Application Development Using Modifiable Off-the-shelf Software: A Model and Extensions

Qi Feng, Vijay S. Mookerjee and Suresh. P. Sethi
School of Management
The University of Texas at Dallas
Richardson, TX 75083-0688, USA

May 6, 2005

Abstract

In recent times, there has been considerable emphasis on the rapid development of application software using modifiable components purchased from a software vendor. Examples include Enterprise Resource Planning Software, Supply Chain Management Software, etc. that can be acquired as a complete suite or a partial set of related sub-systems. Typically such software can be configured and modified to capture the specialized needs of an application being developed. This paper presents a project management policy to develop applications using Modifiable Off-the-shelf Software (MOTS). In this paper, we develop a general model for software development and coordination and propose a policy to manage coordination using system fault reports (e.g., interface inconsistencies, parameter mismatches, etc.). These reports are used to determine the timing of coordination activities that remove faults. We show that under an optimal policy, coordination should be performed only if a "threshold" fault count has been exceeded. We apply the policy to both MOTS and custom-developed software development and compare the management of MOTS projects with custom-developed projects. A series of numerical experiments are conducted to provide additional insights.

(*Off-the-shelf Software; Software Development; Optimal Coordination Policy*)

1. Introduction

In recent times, there is considerable attention is being paid to the development of software applications using Modifiable Off-the-shelf Software (MOTS). This approach is popular because, to compete in today's economic environment, companies need to respond quickly to changing client demands. MOTS-based development can reduce implementation risk and ensure the quality of project execution. There is also the added promise of simplified upgrades and migration paths.

At a conceptual level, it can be argued that the implementation of an application using MOTS is no different from building a custom application from scratch. However, while MOTS development clearly shares similarities with custom software development, it has its own special requirements. MOTS-based development is more a process of composition and reconciliation, rather than one of creation. In custom-developed software, the producers of the code are usually not the consumers. In contrast, the developers in MOTS-based software are sometimes the end consumers, who configure and modify the software to meet the needs of a business application.

Another difference lies in the nature of programming tasks in the two development paradigms. While MOTS and custom development both require programming, MOTS programming is of a higher order and typically does not disturb the basic code obtained from the vendor. At the same time, similar to custom software, MOTS programming introduces faults that affect the integrity and functionality of the system. The influence of MOTS programming errors, however, is more controlled. MOTS errors are typically restricted to the modifications made to the basic code, and hence the core of the application remains stable. In custom built software, the effects of errors could propagate widely and deeply, and result in significant instability of the code.

Although a sizeable portion of the research on software development has been devoted to issues that deal with the productivity of the software development team (Blackburn, Scudder, and van Wassenhove 1996, van Genuchten 1991), little is known about productivity issues pertaining to MOTS development. Previous work on traditional software development can be broadly classified into two streams: software productivity and system testing.

There has been intense discussion in the research community on factors influencing the productivity of software development team (e.g, Krishnan et al. 2000, Guinan et al. 1998,

Faraj and Sproull 2000). Various models have been proposed to improve software development productivity. Two of the most prominent ones are: Software Capability Maturity Model (CMM) from Carnegie Mellon's Software Engineering Institute (Paulk et al. 1993, Humphrey 1992) and Recommended Approach to Software Development from the NASA Software Engineering Laboratory (NASA 1992).

There has also been a considerable amount of research done on software testing and reliability. Various models have been developed to estimate the number of remaining system faults after some amount of system testing has been done (e.g., Musa 1975, Yamada and Osaki 1985, Goel 1985 and Wood 1996). These models have been extended to consider the effect of imperfect fault detection (Ammann et al. 1994) and incomplete fault removal (Xia et al. 1993). Also, some attention has been paid to the economics of system testing (e.g., Dalal and McIntosh 1994, Singpurwalla 1991).

Software development often proceeds as a series of construction cycles. Each construction cycle consists of a development phase followed by a coordination phase. The main activity during the development phase is to code and unit-test new software modules. In the coordination phase, the newly developed modules are corrected to remove inter-module inconsistencies and integrated with the rest of the system. At the end of each construction cycle a stable (albeit incomplete) version of the system is produced. The question arises: how much (or how long) should the development phase continue before coordination is initiated?

Most previous research to address the above question has proposed models that initiate coordination after a certain amount of development is performed or after a certain period of time has elapsed since the last coordination phase (e.g., Tamai 1992, Solheim and Rowland 1993, Koushik and Mookerjee 1995). However, most existing work has not used fault data to monitor the consistency of the development process. The coordination decision would be more informed if fault data is used to manage an on-going project. Chiang and Mookerjee (2004) propose a quantitative model to examine the interaction between coordination decisions and the occurrence of system faults. Their research uses a heuristic approach to manage custom developed projects. In contrast, we present an optimal solution and also consider the distinction between coordination policies for MOTS projects and custom developed projects.

There are a number of studies in the product development literature that are closely related to our current work. Ha and Porteus (1995) propose a model of concurrent product design

and process design to derive an optimal sequence of predetermined design review points that minimize the time taken to complete both activities. Loch and Terwiesch (1998) also provide an analytical model of concurrent engineering, where upstream and downstream tasks are overlapped to minimize the total time-to-market. Thomke and Bell (2001) address the optimal timing, frequency and fidelity of sequential testing activities. They derive an EOQ-like result and show that the optimal number of tests is the square root of the ratio of the avoidable cost and the cost of a test. The above studies focus on the extent of overlap between downstream and upstream activities so that the total time to complete a project can be minimized. While our model shares certain similarities with concurrent development, our focus is on determining the optimal points during the course of development at which corrective action (fixing of faults) is taken. Unlike the work on concurrent engineering, our model is one of reciprocally, rather than sequentially dependent, activities. The question is one of taking corrective action at optimal points in the course of development. These points are not predetermined in our model but are contingent on observing the state of the system (the number of accumulated faults) at the time of a module release.

In this paper, we develop a general model of software development and derive an optimal fault threshold coordination policy in which the total development effort is minimized. We analyzed a policy in which the project manager allows development to continue if system faults are below a certain (tolerable) level. At any stage, coordination is initiated if system faults have exceeded the threshold for that stage. The stage of development is measured by the number of modules released. Since the number of faults at each stage is random, the release points at which coordination will be performed is not known in advance. Thus, the coordination decisions are dynamic and contingent on the result of system test.

Our study has several noteworthy aspects: (1) An analytical model is developed and the optimality of a threshold policy is established under fairly general conditions; (2) coordination decisions utilize fault data rather than the amount of work performed; (3) we focus on effort minimization rather than time minimization as the objective; and (4) our model captures the effect that correction becomes more costly if coordination is postponed.

The main trade-off in our model is between system integrity and maintaining the flow of development work. Our result shows that the thresholds are more unstable in custom development projects than those in MOTS projects. Moreover, the fluctuation in the threshold values

increases over time. This instability of the thresholds in custom developed projects increases the complexity of the solution procedure, and that of managing the project.

In a MOTS development environment, we show that the control parameters (i.e., the thresholds) are relatively stable during the early stages of the project. Near the end of the project, however, the thresholds reveal an unstable trend with a pattern that depends on the terminal conditions of the project. Results from our study indicate that a skillful team should postpone coordination toward the end of the project, especially when the overhead of switching between development and coordination is high. We also show that a large team should use a low-frequency, resource intensive coordination scheme to manage the development process.

From a theoretical perspective, our model uses a dynamic programming formulation to study the interaction between a general stochastic release process (that provides coordination opportunities) and exponentially growing errors. We believe that this is the first time that such a specific model has been analyzed. While we do get a threshold policy, the thresholds are not monotonic (even under stationary assumptions) as they often are in other models, including those found in the inventory literature. From a practical standpoint, many managerial insights are obtained from our model, and we believe that our model has practical significance and aligns closely to actual practice in software development.

The paper is organized as follows. Section 2 presents the model and proves the optimality of the threshold policy. In Section 3, we compare the optimal threshold policies for custom-developed software and MOTS management and explore project management issues through numerical analysis. In Section 4, we discuss the estimation of the release process and examine the policy behaviors under different inter-release time distributions. Section 5 summarizes and concludes the study.

2. Model Development and the Optimal Coordination Policy

In this section we present a general model of software development. Section 2.1 defines the notation and states the assumptions. Section 2.2 sets up the objective and the dynamics of the problem. In Section 2.3, we characterize the optimal coordination policy. The detailed derivations and proofs are relegated to the appendices.

2.1 Assumptions

We make three basic assumptions about the software development process.

1. *Fault accumulation process.*

   The number of faults $B(\tau)$ in the system that occur if development continues without coordination for τ time units, is given by

   $$B(\tau) = k_c(e^{k_d S\tau} - 1) = k_c(e^{\beta\tau} - 1). \tag{1}$$

   The system faults accumulate at an exponential rate that is proportional to the number of developers S and time τ. The number of the existing faults in the system increases with the fault accumulation rate $\beta = k_d S$. This is consistent with real world experience (Goel 1985). Factors such as system complexity, the team's familiarity with the project domain, and the tools and development environment used, influence the values of fault scaling coefficient k_c and fault compounding coefficient k_d.

2. *Module release process.*

   We assume that modules are released to the system according to a counting process $\{N(t), t \geqslant 0\}$, where t is the time since the beginning of the project. Given that the $(m-1)^{st}$ module is released at time t (i.e., $N(t) = m - 1$), the inter-release time T_t^m to the m^{th} release is assumed to depend only on (m, t). We denote $\Phi_t^m(\cdot)$ as its distribution function.

3. *Coordination cost structure.*

   System testing occurs every time a new module is updated. In other words, no system testing occurs between successive module releases. Most modern development environments use automatic testing tools that make frequent system testing viable. Based upon the test results, project managers decide whether or not to switch the project team from development to coordination. The expected cost of coordination with B_m faults at the m^{th} release is of the form

   $$E(C_m|B_m) = K_m + c_m B_m. \tag{2}$$

Fixed costs K_m favor postponing the coordination decision. These costs consist of the cost for setting up the coordination environment (e.g., compiling and analyzing software metrics, updating project status, and rearranging software and hardware tools), communication costs among team members, and the cost for program comprehension. Fault correction costs $c_m B_m$ represent the effort needed to fix B_m faults (we use *effort* and *cost* interchangeably throughout this paper) at the m^{th} release. Since the complexity of fault correction typically increases with the size of the system, it is reasonable to assume that the average cost of correcting a fault is nondecreasing with the release number, i.e., $c_1 \leqslant c_2 ... \leqslant c_M$. From (1), the faults accumulate exponentially over time. Thus, postponing coordination not only leads to an increased unit correction cost, but also a faster growth in the fault count. This is consistent with real world projects where it is generally observed that the cost of fixing an error increases with the time at which the error is corrected.

2.2 The Objective and Dynamic Programming Equations

Our goal is to minimize the total effort to construct a software system given that M modules are to be developed. Total construction effort is the sum of the *development effort* to build new modules and the *coordination effort* to correct the inter-module errors. Since the total number of modules M is planned a priori, the development effort is fixed. Thus we focus on optimizing the expected value of the latter, i.e., the objective is to minimize

$$J(x_1, ..., x_M) = E\left[\sum_{m=1}^{M} C_m(B_m|x_1, x_2, ..., x_{m-1})\delta(x_m)\right], \tag{3}$$

where x_m, $m = 1, 2, ..., M$, is a binary decision variable representing whether or not to coordinate at the m^{th} release, $\delta(x_m)$ is an indicator function whose value is 1 when x_m equals coordination and 0 otherwise, and B_m is the number of faults observed at the m^{th} release. Except for Section 3.4.3, where alternative situations are considered, we have assumed throughout this paper that coordination is enforced at the last release. Hence, x_M is fixed and $\delta(x_M) = 1$. Also note that the random variable B_m is the number of faults that have accumulated since the most recent coordination.

We are now ready to set up the dynamics of the coordination policy. Let $V_m(B, t)$ be the minimum expected cost at the m^{th} release, given that B faults are observed in the system and t

units of time has been spent in devolvement. Also define $V_m^n(B,t)$ and $V_m^c(B,t)$ as the minimum expected costs of continuation (i.e., no coordination) and coordination, respectively, at the m^{th} release. The objective is to find the optimal policy that minimizes the total cost

$$V_m(B,t) = \min\left[V_m^c(B,t), V_m^n(B,t)\right]. \tag{4}$$

Note that if B were equal to zero, we would not coordinate because of the presence of switching costs. Thus faults will start accumulating in the system according to (1). In the absence of discounting, the cost for the m^{th} release with zero faults is given by the expected cost for the $(m+1)^{st}$ release, i.e.,

$$V_m(0,t) = V_m^n(0,t) = EV_{m+1}[k_c(e^{\beta T_t^{m+1}} - 1), t + T_t^{m+1}]. \tag{5}$$

If B faults are found and coordination is carried out, then the total of the cost of coordination at the m^{th} and the future cost after this coordination is given by

$$V_m^c(B,t) = K + cB + E\left[V_{m+1}[k_c(e^{\beta T_t^{m+1}} - 1), t + T_t^{m+1}]\right] = K + cB + V_m(0, t + T_t^{m+1}). \tag{6}$$

If instead we continue the development process, then the cost at the m^{th} release will be the expected future cost of carrying B faults to the next release, i.e.,

$$V_m^n(B,t) = E\left[V_{m+1}[B + (B + k_c)(e^{\beta T_t^{m+1}} - 1), t + T_t^{m+1}]\right]. \tag{7}$$

Equations (4), (6) and (7) constitute the dynamic programming equations for the problem under consideration.

2.3 The Optimal Coordination Policy

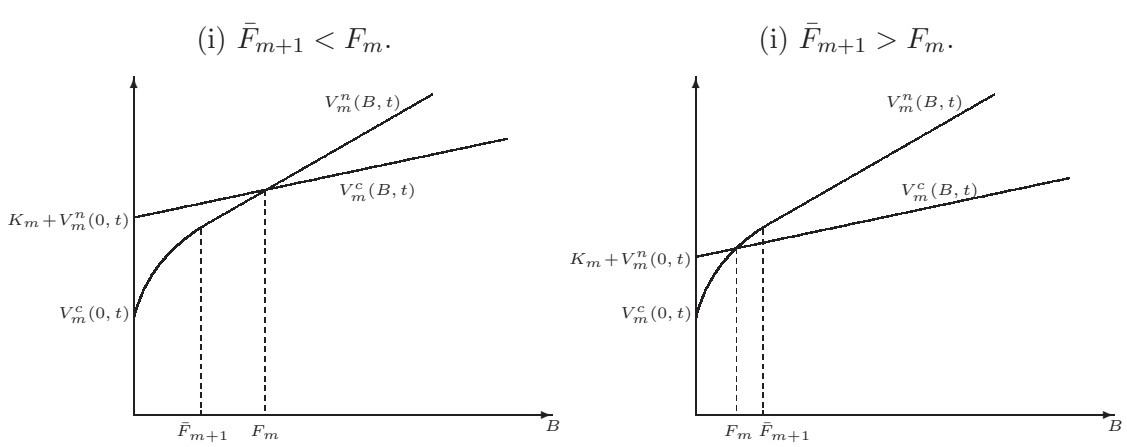
In this section we derive the threshold policy and prove its optimality. As shown in Appendix A, there is a closed-form expression for the threshold at the $(M-1)^{st}$ release (c.f. Lemma 1), and the cost functions can be derived explicitly up to the $(M-2)^{nd}$ release. For a general value of m, however, the corresponding expressions involve transcendental equations that need to be solved numerically.

THEOREM 1 *There exists a unique threshold F_m for the m^{th} release, $m = 1, ...M$, such that it is optimal to coordinate if $B \geqslant F_m$ and not coordinate otherwise. Also,*

$$V_m(B,t) = \begin{cases} V_m^n(B,t) & \text{when } B < F_m, \\ V_m^c(B,t) & \text{when } B \geqslant F_m. \end{cases} \tag{8}$$

It is easy to see that both the coordination and the continuation costs at each release are increasing with the number of faults found in the system. From (6) we see that the cost of coordinating at the m^{th} release with B faults increases linearly in B at the rate c (the expected cost of fixing a fault), beginning at $K_m + V_m(0,t)$. The cost function $V_m^n(B,t)$ starts at $V_m(0,t)$ and increases in a concave manner in B up to $B = \bar{F}_{m+1}$, where $\bar{F}_{m+1}$ is the least upper bound of F_{m+1} evaluated at the m^{th} release. After the number of faults exceeds $\bar{F}_{m+1}$, this cost increases linearly in B at the rate $q_t^{m+1} (= cEe^{\beta T_t^{m+1}} > c)$. At $B = 0$, the cost of continuation is lower than the cost of coordination. However, the continuation cost increases faster than the coordination cost in the entire range of the number of faults. Therefore, once the continuation cost exceeds the coordination cost, it always stays above the coordination cost. This suggests that there is a threshold number of faults beyond which it is always optimal to coordinate. Fig. 1 depicts the optimal cost functions and illustrates the uniqueness of the optimal threshold.

Figure 1: Cost functions and optimal threshold policy.



Below we provide a sketch of the proof for Theorem 1 while referring the reader to Appendix A for the complete proof. We first show that the threshold at the $(M-1)^{st}$ release is unique (c.f. Lemma 1). To give an inductive proof, we next allow the threshold to be unique at any general m and prove that the threshold is also unique at $m-1$. To establish this, we use the fact that the optimal cost function is increasing and concave in B (c.f. Theorem 2).

The uniqueness of the thresholds makes the fault threshold policy easy to implement. Intuitively, the existence of such a policy tells project managers that they can focus on development when the system has a low number of faults, but need to stop development and correct the faults when the number of faults in the system has exceeded a certain level. The intuition behind

the threshold policy can be explained as follows. Because of the presence of (fixed) switching costs, it is not economical to coordinate when the system has a low number of faults, i.e., the cost for coordination is greater than that for continuation. On the other hand, because new faults accumulate exponentially with existing faults, the costs of fault correction can increase dramatically at higher levels of existing faults. Hence after the fault level reaches a certain point, the cost of coordination is lower than that of continuation; hence coordination should be performed.

3. Policy Behavior: Custom Versus MOTS Development

We describe models for custom-developed projects and the MOTS projects in Section 3.1 and Section 3.2, respectively. Superscripts I and II are used to represent custom-developed and MOTS projects, respectively. We compare policy behavior of the two development processes along several discussions. The structural properties of the threshold curves are analyzed in Section 3.3. In Section 3.4, we examine the sensitivity of the optimal threshold curves with respect to the model parameters. Project scheduling issues are discussed in Section 3.6.

3.1 Custom-Developed Projects

In custom software development environments, the effort required to correct inter-module inconsistencies gets higher as more modules are released. Thus, at the m^{th} release, the fixed cost consist of two parts: a switching cost K and a module integration cost bm. The switching cost $K = k_a S$ is proportional to the number of developers in the team S, which represents the cost of setting up the coordination environment. The module integration cost $bm = k_b S^2 m$ represents the effort of the team members to understand each other's work, identify inter-module inconsistencies and integrate the m modules. This cost increases with the square of the team size S (Brooks 1995). The program comprehension coefficient k_b measures the effectiveness of the team to identify and correct inter-module inconsistencies. The value of k_b is influenced by the tools and the technology used. For custom-developed projects, we can rewrite the cost of integration at the m^{th} release as follows:

$$E(C_m^I|B) \quad = \quad k_a S + k_b S^2 m + c_m B = K + bm + c_m B. \tag{9}$$

3.2 MOTS Projects

Due to the nature of MOTS development process, the core of the application remains stable during the development process. Because the basic code obtained from the vendor has clearly defined interfaces, the program comprehension effort spent in a MOTS project is much lower than a custom-developed project. Thus, neglecting program comprehension effort, the coordination cost at a given release m with B faults is given by

$$E(C_m^{\text{II}}|B) \;\; = \;\; k_a S + c_m B = K + c_m B. \tag{10}$$

3.3 Optimal Threshold Curves

In this section, we analyze the properties of the threshold policy for both custom-developed and MOTS projects. For comparison purposes, we focus on the problems where module inter-release times are independent and identically distributed, i.e., $\Phi_t^m = \Phi$. Also $c = c_1 = ... = c_M$. Note that under these conditions, the state variable t can be dropped from the dynamic programming equations (4), (6) and (7) and $V_m(B)$, $V_m^n(B)$ and $V_m^c(B)$ can be used to denote optimal cost functions.

In Section 3.3.1 and Section 3.3.2, we characterize the structural properties of the threshold curves. In Section 3.4, we illustrate these properties through numerical examples and analyze sensitivity of the optimal policy with respect to the model parameters.

3.3.1 Boundedness of the Thresholds and the End-of-Horizon Effect

We first observe that the optimal thresholds are bounded by F_{M-1}.

PROPOSITION 1 $0 < F_m^{\text{I}} < F_{M-1}^{\text{I}} = [K + b(M-1)]/c(q-1)$ *and* $0 < F^{\text{II}} < F_{M-1}^{\text{II}} = K/c(q-1)$ *for* $1 \leqslant m < M-1$, *where* $q = Ee^{\beta T}$.

The proof is stated in Appendix B, where we assume $F_m^i > F_{M-1}^i$, $i = \text{I}, \text{II}$, for some m and get a contradiction. The boundedness of the threshold values reduces the complexity of computation.

Proposition 1 suggests that the threshold for the $(M-1)^{st}$ release is high, and therefore we care less about controlling the fault level when there is only one more module to build. Since $F_{M-2} < F_{M-1}$, if we coordinate at the $(M-2)^{nd}$ release, we may be able to avoid one coordination, and the associated coordination costs at the $(M-1)^{st}$ release. However, at the

$(M-1)^{st}$ release, we know that coordination will surely take place at the next release; hence we are more likely to postpone coordination at the $(M-1)^{st}$ release.

The *end-of-horizon effect* is closely related to the notion of a terminal condition in dynamic programming. We will revisit this point later in Section 3.4.3, where different terminal conditions are considered.

3.3.2 Stability and Scalability of MOTS Projects

In a MOTS project, one might expect the threshold curves to be monotonically increasing because of the *end-of-horizon effect*. However, this is not the case. Monotonicity is disturbed because the decision points are "discrete" in the sense that we cannot carry out coordination at any time, but only when a new module is released. In general, therefore, the optimal coordination policy is not a monotone policy.

Nevertheless, as described in Propositions 2 and 3 below, the optimal threshold curve at the early stages of a MOTS project reveals some useful structural properties. As shown in our numerical study, these properties do not carry over to custom-developed projects, indicating that managing a custom-developed project is more complex than managing a MOTS project.

As we will observe from the examples presented in Section 3.4, an interesting fact about the threshold policy for MOTS projects is that the thresholds are not very sensitive to m near the beginning of the project. This property is stated precisely in Proposition 2 below.

PROPOSITION 2 *In a MOTS project, define $l = M - m$ and $f_l = F_m^{\text{II}}$, where F_m^{II} is the optimal threshold value in Theorem 1. When $M \to \infty$, $\lim_{l\to\infty} f_l$ exists.*

The convergence of the threshold values follows from the convergence of the optimal cost function. Let $W_l(B) = V_m^{\text{II}}(B) - V_m^{\text{II}}(0)$ with $V_m(\cdot)$ defined in (4), (6) and (7). Then, as shown in Appendix B, $W_l(\cdot)$ converges uniformly. This is a direct result of applying the Arzela-Ascoli theorem (c.f. page 245 of Rudin 1987).

Proposition 3 below indicates that the threshold values are not affected by the project size.

PROPOSITION 3 *Consider two otherwise identical MOTS projects with M^1 and M^2 modules to be released. Let $\{F_i^1\}_{i=1}^{M^1-1}$ and $\{F_i^2\}_{i=1}^{M^2-1}$ be the optimal threshold values for the two projects, respectively. Then $F_{M^1-k}^1 = F_{M^2-k}^2$ for $1 \leqslant k < \max\{M^1, M^2\}$.*

The size M of the project determines the number of recursions in the dynamic programming procedure. However, project size does not affect the instantaneous costs at each stage of the dynamic program. Hence, the result in Lemma 3 is understandable.

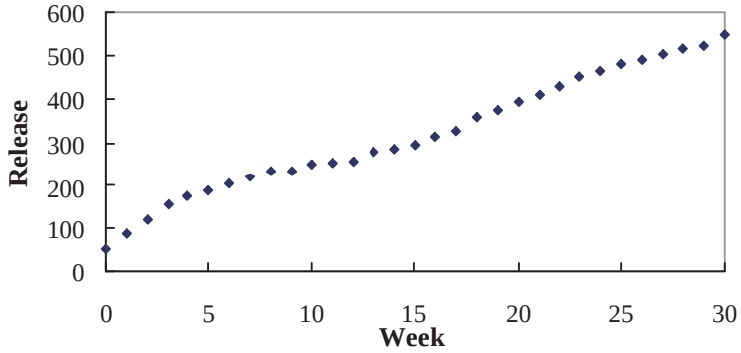
3.4 Numerical Examples and Sensitivity Analysis

In this section, we first analyze the module release process of a custom-developed project. We then present a computational method to obtain optimal threshold values and study how these values are influenced by various model parameters.

3.4.1 Goodness-of-Fit: An Empirical Example

We study data obtained from the Software Engineering Laboratory NASA's GOES Attitude Ground Support System project. The project data is collected over a 30-week period. During this period, the total number of modules in the system increases from 50 to 548. In each week, 1 to 37 new modules are added to the system. Fig. 2 shows that the total number of modules at the beginning of each week increases approximately linearly over time. Thus, we conjecture that the release process is stationary over time and test whether the data supports a Poisson process with a mean release rate of $((548 - 50)/30 = 16.6)$ per week.

Figure 2: Release number over time.



The test results are presented in Table 1, where we note $\chi^2 = 29.14 < \chi^2_{0.005,26}$. Thus, we cannot reject the hypothesis that the observation comes from a Poisson distribution. Indeed the close agreement between the observed and expected frequencies suggests that the Poisson distribution is a good fit.

Table 1: Test of poisson distribution.

Number i	Sample Frequency f_i	Poisson Probability	Expected Frequency e_i	χ^2 test
$\leqslant 10$	5	0.05912	1.8917	
11	8	0.09992	3.1975	
12	9	0.15637	5.0040	
13	11	0.22846	7.3107	
14	14	0.31393	10.0457	
15	17	0.40852	13.0725	$\chi^2 = \Sigma_i (e_i - f_i)^2 / e_i$
16	18	0.50665	16.2128	$= 29.14$
17	19	0.60248	19.2792	$\chi^2_{0.005,26} = 44.31$
18	20	0.69085	22.1071	
19	21	0.76806	24.5778	
20	22	0.83214	26.6285	
21	23	0.88280	28.2495	
22	23	0.92102	29.4727	
23	25	0.94861	30.3555	
24	28	0.96769	30.9660	
25	28	0.98036	31.3715	
26	28	0.98845	31.6303	
27	28	0.99342	31.7895	
28	28	0.99637	31.8838	
29	28	0.99806	31.9378	
30	28	0.99899	31.9677	
31	28	0.99949	31.9837	
32	29	0.99975	31.9920	
33	30	0.99988	31.9962	
34	31	0.99994	31.9982	
35	31	0.99998	31.9992	
36	31	0.99999	31.9996	
$\geqslant 37$	32	1.00000	31.9998	

3.4.2 Sensitivity Analysis

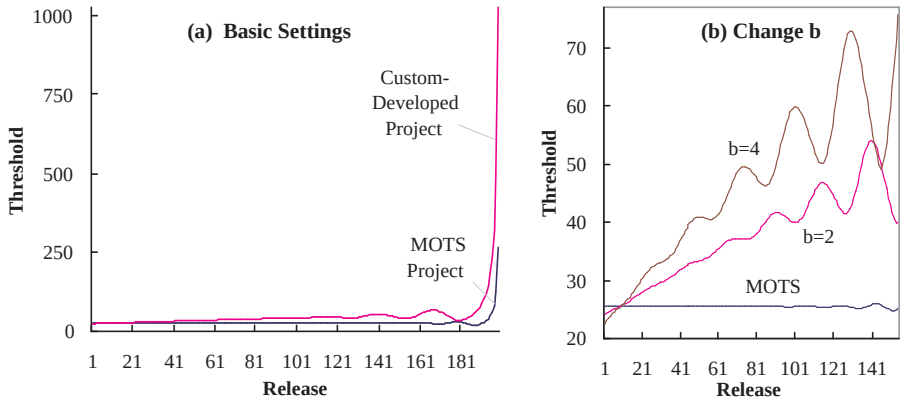
Here we provide an overview of our numerical procedure to obtain the thresholds. Since we have closed-form expressions for $V^n_{M-1}(B)$, $V^c_{M-1}(B)$ and F_{M-1}, we start by directly computing these values. Next, using the dynamic programming equations (4), (6) and (7), we compute $V^n_{m-1}(B)$ and $V^c_{m-1}(B)$ for each B using the values of $V^n_m(B)$, $V^c_m(B)$ and F_m. We then compare $V^n_{m-1}(B)$ and $V^c_{m-1}(B)$ and get the threshold F_{m-1}. Note that we only need to calculate the optimal cost values for $B \in [0, F_m]$. At the end of the procedure, we get the complete curve representing the threshold policy.

In the following examples, we assume that the developers are working independently and in parallel, and the release process follows a Poisson process with estimated rate of λ modules per week. Alternative distribution functions are treated in Section 4, where methods of estimating

non-Poisson release processes are discussed.

The basic parameter settings are as follows. The project has a total of M $(= 200)$ modules and the team has $S(= 10)$ developers. On average $\lambda(= 0.2)$ modules can be developed each hour. Thus the estimated total development effort is L $(= MS/\lambda = 10,000)$ person hours. A developer needs $k_a(= 14)$ hours to switch between development and coordination. The values $k_d = 0.001$ and $k_c = 20$ imply that, starting with a consistent system, a 10-person team generates about 10, 25, 46 faults in the first, the second, and the third week, respectively, of development without coordination. It takes a developer an average of $c(= 10)$ hours to fix one fault during the coordination phase. In the custom-developed project, the value $k_b = 0.02$ indicates that it takes the team $2(= k_b S^2)$ hours per module to comprehend each other's work and to understand inter-module inconsistencies. Fig. 3 depicts the optimal threshold curves under the basic setting. Since the threshold policies exhibit an end-of-horizon effect and closed form solutions are available for the ending thresholds (see Proposition 1), we will focus on examining the behavior of the threshold policies at early stages of a project. Next, we consider a number of parameter values and study their effects on the optimal threshold policy.

Figure 3: The optimal threshold curves.



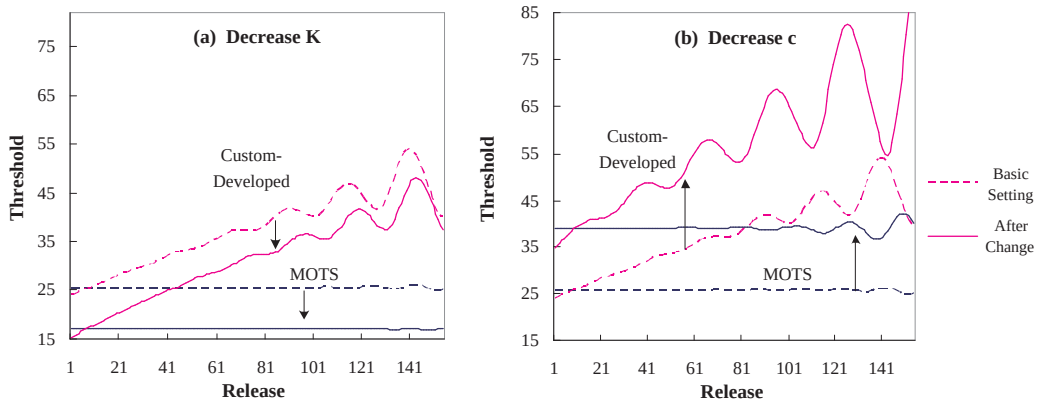
1. **Coordination costs.**

   In a MOTS project, $k_b = 0$ and the threshold curve of the first 100 releases is stable at 25.64, which is consistent with our result in Proposition 2. In the custom-developed project, $k_b > 0$ and the threshold values are not stable. To take a closer look at the effect of the program comprehension effort b, we plot the policy curves at early stages (from $m = 1$ to $m = 141$) of the project in Fig. 3(b). A larger value of b results in a larger

fluctuation of the thresholds and a steeper increasing trend of the threshold curve. Thus, for custom-developed projects, as program comprehension effort increases, the project becomes more difficult to manage.

According to (1), the ending threshold value F_{199} is proportional to the switching cost K and the reciprocal of the error fixing cost c. Fig. 4(a) shows the threshold curves shift down when the switching cost K decreases from 14 to 7. In custom-developed projects, the threshold value decreases by roughly a constant. In MOTS projects, the threshold policy is more stable. In Fig. 4(b), the correction cost c is decreased from 10 to 5 and the threshold curves shift up. In custom-developed projects, the effect of reducing c is amplified over time. In MOTS projects, the threshold policy also becomes less stable, but the values of the thresholds fluctuate less than those in custom-developed projects.

Figure 4: The threshold curves for different cost parameters.



2. **Fault generation speed.**

   As seen from Fig. 5, decreasing the fault accumulation rate $\beta(= k_d S)$ or the fault scaling coefficient k_c decreases the fault generation speed and increases the instability of the threshold curves. Changing the fault accumulation rate has relatively less impact on the average threshold values. Lowering the fault scaling coefficient lowers the average threshold values.

3. **Module release rate.**

   It is quite intuitive that the module release rate λ and the fault accumulation rate β should influence the thresholds in opposite directions (see Fig. 6).

Figure 5: The threshold curves for different fault generation speed.

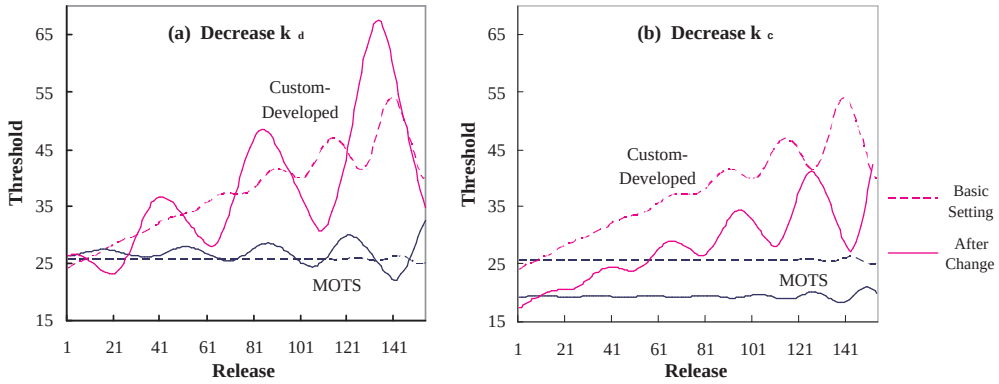
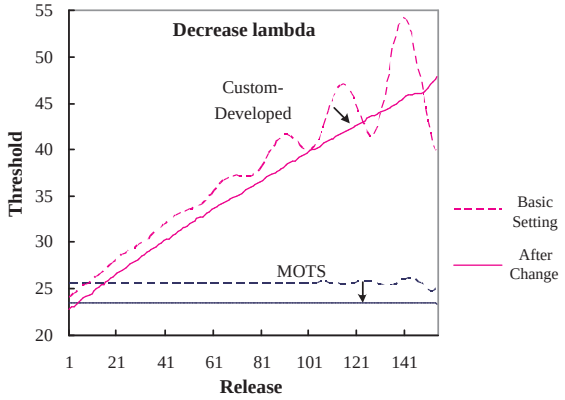


Figure 6: The threshold curves for different inter-release rate.



Overall, a project team should coordinate less frequently when the switching costs and the program comprehension costs are high and the fault correction costs are low. The converse is also true: coordination should be more frequent when the switching costs are low and the fault correction costs are high. Moreover, changing the cost parameters has a greater impact on the threshold curve near the end of the horizon. On the other hand, the impact of changing the fault accumulation rate or the module release rate is slight initially, but quite significant toward the end of the project. Also, changes in the fault scaling coefficient affect the early thresholds. However, this effect diminishes near the end of the project (the ending threshold F_{199} does not depend on k_c). Thus some factors make early project management more difficult, while others contribute more towards difficulties near the end of the project.

3.4.3 Terminal Conditions

As we have seen previously, the threshold curve increases dramatically toward the end of the horizon. This is because coordination is enforced at the last release. To better understand

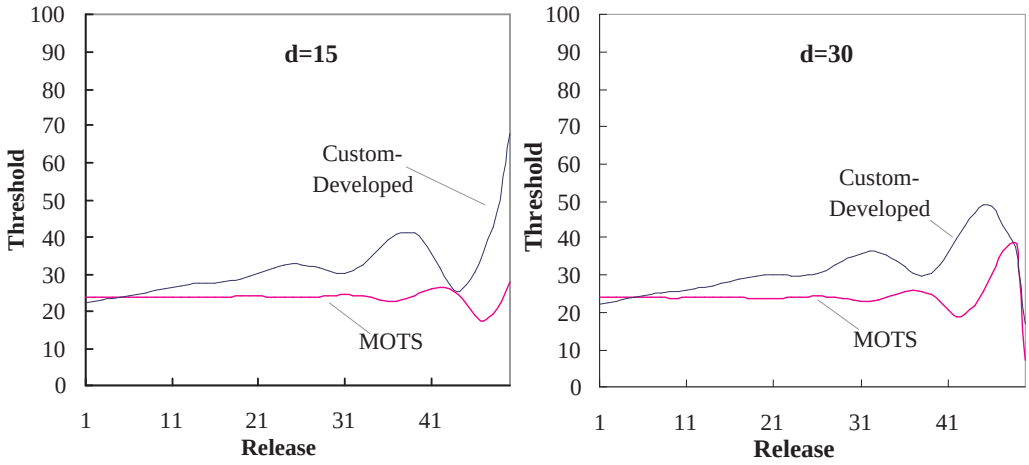
the end-of-horizon effect, we discuss a variation of the model where we allow some decision flexibility at the last release. We consider the option of choosing not to coordinate at the last release, but pay a penalty cost d for each fault left in the final product. Alternatively, we can coordinate at the end and launch the product error-free. Typically, penalty costs (e.g., customer service and goodwill loss) are higher than correction costs. This drives companies to improve product quality and avoid penalties due to errors. Thus, we assume that $d > c$. The cost function for the last release becomes

$$V_M(B) = \begin{cases} dB, & \text{do not coordinate,} \\ K + bM + cB, & \text{coordinate.} \end{cases}$$

From the above, it is easily seen that there is a unique threshold $F_M = (K + bM)/(d - c)$ for the last release.

We notice that the proofs of optimality (Theorem 1), boundedness (Proposition 1), convergence (Proposition 2), and scaling (Proposition 3) of the thresholds do not depend on the terminal condition, i.e., the functional form of $V_M(B)$. Hence these results still hold.

Figure 7: Optimal policy under different terminal conditions.



As shown in Fig. 7, the threshold curves at the early stage of the projects do not depend on the terminal condition. However, the ending threshold value decreases in the penalty cost (other parameters are the same as in Section 3.4.2), which determines the shape of the threshold curve near the end of the problem horizon. Intuitively, if the penalty cost d is very low (say, very close to the correction cost), then F_M is large, and many faults are left in the final product. If the penalty cost is very high ($d >> K + bM - c$), then the product should not be released with

any faults ($F_M = 0$). This gives rise to the situation in the original model, where coordination is enforced at the last release.

3.5 Computational Issues

The scaling property in Proposition 3 together with the convergence property in Proposition 1 allow us to easily compute the threshold values for even large MOTS projects. The optimal threshold policy can be approximated by a *fixed* threshold policy near the beginning of the project. However, in a custom-developed project, the number of modules affects the instantaneous costs, and the convergence or the scaling property does not hold. The computational complexity of the custom-developed project increases dramatically with the size of the project.

In practice, it would be convenient if the optimal policy was easy to compute and would be implemented with relatively less administrative effort. To address these goals, we present and examine the performance of a simple release-based policy for custom-developed projects.

3.5.1 A Release-Based Coordination Policy

In this section, we provide a method to compute an optimal release-based policy and compare it with the optimal threshold policy.

We consider a simple deterministic coordination policy that can be found as follows. Find a sequence of natural numbers $\{n_1, ..., n_l\}$ with $1 \leqslant n_1 < n_2 ... < n_l = M$; the policy calls for coordination at these releases. The value of $\{n_1, ..., n_l\}$ can be obtained by applying a shortest path algorithm (see the details in Appendix C).

The major drawback of a release based-policy is that the decision does not take into account observed fault data. To examine when such a policy would perform well, we calculate the probability of coordination at each release.

PROPOSITION 4 *Suppose that the inter-release times are independent. Let* P_m, $1 \leqslant m \leqslant M$, *be the probability of coordinating at the* m^{th} *release. Also, let* $P_0 = 1$. *Then* $P_M = 1$ *and*

$$P_m = \sum_{i=0}^{m-1} P_{m|i} P_i (1 - P_{i+1|i}) ... (1 - P_{m-1|i}) \quad \text{for } 1 \leqslant m \leqslant M - 1, \tag{11}$$

where

$$P_{m|i} = \int_{\frac{1}{\beta} ln \frac{F_m + k_c}{k_c}}^{\infty} d\Psi(t) \tag{12}$$

and $\Psi = \Phi^i * ... * \Phi^m$ *is the convolution of* Φ^i,...,Φ^m.

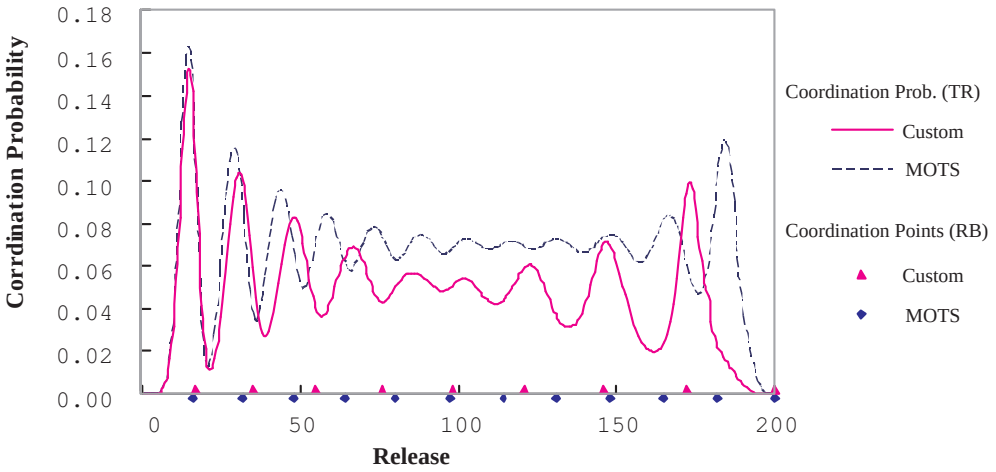
The above proposition allows us to compute the coordination probability at each release recursively. Since we start the project with zero faults (as if we have coordinated right before the project starts), an artificial *release zero* with $P_0 = 1$ is added to serve as an initial condition. In Appendix B, we derive expressions for $P_{m|i}$, the probability of coordinating at the m^{th} release given that the last coordination occurred at the i^{th} release, and $P_{m|0}$, the probability that the first coordination happens at the m^{th} release. When the modules are released to the system according to a Poisson process with a constant rate λ, we have

$$P_{m|i} = \int_{\frac{1}{\beta} ln \frac{F_m + k_c}{k_c}}^{\infty} \frac{\lambda e^{-\lambda t} (\lambda t)^{m-i-1}}{(m-i-1)!} dt.$$

Fig 8 gives coordination probability curves when the optimal threshold policy is applied. The triangular marks and the diamond marks in Fig. 8 represent the optimal coordination points picked by the release-based policy in the custom-developed and MOTS projects, respectively. The probability P_m of coordination at release m is not stable over time. The fluctuation is large near the two ends and small in the middle of the project. Coordination is most likely to take place at the modes of the curve. We observe that the optimal release-based policy picks the coordination points at approximately the modes of the probability curve. For example, in the MOTS project, the coordination probabilities (the dotted curve)under the optimal policy are close to zero for the first 10 releases, increases dramatically to 0.164 at the 15^{th} release, and then decreases from the 16^{th} release. Due to fluctuations in the probability curve, it is very likely that the first coordination in the optimal threshold policy will occur at the 15^{th} release. In this case, the release-based policy captures the averaging behavior of the fault accumulation process, and calls for the first coordination at the 16^{th} release.

The probability curves of both custom and MOTS projects have the property that they are less skewed in the middle than at the two ends. However, we observe certain differences between the two types of project in Fig. 8. For the MOTS project, the probability curve under the optimal threshold policy has a modal points at approximately every 15 releases (except for the ending stage of the project), and the release-based policy coordinates every 16 releases. For the custom-developed project, however, the distance between every two modes of the probability curve increases over time, and the release-based policy calls for less frequent coordination as more modules have been released.

Figure 8: Probability of coordination.



It is not hard to imagine that if the probability curve is stable over time and it is equally likely to have coordination at any release, then applying the released-based policy would lead to poor cost performance. This suggests a release-based policy may behave efficiently when P_m curve highly skewed or unstable. Table 2 compares the total coordination costs $V_0(0)$ and the total number of coordination instance N for custom-developed projects under optimal release-based policy with those under the optimal threshold policy. We denote TR and RB to represent the threshold policy and the release-based policy. The release-based policy behaves efficiently when the fixed coordination costs are high and error correction costs are low. Also, slow fault accumulation and fast module release leads good performance of release-based policy.

Table 2: Comparison of release-based policy with threshold policy.

	Base	b=0	$k_a = 7$	$c = 5$	$\beta = 0.005$	$k_c = 10$	$\lambda = 0.1$
$V_0^{TR}(0)$	6892.64	4820.38	6254.25	4791.58	3552.05	4791.62	13598.3
$V_0^{RB}(0)$	7523.28	5050.92	6807.62	5376.71	4097.11	5376.71	14671.5
% Difference	9.15%	4.78%	8.85%	12.21%	15.34%	12.21%	7.89%
EN^{TR}	9.38	12.90	10.69	7.10	4.01	7.10	20.22
N^{RB}	9	12	11	6	4	7	18

3.6 Project Scheduling

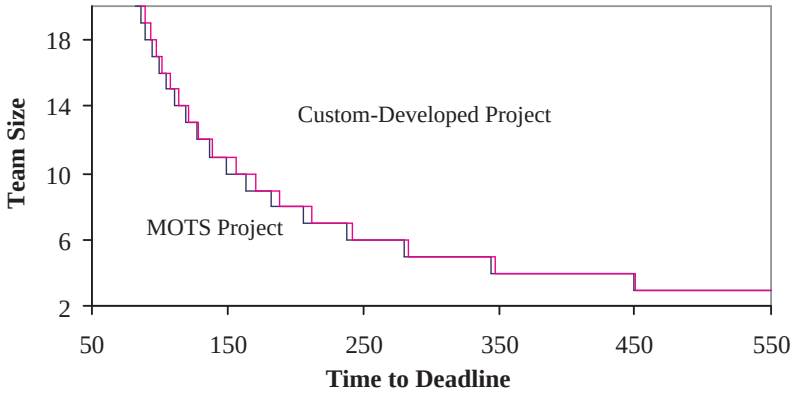
An interesting question is: Given a certain development environment (e.g., technology, tools, hardware, developer skill, etc.), how many developers will be needed to complete the project within a given time deadline? Moreover, how does the project time deadline influence the optimal coordination strategy? To answer this question, we have to consider both the development and the coordination efforts under the optimal threshold policy. By assumption, the

expected total development effort is simply $ED = L$ and the expected total coordination effort is $EC = V_0(0)$.

1. **Minimal team size for a given project time frame.**

   Consider a small project of $M = 20$ modules that requires $L = 1000$ person hours of development effort. The modules are released according to a Poisson process. Other parameter values are the same as in Section 3.4. Fig. 9 is a plot of the smallest feasible team size S against the given project schedule T. A tighter schedule requires a larger team size. A larger team can usually finish the project earlier, but the marginal productivity (coordination effort per unit development) decreases with the team size. Thus, when the project schedule becomes more stringent, more developers are needed to compensate for the fall in productivity as well as the stringency of the schedule. We note that the marginal productivity decreases faster in the custom-developed project than in the MOTS project due to higher program comprehension effort involved.

Figure 9: Required team size for a given project schedule.



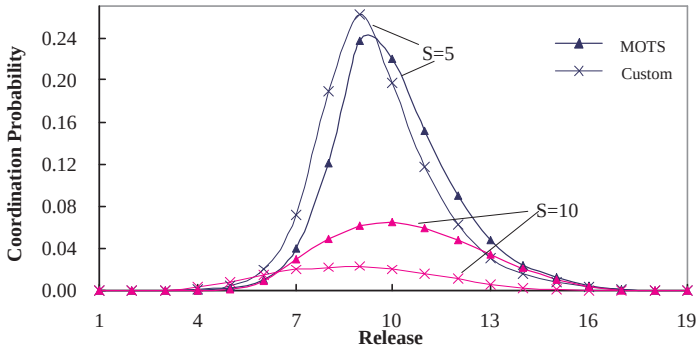
2. **Coordination Probabilities.**

   Increasing team size has two effects: The development and fault correction processes speed up, but faults are also generated faster. In addition, more effort is required to switch between the development and coordination environments. In order to further explore policy behavior, we compute the probability of coordination at a given release under the optimal threshold policy.

   Fig. 10 shows coordination probability at each release — the coordination probabilities become more uniform as the team size increases. In this example, the coordination

probabilities exhibit a bell-shaped character, indicating that the fault level is more likely to exceed the threshold in the middle of the project than it is at the two ends. We also observe that the coordination probability decreases with the team size faster in custom-developed projects than in MOTS projects. Also, the modes of the probability curves for custom-developed projects are always smaller than those for MOTS projects, indicating that the optimal policy tends to coordinate earlier in custom-developed projects.

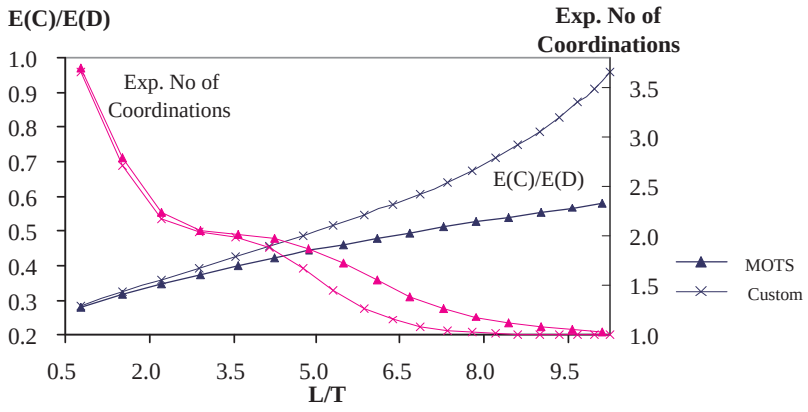
Figure 10: Coordination probabilities for different team sizes.



3. **Frequency of Coordination.**

In Fig. 11, we examine the coordination effort against the project schedule. The ratio L/T is an indicator for how stringent the project schedule is, and EC/ED is a measure of the productivity of the project team. As shown in Fig. 11, the optimal policy for a tighter project allocates more resources to coordination. More effort is required in each coordination, but coordination is carried out less frequently. Thus, an intense, low-frequency coordination scheme is the optimal choice for tight projects.

Figure 11: Expected number of coordination instances.



As the project schedule becomes tighter, the portion of coordination effort allocated in a custom-developed project increases faster than that in a MOTS project. This is due to the increased program comprehension effort involved in a custom-developed project. In terms of the expected number of coordination instance, the difference between a custom-developed project and a MOTS project increases and then decreases as the project schedule gets tighter.

At first glance, the intense and low-frequency coordination scheme for tight projects suggested in Fig. 11 may appear to be in direct contradiction to Microsoft's approach of using daily builds (Cusamano and Selby 1997). Note, however, that the objective in our study is to minimize the total *effort* to develop a system subject to a completion time constraint. In Microsoft's case, project managers may care more about reducing the time-to-market instead of lowering development costs. Finally, we note that Microsoft's practice of using daily builds may occur in our model, i.e., a low threshold-curve that prescribes daily builds may minimize the total development effort in some projects.

4. Non-Poisson Release Processes

In reality, the inter-release time distribution can be obtained by fitting the historical data to a known distribution as in Section 3.4.1. When no historical data are available, we discuss a different approach to estimate the release process. In addition, we analyze the optimal threshold policy under different inter-release distributions.

When no historical data are available, a method commonly used in project management is PERT (Project Evaluation and Review Technique). The project managers first estimate the maximum inter-release time T_{max}, the minimum inter-release time T_{min} and the most likely inter-release time T_{mode}. The mean μ and the variance σ^2 of the inter-release time are estimated as

$$\mu = \frac{T_{min} + 4T_{mode} + T_{max}}{6}, \quad \sigma^2 = \frac{(T_{max} - T_{min})^2}{36}. \tag{13}$$

The above estimates are used to fit a *Beta* distribution function with parameters (a, b), where

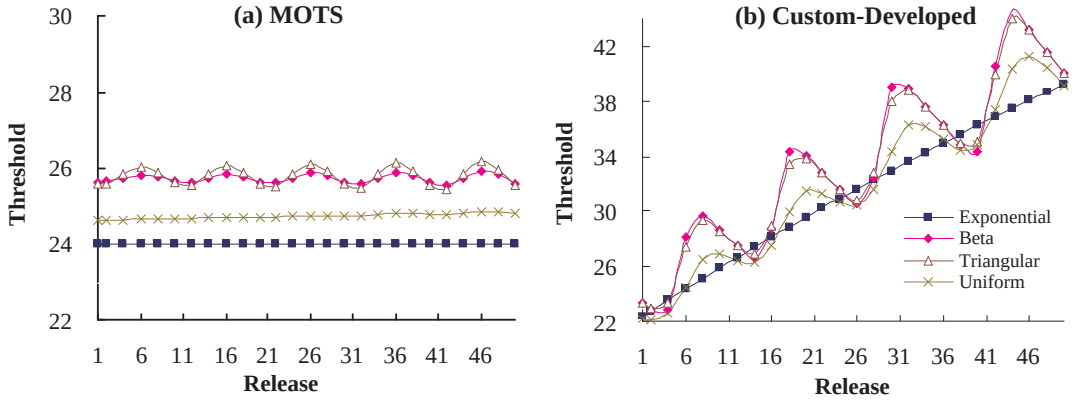
$$a = \frac{\mu - T_{min}}{T_{max} - T_{min}} \left(\frac{(\mu - T_{min})(T_{max} - \mu)}{\sigma^2} - 1 \right) \quad b = \frac{T_{max} - \mu}{\mu - T_{min}} \times a.$$

The advantage of the beta distribution is that it is defined on a finite interval and could have its modal value anywhere on the interval. As a result, the beta distribution provides a good approximation under fairly general situations and is commonly used in project management. Sometimes in order to simply computation, the triangular distribution may also be used. When T_{mode} cannot be specified, the Uniform distribution function can be used. The uniform distribution is a special case of Beta distribution when $a = b = 1$. Note that the estimated variance of a uniform random variable $(T_{max} - T_{min})^2/12$ is always larger than that of a beta distribution $(T_{max} - T_{min})^2/36$ defined on the same range.

1. **Comparison of Different Distribution Families.**

   As an example, suppose that inter-release times are independent and identically distributed. Assume $T_{min} = 2$ hours, $T_{max} = 15.4$ hours, and $T_{mode} = \mu = 8.7$ hours. The cost parameters and faults accumulation parameters are the same as in the basic example described in Section 3.4. Fig. 12 shows the optimal threshold policy when different inter-release distributions are used. The threshold curves under different distributions follow a similar pattern. The threshold values are more stable at the early stages when exponential inter-release times are used. We also note from Table 3 that when the variance σ^2 is larger, the threshold curve tends to be more stable, while the expected total coordination effort $V_0(0)$ becomes larger.

Figure 12: Different distributions of inter-release time.



2. **Impact of Mean and Variance.**

   In Table 4, we consider the optimal policy behavior with different estimates of T_{min}, T_{max}, and T_{mode} for the Beta distribution. The results for other distributions are similar. The

Table 3: Comparison of different distributions.

Distribution		Exponential	Beta	Triangular	Uniform
T_{min}		0	2	2	2
T_{max}		∞	15.4	15.4	15.4
μ		8.7	8.7	8.7	8.7
σ^2		75.69	5	7.48	14.96
Custom-Developed Software	cost	14616.7	14499.7	14506.8	14359.9
	F_1	22.37	23.38	23.42	22.21
	F_{199}	982.26	1026.65	1025.14	1020.57
MOTS	cost	2116.37	2090.71	2092.14	2134.38
	F_1	24.00	25.65	25.60	24.61
	F_{199}	146.92	153.33	153.56	152.65

basic setting is the same as in Section 3.4. Again, we observe cost reductions with a lower mode and a lower variance of the inter-release time distribution.

Table 4: Beta distribution with different mode and range.

Setting		basic	Mode= 3.1	Mode= 12.3	$[4, 13.4]$	$[6, 11.4]$
μ		8.7	6.8	9.9	8.7	8.7
σ^2		4.99	4.99	4.99	2.45	0.81
Custom-Developed Software	cost	14499.7	8472.45	18387.5	14610.7	14487.6
	F_1	23.38	26.93	24.84	23.38	23.40
	F_{199}	1026.65	1828.84	795.445	1028.23	1029.25
MOTS	cost	8358.13	4773.77	10661.2	8470.3	8351.1
	F_1	25.65	26.97	25.39	25.44	25.20
	F_{199}	153.33	273.54	118.98	153.80	153.95

In Fig. 13, we fix $T_{min} = 2$ and $T_{max} = 15.4$, and change T_{mode}. We observe that the threshold curve fluctuates more with a larger T_{mode}. Under the MOTS development environment, the threshold decreases with T_{mode}. Under the custom-developed environment, the threshold curves for different values of T_{mode} reveal a similar trend.

Figure 13: Threshold policy for different modes of the Beta distribution.

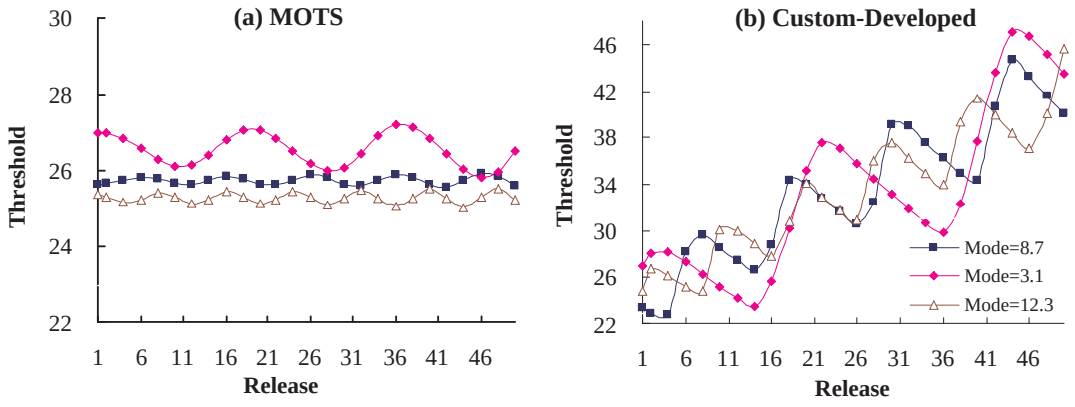
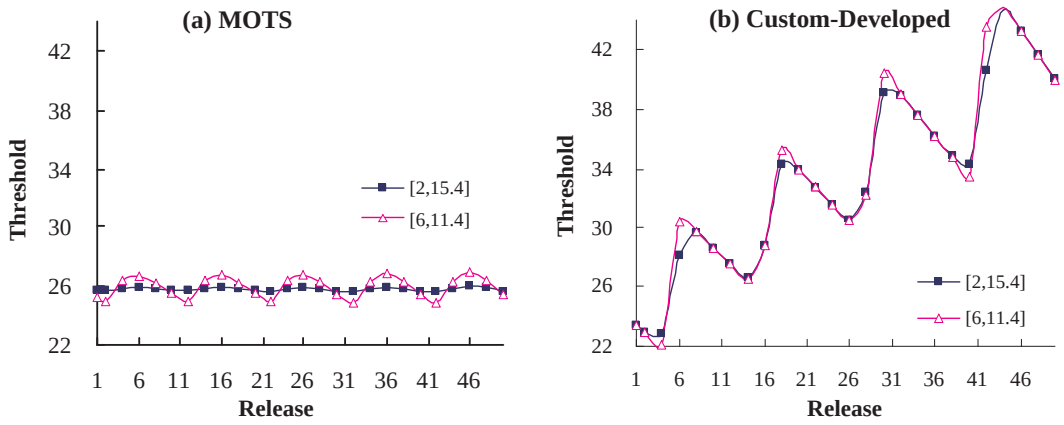


Figure 14: Threshold policy for different ranges of the Beta distribution.



In Fig. 14, we fix $T_{mode} = \mu = 8.7$ and change $[T_{min}, T_{max}]$ to $[T_{min}+\delta, T_{max}-\delta]$ for different values of δ. Clearly, the variance σ^2 increases with $T_{max} - T_{min}$. The threshold curves with different σ^2 values reveal a similar trend and fluctuation pattern. A larger variance in the inter-release time generates a more stable threshold curve. This is consistent with our observations in Fig. 12.

5. Conclusions

We develop a model to examine coordination policies in software development projects. The optimality of a fault threshold policy is established. We also discuss policy behavior in custom-developed and MOTS projects and summarize our findings as follows. In a custom-developed project, the optimal threshold values during early stages of the project reveal a fluctuating pattern with an overall increasing trend. In addition, the fluctuations increase over time. In a MOTS project the optimal threshold levels are relatively stable expect near the end of the project. The shape of the policy curve near the end of the planning horizon depends on the terminal conditions imposed for the project. In general, the computational complexity for the optimal policy of a custom-developed project is much higher than that for a MOTS project. Further, custom-developed projects require more administrative effort to manage.

Several major implications are uncovered by our analysis. First, project managers should pay more attention to test reports at the beginning of the project (i.e., when the thresholds are stable). The optimal threshold curves for custom-developed projects fluctuates over time with an overall increasing trend, while those for MOTS projects are stable. Thus, it is more

important to use system test reports to manage the development processes for custom-projects. On the other hand, coordination can be carried out at approximately fixed time intervals for MOTS projects.

Toward the end of a project, a larger proportion of resources should be allocated to complete the project, i.e., to development. Postponing coordination toward the end of the project is especially beneficial for a skillful development team, that can release the modules quickly and not make too many mistakes. The *end-of-horizon* effect is closely related to what is enforced at the end of the project. If the fault penalty is high, the project ends with a final coordination. If the penalty cost is not very high relative to the coordination cost, a final coordination may not be a good choice. Instead, a final fault threshold $F_M > 0$ can be optimal. We must point out that our model does not address the situation where it is not possible to launch the final product with no faults and still finish the project within the required schedule (i.e., the problem becomes infeasible). However, it is easily seen that a small modification can handle this situation by specifying the lowest feasible F_M. That is, we allow F_M faults in the final product, and compute the resulting project duration under optimal threshold policy. The new formulation of the model would allow us to complete the project in schedule with the smallest number of faults F_M.

Third, the trend and the fluctuation pattern of the threshold curve do not depend on the inter-release time distribution family and the variance of the inter-release time. The stability of the threshold curve increases with the variance of the inter-release time. When the mean release times are low but highly variable, the optimal threshold curve fluctuates with a small amplitude. The indicates that a team with high efficiency but some variability in performance may still follow a relatively stable coordination strategy until the project is close to its end.

Fourth, the position of the threshold curve depends largely on the cost parameters. When the team incurs high overhead in switching between development and low costs in fault-correction, the optimal policy tends to postpone coordination; because of high administrative costs, frequent coordination is not economical. On the other hand, a team with low switching costs can coordinate more frequently and keep the level of system faults relatively low even near the end of the project. The fluctuation of the threshold curve is mainly governed by the interaction between the release process and the fault accumulation process. The project manager should carefully manage coordination along the highly fluctuating threshold curve for a

team with slow fault generation and high development productivity.

A final insight is that while a large team can handle projects with tight deadlines, the administrative complexity associated with a large team makes coordination costly. A less frequent, but more resource intensive coordination policy should be followed here.

The modeling approach in this paper can also be applied to other similar problems in this area. For example, the ideas developed here can be extended to the area of software maintenance. Feng, Mookerjee and Sethi (2003) discuss the use of change request thresholds that strike an optimal balance between maintenance costs and user waiting costs. This stream of research is part of our on-going interest in the use of quantitative models to study problems in the area of software management.

Appendices

A. Optimality of the Threshold Policy

In (6), (7) and (4), we constructed a cost function for the module release process. Before getting into the optimality proof, we need to derive the form of the cost function for each release.

At the last release, $B > 0$ (almost surely), and we are required to coordinate regardless of the number of faults B. Thus,

$$V_M(B,t) = K_M + cB, B > 0. \tag{14}$$

For the $(M-1)^{st}$ release, we have the following.

LEMMA 1 *There is a unique threshold* $F_{M-1} = K_{M-1}/(c_M E e^{\beta T} - c_{M-1})$ *for the* $(M-1)^{st}$ *release, and the optimal cost function is given by*

$$V_{M-1}(B,t) - V_{M-1}(0,t) = \begin{cases} K_{M+1} + cB, & \text{if } B \geqslant F_{M-1}, \\ qcB, & \text{if } B < F_{M-1}\ . \end{cases} \tag{15}$$

PROOF. If we coordinate at the $M - 1^{st}$ stage, then the cost is

$$\begin{aligned} V^c_{M-1}(B,t) &= K_{M-1} + c_{M-1}B + EV_M(k_c(e^{\beta T_t^M} - 1), t + T_t^M)) \\ &= K_{M-1} + c_{M-1}B + K_M + c_M k_c E(e^{\beta T_t^M} - 1), \\ V^n_{M-1}(B,t) &= EV_M(B + (B + k_c)(e^{\beta T} - 1), t + T_t^M) \\ &= K_M + c_M[B + (B + k_c)E(e^{\beta T_t^M} - 1)]. \end{aligned}$$

Then,

$$V^c_{M-1}(B,t) - V^n_{M-1}(B,t) = K_{M-1} + c_{M-1}B - c_M B E e^{\beta T_t^M}. \tag{16}$$

Note that the right-hand side in (16)is strictly decreasing in B. Also $V^c_{M-1}(0,t) - V^n_{0-1}(B,t) = K_{M-1} > 0$ and $V^c_{M-1}(\infty,t) - V^n_{M-1}(\infty,t) < 0$. Thus, there exists a unique

$$F_{M-1} = \frac{K_{M-1}}{c_M E e^{\beta T_t^M} - c_{M-1}},$$

such that (16) is positive when $B < F_{M-1}$ and negative when $B > F_{M-1}$. We deduce that the optimal cost function at the $M-1^{st}$ stage is given by (15). Also note that both $V^c_{M-1}(B,t)$ and $V^n_{M-1}(B,t)$ are linear in B. Hence, $V_{M-1}(B,t)$ is concave in B.

□

THEOREM 2 *$V_m(B,t)$, $m = 1, ...M$, is increasing concave in B.*

PROOF The increasing part follows simply from induction. We know that $V_M(B,t)$ is increasing in B. Let $V_{m+1}(B,t), V_{m+2}(B,t), ..., V_M(B,t)$ be increasing in B. Then from (6), $V^c_m(B,t)$ is increasing in B. And from (7), $V^n_m(B,t)$ is increasing in B. It follows from (4) that $V_m(B,t)$ is increasing in B.

To prove concavity, we first observe that $V_M(B)$ is linear, and thus concave in B. It easily follows that $V_{M-1}(B)$ is concave in B. Let $V_{m+1}(B), V_{m+2}(B), ..., V_M(B)$ be concave in B. From (4), $V_m(B) = \min[V^c_m(B), V^n_m(B)]$. Then, for $0 \leqslant \theta \leqslant 1$, we have

$$\begin{aligned}
& \theta V^n_m(B_1,t) + (1-\theta)V^n_m(B_2,t) \\
= \;& \theta E\left[V_{m+1}\left(B_1 + (B_1+k_c)(e^{\beta T_t^{m+1}} - 1), t + T_t^{m+1}\right)\right] \\
& +(1-\theta)E\left[V_{m+1}\left(B_2 + (B_2+k_c)(e^{\beta T_t^{m+1}} - 1), t + T_t^{m+1}\right)\right] \\
\leqslant \;& E\left[V_{m+1}\left(\theta[B_1 + (B_1+k_c)(e^{\beta T_t^{m+1}} - 1)] + (1-\theta)[B_2 + (B_2+k_c)(e^{\beta T} - 1)], t + T_t^{m+1}\right)\right] \\
= \;& V^n_{m+1}\left(\theta B_1 + (1-\theta)B_2, t + T_t^{m+1}\right).
\end{aligned}$$

Thus, $V_m(B,t)$ is concave in B.

□

PROOF OF THEOREM 1. Since $V^c_m(B,t)$ is linearly increasing in B at a slope of c_m, we only need to show that $V^n_m(B,t)$ is increasing in B at a rate that is larger than c_m. We use induction here.

From Lemma 1, the results holds for $m = M-1$. Assume that there is a unique threshold F_{m+1} for the $(m+1)^{st}$ release. To compute $V_m^n(B,t)$, we first note that F_{m+1} is a random variable at the m^{th} release. Let $\bar{F}_{m+1}$ be the maximum possible value of F_{m+1} evaluated at the m^{th} release. Then we have two cases:

Case 1: If $B \geqslant \bar{F}_{m+1}$, then the optimal policy at the $(m+1)^{st}$ release calls for coordination with probability 1. Thus,

$$\begin{aligned} V_m^n(B,t) &= EV_{m+1}(B + (B+k_c)(e^{\beta T} - 1), t + T_t^{m+1}) \\ &= K_{m+1} + c_{m+1}[B + (B+k_c)E(e^{\beta T_t^{m+1}} - 1)] + V_{m+1}(0, t + T_t^{m+1}). \end{aligned}$$

In this case $V_m^n(B,t)$ is linearly increasing in B at a slope of $c_{m+1}Ee^{\beta T_t^{m+1}}$, which is clearly greater than c_m.

Case 2: If $B < \bar{F}_{m+1}$, then by concavity of $V_m^n(B,t)$, we deduce that $\frac{\partial V_m^n(B,t)}{\partial B} > c_{m+1}Ee^{\beta T_t^{m+1}} > c_m$.

Hence, we conclude the result.

□

REMARK 1 *It is possible to show that $V_m^n(B,t)$ is indeed continuously differentiable in B.*

B. Other Proofs

PROOF OF PROPOSITION 1. We show the result for MOTS project. Suppose $F_m > F_{M-1} = K/c(q-1)$ for some m. Then we must have

$$\begin{aligned} V_m^c(F_{M-1}) - V_m^n(F_{M-1}) &= K + cF_{M-1} + V_m(0) - [K + cqF_{M-1} + c(q-1)k_c + V_{m+1}(0)] \\ &= V_m(0) - K - c(q-1)k_c - V_{m+1}(0) > 0. \end{aligned}$$

But by the minimality of $V_m(0)$, we have

$$V_m(0) = E\left[V_{m+1}\left(k_c(e^{\beta T} - 1)\right)\right] < E\left[V_{m+1}^c\left(k_c(e^{\beta} T - 1)\right)\right] = K + c(q-1)k_c + V_{m+1}(0).$$

Thus, we have a contradiction, which proves the lemma.

□

PROOF OF PROPOSITION 2. In the reverse-time setting, the recursive relation in the dynamic system defined in (4), (7) and (6) becomes

$$W_{l+1}(B) = \min\left\{W_l^n\left(B + (B+k_c)(e^{\beta T} - 1)\right), K + cB\right\}, \tag{17}$$

and

$$\begin{aligned} W_l^n(B) &= \int_0^{t_{l-1}(B)} W_{l-1}^n\left(B+(B+k_c)(e^{\beta t}-1)\right)d\Phi(t) - \int_0^{t_{l-1}(0)} W_{l-1}^n\left(k_c(e^{\beta t}-1)\right)d\Phi(t) \\ &+ \int_{t_{l-1}(B)}^{\infty}[K+c(B+(B+k_c)(e^{\beta t}-1))]d\Phi(t) - \int_{t_{l-1}(0)}^{\infty}[K+ck_c(e^{\beta t}-1))]d\Phi(t), \end{aligned}$$

where

$$t_l(B) = \max\left\{\frac{1}{\beta}ln\left[\frac{f_l+k_c}{B+k_c}\right], 0\right\},$$

and f_l is the optimal threshold value in Theorem 1.

Clearly, if the value functions $W_l(\cdot)$ converge, then the threshold values must also converge. Thus, we try to show that $W_l^n(\cdot)$ converges in l, which, in turn implies the convergence of $W_l(\cdot)$.

Let S denote the collection of all the value functions W_l^n, $l > 2$, in the dynamic system. We first note that every element in S is continuously differentiable. It follows from the discussion in Appendix A that S is *equicontinuous*. That is, for any $\varepsilon > 0$, there is a $\delta > 0$, such that $|w(x) - w(y)| < \delta$ for $y \in (x-\varepsilon, x+\varepsilon)$ and each $w \in S$. Clearly, S is *point-wise bounded*. That is, $w(x) \leqslant K + cx < \infty$ for each $x \in [0,\infty)$ and each $w \in S$. Thus by the Arzela-Ascoli theorem, $\lim_{l\to\infty} W_l^n$ exists (and the convergence is uniform). Thus, we conclude the theorem.

□

Proof of Proposition 3. Follows directly from the discussion in Appendix A.

□

Proof of Proposition 4. Let B_m be the number of faults at the end of the m^{th} release. Also denote t_m as the stopping time to the m^{th} release. Under the optimal threshold policy, the probability of coordinating at release 1 is given by

$$P_1 = P\{B_1 \geqslant F_1\} = P\{k_c(e^{\beta t_1}-1) > F_1\} = \int_{\frac{1}{\beta}ln\frac{F_1+k_c}{k_c}}^{\infty} d\Phi^1(t). \tag{18}$$

If the last coordination is at the i^{th} release, then the random variable $t_m - t_i$, $i < m$, has the distribution $\Phi^i * ... * \Phi^m$. Thus, the conditional probability of coordinating at the m^{th} release is given by

$$\begin{aligned} P_{m|i} &= P\{B_m \geqslant F_m | B_i \geqslant F_i;\ B_k < F_k,\ i < k < m\} \\ &= P\{k_c(e^{\beta(t_m - t_i)}-1) \geqslant F_m | B_i \geqslant F_i;\ B_k < F_k,\ i < k < m\} \\ &= \int_{\frac{1}{\beta}ln\frac{F_m+k_c}{k_c}}^{\infty} d\Phi^i * ... * \Phi^m(t). \end{aligned}$$

Using the conditioning rule, we have

$$\begin{aligned} & P\{B_i \geqslant F_i;\ B_k < F_k,\ i < k < m\} \\ = \ & P\{B_{m-1} < F_{m-1} | B_i \geqslant F_i;\ B_k < F_k,\ i < k < m-1\} P\{B_i \geqslant F_i;\ B_k < F_k,\ i < k < m-1\} \\ = \ & P\{B_{m-1} < F_{m-1} | B_i \geqslant F_i;\ B_k < F_k,\ i < k < m-1\} ... P\{B_{i+1} < F_{i+1} | B_i \geqslant F_i\} P\{B_i \geqslant F_i\} \\ = \ & (1 - P_{m-1|i}) ... (1 - P_{i+1|i}) P_i. \end{aligned}$$

Thus, we can calculate P_m by unconditioning to yield (11).

□

C. Computation of the Release-Based Policy

In this section, we present an algorithm to compute the optimal release-based policy. Let B_m to be the number of faults when development has been continued without coordination for m releases. Denote $C(m,k)$ as the coordination costs at the k^{th} release when the last coordination is at the m^{th} release. Then $C(m,k) = K + bk + cB_{k-m}$. Also define an array $u[m]$ to be the optimal cost from the first coordination after the m^{th} release to the end of the project and an array $t[m]$ to be the index of the next coordination if we coordinate at the m^{th} release. The algorithm proceeds as follows:

1 Initialize: $u[M] = 0$.

2 For $m = M$ and $m \geqslant 0$,

- Set $u[m] = C(m, m+1) + u[m+1]$ and $t[m] = m+1$.
- For $k = m+2$ and $k < M$, if $C(m,k) + u[k] < u[m]$, we set $u[m] = C(m,k) + u[k]$ and $t[m] = k$.
- $m = m + 1$.

At the end of the algorithm, $u[0]$ returns the total coordination cost.

References

Ammann, P.E., S.S. Brilliant, J.C. Knight. 1994. The effect of imperfect error detection on reliability assessment via life testing. *IEEE Trans. on Software Engrg.* **20**(2) 142-148.

Blackburn, J.D., G.D. Scudder, L.N. van Wassenhove. 1996. Improving speed and productivity of software development: a global survey of software developers. *IEEE Trans. Software Engrg.* **22**(12) 875-884.

Brooks, F.P. 1995. *The Mythical Man-Month: Essay On Software Engineering.* Anniversary Edition. Addison-Wesley.

Chiang, I.R., V.S. Mookerjee. 2004. A fault threshold policy to manage software development projects. *Inform. System Res.* **15** (1) 3-19.

Cusamano, C., R. Selby.1995 *Microsoft Secrets.* The Free Press.

Dalal, S.R., A.A. McIntosh. 1994. When to stop testing for large software system with changing code. *IEEE Trans. Software Engrg.* **24**(4) 318-323.

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

Feng, Q., V.S. Mookerjee, S.P. Sethi. 2003. Optimal Policies for Sizing and Timing of Software Maintenance Projects *To appear in Euro. J. Opns. Res.*

Goel, A.L. 1985. Software reliability models: assumptions, limitations, and applicability. *IEEE Trans. Software Engrg.* **11**(12) 1411-1423.

Guinan, P.J., J.G. Cooprider, S. Faraj. 1998. Enabling software development team performance during requirements definition: a behavioral versus technical approach. *Inform. Systems Res.* **9**(2) 101-125.

Ha, A.Y., E.L. Porteus. 1995. Optimal timing of reviews in concurrent design for manufacturability. *Management Sci.* **41**(9) 1431-1447.

Humphrey, W.S. 1992. *Introduction to Software Process Improvement.* CMU/SEI-TR-92-7. Software Engineering Institute, Carnegie Mellon University, Pittsburgh, PA.

Koushik, M., V. Mookerjee. 1995. Modeling coordination in software construction: an analytical approach. *Inform. System Research.* **6**(3) 220-254.

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

Loch, C.H., C. Terwiesch. 1998. Communication and uncertainty in concurrent engineering.

Management Sci. **44**(8) 1032-1048.

Musa, J.D. 1975. A theory of software reliability and its application. *IEEE Trans. Software Engrg.* **1**(9) 312-327.

NASA Goddard Space Flight Center. 1992. *Recommended Approach to Software Development.* Revision 3, SEL-81-305. NASA/GSFC.

Paulk, M., B. Curtis, M.B. Chrissis, C.V. Weber. 1993. *Capacity Maturity Model for Software.* Version 1.1, CMU/SEI-93-TR-024. Software Engineering Institute, Carnegie Mellon University, Pittsburgh, PA.

Rudin, W. 1987. *Real and Complex Analysis.* McGraw-Hill Science.

Singpurwalla, N. 1991. Determining an optimal time interval for testing and debugging software. *IEEE Trans. Software Engrg.* **17**(4) 313-319.

Solheim, J.A., J.H. Rowland. 1993. An empirical study of testing and integration strategies using artificial software system. *IEEE Trans. Software Engrg.* **19**(10) 941-949.

Tamai, T. 1992. Experiment on coordination within software development teams. *Inform. and Software Technology.* **37**(7) 437-442.

Thomke, S., D.E. Bell. 2001. Sequential testing in product development. *Management Sci.* **47** (2) 308-323.

van Genuchten, M. 1991. Why is software late? An empirical study of reasons for delay in software development. *IEEE Trans. Software Engrg.* **17**(6) 528-590.

Wood, A. 1996. Predicting software reliability. *IEEE Computer.* **29**(11) 69-77.

Xia, G., P. Zeephongsekul, S. Kumar. 1993. Optimal software release policy with a learning factor for imperfect debugging. *Microelectronic Reliability.* **33**(1) 81-86.

Yamada, S., S. Osaki. 1985. Software reliability growth modeling: models and applications. *IEEE Trans. Software Engrg.* **11**(12) 1431-1437.