Optimal Coordination in Distributed Software Development

Hao Xia, Milind Dawande, Vijay Mookerjee
Naveen Jindal School of Management, The University of Texas at Dallas, Richardson, Texas 75080
xiah@utdallas.edu, milind@utdallas.edu, vijaym@utdallas.edu

The construction of a software system requires not only individual coding effort from team members to realize the various functionalities, but also adequate team coordination to integrate the development effort into a consistent, efficient, and bug-free system. On the one hand, continuous coding without adequate coordination can cause serious system inconsistencies and faults that may subsequently require significant corrective effort. On the other hand, frequent integrations can be disruptive to the team and delay development progress. This tradeoff motivates the need for a good coordination policy. Both the complexity and the importance of an optimal coordination policy are further highlighted in the context of distributed software development (DSD), where a software project is developed by multiple, geographically-distributed sub-teams. Coordination in DSD exists both within one sub-team and across different sub-teams. The latter type of coordination involves communication across spatial boundaries (different locations) and possibly temporal boundaries (different time zones), and is a major challenge DSD faces. In this paper, we develop an analytical framework to model these two types of coordination activities in DSD and derive the optimal coordination policy for different types of DSD project. The analytical model show that integration activities by one sub-team not only benefit itself as in co-located development but also help the other sub-team, which implies that higher integration frequency is preferred. We also present analytical results and numerical examples to demonstrate how the project types and characteristics affect the optimal coordination policy and the performance.

Key words: distributed software development; project management; coordination in teams

the bulk of the work was done by a student

1 Introduction

Starting in 2001 with a budget of $223 billion, the F-35 Joint Strike Fighter (JSF) program, which is the most expensive defense project in US history, has again been reported to be lagging in progress. The major reason for the delay – according to a 2011 GAO report on JSF to congressional committees – is that the development of the JSF software, which is essential for about 80% of its functionality, suffers from unexpectedly low productivity. As stated in the report, "the officials underestimated the time and effort needed to develop and integrate the software substantially, contributing to the program's overall cost, schedule problems, and testing delays" (GAO 2011). This problem of low productivity in the construction of a complex software system is not a rare occurrence. Through a review of surveys on software effort estimation, Molokken and Jorgensen (2003) show that about 60-80% projects encounter effort and/or schedule overruns. The 2006 CHAOS report by the Standish Group claims that 19% of the software projects were outright failures (Rubinstein 2007). Together, these observations strongly motivate the need to study approaches aimed at improving productivity in software development.

An incremental development strategy is used in most modern software projects, where the system is broken down into smaller parts and these parts are scheduled to be developed over time and integrated when completed (Cockburn 2008). Accordingly, the construction of a software system using incremental development can be assumed to consist of a series of scheduled *construction cycles*, each of which in turn consists of a development phase followed by an integration phase (Chiang and Mookerjee 2004, Mookerjee 2002). In the development phase, the major activities are coding and unit testing of system function units to realize the functionalities required in the design specification. The workload in this phase is primarily the individual effort of developers; the requirement of coordination among developers is minimal. After development is complete, the outcomes of the individual development efforts need to be integrated into a consistent system, which then serves as a baseline for future construction cycles (Humphrey 1989). The developers gain common understanding of their prior work through team communications and perform system-level testing and inconsistency/fault removal. Naturally, a significant amount of coordination is required in this integration phase.

The following tradeoff in scheduling the construction cycles is critical to the productivity of software construction: on the one hand, developers are typically productive when concentrating on individual development work; therefore, switching to an integration phase too frequently interrupts the fluent progress of their work (Brooks 1995). On the other hand, continuous individual development without adequate coordination may cause serious system inconsistencies and faults that may later require significant corrective effort (Keil, Mann, and Rai 2000). The coordination work is often the crux of a software project, especially a large-scale one. Relative to the individual development effort, the coordination effort required often becomes highly unmanageable and escalates over the budget (Brooks 1995). Consequently, designing a good policy that balances this tradeoff to schedule construction becomes essential for the development team.

Coordination becomes even more significant and complex in distributed software development (DSD), where the software system to be constructed is broken down into subsystems that are assigned to geographically-distributed sub-teams. DSD can exploit a variety of economic factors and has become a common practice in the software industry (Aspray, Mayadas, and Vardi 2006, Damian and Moitra 2006). The subsystems, however, are not isolated and need to be integrated into a consistent system. Accordingly, coordination activities in DSD consist of two types: co-located coordination in the same sub-team and remote coordination among developers across sub-teams. Remote coordination in DSD requires communication and cooperation across spatial boundaries and possibly across temporal and cultural boundaries, which further underline the major challenges DSD encounters (Agerfalk, Fitzgerald, and Slaughter 2009, Jimnez et al. 2009).

In DSD, an incremental development strategy is often applied via two-level construction cycles(Sangwan et al. 2006). At a sub-team level, each sub-team has its own construction cycles, which we refer to as *local cycles*. The integration phase in a local cycle primarily involves co-located coordination among local developers. At the level of the entire team, construction cycles that require participation from all the sub-teams are referred to as *global cycles*. During the integration phase of a global cycle, all sub-teams coordinate together to integrate the work increments of their respective subsystems. A coordination policy schedules the local and global cycles for the project, i.e., prescribes when each sub-team should switch to the integration phase in a local cycle and when the entire team should switch to the integration phase in a global cycle.

The rest of the paper is organized as follows. In Section 2, we briefly review related literature. Section 3 presents a detailed discussion on the co-located and remote coordination activities of DSD projects in general and the corresponding mathematical expressions. In Section 4 we develop an analytical framework for the fundamental type of DSD project that involves two symmetric sub-teams. We solve and discuss the models for both centralized decision making case and distributed decision making case. Finally, Section 5 provides concluding remarks of the study and discuses future research directions.

2 Literature Review

Coordination in DSD context is central to our model. In this section we briefly review previous research on the two related topics: (i) the coordination activities in software system construction, and (ii) distributed software development.

The coordination-related themes in software construction have been studied extensively in literature. Here we review several issues that are central to our model. Under incremental development scheme, the finished and already integrated part of the software system is considered to be relatively consistent and deemed as a baseline which can serve as the official standard for subsequent work (Cusumano and Selby 1995). Grounded on the baseline, increment of new codes/modules are developed during the integration phase of the new construction cycle and then integrated as a whole during the integration phase(Cockburn 2008). Coordination-related activities occurs intensely in the integration phase of construction cycles. The amount of effort required increases with system structural complexity and the density and strength of connectivity between system modules (Dhama 1995). Before the actual integration, an necessary preparation task that needs to be performed is program comprehension: developers achieve common understanding about the functions and behavior of the new codes/modules individually developed via team communications (Bhandari et al. 1993, Robson et al. 1991). According to Leveque, Wilson, and Wholey (2001), Crowston and Kammerer (1998), Gorton and Motwani (1996), this common understanding, or "group awareness", requires massive coordination and is essential for effective collaborative work in system integration (Leveque, Wilson, and Wholey 2001, Crowston and Kammerer 1998, Gorton and Motwani 1996).

The DSD phenomenon began in early 1990's and its strategic significance has been recognized by software industry since the new century. (Prikladnicki, Damian, and Audy 2008, Carmel and Tija 2005). DSD is applicable both to in-house projects and outsourcing projects (Prikladnicki et al. 2007) and in the latter case coordination involves differen-t organizations. A DSD project with sub-teams distributed across different countries is referred to as a Global Software Development (Herbsleb and Moitra 2001). DSD takes advantage of lower cost and higher availability of qualifying human resources; however, it also faces difficulties related to complexity and low efficiency in coordination caused by spacial, temporal, and cultural differences (Agerfalk, Fitzgerald, and Slaughter 2009, Jimnez et al. 2009, Krishna, Sahay, and Walsham 2004). An large and increasing body of literature on the strategic, economic, and technical aspects of DSD has appeared in recen-t years (Jimnez et al. 2009, King and Torkzadeh 2008, Prikladnicki, Damian, and Audy 2008). There exists several studies on making optimal coordination policy in co-located software construction via simulations (Solheim and Rowland 1993, Tamai 1992) and quantitative models (Xu et al. 2009, Chiang and Mookerjee 2004, Mookerjee 2002). However, as commented by Prikladnicki, Damian, and Audy (2008), most of the existing studies analyze at the level of organization instead of project. There are no studies that use analytical tool to model and optimize the coordination activities in DSD to our knowledge.

3 Analytical Specification of Activities in DSD

In this section, we present an analytical specification of development and integration activities in a DSD project. For simplicity, we discuss the special case of two sub-teams A and B that have S_A and S_B developers to develop two subsystems respectively. However, our analysis can be extended to $n \geq 3$ sub-teams. We also assume that all developers are almost identical, all modules require about the same effort to develop, and the two subsystems are very similar in terms of structural complexity.

The work procedure of DSD consists of repeated two-level construction cycles until the entire system is constructed, as depicted in Figure 1. A coordination policy (q_A, q_B, n) is predetermined. In the development phase of a local cycle, sub-team A (or B) works on module coding and unit testing until q_A (or q_B) new modules are developed, and then switches to the local integration phase. After each sub-team finishes n local cycles, the two sub-teams launch a global integration, whose completion finishes the current global cycle.

[Insert Figure 1 here]

We denote the construction effort that occurs during the development phases, the local and the global integration phases, by *development effort*, *local coordination effort*, and *global coordination effort*, respectively. Assuming that the coding and unit testing effort required for each module is quite similar and almost fixed, the total development effort is largely predetermined by the software requirements specification. The total coordination effort, however, is affected by the coordination policy, and is the focus of this study.

3.1 Local Coordination Effort

For simplicity, our analysis hereafter focuses on local coordination effort of sub-team A, which can be easily modified to suit sub-team B. During the development phase in a local cycle of sub-team A, developers are able to construct q_A modules that are consistent with local repository, since the local repository is the modules developed and integrated by them before this local cycle. However, the modules newly developed in this local cycle are still inconsistent with each other, so developers will fully integrate them during the corresponding local integration phase. Besides, after the q_A local modules have been integrated internally, if this local cycle is not the first one in the global cycle, the q_B modules developed and integrated by sub-team B during the previous local cycle is available for sub-team A as a reference, which enables sub-team A perform a partial adjustment on the local q_A modules with respect to the remote q_B modules of the previous local cycle.

More specifically, the local coordination effort has one fixed component and two variable components. The fixed component captures the *preparation effort* required for developers to switch from development to integration. Assuming a developer spends k_0 amount of time on this preparation, the total preparation effort per local cycle is $k_0 S_A$.

The first variable component is the *program comprehension effort*. In order to achieve consensus before actual integration, developers review and comprehend each other's newly developed modules and discuss the integration plan. Naturally, the effort required is proportional to the total number of modules require comprehension. Another factor is the communication overhead, which is typically assumed to be a quadratic function of the group size involved (Brooks 1995). In each local cycle, The comprehension effort of sub-team A per local cycle can be expressed as $h_0 S_A^2 q_A$. If the local cycle is not the first one in the global cycle, there is also comprehension effort for remote q_B modules which equals $h_1 S_A^2 q_B$. Here h_0 and h_1 are local and remote comprehension coefficients influenced by a variety of factors, including organizational structure of the team, relationships between the developers within and across sub-teams, the collaboration tools used, etc.

The second variable component is the actual *integration effort* which may contains two parts as discussed above. First, for each local cycle the newly developed modules need to be fully integrated. The effort required increases quadratically with q_A and can be expressed as $r_0 q_A^2$. Second, if the local cycle is not the first one in the global cycle, there is also the partial adjustment on the local q_A modules with respect to the q_B remote modules of previous local cycle. The effort required to fully integrate the two clusters of modules is $r_1 q_A q_B$, of which sub-team A is responsible for $\frac{q_A}{q_A+q_B}$. Assuming that with the use of distributed revision-control systems and telecommunication tools, sub-team A can perform a portion a of this effort, then it amounts to $a \cdot \frac{q_A}{q_A+q_B} \cdot r_1 q_A q_B$. Similarly, r_0 and r_1 are local and global integration coefficients which capture the collaboration efficiency and structural complexity (strength and density of internal relationships between modules) within and across the subsystems.

3.2 Global Coordination Efforts

The global integration phase follows the completion of the n^{th} local cycles in the global cycle. All the developers in the entire team coordinate, either by coming together at a common location or via telecommunication, both of which incur a much higher cost than coordination within a sub-team. The preparation effort for the entire team is $k_1(S_A + S_B)$, given that a developer needs, on average, time k_1 to prepare.

During the global integration, sub-teams A and B work together to achieve full consistency of their two subsystems developed in this global cycle. First, they need to finish the undone partial integration of the modules developed in the last local cycle. Follow the same analysis in previous subsection, the comprehension effort and integration effort total to $h_1(S_A^2 q_B + S_B^2 q_A) + a \cdot r_1 q_A q_B$. Second, to fully integrate the two subsystems which contain nq_A and nq_B modules respectively and have been partially integrated before, the effort required is $(1 - a) \cdot r_1 n^2 q_A q_B$.

3.3 The Model: Parameters and Objective

With the analytical specification of various activities in DSD, we are able to formulate a general model of the DSD problem. Within the scope of this study, we define the DSD problem as following: choose the size of sub-teams together with the coordination policy for a DSD project such that the team can construct the system at minimum total cost and meeting the project schedule. The model parameters and decision variables are summarized in Table 1 and Table 2 respectively. The base values of the model parameters listed in Table 1 will be used for numerical examples in later sections. They are hypothetical, yet plausible values based on our interview with DSD practitioners.

[Insert Table 1,2 here]

Assuming that the average effort required to develop a module is e person-days, then the total development effort is $(L_A + L_B)e$, which is irrelevant to decision variables. Let the total coordination effort under coordination policy (q_A, q_B, n) and sub-teams size S_A and S_B be $C(S_A, S_B, q_A, q_B, n)$ and we have the general model for the DSD problem below. In the following sections, we will investigate several specific variants of it.

Problem DSD

$$\begin{aligned} \min_{(S_A,S_B,q_A,q_B,n)} \quad & C(S_A, S_B, q_A, q_B, n) \qquad \text{(Objective)} \\ \text{s.t.} \quad & C + (L_A + L_B)e \leq (S_A + S_B)T \qquad \text{(Schedule Constraint)} \\ & S_A, S_B, q_A, q_B, \text{and } n \text{ are positive integers} \end{aligned} \tag{1}$$

4 Analytical Model of the Symmetric Case DSD

It is very common and recommended that, in a DSD project, the sub-teams are balanced in team size and workload for easier management. To analyze this basic and yet important scenario of DSD, in this section we accordingly simplify the general **Problem DSD** into a more specific model adopting symmetric setup: the two subsystems consist of the same number of modules; the two sub-teams have the same size and develop the same number of modules in each local cycle. Therefore, we can let $L_A = L_B = L$, $q_A = q_B = q$ and $S_A = S_B = S$. The decision variable is reduced to a triple (S, q, n) and we discuss two cases different in the decision making process.

4.1 Centralized Decision Making

In the centralized decision making case (CDM for brevity), we assume the project leader is the only decision maker who decides the size of sub-teams and makes coordination policy to schedule the local and global cycles for both sub-teams, as we specified in the general DSD model. We will present the solution procedure to the DSD problem for this case in the rest of this subsection,

4.1.1 Total Coordination Efforts

Following the analysis in Section 3 and notice that $L_A = L_B = L$, $q_A = q_B = q$ and $S_A = S_B = S$, we can calculate the coordination efforts in the steps below. For each sub-team, the local coordination effort in the first local cycle when there is no remote modules available for reference is $lc^1 = k_0S + h_0S^2q + r_0q^2$, and that in the i^{th} local cycle for $i > 1$ is $lc = k_0S + (h_0 + h_1)S^2q + r_0q^2 + \frac{ar_1q^2}{2}$. Therefore, the total local coordination effort for each sub-team in one global cycle which contains n local cycles is $LC = S^2\left(nqh_0 + (n-1)qh_1\right) + n\left(Sk_0 + q^2r_0\right) + \frac{1}{2}(n-1)aq^2r_1$. For each global integration phase, the global coordination effort is $GC = 2\left(k_1S + h_1S^2q\right) + ar_1q^2 + (1-a)r_1(qn)^2$. Together the total coordination effort in one global cycle can be expressed as $CE = 2LC + GC = 2\left(nqS^2\left(h_0 + h_1\right) + nSk_0 + Sk_1 + nq^2r_0\right) + \left(an + (1-a)n^2\right)q^2r_1$

Because qn modules are developed in each global cycle by one sub-team, the total number of global cycles required is $N = L/(qn)$. Therefore, the total coordination effort with decision variables (S, q, n) is $C(S, q, n) = N \cdot CE$. For a typical sub-team size $S = 8$, with other model parameters holding the base values specified in Table 1, Figure 2 shows the total construction effort of the project with respect to coordination policy (q, n).

[Insert Figure 2 here]

4.1.2 Optimal Solution

To solve the corresponding DSD problem , we first treat the sub-team size S as given and find the corresponding optimal coordination policy (q, n) as a function of S. In Appendix A, we show that the optimal coordination policy that minimizes the total coordination effort when sub-team size is S is

$$q^* = \sqrt{\frac{2Sk_0}{2r_0 + ar_1}}, n^* = \sqrt{\frac{k_1\left(2r_0 + ar_1\right)}{k_0r_1(1-a)}} \tag{2}$$

With (q^*, n^*), the number of modules developed in one global cycle per sub-team is $q^*n^* = \sqrt{\frac{2Sk_1}{r_1(1-a)}}$ and the total number of global cycles required is $N^* = L/(q^*n^*) = L\sqrt{\frac{r_1(1-a)}{2Sk_1}}$. The smallest total coordination effort to construct the project when sub-team size is S is $C^* = C(S, q^*, n^*) = 2L\left(S^2\left(h_0 + h_1\right) + \sqrt{2(1-a)Sk_1r_1} + \sqrt{2Sk_0\left(2r_0 + ar_1\right)}\right)$. Accordingly, the shortest possible time of constructing the project with sub-team size S is $T^* = (C^* + 2Le)/2S$.

Increasing sub-team size S leads to a reduction in the time required for development, but increases the effort required for coordination due to communication overhead and team switching cost. We observe the following: 1) The smallest total coordination effort C^* is monotonously increasing with S; 2) When S is small, increasing it will decrease the shortest project duration required. However, above a certain point, adding more people to the team will only slower the progress.

Therefore, it is possible that the schedule constraint T cannot be satisfied, irrespective of the number of developers in the team. It is always desirable to choose the smallest team size S^* that (possibly) achieves $T^*(S) \leq T$. Therefore, it is a two-step procedure of solving the DSD problem. First, find optimal S: solve the equation $T^*(S) = T$, take the ceiling of the smaller solution as the candidate and then check its feasibility. If it is not feasible, the project cannot meet the schedule constraint. Second, find optimal (q, n): get (q^*, n^*) from Equation 2 with the smallest feasible S^*. Compare $C(S, q, n)$ at four points: $(S^*, \lceil q^* \rceil, \lceil n^* \rceil), (S^*, \lceil q^* \rceil, \lfloor n^* \rfloor), (S^*, \lfloor q^* \rfloor, \lceil n^* \rceil), (S^*, \lfloor q^* \rfloor, \lfloor n^* \rfloor)$, and select the point that achieves the minimum.

4.1.3 Various Parameter Effects

In this subsection we discuss the various parameter effects on the optimal coordination policy. q^* and q^*n^* indicate the frequency of local and global integrations respectively: more modules are developed in one local (global) cycle, less frequently local (global) integrations will be launched. In equation 2 we can have q^* and q^*n^* as functions of decision variable S and model parameters k_0, k_1, r_0, r_1, and a. The sign of the first order derivative of q^* and q^*n^* with respect to them are concluded in Table 3.

[Insert Table 3 here]

To better understand the implications of these effects, We compose the project characteristics into three macro constructs: schedule tightness, system complexity, and teamwork environment.

SCHEDULE TIGHTNESS. When the time available to construct the system is reduced, we say that the project becomes tighter. An intuitive measurement of project schedule tightness is Le/T which takes the ratio of the total required development effort and schedule constraint. With the same Le/T, the changes in other project characteristics may affect the required coordination effort and then the schedule tightness, which will be discussed later. Tighter project require larger team size. With the base value of model parameters specified in Table 1, we show the numerical example on how the optimal (smallest yet feasible) team size S^* changes with schedule constraint T. Note that when $T < 838$ days, there is no feasible S. Larger team size increases communication overhead during coordination and thus makes integration less efficient. In other words, frequent local and global integrations are favored for more relaxed projects.

[Insert Figure 4 here]

SYSTEM COMPLEXITY. System complexity measures the difficulty of developing the modules individually as well as integrating them into a consistent system via the two-level construction cycles. System complexity is related to model parameters e, r_0, and r_1 and more complex system should have higher values of them. Other project characteristics kept the same, higher e means more development effort is required and thus tighter schedule; then less frequent local and global integrations should be scheduled according to previous discussion. The intra-subsystem complexity is a major factor of local integration coefficient r_0, and the same applies for the inter-subsystem complexity to global integration coefficient r_1. Higher r_1 makes it attractive to schedule more local and global integrations because the inter-subsystem integration effort increase fast over time with high r_1 and it is accomplished in both local and global integration phases. However, higher r_0 only increases optimal local integration frequency since intra-subsystem integration effort occurs only in local integration phases.

TEAMWORK ENVIRONMENT. Various model parameters can be associated with the teamwork environment of the DSD team. If the sub-teams become more efficient in local/global switching, i.e. with lower value of k_0 / k_1, the local/global integrations should be more frequent. The effects of local/remote comprehension coefficient h_0 and h_1 on coordination policy are indirect, i.e. via schedule tightness and then team size. Higher collaboration efficiency (within and across sub-teams) reduces r_0 and r_1, and the corresponding effects on coordination policy accords with the discussion in previous paragraph. Summarizing the above discussions, we propose that in general a better teamwork environment favors frequent integrations. An exception is partial integration coefficient a. Higher a means that higher portion of inter-subsystem integration effort can be done incrementally (thus more efficiently) in local integration phases, namely that now the local integrations bring more benefits to the remote sub-team, making it better to schedule local integrations more frequently; while at the same time, the remained inter-subsystem integration effort for global integration phases becomes less, which allows some delay of global integrations to save the switching effort.

4.2 Distributed Decision Making

In previous subsection we assume that the project leader can choose all the decision variables to achieve global optimum. However, in reality, the project leader may not have such authority over the sub-team leaders, who often only focus on their own benefits. To capture this phenomenon, we now investigate a distributed decision making case (DDM for brevity) . In this case, all other model settings and parameters stay the same except for the process of making coordination policy: first the project leader determine the local schedule constraint T^L, then the sub-teams choose the local coordination policy q and sub-team size S accordingly, and at last the project leader determine the schedule of global cycles n. Sub-teams only care their local effort, whereas the project leader try to minimize the total effort. We will then investigate the optimal solution that sub-teams and project leader will choose for the DDM case

4.2.1 Optimal Solution

The sub-teams choose local coordination policy q and sub-team size S without considering coordination effort incurred with respect to remote subsystem. Follow the analysis in previous subsection, the effort is $lc = k_0 S + h_0 S^2 q + r_0 q^2$ per local cycle and the total number of local cycles is L/q, then total local effort is $C^L(S,q) = \frac{L(k_0 S + h_0 S^2 q + r_0 q^2)}{q}$. Their objective is to minimize $C^L(S,q)$ and ensure corresponding time spent is no more than local schedule constraint T^L. The optimal solution of the sub-teams is S^+, q^+ which is a response to T^L. Knowing that, the project leader try to find the proper T^L and the global coordination policy n that minimize total coordination efforts. To solve this problem, similarly we ignore the schedule constraint and treat S as given in the beginning and analyze how to choose the best S to meet schedule constraint later. Thereby we can easily get that $q^+ = \sqrt{\frac{Sk_0}{r_0}}$ and $n^+ = \sqrt{\frac{2k_1 r_0}{k_0 r_1 (1-a)}}$. The total effort to construct the entire system is then $C^+ = 2L\left(S^2\left(h_0 + h_1\right) + 2\sqrt{Sk_0 r_0} + \frac{ar_1}{2}\sqrt{\frac{Sk_0}{r_0}} + \sqrt{2}(1-a)\sqrt{\frac{Sk_1 r_1}{1-a}}\right)$. The corresponding total constructing time is then $T^+ = (C^+ + 2Le)/2S$. Similar to the CDM case, it is always more efficient to choose the smallest yet feasible team size S^+ if possible. To induce sub-teams to choose the optimal S^+, the project leader can set a proper local schedule constraint T^L that the sub-teams can barely meet with S^+ developers.

Note that $q^+ > q^*$, $n^+ < n^*$, and $q^+ n^+ = q^* n^*$ when the optimal sub-team size S for the DDM and CDM cases of a DSD problem are the same, i.e. $S^+ = S^*$. The sub-teams will choose less frequent local integrations in the DDM case compared with what the subject leader will choose in the CDM case. The reason is that the sub-teams do not take the benefits of local integrations to the other sub-team into consideration, while the project leader does. Given the sub-optimal, higher q^+ chosen by sub-teams, the project leader will try to minimize total effort by setting n, the number of local cycles in a global cycle lower, resulting that the number of modules developed in a global cycle stays the same. However, there will still be an efficiency loss in the DDM case compared with the CDM case. More specifically,

$$\text{Efficiency Loss} = C^+ - C^* = L\sqrt{\frac{Sk_0}{r_0}}\left(\sqrt{2r_0 + ar_1} - \sqrt{2r_0}\right)^2 > 0 \tag{3}$$

The loss percentage is then Efficiency Loss$/C^*$. The factor affecting this percentage is mainly the weight of inter-subsystem integration efforts that can be done in local integrations, namely ar_1 compared with other integration related parameters. With the parameter values specified in Table 1, $S^+ = S^* = 6, q^* = 25, n^* = 3, q^+ = 38$, and $n^+ = 2$, the efficiency loss percentage is only a negligible 0.6%; whereas if we adjust the following parameter values $h_0(0.01 \rightarrow 0.005), h_1(0.03 \rightarrow 0.015), r_1(0.005 \rightarrow 0.01)$, and $a(0.5 \rightarrow 0.7)$, we will find $S^+ = S^* = 5, q^* = 16, n^* = 4, q^+ = 35$, and $n^+ = 2$, with an efficiency loss percentage up to 5.5%.

Moreover, the original optimal S^* might be infeasible in the DDM case since the total effort and constructing time become greater. It is possible that in the DDM case more developers are required to meet the same schedule constraint, i.e. $S^+ > S^*$ for the same T, which will further reduce the integration frequency and cause significant inefficiency compared with the CDM case. In previous example if the schedule constraint is $T = 970$, the optimal solutions for the CDM and DDM cases will become $S^* = 5, q^* = 16, n^* = 4$ and $S^+ = 6, q^+ = 38, n^+ = 2$, with the efficiency loss percentage rising to 36.3%.

4.2.2 Motivate Local Sub-Teams to Choose Global Optimal Policy

As discussed in previous subsection, in the DDM case sub-teams choose sub-optimal local integration policy and the efficiency loss incurred is significant in some scenarios. The project leader might want to apply a incentive mechanism that can motivate sub-teams to choose global optimal policy. The most intuitive way is to provide the sub-teams with a direct monetary incentive for increasing their local integration frequency. In Appendix B, we show that if each sub-team is rewarded with a fixed reward which is equivalent to $aSk_0r_1/(2r_0 + ar_1)$ amount of effort (around 1 person-day for the previous example) for launching a local integration, it will choose the q^* policy instead of q^+, which will then lead the project leader to choose n^* policy and achieve the same global optimum as in the DDM case.

The total reward paid during the construction of the entire system is worth $2 \cdot L/(q^*n^*) \cdot aSk_0r_1/(2r_0 + ar_1)$. If the sub-teams belong to the company, the reward can be seen as transfer within organization. However, for a outsourcing project, the project leader might want to compare the total rewards with the potential saving in total effort. Take the previous example, when the schedule constraint is $T = 1000$, the total rewards would be 125 person-days whereas the efficiency loss saved is only 97 person-days. However, if $T = 970$, the potential saving of the incentive mechanism will increase to 600 person-days which easily compensate the reward required to pay.

5 Summary and Discussion

In this paper, we attempt to devise an analytical framework for coordination activities in DSD and accordingly derive the optimal coordination policies for DSD projects of several important types. We find that in general a smaller team size is preferable to the extent permitted by the project schedule, and investigate how the optimal schedules of the global and local coordination activities are affected by a variety of project parameters. We show that, unlike in co-located projects, in DSD projects local integration by one sub-team benefits not only itself but the entire team, which is critical in determining optimal coordination policy.

There are two natural limitations of our discussion thus far: our models are homogeneous and static. First, heterogeneity is very common in a real DSD project. It is possible that the sub-teams are asymmetric, i.e. with different number of developers and modules in the subsystem to develop. Our model can be slightly modified to investigate this scenario. Beyond that, the modules of a software system usually vary in functionality and importance, making the efforts required to develop and integrate inconstant, and the structure and complexity of the subsystems are often different dependent on system architecture. Besides, the productivity of developers might also be changing overtime due to various factors such as learning effect. Second, since the progress of development, the level of system consistency, and the required specifications, cannot be fully predicted over the duration of construction of a software project, the consideration of uncertainty gains practical importance. We are actively working on incorporating heterogeneity and uncertainty into our research.

References

Agerfalk, P. J., B. Fitzgerald, S. A. Slaughter. 2009. Introduction to the Special Issue–Flexible and Distributed Information Systems Development: State of the Art and Research Challenges. *Information Systems Research* **20**(3) 317–328.

Aspray, W., F. Mayadas, M. Y. Vardi. 2006 Globalization and Offshoring of Software, a Report of the ACM Job Migration Task Force. Association for Computing Machinery, New York.

Bhandari, I., M. Halliday, E. Tarver, D. Brown, J. Chaar, R. Chillarege. 1993. A Case Study of Software Process Improvement During Development. *IEEE Transactions on Software Engineering* **19**(12) 1157–1170.

Brooks, F. P. 1995. The Mythical Man-Month: Essays on Software Engineering, anniv. ed. Addison-Wesley, Reading, MA.

Carmel, E., P. Tija. 2005. Offshoring Information Technology: Sourcing and Outsourcing to a Global Workforce Cambridge University Press, Cambridge.

Chiang, I. R., V. S. Mookerjee. 2004. A Fault Threshold Policy to Manage Software Development Projects. *Information Systems Research* **15**(1) 3–21.

Cockburn, A. 2008. Using Both Incremental and Iterative Development. *STSC CrossTalk (USAF Software Technology Support Center)* **21**(5) 27–30.

Crowston, K., E. Kammerer. 1998. Coordination and Collective Mind in Software Requirements Development. *IBM Systems Journal* **37** 227–245.

Cusumano, M., R. Selby. 1995. Microsoft Secrets. Free Press, New York.

Damian, D., D. Moitra. 2006. Global Software Development: How Far Have We Come? *IEEE Software* **23**(5) 17–19.

Dhama, H. 1995. Quantitative Models of Cohesion and Coupling in Software. *Journal of Systems and Software* **29**(1) 65–74.

Gorton, I., S. Motwani. 1996. Issues in Co-Operative Software. *Information and Software Technology* **38**(10) 647–655.

U.S. Government Accountability Office. 2011. Joint Strike Fighter: Restructuring Places Program on Firmer Footing, but Progress Still Lags. *Report to Congressional Committees* GAO-11-325.

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

Humphrey, W. S. 1989. Managing the Software Process. Addison-Wesley, Boston.

Jimnez, M., M. Piattini, A. Vizcałno. 2009. Challenges and Improvements in Distributed Software Development: A Systematic Review. *Advances in Software Engineering* **2009** 1-14.

Keil, M., J. Mann, A. Rai. 2000. Why Software Projects Escalate: An Empirical Analysis and Test of Four Theoretical Models. *MIS Quarterly* **24**(4) 601–630.

King, W. R., G. Torkzadeh. 2008. Information Systems Offshoring: Research Status and Issues. *MIS Quarterly* **32**(2) 205–225.

Krishna, S., S. Sahay, G. Walsham. 2004. Managing Cross-cultural Issues in Global Software Outsourcing. *Communica-tions of the ACM* **47**(4) 62–66.

Lejter, M., S. Meyers, S. P. Reiss. 1992. Support for Maintaining Object-Oriented Programs *IEEE Transactions on Software Engineering - Special issue on software maintenance* **18**(12) 1045–1052.

Leveque, L., J. Wilson, D. Wholey. 2001. Cognitive Divergence and Shared Mental Models in Software Development Project Teams. *Journal of Organizational Behavior* **22** 135–144.

Molokken, K., M. Jorgensen. 2003. A Review of Surveys on Software Effort Estimation *Proceedings of the 2003 International Symposium on Empirical Software Engineering* 223–230.

Mookerjee, V. S., I. R. Chiang. 2002. A Dynamic Coordination Policy for Software System Construction. *IEEE Transactions on Software Engineering* **28**(7) 684–694.

Prikladnicki, R., J. L. N. Audy, D. Damian, T. C. Oliveira. 2007. Distributed Software Development: Practices and Challenges in Different Business Strategies of Offshoring and Onshoring. *International Conference on Global Software Engineering* 262–274.

Prikladnicki, R., D. Damian, J. L. N. Audy. 2008. Patterns of evolution in the practice of distributed software devel- opment: quantitative results from a systematic review *Proceedings of the 12th Conference on Evaluation and Assessment in Software Engineering* 100-109.

Robson, D. J., K. H. Bennett, B. J. Cornelius, M. Munro. 1991. Approaches to Program Comprehension. *Journal of Systems and Software* **14**(2) 79–84.

Rubinstein, D. 2007. Standish Group Report: There is Less Development Chaos Today. Available at http://www.sdtimes.com/link/30247.

Sangwan, R., M. Bass, N. Mullick, D. J. Paulish, J. Kazmeier. 2006 Global Software Development Handbook. Auerbach Publications, New York.

Schneidewind, N. F. 1987. The State of Software Maintenance. *IEEE Transactions on Software Engineering* **13**(3) 303–310.

Solheim, J. A., J. H. Rowland. 1993. An Empirical Study of Testing and Integration Strategies Using Artificial Software Systems. *IEEE Transactions on Software Engineering* **19**(10) 941–949.

Stevens, W., G. Myers, L. Constantine. 1974. Structured Design *IBM Systems Journal* **13**(2) 115–139.

Tamai, T. 1992. Experiment on Coordination within Software Development Teams. *Information and Software Technology* **34**(7) 437–442.

Xu, S., Z. Li, Q. Lu, G. Li, L. Huang. 2009. An Optimal Coordination Method for Software Development. *IEEE International Conference on Industrial Engineering and Engineering Management* 623–627.

Appendices

A Optimal Coordination Policy for Symmetric and Centralized Decision Making Case

The objective is to choose (q, n) to minimize $C(S, q, n)$ for a given S. Let$F(q, n) = C(S, q, n)$, then we have

$$F_q = L\left(2r_0 + ar_1 + nr_1 - anr_1 - \frac{2Sk_0}{q^2} - \frac{2Sk_1}{nq^2}\right), F_n = L\left(qr_1(1-a) - \frac{2Sk_1}{n^2 q}\right) \tag{4}$$

Choose positive q and n to let $F_q = F_n = 0$, we have the single solution

$$q^* = \sqrt{\frac{2Sk_0}{2r_0 + ar_1}}, n^* = \sqrt{\frac{k_1\left(2r_0 + ar_1\right)}{k_0 r_1 (1-a)}} \tag{5}$$

The hessian matrix of $F(q, n)$ is

$$H(q, n) = \begin{pmatrix} L\left(\frac{4Sk_0}{q^3} + \frac{4Sk_1}{nq^3}\right), & L\left(\frac{2Sk_1}{n^2 q^2} + (1-a)r_1\right) \\ L\left(\frac{2Sk_1}{n^2 q^2} + (1-a)r_1\right), & \frac{4LSk_1}{n^3 q} \end{pmatrix} \tag{6}$$

It is easy to check that

$$F_{qq}(q^*, n^*) = \frac{\sqrt{2}\left(2r_0 + ar_1\right)\left(k_1\sqrt{(1-a)r_1} + \sqrt{k_0 k_1\left(2r_0 + ar_1\right)}\right)}{k_0\sqrt{Sk_1}} > 0, \text{and} \tag{7}$$

$$|H(q^*, n^*)| = 4(1-a)^2 L^2 r_1^{3/2}\sqrt{\frac{k_0\left(2r_0 + ar_1\right)}{(1-a)k_1}} > 0 \tag{8}$$

Therefore, (q^*, n^*) is the local minimum of $F(q, n)$.

B Motivate Local Sub-Teams to Choose Global Optimal Policy

In DDM, let the original local objective be minimizing $f(q) = \frac{k_0 S + h_0 S^2 q + r_0 q^2}{q}$. A reward mechanism will change it to $f(q) + m(q)$. We want that $q^* = \sqrt{\frac{2Sk_0}{2r_0 + ar_1}}$ is the optimum to the new objective function, which requires that

$$f'(q^*) + m'(q^*) = 0 \Rightarrow m'(q^*) = \frac{ar_1}{2} \tag{9}$$

The simplest reward mechanism is to pay each sub-team a fixed reward which is e-quivalent to x amount of effort (person-days) for launching a local integration. In this case, $m(q) = -\frac{x}{q}$ and $m'(q) = \frac{x}{q^2}$ then

$$m'(q^*) = \frac{r_1}{2} \Rightarrow x = \frac{aSk_0 r_1}{2r_0 + ar_1} \tag{10}$$

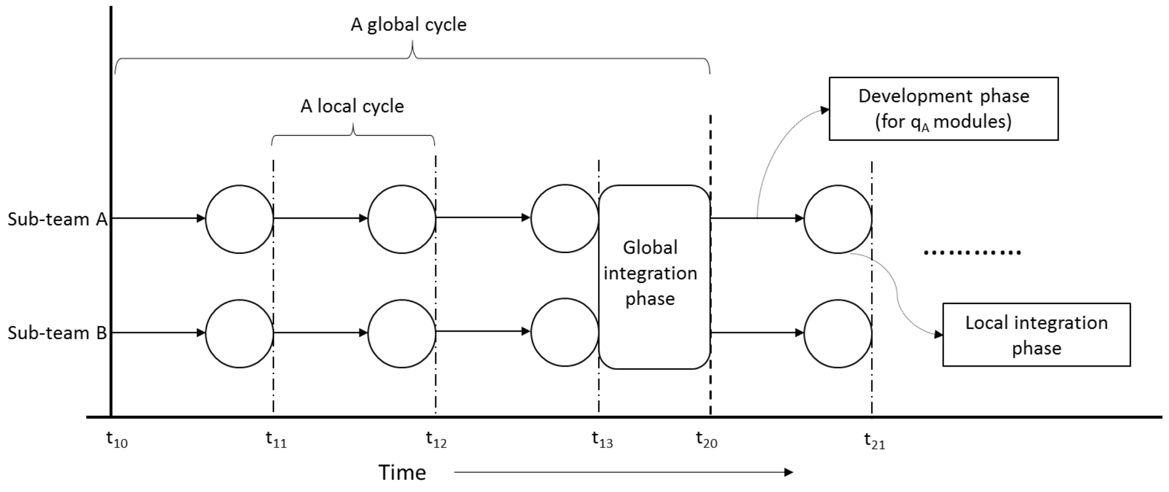

Figure 1: Work Procedure of DSD.

Table 1: Model Parameters.

Model Parameters	**Symbol**	**Base Value**
Schedule Constraint	T	1000 days
Number of Modules in Subsystem A	L_A	1000 modules
Number of Modules in Subsystem B	L_B	1000 modules
Effort Required to Develop a Module	e	4 person-days
Local Switching Coefficient	k_0	0.25 day
Global Switching Coefficient	k_1	1 day
Local Comprehension Coefficient	h_0	0.01 person-days per module
Remote Comprehension Coefficient	h_1	0.03 person-days per module
Local Integration Coefficient	r_0	0.001
Global Integration Coefficient	r_1	0.005
Partial Integration Coefficient	a	0.5

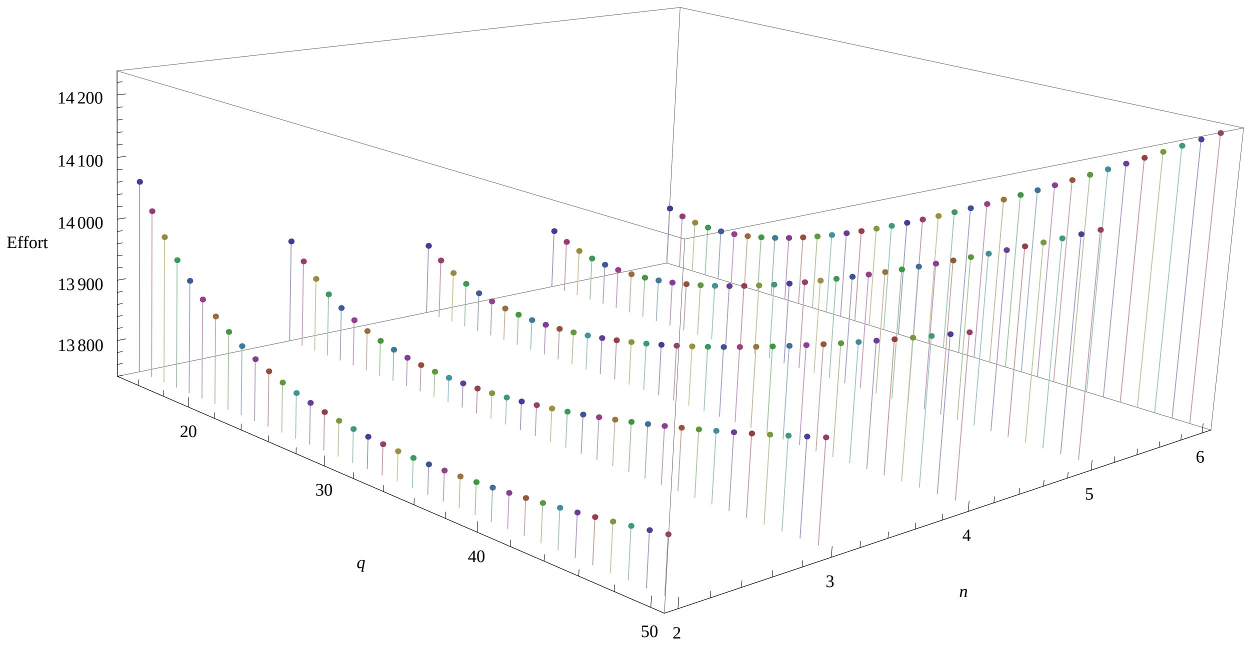

Figure 2: $q, n \rightarrow$ Coordination Effort

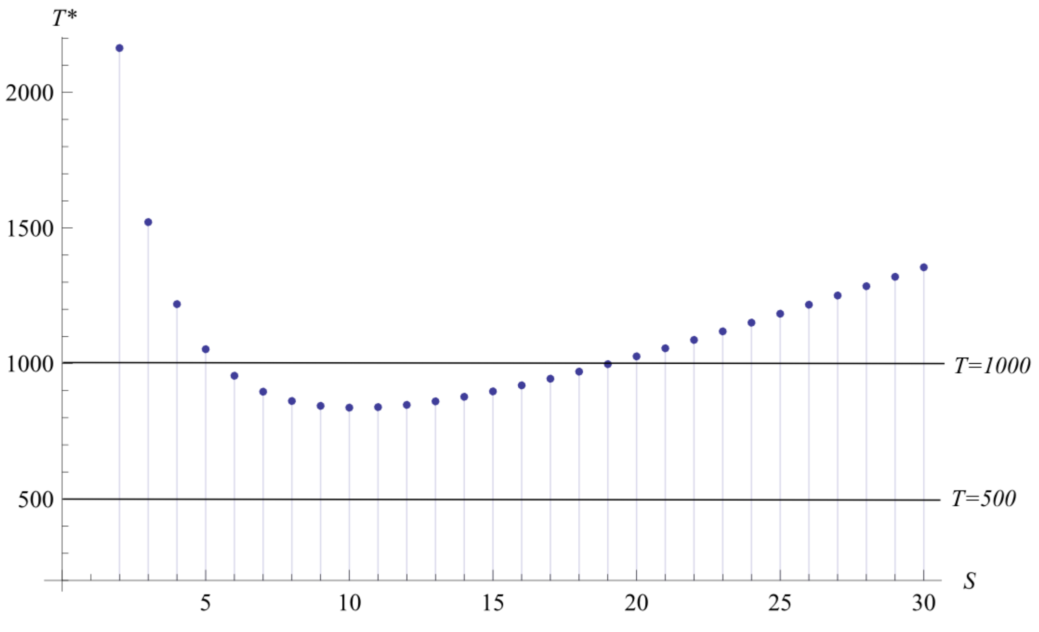

Figure 3: $S \rightarrow T^*$

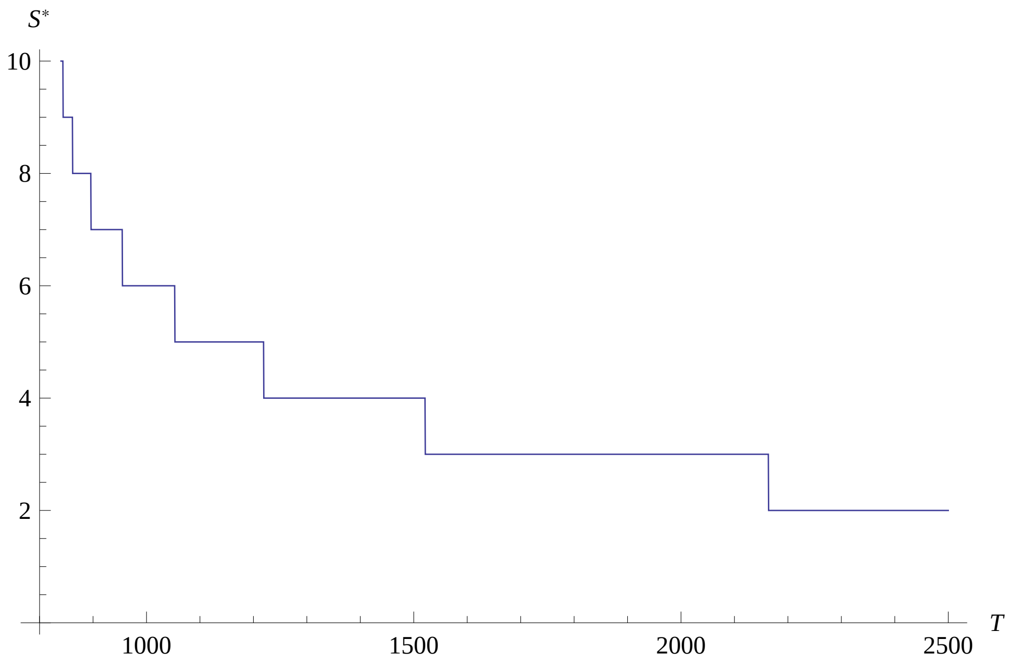

Figure 4: $T \to S^*$

Table 2: Decision Variables.

Decision Variables	**Symbol Used**
Team Size of Sub-team A	S_A
Team Size of Sub-team B	S_B
Number of Modules developed by Sub-team A in a Local Cycle	q_A
Number of Modules developed by Sub-team B in a Local Cycle	q_B
Number of Local Cycles in a Global Cycle	n

Table 3: Parameter Effects on Optimal Coordination Policy.

	∂S	∂k_0	∂k_1	∂r_0	∂r_1	∂a
∂q^*	+	+	0	-	-	-
$\partial(q^* n^*)$	+	0	+	0	-	+