July 2009

Modeling Coordination in Offshore Software Development

Narayan Ramasubbu
Singapore Management University, nramasub@smu.edu.sg

Amit Mehra
Indian School of Business, Amit_Mehra@isb.edu

Vijay mookerjee
University of Texas at Dallas, vijaym@utdallas.edu

MODELING COORDINATION IN OFFSHORE SOFTWARE DEVELOPMENT

Research-in-progress paper

The papers submitted for review MUST NOT contain any information on the author(s). Submission for review has to be anonymous. Papers submitted for review not being anonymous will be rejected without review.

Abstract

Controlling and minimizing coordination costs has been shown to be an important factor to reduce overall project performance in distributed software development. In this research-in-progress paper we investigate the effects of software complexity, software integration, distributed labor division policies, learning effects on software coordination costs. Drawing from data collected on 130 software construction cycles in 34 large projects of a leading offshore development firm, we first present our analysis on how coordination costs relate to team organization factors and complexity of evolving software. We base our analytic model of coordination costs in offshore software development on these empirical relationships, and give an overview of our modeling approach. We apply our model of software coordination costs to develop resource allocation policies in the projects we studied. We consider both waterfall and iterative software development methodologies and also tandem and parallel integration schemes. Our modeling approach helps managers to develop a dynamic coordination policy to aid iterative software development in distributed development environments.

Keywords: Global software development, offshore services, coordination, software engineering.

MODELING COORDINATION IN OFFSHORE SOFTWARE DEVELOPMENT

1. BACKGROUND & MOTIVATION

Offshore software development refers to the mode of information systems development where majority of the development tasks are conducted at a remote development center ('offshore'), away from a customer's premises ('onsite'). Outsourcing development tasks to offshore centers help firms to leverage the significantly low labor costs prevalent in those parts of the world. For example, software labor costs in India are three to seven times lower as compared to those in the USA (Mercer 2004). Countries like India have emerged as an attractive location for software development in the context of the growing business need to develop software applications rapidly, and in a cost effective manner without compromising on quality. Out of the 140-odd companies with SEI-CMM level 5 certification world-wide 42 are from India, the largest pool of high maturity software organizations from any country (Paulk 2003). According to a recent survey, custom software applications worth $ 3.23 billion were jointly developed by software service companies in India and United States. Although in the beginning U.S. software firms took the lead by setting up software factories in India, of late Indian firms have started setting up development centers in the U.S. to provide seamless services to their clients. During 2003-2007 as many as 270 Indian companies set up offices, subsidiaries and alliances in the United States (Nasscom-McKinsey). Considering the phenomenal growth in offshore model, it is important to formally study and understand issues specific to this mode of development.

Though production costs are lower at offshore centers, it is not possible for a software development manager to allocate all her resources to the offshore center because there are certain activities that have to be performed at the customer's premises (where physical hardware and real users are present). For example, Apte and Mason (1995) discuss the need for customer contact and physical presence in a distributed environment for improved customer service. Presence of resources at onsite also helps easier gathering of end user requirements and to get a quick feedback on prototypes. Further, implementation of large scale software is a complex task and often involves several configuration settings and changes that require physical presence at the hardware site. Moreover resources are required to be present at the customer's site to handle urgent production failures. Thus even in the offshore mode of development, resources need to be present at both 'offshore' and 'onsite'. Presence of geographically distributed resources gives rises to additional increase in the total costs of production in the form of higher coordination and integration costs. In contrast to co-located scenario, managers will now have to make decisions on integration and resource allocation policies that govern dispersed tasks.

Prior research on coordination in software does not specifically address the tradeoffs involved in allocating resources to distributed development centers (onsite and offshore). Kraut and Streeter (1995) examined the role of formal and informal communication mechanisms in coordinating work in co-located software development. They argued that a majority of communication support tools facilitated only formal communication and prescribed nurturing interpersonal and informal communications to solve the problems of uncertainty and complexity in software development. Grinter (1995; 1996) reports a case-study of the usage of a configuration management tool to effectively coordinate software development activities. Herbsleb and Mockus (2003) formulate an empirical theory of coordination and explain the role of modular design and information hiding principles in facilitating coordination in software tasks. Koushik and Mookerjee (1995) formally model coordination in co-located software construction based on release cycles and prescribe solutions for determining the integration mechanisms for individual modules. This model was later enhanced to accommodate system stability and team learning factors in the co-located scenario (Mookerjee and Chiang 2002). Other researchers who studied coordination in software development

explain different mechanisms of coordination (Nidumolu 1995) and ways to manage expertise (Faraj and Sproull 2000) also limit themselves to co-located scenarios.

Studies that consider the role of coordination and communication in distributed cross-functional teams are beginning to emerge. Shami, Bos, Wright, *et al.* (2004) use experimental simulation to analyze the effect of social networks on project outcomes in globally distributes software development. Sikora and Shaw (1998) describe the use of a formal framework to model coordination of cross-functional information systems development but do not specifically consider labor-cost differences, and dispersion of individual tasks in software development. Toffolon and Dakhli (2000) describe a coordination meta-lifecycle model and the ways to formally represent global software development dependencies. They too do not consider the influence of resource allocation tradeoff present in offshore software development for prescribing coordination solutions. Overall, there is a dearth of studies in the literature that model tradeoffs involved in allocating resources to 'offshore' and 'onsite' development centers. Also there is a limited understanding of managerial decisions about appropriate coordination and integration policies for distributed development tasks.

In this study we model coordination costs by explicitly capturing the effects of team organization and the complexity of evolving software. Our goal is to help software managers derive optimal policies for resource allocation at onsite and offshore as well as to develop a dynamic coordination policy to aid software construction. The coordination policy in our model involves the appropriate time at which the distributed team members integrate their code.

The rest of the paper is organized as follows. In section 2, we outline the basic set up of the software development activity we study and model. In section 3 we describe our data collection and analysis. In section 4 we discuss our preliminary analytical model in detail and discuss our approach in arriving at a dynamic coordination policy. We outline our next steps in section 5, and conclude.

2. MODELING APPROACH OVERVIEW

A schematic view of the project phases in iterative software development is shown in Figure. 1. The software project begins with the development of code at both offshore and onsite centers. After a certain time 't', the onsite and offshore teams work on integrating their code and reconciling problems. Once they are satisfied, the first release of the system is committed to the global code repository. This chain of events continues for 'j' cycles until the scope of the project is completed. We categorize the effort involved in these activities in to development costs and coordination costs. Coordination costs include the effort spent on integrating the software code between onsite and offshore centers to deliver one working, global system. The unit of analysis for our study is at the level of software construction cycle.

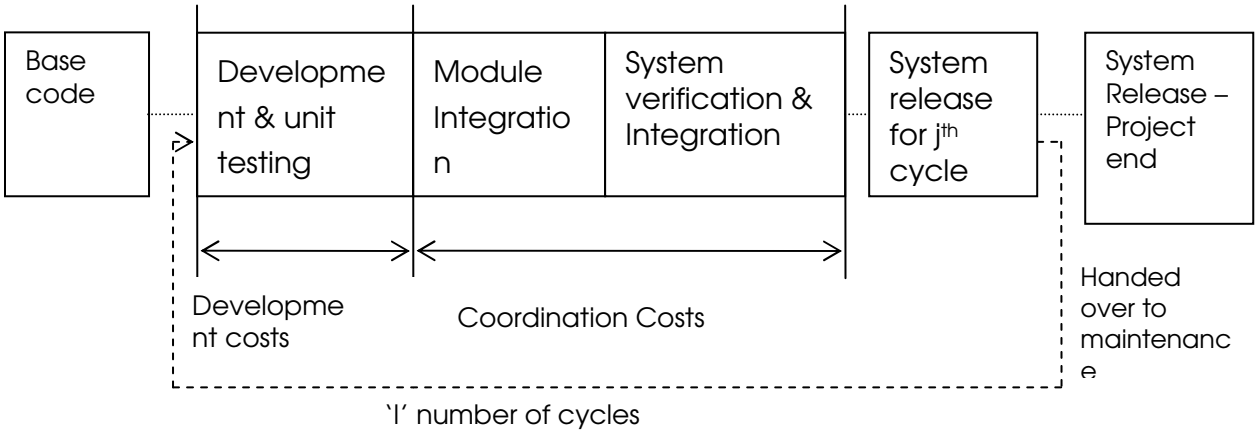

Figure 1. Software Project Activity

Our modeling approach involves two stages. In the first stage, we extend prior analytical coordination models (Mookerjee and Chiang 2002) to address distributed software development. A key factor in grounding this analytical model is to show that the coordination costs in distributed software development consist of fixed and variable elements. The fixed component of the coordination cost is independent of the construction cycle, and is typically influenced by the way the teams are organized and the nature of communication mechanisms between them. The variable component of the coordination cost is dependent on the software code developed during the construction cycle. This separation of the coordination costs in to two components, one related to team organization and the other related to software complexity that has evolved during the particular construction cycle forms the basis for the analytical model we are developing.

Before diving in to the analytical model, we proceeded to empirically test our idea through data collected from real world projects. Empirically testing the planned model gives validity to our approach. Further, the empirical models gives us a good basis to derive the cost distributions of software development activity for each construction cycle and the individual development centers. We revisit the our modeling in detail in section 4.

3. DATA COLLECTION, ANALYSIS & FINDINGS

The empirical data for this research was collected from one of the top five software development firms in the world, in terms of market capitalization for IT services, and is assessed at level 5 of the CMM-I process capability model. This firm employs more than forty thousand personnel across twenty countries. We collected software project data from this research site through a field study. During the field study we extracted detailed project level data from the centralized process database maintained by the quality department of the firm. All the data we had extracted had been previously audited by the quality department. Overall, we were able to observe 130 construction cycles of 34 large projects. For each of these construction cycles, we gathered the development effort, amount of code developed, the coordination effort, the team sizes at onsite and offshore development centers. We also noted the total construction cycles in each of these projects along with the total code size developed for the project and the number of errors discovered in the code. A summary statistics of this data set is shown in Table 1 and the correlation between the variables is shown in Table 2.

Variable	Mean	Std. dev.	Min	Max
Construction Cycle	2.59	1.30	1	6
Development Effort	1017.69	1153.03	14	6910
Coordination Effort	134.90	159.86	1	1152
Defects	13.35	30.06	0	169
Team Size at Location 1	2.19	1.45	1	9
Team size at location 2	8.15	4.86	3	21
Cycle Function Points	223.01	313.55	3	2868
Total Project Function Points	1532.14	1336.94	33	6385
Total Number of construction cycles in sample	130			
Total Number of Projects	34			

Table 1. Summary Statistics

Our data analysis began by exploring the relationship between the coordination effort in each construction cycle and the team size and cycle function points and the construction cycle stage. The observed relationships are presented in Figures 2-6. As depicted in the Figures 2-4, we find that the coordination effort per construction cycle is positively associated with the organization of the teams at the onsite and offshore centers and the interaction between them. Notice that the effect of the team organization is same for all the construction cycles that belong to a single project, i.e., this is independent of the variations in the construction cycles within a project.

Figure 5 depicts the effect of complexity on the coordination effort. Consistent with findings from the software engineering literature, we see that larger the function points, more coordination effort is needed. As depicted in the Figure 6, we find that there is a significant learning effect on the coordination costs between the onsite and offshore teams. As they get advanced in the project, despite the increase in the number of function points to deal with, the overall coordination effort required for the later stage construction cycle is lower. These exploratory results provide an excellent foundation for us to build the analytical model of coordination in distributed software development to derive appropriate coordination policies.

Variables		1	2	3	4	5	6	7	8
Defects	1	1.00							
Total Project Function Points	2	0.77*	1.00						
Construction Cycle	3	-0.06*	-0.11*	1.00					
Development Effort	4	0.39*	0.42*	-0.05	1.00				
Coordination Effort	5	0.38*	0.52*	-0.02	0.67*	1.00			
Cycle Function Points	6	0.49*	0.60*	0.00	0.73*	0.59*	1.00		
Team Size at Site 1	7	0.28*	0.34*	-0.06	0.50*	0.56*	0.15	1.00	
Team size at Site 2	8	0.55*	0.63*	-0.06	0.41*	0.55*	0.39*	0.53*	1.00

Table 2. Correlations

*Note: * Indicates Pairwise correlations significant at 5%*

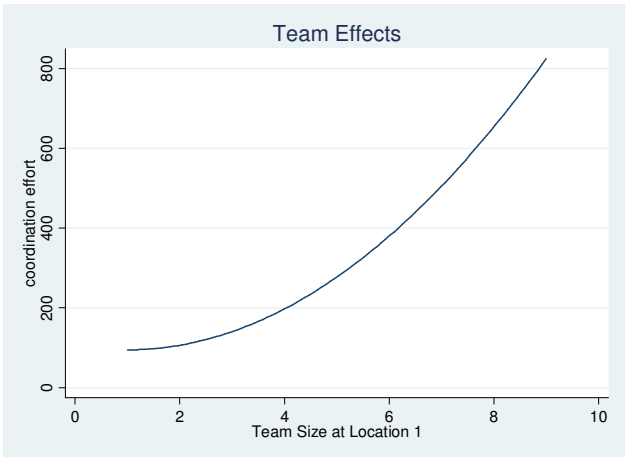

Figure 2. Effect of Team Size Near Client Site on Coordination Effort

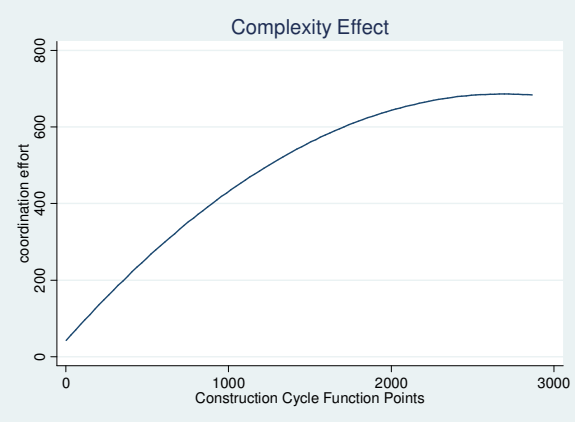

Figure 5. Effect of Complexity (Function Points) on Coordination Effort

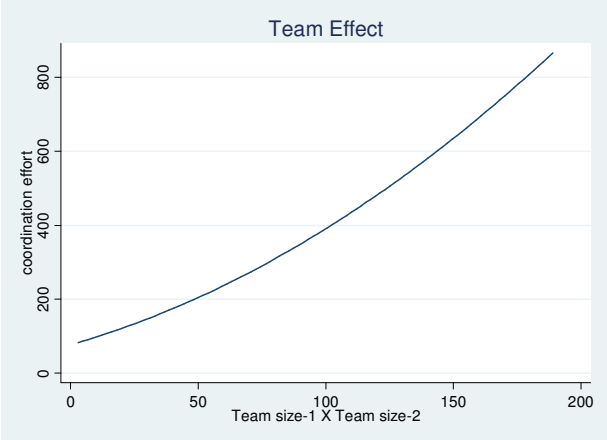

Figure 4. Effect of Offshore-Onsite Team Interaction on Coordination Effort

Figure 3. Effect of Offshore Team Size on Coordination Effort

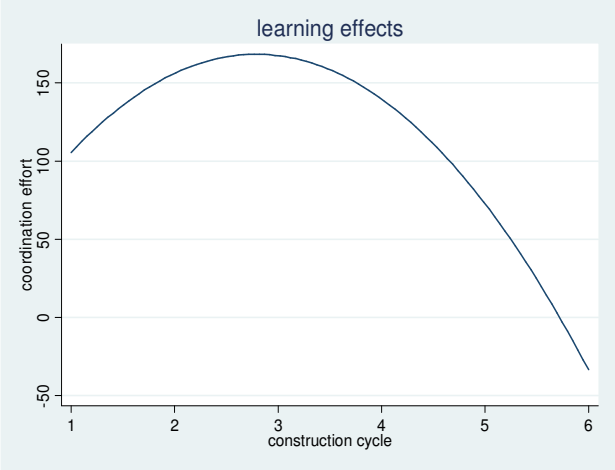

Figure 6. Effect of Learning on Coordination Effort

4. DERIVING DYNAMIC COORDINATION POLICY

4. 1. Basic model in co-located scenario

The following characterizes the co-located development setting:

1. The system to be developed consists of N distinct modules interconnected to each other. A module is defined as a compilation unit of code.
2. There are S developers in the team.
3. Each module is developed by exactly one developer.
4. The project has a fixed time schedule and fixed budget for development. Let T be the available calendar time for development and let C be the overall cost of the project permissible by the budget.

Now, we proceed to develop functional forms for the costs associated with various activities during software development. Let $\check{e}_i$ be the effort in person hours needed to develop the i^{th} module in the system. The variable $\check{e}$ follows some positive distribution with mean $\bar{e}$.

Total development effort for the project $= \sum_{i=1}^{N} \check{e}_i$ …… Eq(1)

Expected total development effort $= E\left(\sum_{i=1}^{N} \check{e}_i\right) = N * \bar{e}$ …… Eq(2)

Letting 'c' be the cost of development per person hour we get,

Total expected cost of development $= c * N * \bar{e}$ …… Eq(3)

Since there are S developers working simultaneously,

Total expected time of development $= N * \bar{e} / S$ …… Eq (4)

Now let us proceed to module integration phase of development. Module integration consists of integrating individually developed modules to rest of the modules in the system. Since there are N interconnected modules in the system, each module has (N-1) connections. The interconnections between modules are primarily due to module coupling (Dhama 1995). Coupling refers to the software property that measures interdependence between modules. Interdependence between modules arises because of data and logical flow between modules. The strength of interdependence between modules depends on the import and export interfaces between modules, the number of parameters passed through these interfaces and the environmental settings that determine logical invoking ('calling or being called') of modules. To model the strength of interdependence between modules we use the notion of pair-wise coefficient of interaction (PCI), $p_{i,j}$. Note that, $p_{i,j} = p_{j,i}$ for any pair of modules i,j. The value of PCI is greater for modules that are tightly coupled with each other and is lower for modules that are loosely coupled. Further the value of PCI is zero if the modules are not connected at all. We model $p_{i,j}$ as a uniform random variable with mean μ_p and standard deviation $\sigma_{p.}$ The effort needed to integrate a module to the system depends on the structural complexity of the system and as well as the functional modifications or enhancements that have been done to the module. Structural complexity of the system is modeled through the pair-wise coefficient of interaction mentioned above. Integration effort required due to functional enhancements can be modeled as a function of development effort that was spent on the module (Koushik and Mookerjee 1995). Thus, we derive module integration effort as,

Module integration effort $= \sum_{i=1}^{N} \sum_{j=1}^{(N-1)} p_{i,j} * \check{e}_i$ …… Eq (5)

Expected Module integration effort $= N * (N-1) * \mu_p * \bar{e}$ …… Eq (6)

Total module integration cost $= c * N * (N-1) * \mu_p * \bar{e}$ …… Eq (7)

Since S developers are working at the same time,

Total module integration time $= (N * (N-1) * \mu_p * \bar{e}) / S$ …… Eq (8)

In the system integration phase of project, overall system is tested for faults using the test conditions developed by the project team. In this phase, interconnections between modules are tested for conformity with agreed upon design specifications. This results in the identification of a certain number of faults with the way module integration was conducted. To correct these faults modules have to be individually analyzed and the interconnections re-adjusted. Moreover, overall system integration effort depends on the stability of the application platform technology that supports the interconnected modules. Application platform stability depends on the extent of usage of application programming interfaces supported by the system, adherence to standards and base technology updates applied to the system. The effort spent on system integration thus depends on the number of faults discovered, structural complexity of the system and overall system stability. Let us suppose that *'f'* number of faults are discovered during the system integration process. Since these faults are at

interfaces, two modules are associated with each of these faults. Effort required to correct a fault is therefore given by,

Effort to correct a system integration fault $= p_{i,j} * (e_i + e_j) * \beta$ …… Eq (9)

Where, β is the application platform stability factor.

Expected effort to correct a system integration fault $= 2 * \mu_p * \bar{e} * \beta$ ……Eq (10)

Because of the adjustment made to the i^{th} and j^{th} module, 2(N-2), other connections have to be verified for conformance with design specification. So the overall expected effort spent on system integration is given by,

Total effort for system integration $= \sum_{k=1}^{f} 4* (N-2) * \mu_p * \bar{e} * \beta$ ……Eq (11)

$= 4f * (N-2) * \mu_p * \bar{e} * \beta$ ……Eq (12)

For each fault, two developers are involved. So 2f out of the S developers have to work in order to fix the 'f' faults discovered for system integration. Thus,

Total time for system integration $= 1/2S * [f^2*4* (N-2) * \mu_p * \bar{e} * \beta]$ ….Eq (13)

Some researchers (Koushik and Mookerjee 1995) have posited that a 'ripple effect' occurs during system integration. That is, an infinite sequence of adaptations results because of continuing modifications of interconnected modules. Modern structured development methodologies stem the ripple effect by freezing a design base line. Project teams usually spend time 'upfront' before coding process starts to freeze the design of interfacing systems. This helps during system integration, because any faults have to be corrected in accordance to the agreed upon standards and not on ad hoc developments. In this study we assume that design base lines are available and that infinite adaptations do not occur during system integration. Apart from system development and integration developers also spend effort on program comprehension, communication and coordination. Communication costs in a software team have been reported to be a nonlinear function of team size in past research (Kraut, Egido and Galegher 1990; Mookerjee and Chiang 2002). Developers also spend considerable amount of effort comprehending different modules and interconnections present in the system. They also spend time in preparing test cases and documenting their knowledge on the system for future use. Program comprehension costs include a fixed cost that captures the effort spent on test preparation and module comprehension as well as variable costs that depend on the number of faults that are discovered. When a fault is discovered, programmers need to perform root cause analysis and understand the reasons for the error.

These costs are modeled as,

Effort spent on communication $= K_1 * S^2$ ……Eq (14)

Where, K_1 is a factor that captures the communication effort per person considering the effect of team structures and organizational hierarchy.

Expected communication cost $= c*k_1*S^2$ ……Eq (15)

Expected time spent on communication $= K_1 *S$ ……Eq (16)

Program comprehension effort $= k_2 * N + K_3 * f$ ……Eq (17)

Where, K_2 is a factor that captures program comprehension effort considering any learning effect in the teams. As programmers become more aware of the system, effort required for comprehension decreases. K_3 captures effort required to discuss and learn about faults taking in to consideration the severity and complexity of faults that are discovered.

Program comprehension cost $= c*k_2*N+K_3*f$ ……Eq (18)

Expected time spent on program comprehension $= 1/S(c*k_2*N) + 2/S(K_3*f^2)$ …...Eq (19)

Overall project costs = development cost + module integration cost + system integration cost + communication costs + program comprehension costs

The objective of a development manager is to determine, optimal team size S* such that the project cost is minimized. This can be formulated as a constraint programming (CP) as following:

Minimize project cost with respect to S and subject to

1) Schedule constraint – Total Project development time < T
2) Budget constraint - Total Project costs < C
3) Functionality constraint – Total of N modules have to be delivered to customer

4.2. Offshore scenario

The software system in this scenario remains the same, however the team that develops the system is dispersed between the primary development center (offshore) and a secondary development center (onsite). Let us suppose that S_1 be the number of developers who are assigned to the onsite development center and that they handle N_1 modules. Assuming that individual onsite and offshore developers have similar amount of work loads we have,

$$S_1/S = N_1/N \quad \text{……Eq (20)}$$

Dispersion in resources between onsite and offshore = $(1-S_1/S)^2=(1-N_1/N)^2$ ….Eq (21)

Development effort = $\sum_{i=1}^{N1} e_i + \sum_{j=1}^{N-N1} e_i$ ……Eq (22)

Expected development effort = $N_1 * \bar{e} + (N-N_1) * \bar{e}$ ……Eq (23)

Letting c1 and c2 be the cost of development in onsite and offshore we get,
Expected cost of development = $c_1* N_1 * \bar{e} + c_2*(N-N_1) * \bar{e} = \bar{e} * (c_1*N_1 + c_2*(N-N_1)$ ……Eq (24)

For any one module owned by a developer at onsite, there are (N_1-1) interconnections located locally (at onsite) and $(N-N_1)$ interconnections with modules located at offshore. Similarly, for a module located at offshore there are N_1 number of interconnections located at onsite and $(N-N_1-1)$ number of modules located locally. Using this, we model module integration effort as shown below:

Module integration effort = $\sum_{i=1}^{N1}\sum_{j=1}^{N-1} p_{i,j} * \check{e}_i + \sum_{i=1}^{N-N1}\sum_{j=1}^{N-1} p_{i,j} * \check{e}_i$ ……Eq (25)

Therefore,
Expected Module integration effort = $N_1 * (N-_1) * \mu_p * \bar{e} + (N-N_1) * (N-1) * \mu_p * \bar{e}$
……Eq (26)
Expected cost of module integration = $[c1 * N_1 + c_2 *(N-N_1)] * (N-1) * \mu_p * \bar{e}$ ……Eq (27)

Assuming that ‘f’ number of faults are discovered during system integration and that the likelihood of discovering faults in any of the modules remains the same we have,

Faults to be corrected at onsite = $N_1 / N * f$ ……Eq (28)
Faults to be corrected at offsite = $(1 – N_1 / N)* f$ ……Eq (29)

Further for each fault corrected 2* (N-2) interconnections each have to be checked by onsite and offshore personnel.

Effort for system integration at onsite = $\sum_{k=1}^{(N1/N)f} 4* (N-2) * \mu_p * \bar{e} * \beta$
$= (N_1/N)*f*4* (N-2) * \mu_p * \bar{e} * \beta$ ……Eq (30)

Effort for system integration at offshore = $\sum_{k=1}^{(1-N1/N)f} 4* (N-2) * \mu_p * \bar{e} * \beta$
$= (1-N_1/N)*f*4* (N-2) * \mu_p * \bar{e} * \beta$ ……Eq (31)

Expected cost of system integration = $f*4* (N-2) * \mu_p * \bar{e} * \beta* [c_1*(N_1/N) + c_2*(1-N_1/N)]$
……Eq (32)

Expected time for system integration = $1/2S * [f^2*4* (N-2) * \mu_p * \bar{e} * \beta]$ ……Eq (33)

Effort spent on communication in the offshore scenario is two fold. There is intra-site communication with team members co-located and there is also inter-site communication with team members located in the remote development center. We posit that inter-site communication is a function of the dispersion of team members and the richness of medium used in communication.

Effort spent on communication = $k_1 S_1^2 + k_1 (S-S_1)^2 + k_4 (1-S_1/S)^2$ ……Eq (34)

Where the factor K_4 measures the effort per person considering the richness of the communication medium

Expected communication costs = $(c_1+c_2)/2 * [k_1 S_1^2 + k1 (S-S_1)^2 + k4 (1-S_1/S)^2]$ ……Eq (35)

Program comprehension effort = $2(k2N_1 + k_2(N-N_1)) + k_3 N_1 / N * f + k_3 (1-N_1/N)*f$ ……Eq (36)

Program comprehension cost = $c_1* ((k_2N_1 + k_2(N-N_1)) + k_3 (N_1 / N) * f) + c_2*((k_2N_1 + k_2(N-N_1)) + k_3 (1-N_1/N)*f)$ ……Eq (37)

The objective of a development manager is now to determine, optimal team size S* and S_1* such that the project costs are minimized. This can be formulated as a constraint programming (CP) as following:

Minimize project costs with respect to S, S_1 and subject to

1) Schedule constraint – Total Project development time < T

2) Budget constraint - Total Project costs < C

3) Functionality constraint – Total of N modules have to be delivered to customer

4.3. Iterative offshore development

Unlike traditional software development methodologies such as the waterfall model of development, iterative development methodologies such as spiral model of development (Boehm 1988), rapid application development approaches such as prototyping and extreme programming (Beck and Andres 1999) encourage building a software system in several cycles. The following characterizes my iterative development setting:

1. There are N interconnected modules in the system.
2. The final system is built in 'I' number of cycles.
3. In any j^{th} cycle 'm_j' number of modules are released with enhancements. These m_j modules have to be integrated with each other and with the system.
4. Prior to the system release in any of the I cycles, remaining $(N-m_j)$ modules are adjusted for these enhancements.
5. There are S developers in the development team, S_1 of them are allocated the secondary development center located at onsite. The ownership of the modules is spread according to $N_1/N=S_1/S$.

Let $\check{e}_{ij}$ be the effort in person hours needed to develop the i^{th} module in the j^{th} cycle. The variable $\check{e}$ follows some positive distribution with mean $\bar{e}$. M modules are released in every cycle. The probability of a module picked has ownership at onsite is N_1/ N and at offshore is $(N-N_1)/N$.

Development effort $= \sum_{j=1}^{I} \sum_{i=1}^{mj*N1/N} e_{ji} + \sum_{j=1}^{I} \sum_{i=1}^{mj*(1-N1/N)} e_{ji}$ ……Eq (38)

Expected development cost $= \sum_{j=1}^{I} [c_1(m_j * N_1/N) + c_2(m_j * (1-N_1/N))] * \bar{e}_j$ ……Eq (39)

At the start of a j^{th} cycle, number of modules available for module integration is given by

$$M_j = \sum_{k=1}^{j-1} m_k$$

So for the first module released during the j^{th} cycle has to be integrated with M_j modules released during the previous cycle. We derive this effort as,

Integration effort for first module released during j^{th} cycle = $\sum_{x=1}^{Mj} p_{1,x} * e_{j1}$ …Eq (40)

Now, the i^{th} module released during the j^{th} cycle, has to be integrated with i-1 new modules released during this cycle and M_j modules released during the previous cycles. Thus the overall module integration effort for the j^{th} cycle is given as,

Total module integration effort = $\sum_{j=1}^{I} \{ \sum_{x=1}^{Mj} p_{1,x} * e_{j1} + \sum_{i=1}^{mj} \sum_{k=1}^{i-1+Mj} p_{ik} * e_{ji} \}$ …Eq (41)

The probability of each of the mj modules being released during the jth cycle having ownership at onsite is N_1/N and at offshore is $(1- N_1/N)$. Thus the cost of module integration is modeled as,
Total module integration cost =

$[c_1*N_1/N + c_2*(1-N_1/N)] * \sum_{j=1}^{I} \{ \sum_{x=1}^{Mj} p_{1,x} * e_{j1} + \sum_{i=1}^{mj} \sum_{k=1}^{i-1+Mj} p_{ik} * e_{ji} \}$ …Eq (42)

Let 'f_j' number of faults are discovered during system integration of the j^{th} cycle. For each fault corrected (N-2) interconnections have to be checked by onsite and offshore personnel. In iterative development, the entire system is not fully connected until all of the I iterations are complete. Thus structural stability of the system increases as the system is developing. During beginning of the j^{th} cycle only $M_j(M_j-1)$ of the N(N-1) interconnections are stable. Thus structural stability of the system can be derived as,

Structural stability of the system = $1-\Delta$, where $\Delta = M_j(M_j-1) / N (N-1)$

We posit that structural stability of a system has an exponential effect on the effort required for integration. Thus,

System integration effort required =

$\sum_{j=1}^{I} \{ \sum_{k=1}^{(N1/N)fj} 2* (N-2) * \mu_p * \bar{e} * \beta + \sum_{k=1}^{(1-N1/N)fj} 2* (N-2) * \mu_p * \bar{e} * \beta \} * Exp (1-\Delta)$ . ...Eq (43)

Expected effort for system integration = $\sum_{j=1}^{I} \{(N_1/N)*f_j*2* (N-2) * \mu_p * \bar{e} * \beta +$

$(1-N_1/N)*f_j*2* (N-2) * \mu_p * \bar{e} * \beta\} * Exp (1-\Delta)$ …Eq (44)

Expected time for system integration = $1/2S * [f^2*2* (N-2) * \mu_p * \bar{e} * \beta] * Exp (1-\Delta)$ …Eq (45)

Program comprehension and test preparation cost for each cycle is given by,

Program comprehension effort = $\sum_{j=1}^{I} k_2m_j + k_3f_j$ ……Eq (46)

Program comprehension cost = $[c_1* (N_1/N) + c_2* (1- N_1/N)] * \sum_{j=1}^{I} k_2m_j + k_3f_j)$ ..Eq (47)

The objective of a development manager is now to determine not only the optimal team size S* and S1* but also the optimal number of iterations I * and the number of modules m_j* to be released in each of the I* cycles, such that the project cost is minimized. This can be formulated as a constraint programming (CP) as following:

Minimize project cost, with respect to S, S_1, I & m and subject to

1) Schedule constraint – Total Project development time < T

2) Budget constraint - Total Project costs < C

3) Functionality constraint – Total of N modules have to be delivered to customer

5. CONCLUSION

Our study identifies the key relationships between coordination effort, team organization and software complexity. We have empirically shown the presence of team effects that affect coordination costs independent of the software complexity effects present in a software construction cycle. Also we showed the presence of significant learning effects that reduce coordination effort at the later stage of the construction cycles, despite an increase in software complexity. These findings form the basis of our analytic model of coordination effort which we use to develop coordination policies for both plan-based, waterfall and iterative models of offshore software development.

6. REFERENCES

Apte, U.M., and Mason, R.O. "Global Disaggregation of Information-Intensive Services," *Management Science* (41:7), 1995, pp 1250-1262.

Beck, K., and Andres, C. *Extreme programming explained: embrace change* Addison-Wesley Professional, Boston, MA, 1999.

Boehm, B.W. "A spiral model of software development and enhancement," *IEEE Computer* (21:5), 1988, pp 61-72.

Dhama, H. "Quantitative Models of Cohesion and Coupling in Software," *Journal of Systems and Software* (29), 1995, pp 65-74.

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

Grinter, R.E. "Using a configuration management tool to coordinate software development," Organizational computing systems, Milpitas, CA, 1995.

Grinter, R.E. "Supporting articulation work using software configuration management systems," *Computer supported cooperative work* (5:4), 1996, pp 447-465.

Herbsleb, J.D., and Mockus, A. "Formulation and preliminary test of an empirical theory of coordination in software engineering," 11th ACM SIGSOFT international symposium on foundations of software engineering, Helsinki, Finland, 2003, pp. 138-147.

Koushik, M.V., and Mookerjee, V.S. "Modeling Coordination in Software Construction: An Analytic Approach," *Information Systems Research* (6:3), 1995, pp 220-254.

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

Kraut, R.E., and Streeter, L.A. "Coordination in software development," *Communications of the ACM* (38:3), 1995, pp 69-81.

Mercer "2003/2004 Global IT Function Salary Differentials," in: *Mercer Consulting Inc*, 2004.

Mookerjee, V.S., and Chiang, R. "A dynamic coordination policy for software system construction," *IEEE Transactions on software engineering* (28:6), 2002, pp 684-694.

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

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

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

Shami, N.S., Bos, N., Wright, Z., Hoch, S., Kuan, K.Y., Olson, J., and Olson, G. "An experimental simulation of multi-site software development," 2004 Conference of the center for advanced studies on collaborative research, Markham, Canada, 2004, pp. 255-266.

Sikora, R., and Shaw, M.J. "A multi-agent framework for the coordination and integration of information systems," *Management Science* (44:11), 1998, pp S65-S78.

Toffolon, C., and Dakhli, S. "A framework for studying the coordination process in software engineering," 2000 ACM symposium on applied computing, Como, Italy, 2000, pp. 851-857.