development, and team reality. The general conclusion, for traditional, co-located teams, is that a leader should delegate to his followers more when the followers are mature [12, 13]. However, effects of adding virtual team as yet another factor are not well understood and there has been limited research resulting in conflicting findings [13]. Coordinators or leaders in open source software, however, are known as good at empowerment by "delegating everything you can" [10]. Managers and leaders should be sensitive and proactive to this issue. Empowerment, after all, can be one of the core forces to achieve creativity and innovation in a global distributed team environment.

[17] Tom Allen, Lecture Notes from course 15.980 "Organizing for Innovative New Product Development", Spring, 2006

In summary, the development of open source software in globally distributed locations has a unique organization structure different from proprietary development. Where in proprietary development hierarchical models are necessary and integral to the survival of the project, a much more flattened approach is undertaken when developing open source software. While the existence of globally distributed teams does assume a certain flattening of the organizational structure, geographically-distributed development certainly demands a more flexible and adaptive organizational structure, as it is more competitive and responsive to the marketplace.

The virtual organization remains a theoretical model today towards which organizations are converging, as the emergence of environments which require inter-organizational cooperation as well as competition, the continued shift from production to service/knowledge work environments, and the changes in workers' expectations of organizational participation. Forrester's Navi Radjou (2006) alleges that "The World isn't Flat till Global Firms are Networked, Risk-Agile, and Socially Adept" [16]. Building a unified team is considered to be one of the best ways for team productivity and risk mitigations, and team building should focus on various aspects and involve multiple activities.

4.3 Collaborative Technologies

Communication, coordination and collaboration are critical in all software projects of any size. When a project is co-located, much of the coordination happens invisibly, embedded in the habits and practices that grow up around the development process, and in face-to-face meetings and informal conversations. As demonstrated in figure 4-6, less than 20 percent of the technology sources for product development projects come from written material. A large portion of messages involve people to people (inside and outside) interaction.

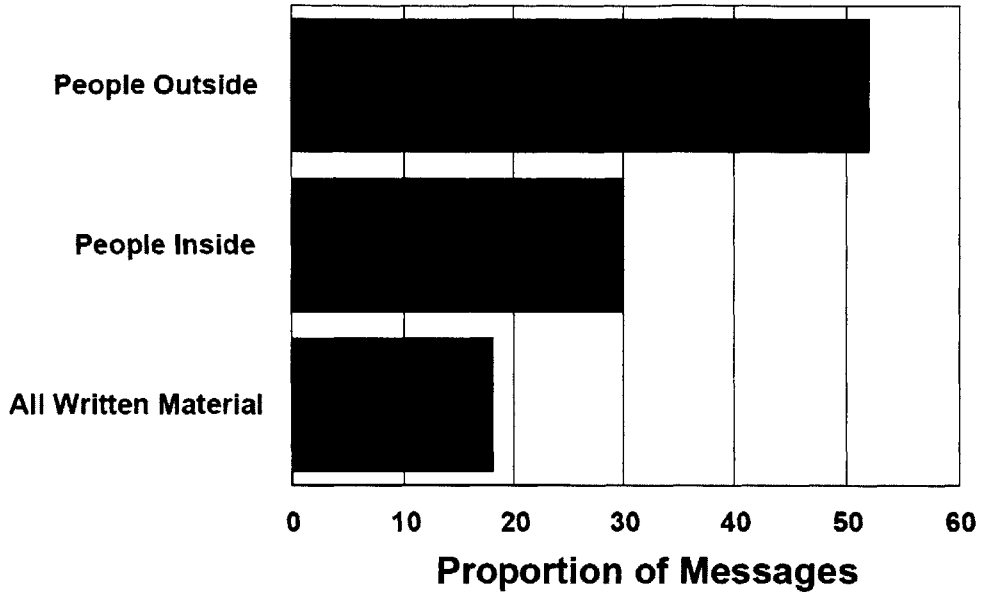

Figure 4-6 Technology Sources for Product Development Projects[18]

However, these "invisible" means of coordination are not consistently available in distributed projects, allowing team members to see - by observing the effects of their absence - the nature of coordination among people that software development requires [17].

Collaborative technology is crucial for successful global software development. It makes the "invisible" communication and coordination which happens in a co-located team visible in a global team. The generally used methods can be categorized into three groups (1) Individual initiatives, including E-mail, Instant Messaging, Blogging, etc. (2) Meetings and Events, including Web conferencing, Audio conferencing, Video conferencing and Webcasts, etc, and (3) Team and Communities, including Wiki, Software Configuration Management (SCM), file sharing and document management and other communities tools including the emerging technology – collaborative development environment (CDE).

As managers, project leads and members engaged in geographically distributed development, knowing the available collaborative tools is a prerequisite. As companies

[18] Tom Allen, Lecture Notes from course 15.980 "Organizing for Innovative New Product Development", Spring, 2006

with global workforce or with global partners, investing in and adopting collaborative technologies can effectively improve productivity and promote innovation.

The author will explore the telecommunication infrastructure, some of the commonly used or emerging collaborative technologies in the context of geographically distributed development, and then argue that companies and development communities should adopt collaborative technologies and structure with measurable goals.

Telecommunications Infrastructure

Given the distance and cultural differences among the global teammates, the foundation of telecommunication is crucial for successful team and telecommunications infrastructure is critical. With well developed telecommunications infrastructure, each team member in global locations can communicate and cooperate in a timely and smooth manner.

The Internet is the true enabler of open source software development. For example, Linux was considered the first project to "make a conscious and successful effort to use the entire world as its talent pool" [11]. Raymond (1999) [10] also argues that it is not a coincidence that the gestation period of Linux coincided with the birth of the World Wide Web, and Linux left its infancy during the same period in 1993-1994 that saw the takeoff of the ISP industry and the explosion of mainstream interest in the Internet.

In the past, a high speed telecommunication network was not reliable and many companies had a private global network. In the past decade, the maturity of Global telecommunications networks has provided a vehicle for relatively inexpensive communication between teams that are separated by great physical distances. For example, a Virtual Private Network (VPN) is available through internet and it is now widely used. Constructing a VPN is cost-effective because it reduces connectivity costs as well as some equipment costs. However, VPN contains security concerns and they are not perfectly solved.

Another thing to note is that even though the internet is ubiquitous now, it is not as available in some areas of the world, such as some parts of Vietnam and Russia. If a company engages in software development in such an underdeveloped part of the world to seek deep cost reduction, the undeveloped telecommunication infrastructure may become one of the biggest concerns and often require significant investment.

In summary, today's telecommunication infrastructure is much stronger than seven years ago when Carmel published his book. Advances in global network connectivity over the past 10 years have significantly reduced the effects that physical separation has on geographically distributed development teams. However, there is still a general concern in terms of network latency, firewalls, authentication and data security issues.

Collaboration Tools

E-mail, as a killer application in the communication, is probably the most popular collaborative tool. However, considering cultural aspect, Carmel (1999) regards e-mail as not effective to teams in some cultures. Americans may be comfortable allowing e-mail participants to get into a heated "fight." Such an open conflict is acceptable for Americans, but not for Japanese who operate in another cultural context. In cultures like Japan, coordination is sometimes more important than open discussion. When you collaborate with a team in other countries, you should take into account the different way of collaboration and may have to think about other types of collaborate technology. The lack of this kind of cultural awareness has significantly hampered many attempted collaborations; alternative approaches should be considered. Software configuration management systems, for instance, can be used to mitigate this problem.

Instant Message is starting to find its way into the business world as a valuable tool. It is more interactive and responsive than Email, and thus is highly recommended for co-located teams and for teams within certain time zone. However, it proves to be less useful when you have a team in USA and a team in China with 12 hours time difference, and with language barriers. Another problem with Instant Message and E-mail is that even though

the content of discussion and information exchange can be saved for later review, it is very difficult (if not impossible) to find, locate or search when needed, and is often lost.

Web logging or "blogging" is the process of instant publishing to a Web page. "Blogs" typically contain short messages that follow a chronological order. Global team members can use blogging to publish their progress, and the Web interface makes it easy for everyone to see each other's notes. Evidences show that "blogging" can be an effectively way to understand cultural difference and to learn how to anticipate different behaviors.

Wiki was invented by Ward Cunningham in 1995. It was created as a means to develop collaborative web pages by allowing information and associations to be added freely by any or selected users. Because a history of all wiki changes is maintained in a database and logs of changes can be obtained, the system is essentially self-regulating and has a survival-of-the-fittest but evolutionary feel [18]. Wiki has proven to be a very useful asynchronous collaboration tool for a geographically distributed team. However, some of the problems with the use of wiki should be considered and addressed. Literature, interview with wiki users and personal experience reveal that many team members seem uncomfortable with the idea of shared ownership of data within wiki. This problem evidences itself in two phenomena. On the one hand, some original authors of wiki pages want to have control over the content; on the other hand, contributors have concerns about erasing or deleting other's work.

When real-time communication is needed, conference calls can be significantly enhanced with Web conferencing. Web conferencing can be used for group presentations, interactive planning and review, and even unstructured brainstorming sessions. Today, most web conferencing products include collaboration features such as white boarding, file sharing, instant messaging, and cooperative editing.

According to a study at IBM, the web conferencing usage increased steadily over the past several years because of the rapid growing of geographically distributed teams.

- 2002 - Average daily ~1500 participants, ~300 meetings

- 2003 - Average daily ~3000 participants, ~500 meetings
- 2004 - Average daily ~4000 participants, ~900 meetings
- 2005 (trend) = ~5000+ participants, ~1000+meetings!

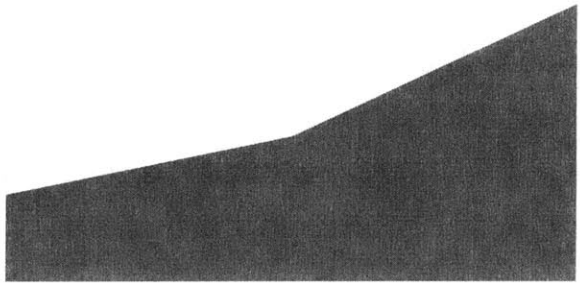

Figure 4-7 Web Conferencing Usage at IBM

Carmel (1999) explains that a collaborative technology to support software engineering (CT-SE), such as Software Configuration Management (SCM) and Collaborative Development Environment (CDE), works better for global software development teams. He argues that CDE presents five important objectives, such as serving as a team memory center, provision of team 360 views, support of coordinating workflow, reduction of duplicated effort, and support of quality assurance.

Pressman defines software configuration management (SCM) as a "set of activities designed to control change by identifying the work products that are likely to change, establishing relationships among them, defining mechanisms for managing different versions of these work products, controlling the changes imposed, and auditing and reporting on the changes made." [19] Geographically distributed development involves mainly software code, and SCM tools can support an in-context collaboration that tracks changes, and mitigates the impact of cultural difference, distance, and time zone. Literature and empirical experience from both the company-led and the OSS-style global software development have both validated the importance of using software configuration management (SCM) tools. Section 4.5 will explore SCM in more details.

As an emerging collaborative technology, Collaborative Development Environment (CDE), a term coined a few years ago by IBM fellow Grady Booch, allows geographically

distributed project team members and distributed stakeholders to share ideas and knowledge and to work together on a common task. With large projects across the globe and with the looming of temporally and geographically distributed development, Booch proposes a next generation collaborative development environment that changes the focus from the needs of the individual developer to the needs of the team.

Typically, software developers spend a majority of their time on code-centric activities supported by an Integrated Development Environment (IDE) offering a range of code development and manipulation features. Other aspects of their task involving interaction, communication and coordination within and across teams are generally supported by a discrete combination of capabilities including configuration management systems, issue tracking databases, instant messaging systems, project websites, and so on. Assembled in a coherent fashion, this latter set of capabilities can compose a collaborative development environment for software engineers.

Whereas traditional IDEs focus upon improving the efficiencies of the individual developer, CDEs focus upon improving the efficiencies of the development team as a whole. Today, there exists only a few commercial CDEs focused primarily on the problem of distributed software development over the Web, most notably, SourceForge[19] and CollabNet[20]. But there are many more that have been created for other domains or that address one specific element of the software CDE domain. The openCollabNet[21], a new online community based on Web 2.0 principles and integrated with SCM capability, claims that it connects about 1.5 million software developers all around the world, allowing them to share innovations and software development best practices applied to open source projects and to tightly protected enterprise and government intellectual property. IBM's alphaWorks has launched a portal[22] about CDEs and predicts CDE as "the next big thing" following the journey of development tools from command line interface (CLI)s to Integrated Development Environment (IDE)s, to Extended Development Environment (XDE)s.

[19] http://sourceforge.net/
[20] http://www.collab.net/
[21] http://www.open.collab.net/
[22] http://www.alphaworks.ibm.com/topics/cde

It is important to articulate some measurable goals when investing in collaborative technologies and deploying collaboration tools. Some measure goals could include:

- Cost – What is the total cost of ownership (TCO) associated with each tool? What portion of project budget should be invested in collaboration technologies?
- Scalability – The scalability of the tools is critical to anticipate the growth of the global team, to enable regional communication and distributed decision-making, and to improve the accountability in proportion to authority [38].
- Security – The security of the tools determines whether 3rd party or extra-organizational partners or customers could be included in the collaborations.

In summary, collaborative technologies are intended to resolve coordination issues in any context, co-located as well as distributed, but are especially useful for enabling a global team to communicate timely and efficiently. Adopting a rich set of collaborative tools with measurable goals and with security taken into account will fundamentally increase the productivity of the global team and promote innovations.

4.4 Development Methodology

The development methodology is the map that guides the team through the software development cycle. It also becomes the common language bridging developers at different sites. When one writes that "we'll do that during the X stage," exactly what is expected in the X stage should be understood between the two developers in distant areas. In addition to providing team members with a common language of tasks and activities, these methodologies reduce redundant activities and excessive work as well as enhance quality assurance.

Understanding the various methodologies and adopting them to tackle the dynamics in geographically distributed development is one of the keys of managing a global team.

4.4.1 Waterfall Methodology

Challenges that globally distributed teams face had led this type of development to favor the waterfall model, a development model introduced in the late 1960s in response to the problems of managing large custom software development projects. Its aim was to bring control and discipline to what had previously been a rather unstructured a chaotic process (MckCormack *et al.*, 2003) [20]. The waterfall model is a sequential software development model in which a project is completed in a series of steps, such as analysis, design, coding, testing and deployment. Each step, such as requirements gathering, is undertaken only once and must be completed and verified before the next phase. Waterfall model is sometimes referred as Classic Life Cycle Model, or Linear Sequential Model, or "Japanese Factory" [21]. As illustrated in Figure 4-8, the waterfall model minimizes the need for communication and coordination because there is clear handoff from one phase to another. This method also makes tasks allocation easily to manage, for example, mapping phases to responsible development sites. There is less concern over shared context among geographically distributed teams since one phase's output becomes next phase's input. However this is true only if the handoff happens at phase boundaries – the more critical issues are often driven by distributed development in parallel during the same phase.

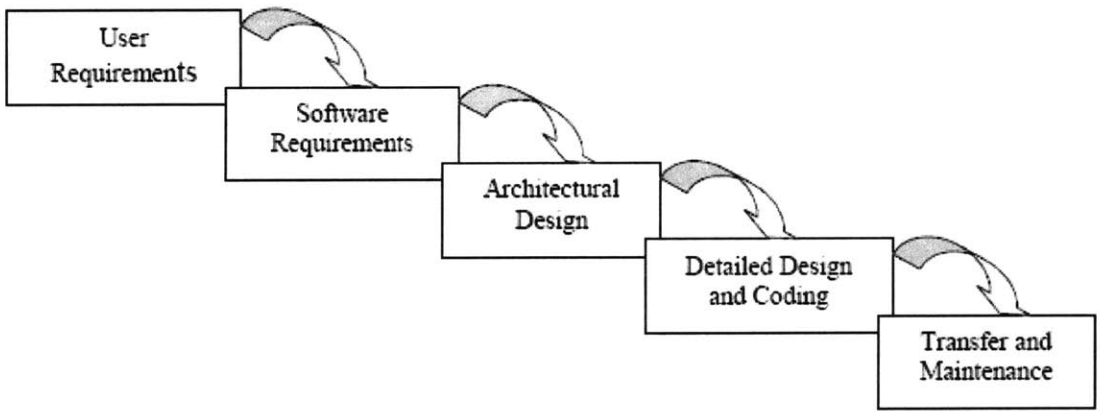

Figure 4-8 Waterfall software development model

It has been believed that waterfall development model works better with a geographically distributed development project to mitigate the impact of cultural differences and avoid

uncertainty among remote teams (a study in 2003 comparing Japan, US and India). While implementing the waterfall development model may not be completely ghastly, this model makes it difficult to scale up to meet the challenges of increased size and complexity of newer software products.

4.4.2 Iterative or Agile Methodology

Recent literatures [20][21], research, and industry reports, however, have concluded or suggested that the waterfall model makes it difficult to scale up to meet the challenges of increased size and complexity of newer software products. Moreover, the complex global environment developers are facing today would warrant the adoption of the agility principles in software development teams more than the values of enforcing discipline in software development. Iterative development, especially a variation called the "agile development" model, gained a lot of attention in the GDD field. The term Agile Method of software development was coined in the 1990's. It incorporates autonomy into software development to nurture creativity and resourcefulness. For examples, a "synch-and-stabilize" (Selby and Cusumano) [21] style development is intended to adapt easily to change during a project at the features level and produce incremental releases relatively quickly. These capabilities are crucial to GDD because of the high requirement uncertainty and the pressure of "time-to-market". Cusumano argues it is inherently superior compared to the over structured "factory-like" or waterfall approaches [21]. Paasivaara and Lassenius [22] reported that a Finnish company was developing software using both several onsite subcontractors and a subcontractor that had onsite developers working in the customer's premises and offshore developers. Earlier, the company had used a waterfall type process with its main subcontractor. This development model had many problems, which led the company to experiment with the usage of Scrum, an agile method for software development and project management.

However, agile and (traditional) distributed development approaches differ significantly in some key tenets. For examples, agile development relies on informal processes to facilitate coordination [23] and advocates on peer programming and frequent code review, all of which can be hard to implement in distributed development. Thus despite the

suggested benefit of being more responsive, agility methods have a problem in managing the vast amount of information and knowledge, which were relevant. Communication of the ACM October 2006 is dedicated to agility in global software development.

In an effort to achieve or maintain an iterative and agile development model, companies seeking to undergo globally distributed software development should identify and become aware of the challenges plaguing the distributed teams. Specifically, some of the higher-risk challenges in distributed projects that dissolve the ability to implement an iterative model include differences in culture, organization, infrastructure, difficulty in work allocations, and different time zones, among others.

4.4.3 Capability Maturity Model

Capability Maturity Model (CMM), as one of the advanced development methodologies, was originally created by System Engineering Institute (SEI)[23]. It provides a conceptual structure for improving the management and development of software products in a disciplined and consistent way. The staged Software CMM Integration (CMMI)[24] model identifies five levels (ranking from 1 ~ 5) of software development practices that provide organizations with a roadmap for software process improvement, taking them from chaos, ad hoc, ill-defined processes to mature, disciplined software practices.

Despite criticisms regarding the lack of formal theoretical basis (Bach, 1994) [24] emphasizing control over flexibility and learning (Senge, 1990) [25], undermining intrinsic motivation and ignoring many factors that contribute to the productivity of individual engineers (Jones, 1994) [26], numerous organizations are adopting the CMMI model as a way to improve processes and to manage the development, acquisition, and maintenance of software and services. In global software development context, offshoring and outsourcing vendors are mandating CMMI to drive competitive advantage, to increase effectiveness and efficiency, to increase customer satisfaction by improvement on performance, cost and schedule, and also to stimulate cross-border collaboration. According to SEI CMMI profile reports (Table 4-1), India and China now have second and third largest number of CMM

[23] http://www.sei.cmu.edu
[24] http://www.sei.cmu.edu/cmmi

appraisals, next to USA. Chinese software enterprises now rush to CMMI. As an additional promotion, Chinese government even promised to provide up to 50% cost subsidy to CMM certification (Ju, 2005) [27].

Country	March 2002	March 2004	March 2005	March 2006
China	18	152	243	354
India	153	330	387	422
USA	1498	1838	1947	2035

Table 4-1 Number of CMMI Appraisals [25]

CMMI, as a development methodology, may spread from the offshore software or service vendors back to the home organization, as a result of the desire of match that of the vendor's. This is reported by Pitney Bowes[26] during the outsourcing of the entire portfolio of applications development and maintenance. At that time, Pitney Bowes was evaluated at CMMI model level 1 (Initial), while the outsourced service provider was at CMMI level 5 (Optimizing). The big process maturity gap required increase resources to resolve issues and rework, making business benefits not realizable, and hazarding a value-added partnership. This forced Pitney Bowes to adopt CMMI processes [29].

4.4.4 Bazaar-style Methodology

The open source development mode is quite different from the traditional commercial software development model. Raymond likens the corporate or traditional model, whereby a corporation produces and sells proprietary software, to a cathedral and the open source model to a bazaar [10]. In the corporate model, individuals or small groups of individuals quietly and reverently develop software in isolation, without releasing a beta version before it is deemed ready. In contrast, the open source model relies on a network of "volunteer" programmers, with differing styles and agendas, who develop and debug the code in parallel. The software is released early and often.

[25] http://www.sei.cmu.edu/appraisal-program/profile/pdf/SW-CMM/2006marSwCMM.pdf
[26] http://www.pb.com/

The "Bazaar-style" development is not in itself a new process [30]. Most software development initiatives are carried out by a limited (often just one) number of developers. This observation has been corroborated by several studies of open source repositories such as Souceforge.net. Large open source projects such as Linux and Apache do have a very well structured and organized process, which resembles those of other proprietary products [31]. The characteristics of OSS' "release early and often" strategy is also observed in Microsoft's practice of daily build and feature orientation [21]. Iterative and Extreme programming stress the notion of incremental and evolutionary development, and can be equally applied to proprietary and open source software.

In summary, the goal for adopting a development process is to create enough structure to keep projects under control but not so much that 'the process" stifles creativity and flexibility [21]. Given the complex and dynamic nature of the geographically distributed development, the desire of gaining competitive advantage and the pressure of "time-to-market", the author proposes an iterative, agile methodology with an architecture that supports distributed, model-driven development. CMMI should be also considered as a software engineering standard especially when outsourcing is involved.

4.5 Software Life-Cycle Management

Software Life-cycle Management (SLM) or Application Life-cycle Management (ALM) regards the process of delivering software as a continuously repeated cycle of inter-related steps: requirement, architecture & design, development, testing, deployment and management. Each of these steps needs to be carefully monitored and controlled. For most software projects, the coordination of life-cycle development activities remain a manual process, even with co-located project team, and becomes one of the big concerns in geographically distributed development mode.

The author thus argues that in geographically distributed development, software life-cycle management, as the act of managing all artifacts related to a development project including versioning, relationships, and workflow, along with project-based task management, should be carefully considered, evaluated and deployed.

4.5.1 Architecture Decision

Software architecture gives companies intellectual control by establishing a means of effective communication throughout the project by a common vocabulary. It makes large-scale reuse possible by clearly articulating its components and interfaces. And it provides the basis for project development and management, orienting development teams in systematically producing different parts of a system. It is known that an architecture-based approach can bring about systematic management and productivity for conventional processes. In geographically distributed development, maintaining a common architecture to support distributed development, orienting all the software life-cycle processes, can ultimately contribute to the success of the software project and sustainability of the software system.

4.5.2 Project Management

The project architecture in a global software development context, in general, could result in four types of distributed processes [39]: module-based, phase-based, integration-based, and follow-the-sun. As project managers, the most important job is to understand these distributed development process, and determine what is required, in each case, to manage the process.

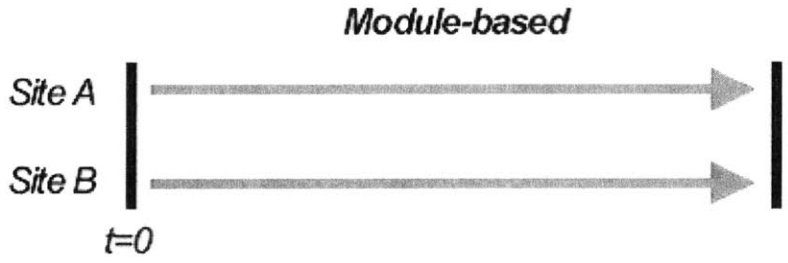

Figure 4-9 Module-based project architecture [39]

A module-based (or component based) process assigns tasks or modules to each site, where the entire software life-cycle management, including planning, developing, testing and maintaining, from start to finish, was carried out. The principle of modularity is to design software components being self-contained and having few interdependencies with other modules.

As a commonly adopted process, it generally requires a decoupling of the assigned modules such that there is minimal interdependence among sites and teams, and in some cases even require different development sites to have a vertical field of expertise to enable the work split be carried out to maximize and leverage each site's strengths.

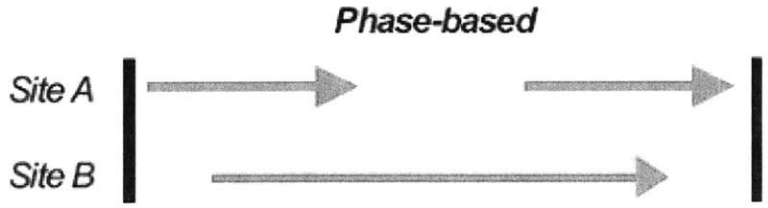

Figure 4-10 Phase-based project architecture [39]

In a phase-based process, the work is passed from site to site at the end of a major phase. It has the benefit of balancing the disparity between sites in the technical capacity to perform some of the more complex phases for a particular project. This is a typical, traditional model of leveraging offshore capabilities, where the design work is carried out on-site, and the coding or testing work is carried out at a low cost offshore center. Inter-site communications are required at the points where the project is handed off from one site to the next. The physical separation is typically perceived as a disadvantage of the distributed structure model; however the structure can instill an element of formality of communication and encourage teams to clearly define entry and exit criteria for each phase, and actually improve the quality of information [39]. This doesn't seem to do much for accelerating development – it focuses different types of work where the expertise exists, but doesn't allow significant parallel efforts.

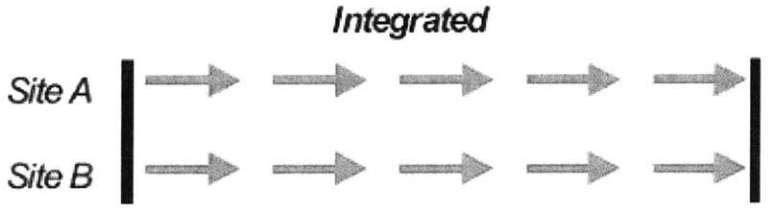

Figure 4-11 Integrated project architecture [39]

In an integrated process, the tasks are tightly integrated between sites and work passes, as often as daily, between sites. This approach is the most difficult one to manage and requires almost constant collaboration. This situation may occur due to the lack of enough suitable resources for a given tasks which compels the organization to execute the project with a virtual team scattered across multiple sites.

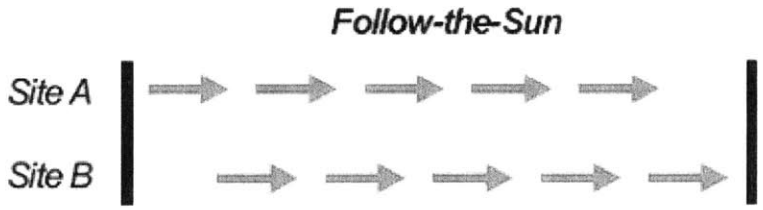

Figure 4-12 Follow-the-Sun project architecture [39]

The follow-the-sun (or "around-the-clock") methodology, as a special case of phase-based process, is the approach of attempting to create a virtual twenty-four hour workday by leveraging teams/resources in sites distributed across the globe, which allows for hand-over of the work in progress across sites at the end/beginning of their respective work shifts, thus theoretically creating a continuous engineering environment which promises to reduce the time to market. In reality, depending on the distance, region and time zone difference of the global teams, there are tasks which are more or less suitable for the application of this model than others [39]. One of the scenarios where this model has been proved to work well is in the iterative development and testing phase of an engineering project. Developers at one site do the development and another site does the required testing for the project during the night, to have results ready for the developers in the morning. Clearly, this methodology, along with the integrated process, is where the radical payoff may be for large, globally distributed workforces – but it is also the hardest to execute.

Although each type of project architecture contains different weakness, such as possible initial design flaw for module-based, hand-over mistakes for phase-based, and too many

transitions for integration-based, and management challenges for the "follow-the-sun" process, different types of project architecture should be adopted according to the character of different tasks. In reality, most distributed software projects will be implementing a distributed methodology with a hybrid of two or more models discussed above. The software engineering process is complex, and its component phases have greatly varying requirements with respect to collaboration. It is imperative for any engineering project to be planned accordingly, and it is felt that time taken upfront to layout viable project architecture will be rewarded by the reduction of related problems in the implementation phase.

4.5.3 Managing Requirement

Requirements analysis is an important sub-process in software development. Failure to define, communicate and validate requirements is the primary cause of most software project disasters. Among all the problems that contribute to a project failure, most of them can trace back to lack of effective validation of business requirements. Responses to a survey of top risks facing software projects include unstable, constantly changing requirements (66%), poor requirements specification (55%), client approval delays, changes, poor communications (42%), and reveal that 30~40% of software project effort is rework and about 50% of defects originate in the requirements[27]. Reports show that requirements errors cost U.S. businesses over $30 billion annually. How to manage requirement has become a widely recognized pain point for project leads, managers, and even CEOs and CIOs. Sadly, as illustrated in Figure 4-13, while other failure factors of software project decline in the past ten years, requirement errors caused failure remain steady. This is true of most large-scale system projects combining hardware and software, too.

[27] Source: adapted from http://www.processimpact.com/pubs.shtml#requirements

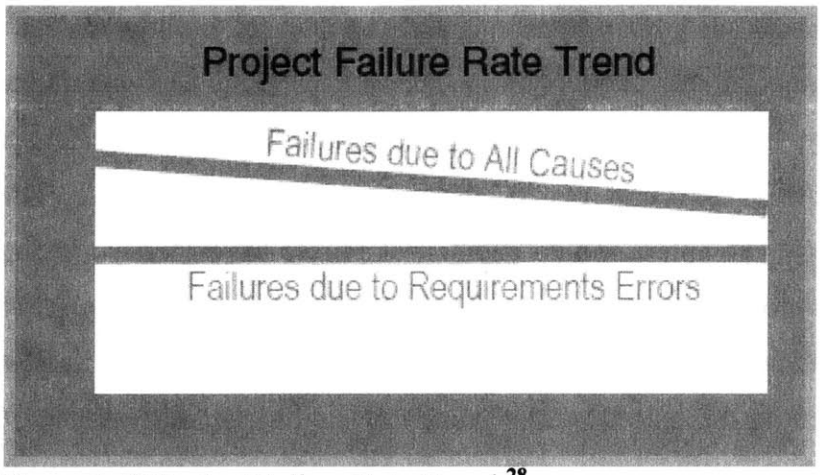

Figure 4-13 Project Failure Rate Trend [28]

In geographically distributed environments, requirements management becomes critical due to the characteristics of the distributed development - physical distance, cultural differences, trust, communication, etc. which remove the informal interactions that normally promote awareness about requirements information. Even though requirement information could be maintained in organizational and project documents, this has been shown to be very poor communication media (Curtis *et al.*, 1988). In particular, when requirements change, formal mechanisms such as documents do not react quickly enough and often news is propagated informally (Herbsleb *et al*, 2000).

Another goal of requirements management with a distributed workforce is to capture, reflect and record information that may be lost with the distance or disconnection. In the global context, requirements questions may come from all areas and all aspects of the project, in any form: hallway discussions, email chains, phone calls, interaction with local customers, etc. This goal is also related to promoting and managing global innovation.

To reduce the complexity of requirements management, agile-style methodology argues that only specifying things that get implemented in the short term and only specifying things at the appropriate level of detail. It also advocates empowering the stakeholders and users to take ownership and control of the project scope and using the requirements to help

[28] Chaos Reports, Standish Group, SEI and National Institute of Standards & Technology, Karl Wiegers

the project respond to change. It stresses the importance of iteratively evolving the requirements specification alongside the implementation, using requirements management to facilitate and improve communication between all parties involved in the project.

Use case model evolution, as a particular proposal of this agile-style requirement management throughout the iterative development process, is illustrated in Figure 4-14.

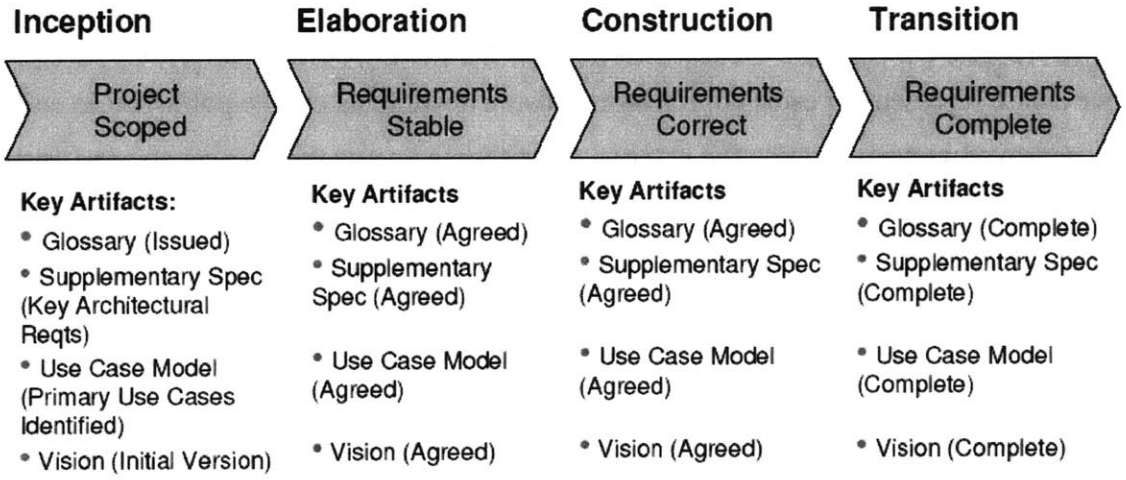

Figure 4-14 Use Cases and Unified Process Lifecycle for Requirement Management [2]

As shown in table 4-2, in each iterative stage – inception, elaboration, construction and transition, use case model that reflects the software requirement goes through all the states including Identified, Outlined, Authored, Analyzed, and Designed, Implemented and Verified, with different expected percentage.

Use-Case State	Inception	Elaboration	Construction	Transition
Identified	60%	>80%	100%	100%
Outlined	50%	20% - 60%	0%	0%
Authored	10%	40% - 80%	100%	100%
Analyzed	< 10%	20% - 40%	100%	100%
Designed, Implemented & Verified	< 5% *	< 10%	100%	100%

* A small percentage may be addressed for proof of concept purpose

Table 4-2 The use case model evolution [29]

[29] Spence I., Adapted from the Unified Software Development Process, Jacobson et al (page 358)

From the project architecture point of view, to mitigate the requirements management issue, it is recommended that headquarters or a single development site (on-site if offshored or outsourced teams are involved) controls the requirements gathering and analyses. This is to make sure that new development effort aligns with the existing architecture, and maintains similar features to an existing process or software product. Evidence also shows that true business reengineering is more likely to occur when in-house or on-site staff lead the requirements process. Meanwhile, it is important to make sure that the other teams have either onsite representatives or coordinators, ensure documents can be published in such a way, approved viewers outside the external firewall can access the documents, and ensure "clarifying questions" from the offsite team can be answered timely. Last but not least, it is critical to trace and track the dependencies among requirement-related documents and database, and to enable cross-repositories reporting.

Besides integrating requirements management in the agile iterative development process and using it as a way of tracking the project and promoting communication and coordination among different development sites, it is also critical to be able to trace project outcomes with requirements. Gotel (1994) defines requirements traceability as "the ability to describe and follow the life of a requirement, in both a forward and backward direction, i.e. from its origins, through its development and specification, to its subsequent deployment and use, and through periods of ongoing refinement and iteration in any of these phases." [32] It is a characteristic of a system in which the requirements are clearly linked to their sources and to the artifacts created during the system development life cycle based on these requirements [33], and it is one of the indicators of the success of a distributed development.

4.5.4 Software Configuration and Change Management

Carmel's six forces framework and empirical experience from both the company-led and the OSS-style global software development both emphasize the importance of using software configuration management (SCM) tools and change management (CM) or defect tracking tools as collaboration media, as infrastructure of versioning software and supporting parallel development, and as way of capturing requirements.

A correct decision on the SCM and CM toolset manages and controls software assets across the software lifecycle. Parallel development support enables developers to work on the same code or release, more easily resolve conflicts and reduce confusion, and get more done in a shorter amount of time.

Given the complexity of geographically distributed development, there are a myriad of ways to enhance the development efforts and software asset management through SCM. Here are the some attributes that a good SCM system possesses - stability & reproducibility, safety, control, auditability, governance & traceability, (Milligan, 2003) [34], and accessibility & scalability. Though all important, some of these attributes are more critical than others in the context of geographically distributed development.

- **Stability & Reproducibility**

A stable development environment is indispensable both for the team and for individual developers, True stability has two essential elements, guaranteed stable work area and the ability for individual to control when and what new code (which may not be the most stable) is introduced into a work area. When instability is introduced into a developer's environment, it can cause a downward spiral, whereby developer and team productivity dramatically suffer. This will have a negative impact on morale, cause schedule delays and quality problems. Maintaining a stable environment negates these problems and adds value across the board [34].

Among the many changes made over the course of a project, not all are for the better. Sometimes, previous versions are superior to the latest. Reproducibility in the context of software development requires the ability to reproduce the configuration of the entire development environment, including tools, tests and documentation at any point in the project lifecycle. It represents a distinct time and a detailed, broad view of the project. Reproducibility in terms of rolling a project back to a milestone or an arbitrary point in time may only yield periodic value. But when used routinely to ensure that what is being built also has been tested, its risk reduction value is incalculable.

These two attributes, as the basis of a SCM system, do not pose any additional requirement with a geographically distributed development.

- **Safety**

The primary purpose of any SCM system is to keep software assets (design models, source code, test cases, documentation, and so forth) safe from corruption, unintentional damage, unauthorized access, or even a disaster. This requires two things, secure access, meaning those who can view or change software assets must be only those persons explicitly authorized to do so, and reliable recovery, referring to the ability to recover work lost in the event of an authorized user's mistake, such as the accidental deletion or overwriting of source code files.

Safety features are the basis for risk mitigation in the software development process. If work is not secure from deliberate or accidental loss, an unacceptable level of risk to code and other critical project assets is ever-present. This potential loss can temporarily cripple a project and, at worst, derail a project for months or kill it altogether.

Creating safety in the geographically distributed development environment means having the ability to keep unauthorized users out and being able to quickly restore code that gets corrupted or overwritten which is not trivial to implement.

- **Control**

A major role of any SCM system is to help manage change throughout the development lifecycle. The system must strike a balance between enforcing appropriate controls on the overall workflow of a project without imposing bothersome constraints on individual team members.

In a geographically distributed development model, developers and teams are dispersed in different offices, in different countries, and in different time zones. Trying to integrate contributions from many diverse development teams requires a system that can control

who works on a change request, how changes flow from development to integration, and who can work on a particular development stream.

Alternatively, the SCM solution has to be flexible enough to allow a whole team to work on a common branch of code, or allow individual members of the team to work on private branches. In the case where private branches are desired, then the system requires the ability to control the flow of changes between those private branches and the projects integration area. This can be difficult in a globally distributed team, particularly if infrastructure requires distributed data or artifact repositories

Today, software reuse is important when it comes to reducing costs, and module-based architecture is crucial when it comes to promoting reuse. Therefore, it is also invaluable to have the ability to enforce a food chain or producer-consumer approach to development. This approach is taken when a team (for example, an on-site team who is developing a core component) produces deliverables that are meant to be consumed by another team (for example, an outsourced team who is writing a GUI - graphical user interface – component by calling the core library) on the project. Such components should be modifiable only by the team that produces them. Properly understood and implemented, these features create a controlled environment for development, which results in more productive developers and more predictable project schedules.

- **Auditability & Traceability**

Developing software is an iterative, complex process that requires a keen eye to understand all that is happening. Auditability refers to the ability to answer the "who, what, when, and where" questions about a specific version or configuration of the software release at any time. It also must be able to highlight the differences between releases. The answers are critical for smoothly resolving familiar occurrences such as errors in build scripts or include files. Finding the information manually consumes valuable time. Indeed, developers frustrated by an inability to interrogate the SCM system for this critical information may simply let build and testing teams sort things out - wasting even more time and greatly diminishing project efficiency.

Auditability requires that an SCM system track and record metadata about changes when they are being made geographically and distributed, and provide queries easily and quickly.

Many SCM systems offer some degree of traceability. A comprehensive definition of traceability involves the ability to identify the version of a released product deployed, in a way that makes it possible to re-create the development environment (code, use cases and testing suites) required to rebuild, test, and/or run that version. It also involves the capability of tracing backward from a specific software version to the original requirements. Incidentally, this capability is now mandated on projects involving government agencies, such as the Food and Drug Administration (FDA), Federal Aviation Administration (FAA), or Department of Defense (DoD), all of which build strict traceability requirements into their bid requests.
Similar to auditability, traceability is a way of finding out how you arrived at where you are in a project. Given the intricate complexities of software development and the high risks involved, traceability is essential. And like auditability, traceability saves time, recording metadata that would otherwise have to be done manually or probably not at all.

- **Accessibility & Scalability**

As the foundation for the entire development platform, the SCM system must support geographically dispersed teams in various forms – co-located teams, teams in remote sites, mobile users, telecommuters and/or outsourced partners, and support geographically distributed projects of any scope – while scaling up to large teams from start to finish, it must not impose a burden on small teams.

A global accessible and scalable SCM system should be configurable and functional when few controls are necessary and be versatile to manage growth. The presence of off-site contributors adds significant stress to the SCM environment, because of the need to coordinate and manage distributed collaboration, whether centrally or via replication. Scalability also implies that reasonable performance standards should be achieved.

Increasing demands on an SCM system should not compromise reliability or impede basic operations. Figure 4-15 illustrates such a flexible and scalable requirement to SCM systems.

- Choice 1-1: "Remote access to a centralized repository"

In this approach, users connect to the centralized repositories (normally located at headquarter) via web interfaces or light-weighted clients. This approach has the lowest TCO (only the site with central repository requires admin staff, end user may only need browsers) and highest potential security (all the critical data can be put behind a firewall), but does not scale very well for large projects. It can be used to extend legacy application outside the firewall. It is suitable for part-time remote access, telecommuters, off hours (nights and weekends) access, working from home, consultant, moving between different customer locations, full-time remote access, users working out of small satellite offices, users who are not co-located, remote but small development teams, "Off Shoring" and "Out Sourcing". This approach is usually used in conjunction with other virtual workplace software, such as Citrix[30], pcAnywhere[31], Remote Desktop Connection (RDC)[32], VNC[33], etc, to provide additional accessibility of the central repositories.

- Choice 1-2: "Replicated repositories"

In this approach, multiple offices and multiple development teams each have their own copy of the repository and all the copies are synchronized in certain intervals. It is commonly used between big development sites or centers, and used with development teams that came from acquisitions. Sometimes, it is also chosen as an alternative to data backup and/or disaster recovery. Though highly scalable, this approach suffers from a high TCO since all the replicated sites requires installation, setup, maintenance, and other administration overheads.

- Choice 1-3: "Broker-mediated remote access"

[30] http://www.citrix.com
[31] http://www.symantec.com/home_homeoffice/products/overview.jsp?pcid=pf&pvid=pca12
[32] http://www.microsoft.com
[33] http://www.realvnc.com

This approach aims to provide a trade-off between centralized repository access and replicated repository access. But the broker-mediated technology is still in its infancy, and has not had proven evidence in supporting the values over the other two approaches.

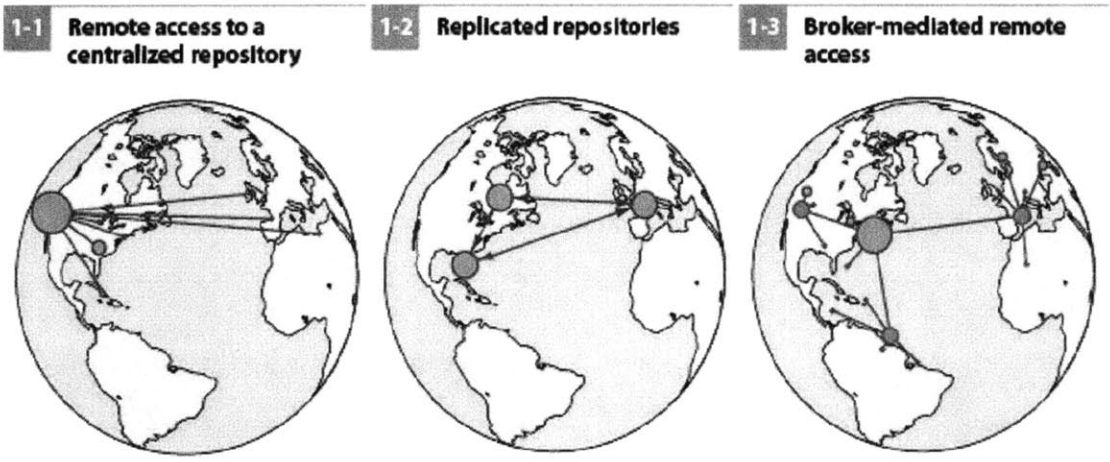

Figure 4-15 Distributed Access to Development Assets[34]

The author argues that software configuration and change management system, comprised of multiple services, including code management, parallel and distributed development management, build and release management, test environment management, should be a centralized policy and should be independent from the development organizations and teams. Attributes mentioned above - stability & reproducibility, safety, control, auditability, governance & traceability, and cost-benefit with various technologies, such as server-based remote access approach and replicated access approach, should be considered carefully, as this recommendation may have significant IT infrastructure implications.

4.5.5 Managing Build and Release

Software build is the process of creating the application binaries for a software release, which is normally done in a periodic manner. Build management has typically or traditionally been viewed as a just a subcategory within software change and configuration management (SCM), but has gained substantial attention in recently years. Distributed development using offshore, outsourced, and internal resources demands more effective

[34] Source: "Don't Rely On Just One Method For Distributed Access To Development Assets", Forrester, 2005

build management because of the need for better communication, collaboration, and coordination. Regulatory compliance initiatives resulting from legislation mandate auditability, consistency, and reproducibility from development to production, necessitating effective build management. The complexity of emerging new development paradigms, such as service oriented architecture (SOA), require close coordination across business and software or IT groups and a higher level of quality, change, and build management to deliver services. IDC[35] thus advocates that build and release management should be considered a distinct category and process within application life-cycle management (ALM), with a similar level of focus to other ALM phases. Not doing so can jeopardize successful software implementations.

The author agrees that build management, as one critical phase in the software life-cycle management, requires a focus on consistent processes and organizational strategies as well as appropriate functional capabilities to automate core build management functions. These functions include rigorously managing build processes across multiple projects, auditing release contents, coordinating parallel tasks and systems, applying consistent configurations, reusing and replicating build processes, and integrating with other ALM systems, especially the software configuration and change management tools.

4.6.6 Assuring Quality

Software quality can suffer in a geographically distributed development environment when source code is contributed by teams and individuals inherently (distance, culture, different organization and standards, etc.) disconnected in nature. The discrepancies in understanding introduce delay and may necessitate substantial re-work during integration [36]. However, only a few case studies report measurable impact of distributed development on product or project attributes, such as quality, cost, productivity and time-to-market. A study of modification requests (MRs) by at Lucent Technologies showed that single-site MRs took in average 5 days to complete, in contrast to 12.7 days for multi-site MRs (not related to size or number of changed modules, but to number of people involved) (Herbslebet al. 2001), which suggests that delay is due to the additional time it takes to

[35] www.idc.com

resolve an issue when more than one site is involved. There is also a report on reduced productivity due to asynchronous communication (Boland et al , ICSE workshop, 2004), and on improved efficiency in collocated teams during initial validation activities of over 50 percent (Ebert, 2001).

It is ironic that, while company-led geographically dispersed developments often suffer from "quality" issues, especially with offshored or outsourced teams, some of the open source software projects, as cases of global distributed development, have gained the reputation of "quality" with reduced defect-density. What Raymond described as "Linus' Law" of testing - "Given enough eyeballs, all bugs are shallow" or as J. Dutky put it "Debugging is parallelizable", may contribute to the quality assurance in OSS practice. OSS-style release and test strategy stresses the importance of having users and listening to users. "Treating your users as co-developers is your least hassle route to rapid code improvement and effective debugging" [10]

What exactly can company-led software learn from the OSS-style "release early and often, assuring quality by having enough eyeballs" strategy is not exactly clear at this point. But more and more companies are trying to mimic this strategy by utilizing tech preview, early beta, or in-house deploy and to allure more people to use the software as early as possible.

Another common trend of testing in company-led geographically distributed development is to outsource the testing function to a 3rd party or internal lab in another country. Testing is recognized as a specialized functional area within the development lifecycle that is critical to software success. Experts advocate that testing be conducted independently, and industry best practices include setting up independent testing teams that function autonomously. Some of the benefits of outsourcing testing include lower cost, greater efficiency, improved time-to-market, focus on core business, enhanced coverage and quality of delivery, skill generation and training, staffing flexibility and technological advancement.

Combining the OSS-style testing approach, the "Test, Test, Test" strategy advocated by agile development, and the trend of outsourcing testing function, the author proposes test-

driven development (TDD) methodology to address the quality challenges in global software development. The basic idea is to create test suites of the different modules prior to development, share them across all sites and use them as a medium of communication and as lightweight specification that guides subsequent implementation between development teams. For example, changes in these test suites may be used to reflect changes in requirements or in module behavior. The shared understanding that would result from this should help preserve overall consistency. Test suites are unambiguous; from the perspective of global development, which means that a test suite should be interpreted in the same way by different remote development teams [36].

The idea behind Test Driven Development is quite simple. Before implementing a new functionality, first write executable unit test cases for it. Once you have written enough test cases to thoroughly check the new feature, write the actual code to pass these test cases. The test cases thus become a mini-specification of the functionality that was implemented. This then goes continuous as more and more functionality is added. At the end, one has not only the complete implementation, but also an efficient regression test bed capturing all the new functionalities that were added, which can be used to identify if subsequent changes break anything in the existing system.

Having a regression test bed is the basis for ensuring the quality of distributed development. Any development made at one site should not break the regression test, and any outsourced testing function can use it as a basis to develop more sophisticated tests, such as integration test, performance test, localization/globalization test, etc. Another advantage of Test-Driven Development is to allow the implementation of a single use case from the end user interface (client – normally a GUI component) to the core functionality (server – normally a C/C++ or Java component), which makes early releases (beta versions, tech preview version or even bits from iterative milestones) and getting user inputs possible.

In summary, the author argues that in a geographically distributed development, testing should be taken into account from the beginning and integrated in the whole software life-

cycle using test-driven development methodology. The goal of this approach is to use test-suite as a channel for collaboration across multiple sites, a way for catching regression introduced during software life-cycle, a driver for enabling early adoption, and a basis for outsourcing the testing function.

4.7 Summary

In this chapter, the author has proposed a five-force framework based on Carmel's six-force framework to address both the strategic and tactic problems when dealing with geographically distributed development and global teams. Mastering and utilizing the five forces - strategic vision and management skills, organizational structure and team building, collaborative technologies, development methodology, and software life-cycle management, can ultimately leash out the potentials of distributed development in a global context.

In the next chapter, the author will present a real case study on company-led geographically distributed development practices to validate the revised framework.

5. Case Study

The case study used in here is a geographically distributed development by IBM Rational Software[36]. It is presented to illustrate some of the main theoretical and practical points made throughout the thesis. This is a software project called "iTeam" during which a global team (Figure 5-1) is delivering a J2EE (Java 2 Platform, Enterprise Edition) application.

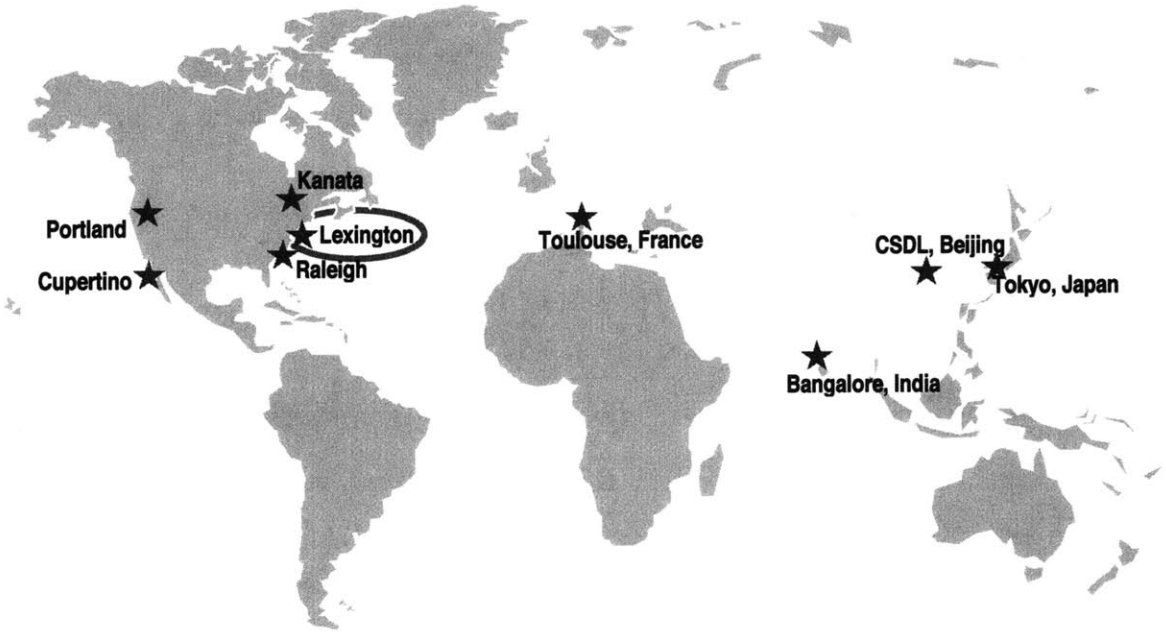

Figure 5-1 The global IBM Rational team of delivering "iTeam" project

5.1 Background Information

IBM Rational Software was founded in 1981 by Paul Levy and Mike Devlin to provide tools to expand the use of modern software engineering practices. It markets and supports a comprehensive solution for developing and managing complex software systems. It provides an integrated family of open, industry standard tools that spans the critical phases

[36] http://www-306.ibm.com/software/rational/

of the software development process. It also provides best practices and services for developing business applications and building software products and systems, including embedded software for devices such as cell phones and medical systems.

After the acquisition by IBM on Feb. 2003, Rational software became one of IBM's six software brands (others include Lotus, DB2, WebSphere and Tivoli). Since then, Rational Software continues its leadership in software development, application lifecycle management tools, project and portfolio Management and SOA infrastructure solutions, which helps IBM hold 25.4% market share in Application Development and Project and Portfolio Management segments, exceeding the second leading competitor by 15%[37].

5.2 Strategic Investment in Global Development

IBM is a pioneer in global software development. In recent year, IBM has been undertaking a series of initiatives to transform itself from a multi-national company to a true global company, including setting up a global software work communities. In particular, IBM's software business and employee base is increasingly global and today includes 15,000 customer facing resources around the world. It continues to shift its investments to put more skilled specialists and technical sales resources directly in front of the customers, especially in high growth countries such as India, China, and Brazil. It has 26,000 software developers in over 50 labs around the world. This distributed resource model allows it to improve its development productivity and, at the same time, improve its competitiveness by growing skilled resources closer to the customers, providing a significant competitive advantage over others in the IT industry.

Meanwhile, IBM has been heavily invested in infrastructures, platforms and tools to tackle the challenges and complexities of geographically distributed development. In June, 2005, IBM Rational Software announced the latest version of its software development platform,

[37] Source: Gartner Dataquest, "Market Share: Application Development and Project and Portfolio Management, Worldwide, 2005, Table 2-1," L. Wurster and F. Biscotti, May, 2006.

known as Release 7 (code name "Baltic") of Rational Team Products, with a focus on "trying to accelerate global software delivery."[38]

Global development represents part of a more general strategy to become a true global company, with global presence and proximity to customers as the main business factors driving global software development at IBM. At business unit levels, however, this phenomenon tends to be more cost-reduction and resource driven. For example, workforce in Lexington (Headquarters of Rational) represents greatest concentration of product expertise, density of senior talent and understanding of market niche and competition. Its success has been however significantly challenged by the fact that the workload was often overwhelming and headcount expenditures too expensive to further expand local workforce. Executive managers have also mentioned the disadvantages associated with having product knowledge that is too localized and single center of competency being too risky for global ecosystem.

This illustrates that, while additional benefits are recognized, cost considerations are essential in driving global expansion of the workforce. The escalating relative importance of cost factors in a context of intensified competition and commoditization of software is also closely associated with the increasing popularity of open source software development that relies on the use of a globally free workforce.

For IBM Rational "iTeam" project, telecommunication infrastructure, as backbones for collaboration, is not much of an issue since all the major global sites have high speed internet connections. At the end of 2004, IBM completed a worldwide project to migrate all IBM facilities to Ethernet for network connectivity. This means that Ethernet is the standard form of network connectivity in the offices and common areas within IBM. The Ethernet technology provides the maturity and performance to support IBM's standard services, and provided IBM the opportunity to standardize the campus network architecture for all IBM sites worldwide. Consistent global architecture provides the underlying framework for emerging IT technologies, like Voice over IP, Video over IP, and Wireless

[38] Lee Nackman, VP of product development and customer support for IBM Rational RSDC (Rational Software Development Conference), Orlando, Florida, , June 2006,

LAN. With built-in Ethernet adapters available in standard workstation platforms, all one needs is an Ethernet cable to connect to the IBM network.

Wireless LAN technology, also known as WLAN, Wi-Fi and 802.11 Wireless, provides a flexible network environment that greatly benefits IBM's mobile population and devices that are hindered by cabled connections. By providing network connectivity to workstations without being cabled to a network port, WLANs can enhance productivity and collaboration capabilities. WLAN is installed as a complementary service to standard wired Ethernet network connectivity, since Ethernet still provides the underlying infrastructure for WLANs and offers the performance, maturity and support across all system platforms needed to support emerging technologies and applications.

IBM's strategy is based on leveraging investments into Wireless LAN to expand coverage, while delivering a secure and reliable service. As WLAN technology continues to mature and standards and products evolve rapidly, IBM will continue to update the standard solution to incorporate enhancements to the 802.11 protocols, security, management, and service levels. While convenient, Wireless LANs (WLANs) can also pose potential security threats to the IBM network. In order to maintain a consistent architecture globally and ensure protection of the IBM environment, all WLAN installations must be performed through the IBM Standard WLAN Offering. Any business-justified requests for WLANs outside the standard offering must be approved through the WLAN Exception Process prior to installation.

An old joke used to proclaim I.B.M. standing for "I've Been Moved," remarking at the company's strategy to relocate employees around the world as needed. Today, some quip that I.B.M now means "I'm By Myself." A growing number of IBM's workforce is now considered mobile, often with only 50% of our population physically present at an IBM site. Individual developers gain access to the data they need for their job via VPN (Virtual Private Network) with secure applications that allows them to "tunnel under" the company's safety precautions (firewalls, etc.) and enter its intranet.

Overall, IBM employees at all levels connect and work with colleagues across the company. Cross functional teaming is paramount to delivering to the marketplace with teams frequently forming around specific projects and client requirements and disbanding when objectives are met. Employees in general expect to work with colleagues outside their immediate organizational structure, country and culture whenever their expertise is required to solve a problem or deliver on a customer obligation. This is inherent to one of IBM's core values[39]- Trust and personal responsibility in all relationships.

IBM's culturally diverse workforce and its global business environment make it necessary that cultural differences are recognized and taken into consideration when managing people, doing business and working with multicultural teams. Recognizing cultural differences is the necessary first step to anticipating potential problems and opportunities for business encounters. In order to go beyond the awareness and to create successful interaction, cultural differences need to be open to discussion and flexibility is required in order to adapt to each other. The managers and some team members in "iTeam" project have taken IBM's diversity training programs or workshops that aim to heighten the awareness of cultural biases, increase sensitivity to other cultures and provide tools to work effectively within a multicultural environment. These trainings help managers and employees understand the importance of cultural competence in the context of IBM's globalization, explore and frame key cultural differences on national, functional, and interpersonal levels, and acquire specific cultural skills and strategies for bridging cultural gaps and resolving salient issues. The primary way to learn about managing and working with a multicultural team is through experience, as "iTeam" members summarized.

Resolving difficulties related to cultural differences still represents one of the main challenges to "iTeam" managers and team members. This is particularly the case in the context of collaborations with teams located in China. The tendency of the Chinese technical staff to provide quick and almost automatic "Yes" responses to all questions and demands and withhold further comments or questions until it is too late and conflicts have already escalated, represent one of the major challenges in communication across the

[39] http://www.ibm.com/ibm/values/us/

borders. According to one manager's experience, meetings can take a lot longer when various nationalities take part. In one culture, a position can be stated at the start of the meeting and moved forward from there, whereas in the another culture, it's more likely that everyone will add their opinion and express agreement on the current position before moving towards a resolution.

Having co-workers in another region or country evidently causes climate change to the local teams. For example, the Lexington team has the tradition of "holiday potluck" during which folks sit down and enjoy the annual face-stuffing event. This event used to run from noon to about 3PM in the afternoon. But in the recent two or three years, folks are all gone even before 1:30PM. The pressure of having to get back to the remote team members is identified as the major concern that has cut short the local event. More and more people are checking emails and responding to co-workers in other sites or countries during nights or vacations. In order to cope with the time zone difference, the Lexington team has to call early meetings if the number of participants in remote site is dominating (For example, 8AM EST time with India Software Lab, ISL). In the opposite situation, representatives from remote sites have to sacrifice their after-work hours to dial in to meetings (For example, 10AM EST time with China Software Development Lab). From time to time, travel is also necessary to allow the acquaintance between the team members and there normally are planned trainings during visit. With budget constrains, other types of travel are also used. For example, during the stay in Beijing to attend the IBM Software User Conference in China, a conference-attendee from Lexington and the author managed to meet with the local sales, service and development teams. Web conferences are used on a regular basis to enable global participation. Some meetings or presentations are recorded using Camtasia[40] to enable teams in other time-zones to re-play later. Another practice IBM Rational "iTeam" used for team building is the requirement of posting member photos and name pronunciations to the IBM Bluepage directory service. Seeing a remote team member's face and being able to call an unfamiliar name make a big difference in team bonding.

[40] http://www.techsmith.com/camtasia.asp

Distributed team members need personalized access to project knowledge bases, discussions, business processes and the team members essential to their work. Establishing this collaborative environment provides increased responsiveness among colleagues, speeds group decision making and keeps team members involved and up-to-date with project activities. "iTeam" project has used the IBM central topology for collaboration and innovation, which provides clear guideline on when and how to use collaboration tools (CT). The rich set of collaboration tools include tools for meetings and events (Notes 7 calendar, Audio Conference, Web Conference, Video Conference, Webcasts and Conference Rooms), for Teams (Wiki, QuickPlace, TeamRoom Plus, TeamRoom Pro, TeamSpace, Knowledge Café), for Communities (Blogs, CommunityMap, Forum and ActionNet), and for Individual Initiatives (Instant Messages, Notes email and Bluepages). The availability of a rich set of collaboration tools ultimately improved the communication among global teams and helped create innovation. Wiki, in particular, has been used extensively. "iTeam" project wiki contains information on team members and responsibilities, on project status, bibliography, development and debugging tips and guidelines on how to get things done, such as code reviewing, writing tests, submitting defects, triaging service request, etc. It also contains various test results – performance benchmarks, JUnit tests, and code coverage report. This wiki page has brought the global project team together with contributions from all the team members and has become the single place for sharing information and knowledge, keeping work visible, and promoting innovation.

To further promote the collaborations among its global workforce, IBM has set up the Community Source program by learning from some of the tried-and-true software development practices from Open Source Software. The goal is to re-architect some of its most valuable software assets and by converting them from tightly-coupled offerings to a collection of integratable loosely-coupled component-based applications. The componentization enables more code reuse and more flexibility in products to take advantage of the mature of SOA and web services technologies. However, the way of developing components is very different from the way of developing an integrated product.

A key to changing people's thinking and behavior is to optimize the development environment and transparent among the global workforce.

The requirement for better worldwide collaboration and transparency – and the sharing of code and ideas implicit in that visibility gives birth to the Community Source program – a model borrowed from many approaches – tools, processes, collaborative mechanisms and even some culture, found in Open Source Software. The biggest difference between Open Source and Community Source is the definition of "Community". For now, there are no outside developers working in the Community Source projects, and IBM does not intend to release software from Community Source out into the Open Source community. Community Source aims to improve internal development efforts, and has been in existence for more than 2 and half years. As of June, 2006, it has involves over 10% of IBM's 25,000 in-house developers, and involves more than 100 projects.

Some of the "iTeam" project team members have been actively involved in the IBM Community Source initiatives and other Open Source Software projects. For example, some members contribute to an effort of creating a reference implementation (RI) and Technology Compatibility Kit (TCK) for the Workspace Versioning and Configuration Management (WVCM[41]) standard proposed to Java Community Process (JCP[42]).

5.3 Global Development and Delivery

The company's strategic investment in the geographically distributed development, which includes obtaining global workforce, maturing the telecommunication infrastructure, gearing product directions to support GDD, stressing the importance of working in a diversity environment, and supporting a rich set of collaborative technologies, has formed the foundation for the "iTeam" project.

[41] http://www.jcp.org/en/jsr/detail?id=147
[42] http://www.jcp.org

Next, the author will study how the "iTeam" project is delivering the software in a geographically distributed environment.

5.3.1 Disciplined Agile Methodology

The organizational model of IBM Rational "iTeam" project is a combination of intra-organizational hierarchy model and network model.

	Requirement & Analysis	Design & Architect	Construction	Unit Testing	Functional & Performance Testing	Project Management
Lexington, MA	100%	80%	60%	60%	50%	80%
Raleigh, NC		10%	15%	20%		10%
Bangalore, India		10%	15%	20%		10%
Beijing, China					50%	
Tokyo, Japan			10%			

Table 5-1 GSD segment and discipline alignment in "iTeam" project

As shown in Figure 5-1 and Table 5-1, the "iTeam" global project team contains business headquarter (Lexington, MA) that focuses on requirements, architecture and high-level design for all projects. As these elements are created, project specifications are communicated to Raleigh, Bangalore, Beijing and other sites for development and component testing. The Lexington team is responsible for new feature development on existing projects and all new project development, while the Bangalore team is tasked with new .Net client development and maintenance development, and the Raleigh team is tasked with new Eclipse client development. Software translation and localization related work is done by the Tokyo site. Beijing site (CSDL - China Software Development Lab) is responsible for regression testing, platform testing and I18n/L11n testing. Defects found are returned to Lexington, where validation, function and performance tests are executed. All unit level testing is done by the development teams. Requirement maintenance, Project and portfolio management, as a core competency, is handled from Lexington, where all components of application and resource portfolios are tracked and monitored.

The "iTeam" project organizational structure with a combination of intra-organizational hierarchy model and network model is very typical in company-led global software development. The software architecture of this project is highly module-based with clear layers among back-end core functionality (legacy C++) code, middle-tier J2EE components (Java), and client applications (for .Net user and for Eclipse users) utilizing the same web server interface. This module-based architecture makes it possible and easy to delegate tasks among different sites. The development methodology adopted by IBM Rational Software "iTeam" can be summarized as "discipline at the global level, be agile at the local level" (Liu, 2006) [37]. This aligns with IBM's orderly way to step away from its traditional top-down development and its attempt to quickly harness individual creativity and drive high-quality bottom-up by replicating Open Source Software's self-organizing principals.

At the global level, IBM Rational Unified Process (IRUP), a formal process of iterative development methodology, is what the global software workforce follows. IRUP provides a disciplined approach and delivers software best practices via guidelines, templates and tool mentors for all critical software lifecycle activities. Its goal is to ensure the production of high-quality software that meets the needs of its end-users, within a predictable schedule and budget. The process can be described in two dimensions, or along two axes, as shown in Figure 5-2.

- The horizontal axis represents time and shows the dynamic aspect of the process as it is enacted, and it is expressed in terms of cycles, phases, iterations, and milestones.
- The vertical axis represents the static aspect of the process: how it is described in terms of activities, artifacts, workers and workflows.

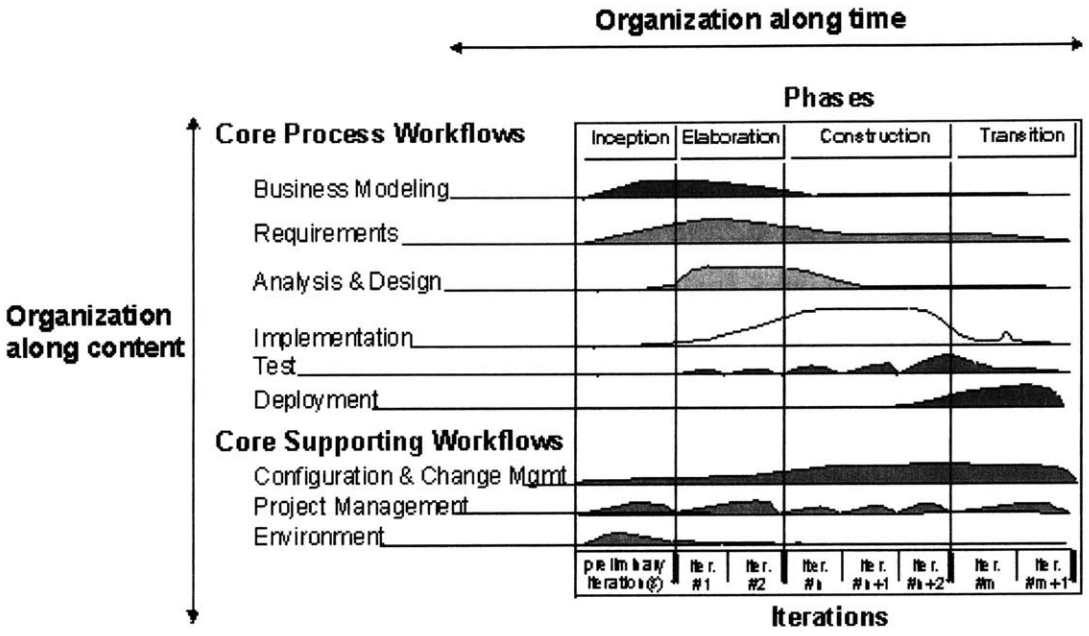

Figure 5-2 IBM Rational Unified Process

Iteration is inherent. Software architecting and software development is not a linear single pass effort. Iterations are one of the essential features of design and development projects. "Developing software iteratively" is one of the six best practices IRUP defined and recommended. As presented in system project management class (EDS.36), there are planned iterations and unplanned iterations. Iteration can be accelerated through information technology (faster iterations), coordination techniques (faster iterations), and decreased coupling (fewer iterations).

IRUP is also integrated with IBM Integrated Product Development (IPD) process, a management system designed to optimize the development and delivery of successful products and offerings. IPD emphasizes on strategic alignment, market penetration and value returns; whereas IRUP focused on the execution of the software development. IRUP is supported by tools, which automate large parts of the process. They are used to create and maintain the various artifacts-models in software engineering process: visual modeling, programming, testing, etc. They support all the bookkeeping activities associated with the software configuration management and change management. Rather than focusing on the production of large amounts of paper documents, this unified process emphasizes the

development and maintenance of models - semantically rich representations of the software system under development.

Another emerging software design approach called "Outside-In Design (OID)" is also used at global level as an attempt to increasing the consumability of product offering and delivering products that are more effective and have immediate value for the customers. . Consumability involves having functions that can be used effectively: right design, installation, configuration, fundamental quality, information, integration, maintenance, and upgrade. OID focuses on everything from the base architectural decisions about an offering to the user experience, and takes IRUP/IPD to a new level by focusing on the challenges and opportunities regarding consumability. From the implementation point of view, system goals ought to be set around consumability, and OID is a unification of four customer-driven best practices - Business Scenarios, User Engineering, Visualization, and Milestones – as shown in Figure 5-3, which helps to deliver the right products to the right customers at the right time.

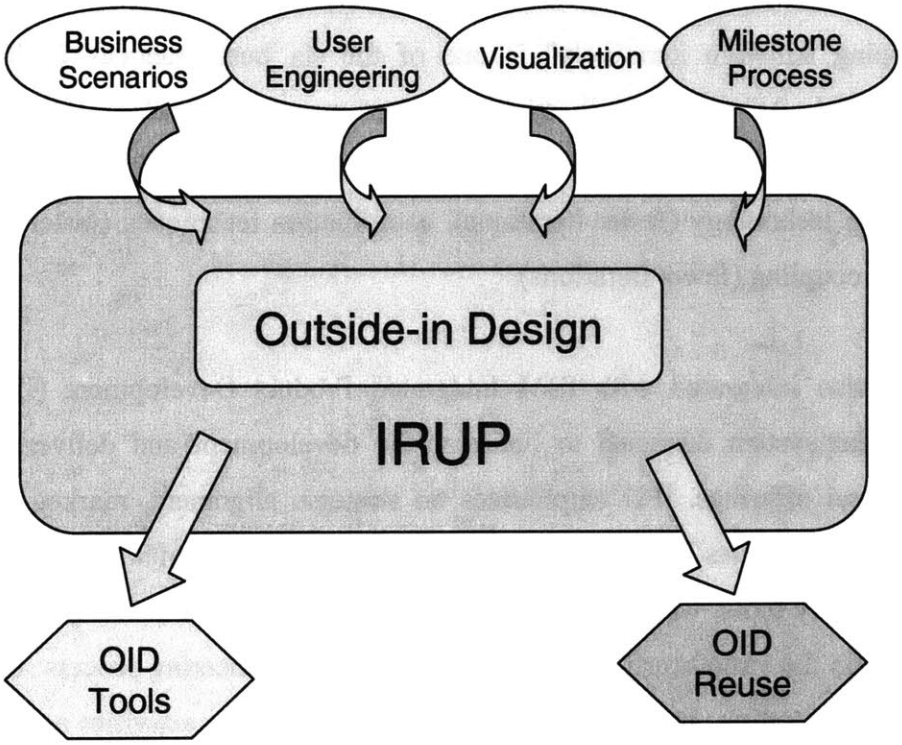

Figure 5-3 IBM Outside-In Design Development Methodology

At the local level, an agile-style (frequent code review, test-driven, daily build, etc) iterative development model is executed. The "iTeam" project has used iterations, at 6 week intervals, to set specific milestone deliveries and to track how much can be realistically accomplished. The six week periods define objective accomplishment and milestone deliverables, and force the team to evaluate effectiveness and productivity. There are special iterations that aim to solve particular issues revealed in previous iterations. For instance, there is an iteration that is dedicated to improving productivity during which training and documenting take priority over implementing new functionality. Evidence shows that the shorter iteration, while imposing urgency, provides an honest feedback mechanism which more accurately indicates the project status and is faster in nailing down and resolving issues among multiple development sites.

Some of the agile development techniques, such as test-driven development and short iterations, however, cause some conflicts with module-driven development. Due to the complexity of the component structure in this J2EE application, the team has faced the issue of effectiveness and efficiency in a global development context. The original proposal is a strict component-based (or module-based) development with well defined component interface contracts, and has separate teams, with each team, by and large working on only a small subset of the components. An alternative approach, experimented in a short period time by the client team and server team, is the methodology of "vertical development" during which each client team member is responsible for a vertical slice of functionality, all the way from client libraries to server implementation. The development tools required in different layers of the software system, in particular, have been problematic and became the No.1 cause for loss of productivity.

The failed experiment shows that "vertical development", while facilitating better overall understanding of the system with immediate learning of the entire set of technologies and sub-systems, and providing better end-to-end semantics from client layers to server layers, suffers from managing additional complexity including the overhead of learning tools required at different layers, which is extremely difficult with geographically distributed

teams. Component-based development has thus been followed and executed, and has proven to be a success.

The "disciplined agile" development methodology adopted by the "iTeam" project is evidence that a flexible development process is desired in a distributed development.

5.3.2 Software Configuration and Change Management

IBM Rational ClearCase[43] and ClearQuest[44] are the two key software configuration and change management tools that enable the geographically distributed development of "iTeam" project.

The "iTeam" project has used "RATLC", a multi-site implementation of IBM Rational ClearQuest, a workflow management solution, to automate and control software development processes among multiple sites. With ClearQuest MultiSite[45], developers at different locations can use the same database repository with its own copies, or replicas. At any time, changes made in one replica can be sent in update packets to other replicas. RATLC is comprised of eleven replicas (see Figure 5-4) located in North America, India, France, and China. The application, deployed in a heterogeneous environment using Oracle, Microsoft SQL Server, contains over 250,000 records and supports more than 2,700 developers across the IBM Rational division 24 hours a day, 7 days a week.

[43] http://www-306.ibm.com/software/awdtools/clearcase/
[44] http://www-306.ibm.com/software/awdtools/clearquest/
[45] http://www-306.ibm.com/software/awdtools/clearquest/multisite/

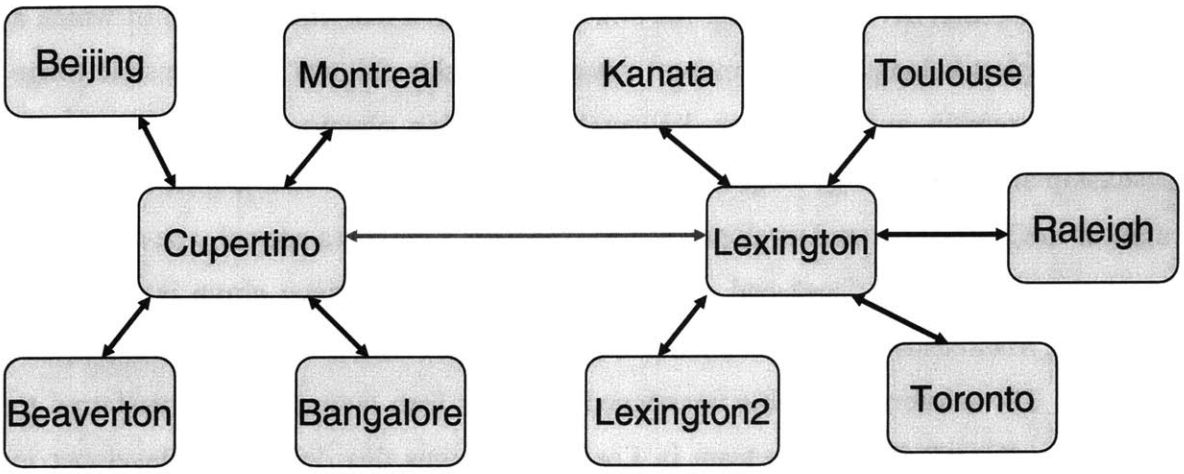

Figure 5-4 RALTC Replicated Topology

RATLC uses a multiple-hub synchronization pattern to define which replicas exchange update packets and the direction of exchanges. The hub replicas (Cupertino and Lexington) exchange packets with all spoke replicas to increase efficiency for the spoke and branch replicas. Many factors have been taken into account to define a flexible synchronization schedule. Such factors include the rate of development at different sites, the connections among sites, the time zone differences, and the coordination with backup schedule. On average, update to RALTC at one replica is synchronized to other replicas at 5 minutes intervals. The automatic replication and synchronization of RATLC repository enables distributed teams on multiple distributed sites to manage and track development activities. It also maintains data integrity by resending information in the event of network failure and automatic recovery of repositories in the event of system failure.

An issue with this distributed repository scheme is how to manage concurrency control across sites - how to deal with two developers at different sites trying to modify the same code or change the same record at the same time. To deal with this issue, ClearQuest MultiSite uses a concept called ***mastership*** - an exclusive right for a site to modify an object. If site A masters an object, such as a ClearQuest record, no other sites can modify the record. By default, an object, such as Change Request or Component, is mastered by the site in which it was created. Mastership can be given away, or in some cases, taken, but only one site masters a specific object at a time. Users and Groups, as special record types

in ClearQuest, also have mastership. RATLC implements a mastership model in which a Change Request belongs to a component whose mastership follows its responsible triage group's mastership and a Change Request's mastership always follows its owner's mastership after this Change Request is assigned to a developer. For example, if a quality engineer in China Software Development Lab discovers a defect (a special type of Change Request) in component "back-end server" whose responsible triage group is located in Lexington, Massachusetts, he/she enters a Change Request record in RATLC from replica "Beijing". The mastership of this unassigned defect is then automatically transferred to replica "Lexington". The triage team in Lexington assigns this defect to a developer in Raleigh, North Carolina. As a result, the mastership of this defect is transferred to the "Raleigh" replica so the developer there can work on it. After the defect is resolved, its mastership is transferred to "Beijing" replica during the process of changing owner from the developer to the submitter - the quality engineer in Beijing, China so the fix to this defect can be verified and closed.

By distributing independent, but related, development efforts across multiple cities, nations, or continents, RATLC serves the centric repository for requirements management, risk management, defect tracking/verification, customer escalation management, and developer task management. To the "iTeam" global team, RATLC is not just a tool, but a process of how the team works. RATLC defines the required states in the process workflow to apply the right level of control. A specific state determines what action can be performed on the specific object, and what state transitions can be done from there. RATLC also defines and controls the correlations between roles, rules, and tasks, and reflects the needs of a team and the different roles within the team, by allowing the definition of various authority groups. Figure 5-5 and 5-6 demonstrate the states and actions defined in the life-cycle of a change request, and the role-based process.

Release requirements, as a means to tracking high level release requirements per software release, are also controlled by RATLC as a series of state-transition processes. By managing requirements in a single but global accessible repository, it provides the

geographically distributed team a consistent view of what is expected out of a software product in a given release.

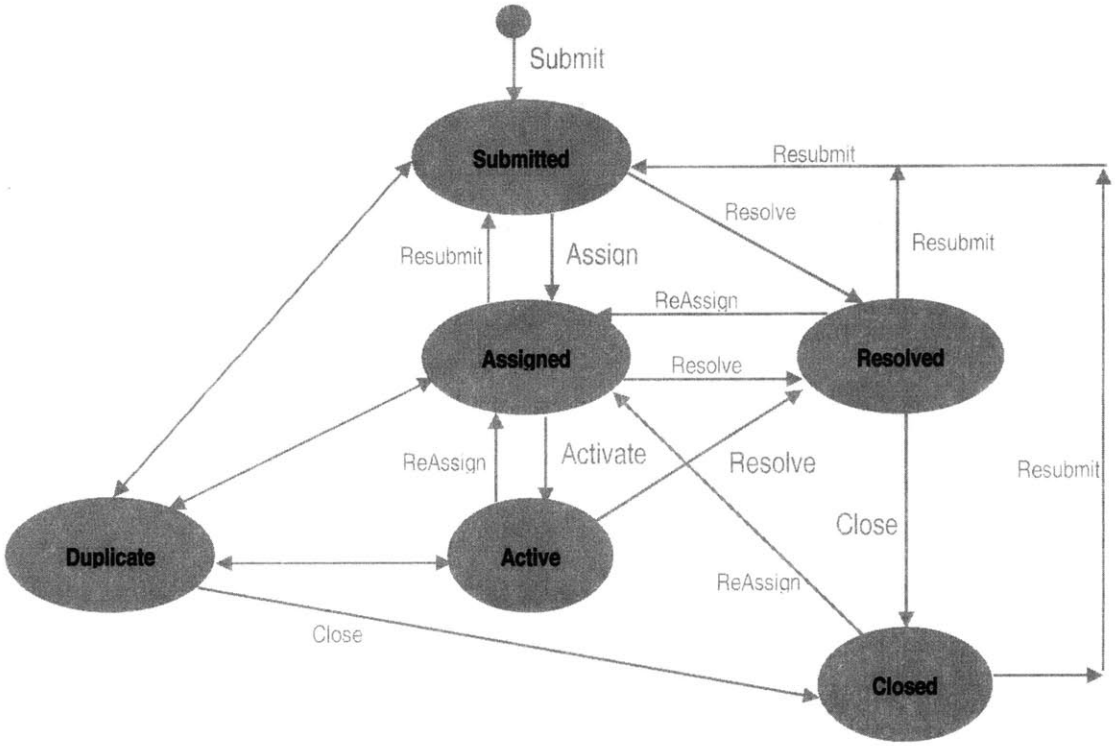

Figure 5-5 The Life-Cycle (states and actions) of Change Request in RATLC

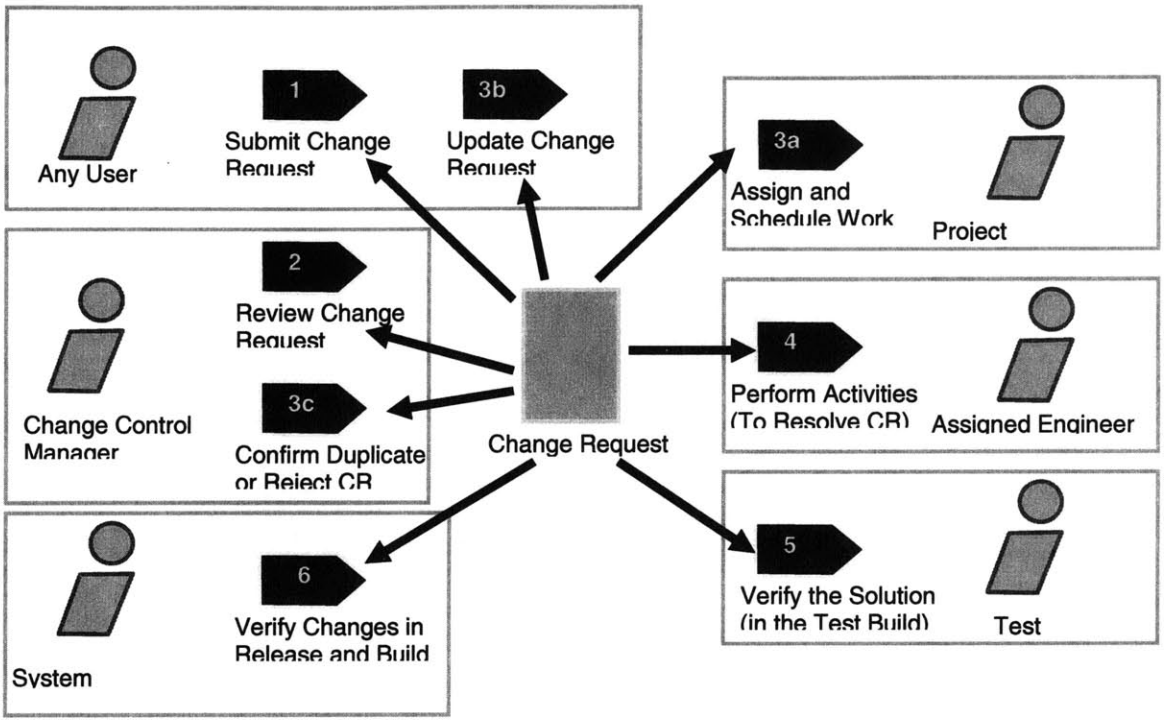

Figure 5-6 Role-based Process Management in RATLC

The "iTeam" project has used IBM ClearCase MultiSite (CCMS) and ClearCase Remote Client (CCRC), in conjunction with IBM ClearQuest MultiSite (CQMS) and ClearQuest Web interface, to implement a flexible topology of configuration management and workflow management. The integrated solution combines the replicated repository access approach and the remote access to a centralized repository approach to balance the need for scalability and reduced total cost of ownership. For the software configuration management, a replicated approach is used at two development sites (Lexington and Bangalore). Other sites (and individuals) remotely access a centralized repository via wide area network (WAN) clients and through Web clients. For the change management repository, RATLC, all replica sites also have dedicated web servers to provide Web access.

With ClearCase, CCMS and ClearQuest, CQMS as the basic infrastructure, the "iTeam" adopts Unified Change Management (UCM), the IBM Rational's "best practices" process, for managing change from requirements, coding to release with a defined, consistent, repeatable, activity-based change management process. UCM simplifies development by

raising the level of abstraction to manage changes in terms of activities, rather than manually tracking individual files. This enables the global development team to easily identify activities included in each build and baseline. With UCM, an activity is automatically associated with its change set, which encapsulates all project artifact versions used to implement the activity. This activity is also mapped to a Change Request defined in RALTC – the workflow repository. This ensures that all code and content changes are delivered and promoted accurately, and can be traced back to the Change Request. For example, from any replica of the RATLC database, the global team can either use native interface or web interface to look at detailed information (description, configurations, customer, test logs, requirement, resolution, etc) of a particular Change Request, as shown in Figure 5-7. They can also look at the source code changes (files, versions) associated with this Change Request, as shown in Figure 5-8. This not only provides a consistent global view of the development, but also adds compliance and governances capabilities which are crucial to geographically distributed development.

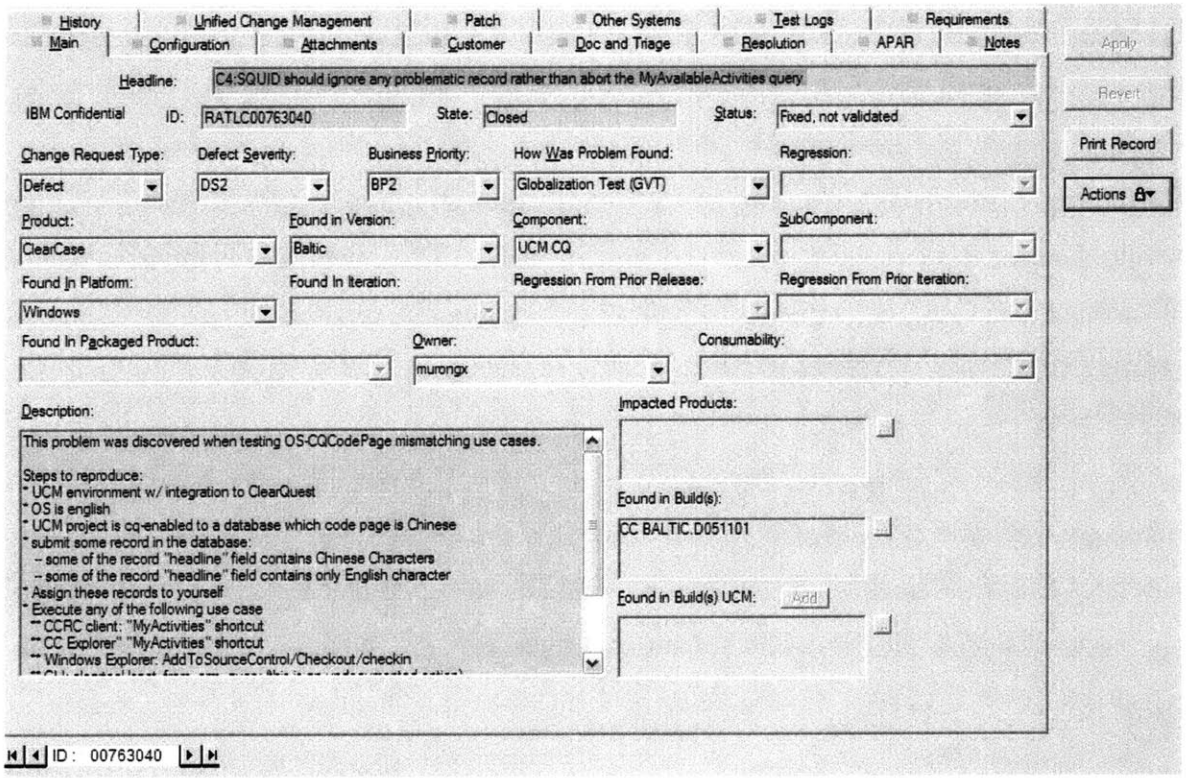

Figure 5-7 A Change Request in RALTC

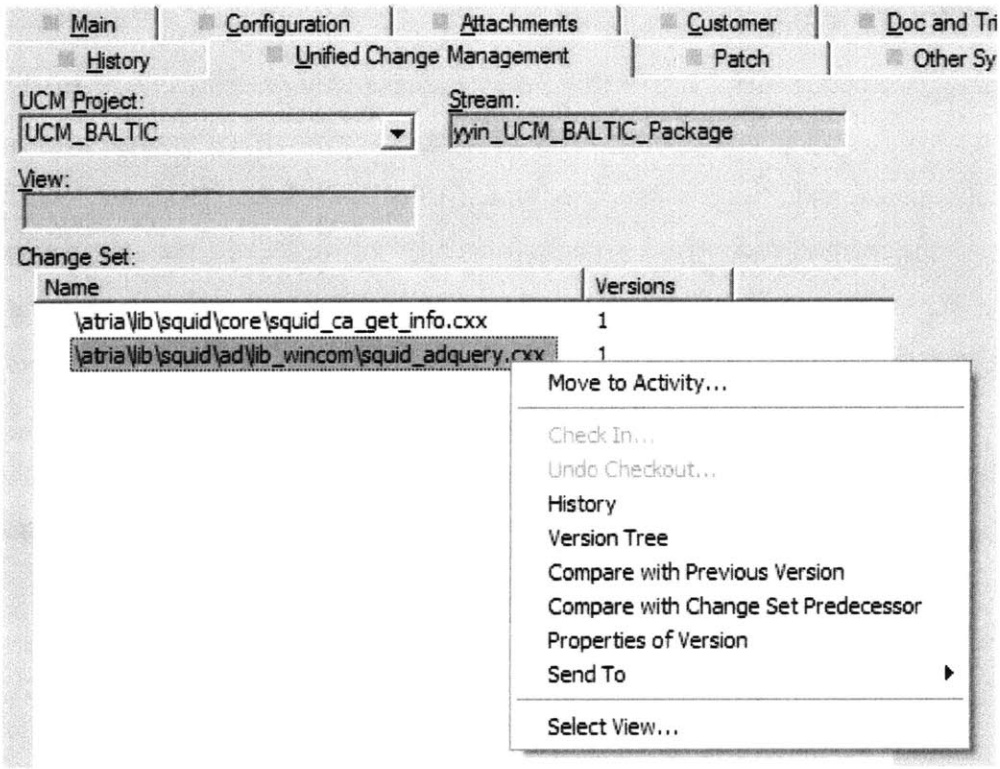

Figure 5-8 View the source code that contribute to this Change Request

UCM also enables structured parallel development with consistent policies and practices. It helps project managers reduce risk by coordinating and prioritizing the activities of developers and by ensuring that the developers work with the right sets of assets. Extending across the lifecycle to accommodate all project domain information requirements, visual models, code, and test artifacts, UCM enables development teams effectively baseline requirements together with code and test assets. This results in accelerated team development in which quality standards are met or exceeded on time and on budget.

As one of the important concepts in UCM model, a stream maintains a list of activities and baselines and determines which versions of elements appear in the workspaces (views) that are associated with it. The UCM stream structures in a UCM project (corresponding to either a software release or a software component) determine the development processes.

A stream strategy is a method to manage project deliveries for a software product. A successful stream strategy needs to determine whether to focus on integration, during

which developers work on the same stream, or isolation, during which developers work on private streams. Procedural flexibility is key – it is not good if the defined strategy does not work for the developers. The right philosophy should be based on the global team's needs and processes while striking a balance between the two philosophies. In a geographically distributed development, the stream structures resulted from a stream strategy are especially critical given the mastership complexity of the UCM streams and RATLC records.

While frequent integration can identify functionality issues early on and mature the release as a whole as opposed to piece meal, continually updating a stream can cause instability and impact the integrity of the release. While isolation allows for more stable workspaces, integration occurs late in the software life cycle or release cycle can cause issues too. The UCM stream spectrum from isolation-centric to integration-centric is shown in Figure 5-9.

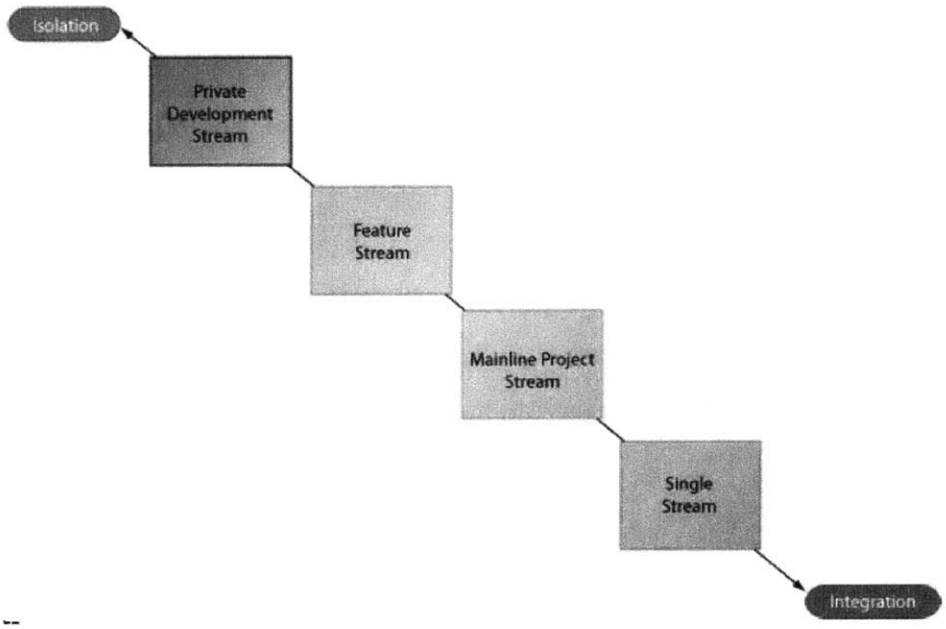

Figure 5-9 UCM Stream Spectrum

- Single stream model

  It appears on the integration side of the spectrum. In this model, all developers work on the same stream, and integrate their changes throughout the development cycle.

- Mainline project stream model

  In this model, the project has one integration stream and multiple development streams that deliver into the single integration stream for collaboration and release.
- Feature stream model

  This model utilizes multiple task streams to allow isolation and integration at a defined level (feature, enhancement, functionality, etc.). It provides a great flexibility in organizing and structuring geographically distributed team based on components and function focuses.
- Private development stream model

  It appears on the isolation side of the spectrum. In this model, developers work on their changes in their own stream and integrate later on in the development life cycle. This model is normally used in conjunctions with other models.

In the delivery of the "iTeam" project, the global team has adopted all models in various stages of the development cycle. At the initial phase of the project, the global team members work in the single stream model for quick integration of design work and early prototyping. After high-level features are well-defined, but before the release goes into the transition phases, feature stream model is used extensively to provide trade-offs between isolation and integrations. In the later phase of the release, mainline project stream model is recommended to reduce the overhead of managing feature streams.

Another aspect of the stream strategy is to cope with the mastership issues associated with replicated repository, especially with ClearCase source code repositories - Version Object Bases (VOBs). This is fundamentally difficult when distributed teams are working on shared code base. For instance, both the Lexington new feature team and the Bangalore sustaining team have the needs to modify the core layer of the product.

Two stream strategies are used to work around the mastership issue. In the first strategy, as shown in Figure 5-10, the integration stream's mastership gets transferred between replica "Lexington" and replica "Bangalore". Pat at Lexington delivers his work to the integration stream locally when the replica "Lexington" has the mastership of the integration stream,

while Raj at India Software Lab delivers his work when the replica "Bangalore" has the mastership of the integration stream. This strategy works better when there is a big time-zone difference between the two replicas (or development sites). For example, the integration stream's mastership gets transferred from replica "Lexington" to replica "Bangalore" at 6PM Eastern Standard Time (EST), and gets transferred back at 7AM EST. In the second strategy, the feature integration stream's mastership is never changed – in this case it is always mastered by replica "Lexington". The key is to set up locally mastered task integration streams to enable local deliveries. As shown in Figure 5-11, all the developers in India Software Lab can deliver their work locally to the Bangalore task integration stream mastered at replica "Bangalore", and all the developers at Lexington can do local deliveries as well. The locally mastered task integration stream is also used to stabilize the code delivered by the local development site. Whenever there is a need to deliver from the task integration stream to the feature integration stream, a UCM post delivery operates in which the delivery initiates at replica "Bangalore" but completes at replica "Lexington". This strategy is recommended since it provides greater flexibility and stability, but it does have the overhead of setting and maintaining addition layers of stream structure.

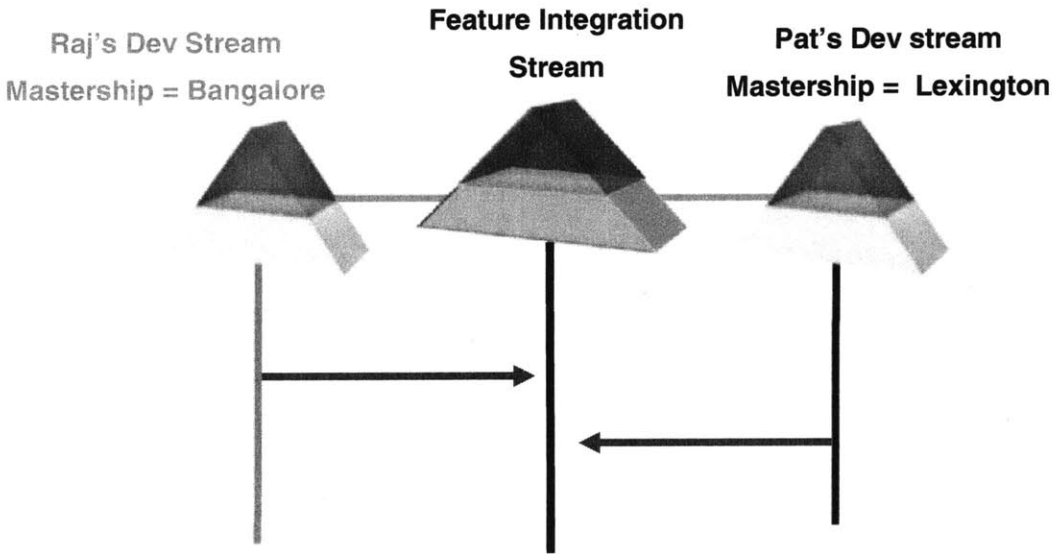

Figure 5-10 Transferring the mastership of the integration stream

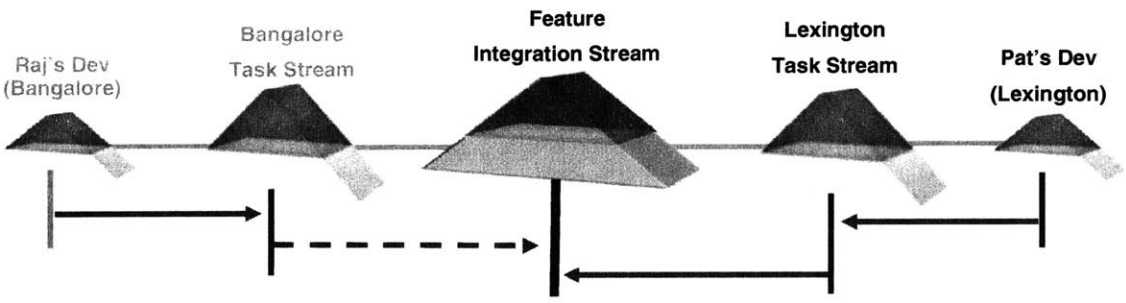

Figure 5-11 Setting up locally mastered task integration stream

With a flexible and integrated software configuration and change management solution that provides both replicated local access and centric web access to the software assets and other repositories and with process and workflow automation with ClearCase UCM and ClearQuest RALTC, the global development follows repeatable, enforceable, and predictable processes.

5.3.3 Automated Build/Test/Deploy Process

In the "iTeam" project, all the unit-level tests are handled by developers, while all other tests, such functional tests, platform tests, integration tests, performance tests and I18n/L11n tests are owned either by China Software Development Lab or by a local testing organization. Automated unit-level testing is an essential ingredient to test-driven development, continuous integration and extreme programming for "iTeam" project's short-iteration (with about 6-week intervals) development process. Even though the team constantly makes tradeoffs between spending time on developing new code and on writing tests, it is not arguable that productivity is directly related to the stability of the code base. The fewer test cases the developers write the less stable the code base becomes. The team has adopted the JUnit[46] test framework which makes unit test creation a quick and easy process that doesn't dominate development time. All new features in the J2EE middle-tier components and C++ backend servers are accompanied by working unit tests upon delivery to the integration stream. Developers are expected to perform quick, informal unit test "spot checks" at any point in the development cycle. Developers gain confidence in

[46] http://www.junit.org/index.htm

the quality of code changes by performing spot checks during development and integration. By spot-checking with the full unit test suite, significant refactoring work can be done with more confidence and less risk. The full JUnit test suite is executed automatically as part of the nightly build process. The test results along with its configuration information (i.e. platform, version, DB, etc) are then stored on a server and published on team wiki pages.

To achieve the goal of having "daily build" and kicking off build globally, the "iTeam" project has used IBM Rational BuildForge[47] to manage build orchestration. BuidForge ties all build steps together, and allows complete build process to be managed via web console. It also provides the capability of restarting a build to complete only for failed components. It coordinates build distribution based on machine capacity and its agents can be compiled for different platform needs. Most importantly, BuildForge supports geographically distributed development teams through a centralized web interface - accessible virtually anytime, anywhere. For instance, any one in the global team can check build status (as shown in Figure 5-12) or kick off a new build (as shown in Figure 5-13) via the BuildForge web interfaces.

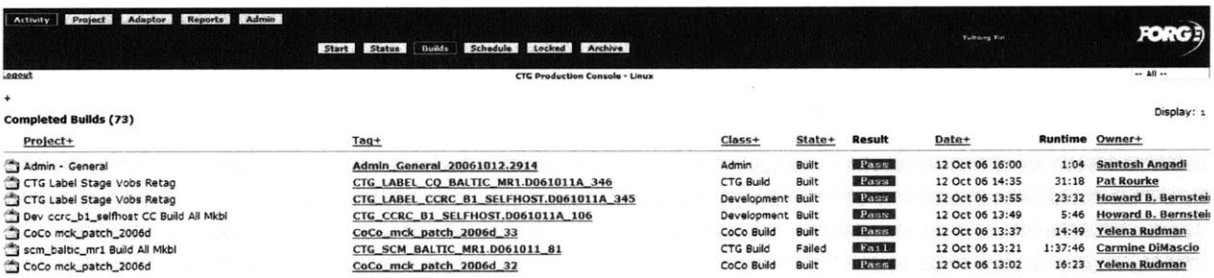

Figure 5-12 Check the build status of individual feature streams

[47] http://www-306.ibm.com/software/awdtools/buildforge/

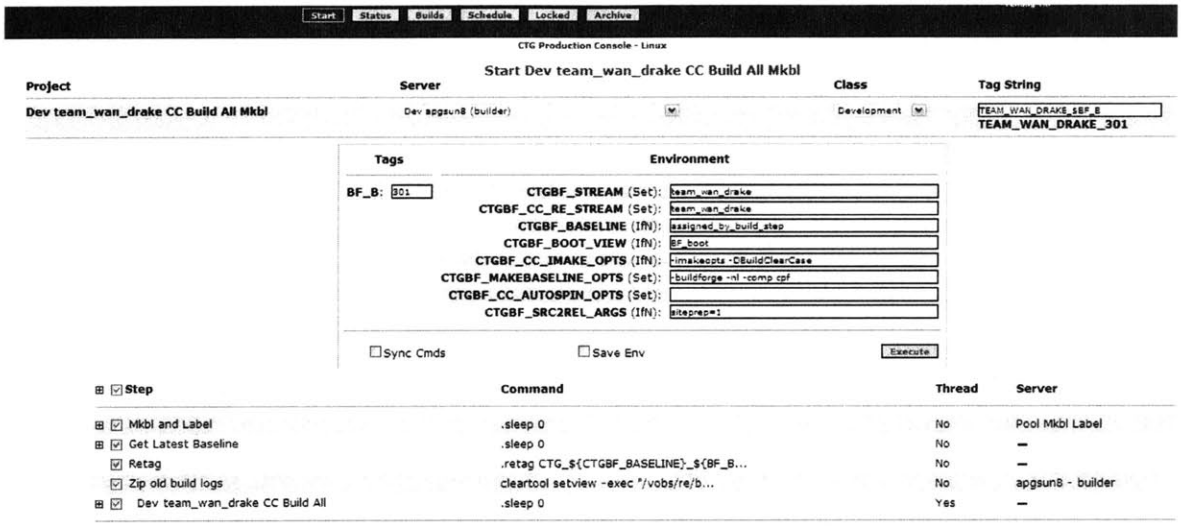

Figure 5-13 Start a build for a particular feature stream

To support feature stream and daily build, the global development community in the "iTeam" project has participated in the build process by working as build meister (or feature stream manager). This job was previously owned by the release engineering (RE) organization but is now taken over by development team with the assistance of release engineers.

The role of "build meister" is rotated on a weekly basis with responsibilities of:

- Improving communications

  The build meister sends out emails, delivers presentations, and holds meetings, as needed, to ensure that the development community using the feature stream understands the current status of the stream and the Rules of the Road, events, and activities that surround the stream. This includes: build and install status and schedule, stream locked or unlocked, any Change Control Board (CCB) in effect, merge related activities, relevant up-coming milestones, etc.
- Keeping the build meister Wiki pages up to date and accurate

  The build meister wiki pages contain detailed instructions on how to run a build and how to debug problems. It also contains useful information such as JUnit testing result, currently recommended baseline, the latest and greatest spin, and any known problems.
- Monitoring build and spin status

The build meister is responsible for tracking down development personnel to resolve build and spin issues on the feature streams, monitoring testing status to ensure an understanding of stream / build / install stability and quality. This is important to aid in making good decisions about when to accept deliveries, merges, and so on.

- Managing the stream

  The build meister interfaces and coordinates with the development teams to ensure that they understand rebase and deliver coming from and going into the stream, with other stream managers to deliver to other streams and accept deliveries from other streams into the stream. The build meister also manages the access to the stream (keeping it open for all, or locking it and adding only required people to ensure issues are fixed in isolation). He/she is also responsible for rebasing and delivering the stream based on the development team's needs and the stream's quality and stability, and recommending baselines as appropriate.

Based on the success of applying BuildForge with feature streams and of rotating developers as build meisters, there is a plan to open up the system to the individual development team. Using BuildForge built-in role-based security, developers can be given limited access to approved production processes directly from their IDE environment. Developers will be able to validate their work, both before and after IBM Rational ClearCase source check-ins, to identify and resolve errors before they impact the nightly run. Developers will gain greater confidence in their code, and the team expects even further productivity and quality gains.

By implementing a scalable build and release management infrastructure, the "iTeam" project team has improved productivity, quality and resource utilization. The team has also automated the life cycle of baselining versions, creating build, generating install image, deploying, testing, publishing test results and recommending baselines, which adds reliability and visibility into the end-to-end development processes. With the increased speed of producing quality builds and spins, the team has more cycles to test the software. The quality engineers in China and in US get immediate access to the latest build and can see exactly what they need to test.

5.4 Summary

The geographically distributed development by IBM Rational represents a typical organization-led practice. GDD is a key strategy for IBM to become a true global company, with global presence and proximity to customers as the main business factors driving global software development at IBM. At business unit levels, global development is more cost-reduction and resource driven. The case study shows that even when led by a big company with many years of multi-national experience, geographically distributed development still presents challenges in strategies and tactics. The heavy investment in global workforce, telecommunication infrastructure and collaborative technologies, the disciplined agile development methodology, and the deployment of software life-cycle management toolsets and infrastructure have made it possible to overcome the challenges and leashed out the potential of geographically distributed development.

This case study validated the business drivers and challenges identified in chapter 2 and 3 and best practices proposed in chapter 4.

6. Conclusions and Recommendations

The global business environment presents emergent business opportunities as well as challenges. Geographically distributed development (GDD) is not a new business mode, but rather a practice that has become a norm or reality since 1990s in software industry. It is likely that GDD will continue to be an important and challenging phenomenon driven by the traditional goals of software development as well as additional objectives such as global presence, but with a relative increase in the importance of cost considerations as the industry is becoming more mature.

The key business drivers for company-led GDD practice include managing operational cost, improving competitive advantage by organizational re-focus and cost reductions, growing revenue by market globalization or by presence in the local market, as well as attracting a large pool of global talents and implementing a "Follow-the-Sun" development model.

Company-led geographically distributed development is facing multiple challenges including uncertainty in strategies, inadaptive in organizational structure, lacking communication and coordination due to distance and cultural differences, and inadequate support in knowledge management and information sharing.

The author has then proposed a five-force framework based on a modification of Carmel's six-force framework to address both the strategic and tactic problems when dealing with geographically distributed development and global teams. Mastering and utilizing the five forces - strategic vision and management skills, organizational structure and team building, collaborative technologies, development methodology, and software life-cycle management, can ultimately leash out the potentials of distributed development in a global context.

The author anticipates that in the future (15 ~ 20 years from today), challenges associated with GSD or GDD will not be so unique or evident, and benefits and disadvantages of practicing GSD or GDD will be well understood and easy to evaluate, thus terms like GSD or GDD may go away. People will talk about software development and software team, but will not be very concerned about so much on whether the development is distributed or whether the team is remote. Software companies and communities will deal with virtual teams, all the time.

References

[1] E. Carmel, *Global Software Teams*. Prentice Hall PTR, Upper Saddle River, NJ, 1999.

[2] B. Cammarano, *Geographically distributed development: IBM's unified lifecycle approach*, IBM developWorks, September, 2004
http://www-128.ibm.com/developerworks/rational/

[3] T. Friedman, *The World Is Flat: A Brief History of the Twenty-first Century*. Farrar, Straus and Giroux. April, 2005.

[4] E. T. Hall, *Beyond Culture, Anchor*. Reissue Edition, December, 1976

[5] G. J. Hofstede, P. B. Pedersen, G. Hofstede, *Exploring Culture: Exercises, Stories and Synthetic Cultures*. Intercultural Press, August, 2002

[6] D. Moitra, Global *Software Development & Business Competitiveness*, IEEE International Conference on Global Software Engineering, 2006

[7] F. Karamouzis and A. Young, *The Offshore Outsourcing Journey Goes Beyond Labor Arbitrage*, G00131094, Gartner, September, 2005

[8] S. Schneider, J. Barsoux, *Managing Across Cultures*, Prentice Hall Europe, 1997

[9] D. German, *The GNOME project: a case study of open source*, global software development

[10] E. Raymond, *The Cathedral and the Bazaar*, O'Reilly, October 1999

[11] B.M. Bass. *Bass & Stogdill's handbook of leadership (3rd ed.)*. New York: The Free Press, 1990

[12] P. Hersey and K. Blanchard. *Management of Organizational Behavior: Utilizing Human Resources (5th Edition)*. Englewood Cliffs, NJ: Prentice-Hall, 1988

[13] Foels, R., Driskell, J. E., Mullen, B., and Salas, E. (2000). *The Effects of Democratic Leadership on Group Member Satisfaction: An Integration. Small Group Research*, 31(6), 676-701

[14] B. D Janz, J. A. Colquitt, and R. A. Noe. Knowledge Worker Team Effectiveness: The Role of Autonomy, Interdependence, Team Development, Contextual Support Variables. Personnel Psychology, 50, 877-905, 1997

[15] S. Zhang, M. Tremaine and J. Fjermestad, Delegation in Virtual Team: the Moderating Effects of Team Maturity and Team Distance, IEEE International Conference on Global Software Engineering, 2006

[16] N. Radjou, E. Daley, M. Ramussen, H. Lo, The World Isn't Flat Till Firms Are Networked, Rish-Agile, And Socially Adept, Forrester, December, 2006

[17] J. Herbsleb, Coordinating in GSD: Making the Invisible Visible, IEEE International Conference on Global Software Engineering, 2006

[18] J. Davies, *Wiki Brainstorming and Problems with Wiki Based Collaboration*, thesis submitted to the Department of Computer Science at the University of York, September, 2004

[19] R. Pressman, *Software Engineering: A Practitioner's Approach (*6 edition), McGraw-Hill Science/Engineering/Math, March, 2004

[20] A. MckCormack, C. Kemerer, M. Cusumano, B. Grandall, *Trade-offs between Productivity and Quality in Selecting Software Development Practices*, IEEE Software, 2003.

[21] M. Cusumano, *The Business of Software: what every manager, programmer and entrepreneur must know to thrive in good times and bad*, Free Press, 2004

[22] M. Paasivaara and C. Lassenius, *Could Global Software Development Benefit from Agile Methods?* IEEE International Conference on Global Software Engineering, October, 2006

[23] B. Ramesh, L. Cao, K. Mohan and P. Xu, *Can Distributed Software Development Be Agile?* Communications of the ACM, October, 2006/Vol.49, No. 10

[24] J. Bach, *The Immaturity of CMM*, American Programmer, September, 1994
http://www.satisfice.com/articles/cmm.shtml

[25] Senge, *The Fifth Discipline*, Doubleday, 1990

[26] C. Jones, *Assessment & Control of Software Risks*, Prentice-Hall, 1994

[27] D. Ju, *A Concerted Effort towards Flourishing Global Software Development*, ACM Press, 2006

[28] SEI, *Capability Maturity Model Integration*, 2006

[29] C. Fung, *Case Study: Using IBM Rational Tools to Achieve Outsourcing and CMMI Objectives,* IBM Rational Software Development Conference Proceeding, June, 2006

[30] A. Fuggetta, *Open Source and Free Software: a New Model for Software Development?* Unpublished paper, Politecnico di Milano, July 2004

[31] A. Mockus, R.T. Fielding, J. Herbsleb. *Two case studies of open source development: Apache and Mozilla.* ACM TOSEM, Vol. 11, Issue 3, July 2002

[32] O. Gotel, A. Finkelstein, *An Analysis of the Requirements Traceability Problem, Proceeds of First International Conference on Requirements Engineering, 1994*

[33] B. Ramesh, M. Jarke, *Towards Reference Models for Requirements Traceability,* IEEE Transactions on Software Engineering, Vol. 27, No.1, January, 2001

[34] T. Milligan, *Seven Keys to Improving Business Value*, IBM White Paper, 2003

[35] M. Ballou, *Establishing Build Management for IT Efficiency and Business Adaptability,* IDC White Paper, January, 2006

[36] B. Sengupta, V. Sinha and S. Chandra, Test-Driven Global Software Development, the 3rd International Workshop on Global Software Development, May, 2004

[37] C. Liu, R. Wong, Y. Chen and H. Huang, Managing Critical Knowledge Management Issues in Global Software Development Project, Issues in Information Systems, Volume VII, No. 2, 2006

[38] R. Rogowski, *Make Your Global Site Committee More Effective,* Forrester Research, Inc, June, 2006

[39] S. Fukushige, H. Trainor, *A Primer on Distributed Development Process,* White paper, the Symbio Group, 2006