An Economic Model to Estimate Software Rewriting and Replacement Times

Taizan Chan, Siu Leung Chung, and Teck Hua Ho

Abstract—The effort required to service maintenance requests on a software system increases as the software system ages and deteriorates. Thus, it may be economical to replace an aged software system with a freshly written one to contain the escalating cost of maintenance. We develop a normative model of software maintenance and replacement effort that enables us to study the optimal policies for software replacement. Based on both analytical and simulation solutions, we determine the timings of software rewriting and replacement, and hence the schedule of rewriting, as well as the size of the rewriting team as functions of the 1) user environment, 2) effectiveness of rewriting, 3) technology platform, 4) development quality, 5) software familiarity, and 6) maintenance quality of the existing and the new software systems. Among other things, we show that a volatile user environment often leads to a delayed rewriting and an early replacement (i.e., a compressed development schedule). On the other hand, a greater familiarity with either the existing or the new software system allows for a less-compressed development schedule. In addition, we also show that potential savings from rewriting will be higher if the new software system is developed with a superior technology platform, if programmers' familiarity with the new software system is greater, and if the software system is rewritten with a higher initial quality.

Index Terms—Software maintenance, software replacement, economic modeling, optimization, project management.

1 Introduction

Application software maintenance has always been a resource-intensive Information System (IS) activity. In the '80s, it was estimated at US$30 billion worldwide annually and to comprise 50%-80% of the corporate IS expenditures in the United States [25], [33]. While no recent figures are available, it is believed that this trend of high maintenance cost is likely to continue in the foreseeable future as new software systems continue to be developed at a faster rate than old software systems are discarded [3].

An organization incurs a huge software maintenance cost because of

1) a volatile user environment and
2) deteriorating software maintainability.

A volatile user environment generates new user requirements. These requirements are translated into maintenance requests which call for modifications or enhancements to the existing software system. Consequently, the higher the volatility of the user environment, the higher is the maintenance effort. Also, as the software system is being modified or enhanced, its maintainability degrades. With frequent enhancements, the number of inputs and outputs, the number of functions, and inter-module interactions in the software system increase, leading to higher complexity [4]. In addition, these changes are often neither well-integrated into the existing software design nor well-documented. This results in a deterioration of system structure and quality [22]. Consequently, the effort required for each maintenance request increases as the software system ages [15], [20], [25], [33]. The increase in the effort per request exacerbates the total cost of maintenance.

There are two major ways for improving the maintainability of an *existing* software system. One way is through software restructuring [18]. This method improves the structural quality of the software system but at a cost of increasing its size (in SLOC).[1] In addition, this technique may not be viable for some language platforms as restructuring tools are only currently available for Cobol, Fortran, PL/I, and C.

Another way, which is particularly appropriate when the software system concerned is old and the associated documentation is outdated, is to replace the software system by rewriting it. The key idea in software replacement is that over the expected life time of an application, there may exist a time when it is more economical to rewrite the software system than to continue maintaining it so that its overall maintenance cost is reduced [16]. This is because the new software system will often have a higher quality than the aged software system. In addition, the software system may be rewritten with a superior technology platform[2] so that future maintenance can be done more efficiently. Swanson and Beath found that replacement of aged soft-

• *T. Chan is with the National University of Singapore, Lower Kent Ridge Road, Singapore 119260. E-mail: chantz@iscs.nus.sg.*
• *S.L. Chung is with the School of Business & Administration, Open Learning Institute of Hong Kong, Trade Dept. Tower, 700 Nathan Road, Hong Kong. E-mail: slchung@oliv1.oli.hk*
• *T.H. Ho is with the Anderson School at UCLA, University of California at Los Angeles, 110 Westwood Plaza, Box 951481, Los Angeles, CA 90095-1481. Email: teck.ho@anderson.ucla.edu.*

Manuscript received Aug. 3, 1994; revised Feb. 18, 1996.
Recommended for acceptance by H. Muller.
For information on obtaining reprints of this article, please send e-mail to: transse@computer.org, and reference IEEECS Log Number S95592.

1. Empirical research has shown that software size is a significant predictor of the magnitude of maintenance effort required (see, for example, [12] and [20]).
2. In this paper, a technology platform refers to a language platform and its associated development environment.

ware system is indeed a significant IS activity [33], thereby emphasizing the importance of studying the economic ramifications of various replacement policies.

The study of optimal policies for hardware replacement has a long tradition in the operations research literature ([28] and [35] are comprehensive surveys of hardware maintenance/replacement models developed in the '70s and '80s, respectively). These hardware replacement models are not directly applicable to software replacement because new hardware is "bought" whereas new software system is often "developed" in-house. Consequently, hardware replacement occurs instantaneously but software replacement often requires a dedicated team of programmers working over an extended period of time. This fundamental difference necessitates a different model for software replacement.

The use of quantitative models in the study of software maintenance is not new. Lehman and Belady developed an analytical model for explaining the growth pattern (or the evolution dynamics) of a system software as a function of its release version [22]. Their model explains why a system software grows in size in some releases but not in others. They found that a software system grew when maintenance effort was expended on *progressive work*, which involved extending the functionality of the software system. Progressive work or enhancement, however, caused degradation in the software structure which led to lower productivity in any future progressive work. To reduce degradation in productivity, effort must be expended on *anti-progressive work*, which would improve the software structure but would not extend the software functionality.

Arnold and Parker proposed a set of criteria against which an IS department's software maintenance performance could be assessed [2]. For example, one proposed criterion is the percentage of enhancements and perfective requests that should be completed within one person-week or less. The levels of the criteria against which an IS department is benchmarked depend on the specific IS environment and its priorities. While these criteria could be used to determine if an existing software system has become too costly to maintain, they cannot be used to assess whether or not software replacement could reduce the total cost of maintenance.

Sneed proposed a cost-benefit model for evaluating the benefits of software replacement against those of software reengineering and doing nothing at all [29]. Sneed's methodology could be used to help an IS manager to decide if software replacement may be more economical than the other two approaches. The methodology, however, does not prescribe the optimal policy for doing so (i.e., when to replace decision).

Gode, Barua, and Mukhopadhyay provided the first formulation for analyzing the optimal timing to replace a software system [16]. They showed that it would be optimal to replace a software system before the cumulative number of requests reached half the total number of requests expected over the planning horizon.[3] In addition, they proved that the timing of software replacement should be earlier if the initial software size was larger. However, their model assumed, similar to the hardware replacement problem, that rewriting and replacement of software system occurred instantaneously. In addition, they did not explicitly model the user environment.

In [10], we extended Gode, Barua, and Mukhopadhyay's model [16] and allowed the rewriting to take place over an extended period of time and captured the volatility of the user environment. We showed that if the rewriting of the new software system has constant return to scale[4] and there was an unlimited capacity of programming resources, rewriting and replacement should occur instantaneously (i.e., the development schedule should be as small as possible to reduce the period of duplication of maintenance effort on both the existing and the new software systems).[5] Thus, Gode, Barua, and Mukhopadhyay's model was shown to be a special case of the proposed model. In addition, we considered a special case where the number of programmers available for rewriting was fixed and there was a diminishing return to rewriting. The assumption of fixed rewriting team size allowed us to reduce the problem of finding the optimal timings of rewriting and replacement (a two-variable optimization problem) into one that involved only a single variable. Here, we showed, among other things, that the development schedule should be more compressed when the new technology platform was superior to the existing technology platform.

In this paper, we determine both the optimal timings to rewrite and to replace an aged software system with no assumption made on the size of the development team. This allows us to study the optimal level of programming resources to be assigned to rewrite the software system, an important issue which has not been addressed previously. In addition, we explicitly model the software degradation process. Prior research assumes that maintenance only deteriorates the quality of the software system.[6] However, a large proportion of software maintenance jobs involves enhancements [24], and these enhancements not only degrade the quality of the software system but also increase its functional complexity. The distinction between functional complexity and system quality is important because rewriting a software system will improve its *quality* but will not reduce its functional complexity. Furthermore, we analyze the potential savings derivable from software replacement. We found that savings will be higher if the new software system is developed with a superior technology platform, if the staff assigned to maintain the new software system is more familiar with the software system, and if the new software system has a higher initial quality.

The rest of the paper is organized as follows. Section 2 describes the model framework. Section 3 considers the case when the speed of rewriting is approximated by a linear function of the development team size. In this case, we are able to derive close-form analytical solutions for opti-

3. The total number of requests expected could be calculated from the product of rate of arrival of requests and the length of the planning horizon.

4. A production process has constant return to scale when the output increases proportionately with the amount of input. That is, doubling the input doubles the output. This is possible in software development context when the development task is perfectly partitionable [8].

5. This is so because, under constant return to scale, there is no penalty associated with compressing the rewriting schedule.

6. This assumption is common in the hardware maintenance literature.

mal timings of rewriting and replacement. We also provide a formula to quantify the benefit associated with software replacement. Section 4 considers the case of a concave rewriting speed. Here, we use simulation approaches to understand how optimal timings of rewriting and replacement vary with the problem parameters. Section 5 presents an example that illustrates the key results. Section 6 discusses the results from a managerial perspective and suggests future research directions.

2 Model Framework

We consider the cumulative maintenance effort of an independent application over a planning horizon T (measured in months). The planning horizon starts at the time when the application is operational and ends at the time when it is obsolete.[7] It is assumed that the application manager receives and fulfills an incoming stream of maintenance requests from users. According to [31], maintenance requests can be classified into

1) adaptive,
2) perfective, and
3) corrective.

In our study, we focus on adaptive and perfective maintenance requests, which account for 75% or more of the total maintenance effort in most IS environments [24] and refer to them collectively as enhancements.

The software maintainability deteriorates as more and more enhancements are made to the software system. At T_R (measured in months), the application manager begins the development of a new software system whose functionality is equivalent to the existing software system at T_R.[8] This development is scheduled to end at T_N (also measured in months), the time when the existing software system is withdrawn and the new software system is operational. Note that the time interval $(T_N - T_R)$ is the development schedule, during which there is a duplication of maintenance effort. Fig. 1 shows the problem scenario we wish to model here.

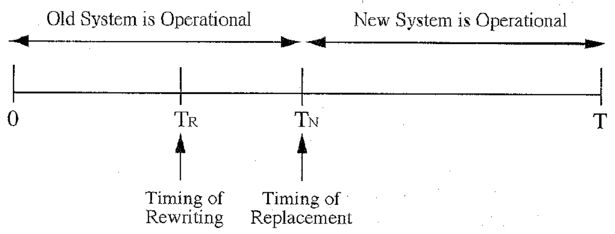


Fig. 1. The problem scenario.

The total effort required for maintaining the application is the sum of two components:

1) the cumulative maintenance effort (of the existing and the new software systems) and
2) the rewriting effort.

In the first two subsections, we shall describe the request arrival process and the software degradation process. In the third subsection, we shall determine the cumulative maintenance effort and the rewriting effort and show that the software maintenance/replacement problem can be formulated as a constrained nonlinear optimization problem.

2.1 Request Arrival Process

We model the business environment explicitly by considering the rate of arrival of maintenance requests. We assume a constant rate of arrival of requests, denoted by λ. This assumption is empirically supported by our field data on the arrival rates of 10 applications over a seven year period [9]. We regressed the number of requests arrived per month against the month period for each of the 10 applications and found that only two applications had slopes significantly different from zero at the 5% level. Furthermore, the slopes were very small (0.0269 and 0.0161, corresponding to a growth rate of approximately one request every three and five years, respectively). A more volatile business environment is represented by a larger λ.

In addition, we assume that each request entails a task complexity denoted by θ_m. θ_m measures the number of function points (FP)[9] that must be added to the software system to fulfill the request [19].

Let $M(t)$ be the total number of arrivals by time t, and $N(t)$ be the total number of FPs added to the software system by time t. Then, $M(t) = \lambda t$, and

$$N(t) = M(t)\theta_m = \lambda t \theta_m. \tag{2.1}$$

2.2 Software Degradation Process

The maintainability of a software system tends to deteriorate as more maintenance requests are serviced. This phenomenon is so pervasive that Lehman and Belady have termed it as the third law of software evolution: "the law of increasing entropy" [22]. We model software maintainability as a function of the functional complexity and the quality of the software system. Software maintainability is assumed to decrease with the functional complexity [19] and increase with the quality of the software system [15], [20]. Functional complexity measures the number of functions that a software system serves. It is measured in terms of the number of function points.[10] Software quality refers to the programmer-oriented (as opposed to user-oriented) features of the software system that relate to maintenance effort. It is measured in terms of characteristics such as its structuredness [15], modifiability [25], and under-

7. We make a distinction between application and software system. The existing software system is obsolete when it is withdrawn and the application is only obsolete at the end of the planning horizon.

8. It is possible that the organization may take the opportunity to discard outdated functions and develop a software system with only a proportion of the original functionality at T_R. Conversely, the organization may incorporate new functions during rewriting so that the new software system is in fact larger than the current one at T_R. We can model either situation by modeling the size of the new software system as a product of the size of the existing software system at T_R and a proportional parameter. The resultant model, while more complex, is qualitatively similar to the current model.

9. Function point is a measure of the functional complexity for an MIS-type system [1]. This raw measure is a weighted function of the number of inputs and outputs, the number of files, and the number of interfaces a software system has. The raw measure is adjusted by a set of 14 technical characteristics of the software system to yield the final measure. For further discussions of function point measurement, see [14], [19], [21].

10. Other functional measures, such as Demarco's Function Bang Matrics [13], are possible. However, Function Point measurement appears the most popular [14], [19].

standability [5]. It is a reflection of the technology platform used [6] and the discipline embedded in the maintenance procedure [25]. A comprehensive treatment on the measurement of these attributes can be found in [26].

Let $F_0(t)$ and $F_1(t)$ be the functional complexities of the existing and the new software systems at time t, respectively. $F_0(t)$ is the sum of the initial functional complexity of the existing software system when it was first installed (θ_0) and the functionality added to the software system (as a result of the enhancements) up to time t. From (2.1), we have

$$F_0(t) = \theta_0 + \lambda t \theta_m. \tag{2.2}$$

That is, the software system climbs up a *functional complexity ladder* with a step size of θ_m. Each maintenance request adds, on average, a complexity of θ_m. This linear assumption is consistent with Lehman and Belady's observation that the size of a software system tends to grow linearly with time [22]. A typical value of an initial application size is 500 FPs (i.e., $\theta_0 = 500$) [19]. If the arrival rate is four requests per month (i.e., $\lambda = 4$) and the software system increases in functional complexity by approximately 12% with respect to its initial size every year [33], then $\theta_m = 1.2$ FP.

At T_R, when rewriting begins, the specification for the new software system is "frozen" based on the functionality of the existing software system at T_R. That is, the initial complexity of the new software system is given by $F_0(T_R)$. Any requests that arrived *after* T_R must be fulfilled to keep the software system current. The expected total number of requests (at time t) that arrived after T_R is given by $M(t) - M(T_R)$. Thus, the expected functionality of the new software system at time $t > T_R$ is given by

$$\begin{aligned} F_1(t) &= F_0(T_R) + (\lambda t - \lambda T_R)\theta_m \\ &= \theta_0 + \lambda t \theta_m. \end{aligned} \tag{2.3}$$

Let $Q_0(t)$ and $Q_1(t)$, both $\in [-1, 0]$, be the code qualities of the existing and the new software systems at time t, respectively. A '0' reflects a perfect software system and a '−1' means a highly unstructured software system. $Q(t)$ is given by the initial quality of the software system minus the deterioration in quality due to changes made to the software system up to time t. Let q_0 be the *initial* quality of the existing software system and q_1 be the initial quality of the new software system when they were first installed, then

$$Q_0(t) = q_0 - \delta_0 \lambda t, \tag{2.4}$$

$$Q_1(t) = q_1 - \delta_1(\lambda t - \lambda T_R). \tag{2.5}$$

Our formulations of Q_0 and Q_1 are similar to those of Woodside [36], who developed a model to account for the growth in disorder in software system observed by Lehman and Belady [22]. In [36], the magnitude of "disorder" of a software system (a positive value in his model) at time t was formulated as the sum of the software disorder at time $t - 1$ and disorder introduced during time t.[11]

The initial software qualities q_0 and q_1 reflect the control for quality during the development of the existing and the new software systems respectively. More control exercised in ensuring a quality software system should yield higher values of q_0 and q_1. The value, q_0, for example, could be determined by first measuring the relevant software attributes with the appropriate metrics described in [26] and then combining these metric values into a single quantity. Coleman et al. [11] and Oman and Hagemeister [27] proposed various means by which values from different software metrics could be combined into a single maintainability or quality index. The index value could then be normalized to fall in the interval [−1, 0] by first subtracting from it the maximum possible index value and then dividing the difference (a negative value) by the maximum value.

Each maintenance request is assumed to cause a constant deterioration in the quality by δ. δ reflects the discipline imposed on the maintenance procedure. A more stringent maintenance procedure has a lower δ [6]. An estimate for δ could be obtained by calculating the difference in the quality index values before and after a specific number of requests and then dividing the difference by the number of requests [20]. For example, assume that the same software system we considered earlier has an initial quality q_0 of −0.01 when it was first installed. If its quality value is assessed as −0.06 a year later, δ_0 could be approximated as 0.001 (0.05 divided by 4 * 12 requests).[12]

The effort required to fulfill each maintenance request is a product of the maintenance productivity of the programmers (in person-hours per function point) and the task complexity of each request (that is, θ_m). The maintenance productivity is determined by the structuredness of the technology platform, the functional complexity, and the quality of the software system. The degree of impact of functional complexity and software quality depends on how familiar the programmers are with the software system [17]. Let p_0 and p_1 be the maintenance productivity on the existing and the new software systems respectively. Then p_0 and p_1 are given by

$$\begin{aligned} p_0(F_0(t), Q_0(t)) &= \alpha_0 + \beta_0 F_0(t) - \gamma_0 Q_0(t) \\ &= (\alpha_0 + \beta_0\theta_0 - \gamma_0 q_0) + (\beta_0\theta_m + \gamma_0\delta_0)\lambda t, \end{aligned} \tag{2.6}$$

$$\begin{aligned} p_1(F_1(t), Q_1(t)) &= \alpha_1 + \beta_1 F_1(t) - \gamma_1 Q_1(t) \\ &= [\alpha_1 + \beta_1\theta_0 - \gamma_1 q_1 - \gamma_1\delta_1\lambda T_R] + (\beta_1\theta_m + \gamma_1\delta_1)\lambda t. \end{aligned} \tag{2.7}$$

The parameter α may be interpreted as the person-hours spent in coding a function point worth of code in some technology platform. It is similar to the *development* productivity of writing a new independent program, unconstrained by any existing software system (i.e., when both $F(t)$ and $Q(t)$ are zeroes). It is a reflection of the productivity of the maintenance staff with respect to the given technology platform. A superior technology platform (such as a Fourth Generation Language (4GL), as compared to Cobol) should

11. More specifically, Woodside's formulation of software disorder also included a term representing a possible reduction in disorder that may be brought about by effort expended on restructuring the code. This term is set to zero if no effort is expended on code restructuring, which is the situation considered in this paper.

12. Kafura and Reddy [20] found that an enhancement request could deteriorate the structure of a software system by between 10% to nearly 50%, depending on which structural metrics is used to measure the deterioration.

yield a lower α [18], [19]. α_0 and α_1 are constants because we assume that the same technology platform is used to maintain a software system throughout the operational life of the software system.

β measures the marginal effort required to deal with a functionally more complex software system. This is the additional effort needed to understand a more complex software system in order to determine how and where a change should be implemented. It depends on the extent to which the maintenance staff is familiar with the functionality of the software system [17]. In [17], it was found that servicing a request on a software system could reduce the effort required to service the next request (of the same task complexity) on the same software system by an average of 33%.

γ is the marginal effort required to deal with a lower quality software system. A more unstructured code will require greater effort in changing the code and evaluating the impact of the change [23]. Similar to β, γ also depends on software familiarity of the maintenance staff. The effort required to deal with unstructured code is less if the staff is more familiar with the software system [17]. Thus, greater familiarity implies a lower γ.[13]

The potential values for p_0 and p_1 could range from 17.9 person-hours/FP to 76 person-hours/FP (based on a data set of 4200 software development, enhancement, and maintenance projects in [19]).[14] As an illustration of the possible values that α, β, and γ may presume, consider the same software system which we have discussed earlier. Given that the software system has an initial functional complexity of 500 FP, the initial productivity for fulfilling an enhancement request on this software system would be approximately 25 person-hours/FP [19]. Assuming that the software system is developed in a third-generation language such as Cobol, α_0 is estimated at 20.3 person-hours/FP [19]. Since the initial functional complexity and initial quality is respectively 500 FP and –0.01, the potential values for β_0 and γ_0 are 0.005 and 200, respectively. That is, the marginal effort required in an enhancement request to deal with the initial functional complexity of 500 FP is 2.5 person-hours and the marginal effort needed to deal with the initial quality of the software system is two person-hours.

Given the productivity values p_0 and p_1, the effort required per maintenance request on the existing software system at time t is given by

$$\theta_m p_0(Q_0(t), F_0(t)), \tag{2.8}$$

and the effort required per maintenance request on the new software system at time t is given by

$$\theta_m p_1(Q_1(t), F_1(t)). \tag{2.9}$$

2.3 The Maintenance Planning Problem

During the planning horizon, three types of effort are expended. First, effort is needed to maintain the existing software system. The total effort required for the maintenance of the existing software system is

$$\int_0^{T_N} \lambda\theta_m p_0\big(F_0(t), Q_0(t)\big)dt. \tag{2.10}$$

It is the integral sum of all the effort required for fulfilling all the maintenance requests from the time the existing software system starts operating to the time at which the existing software system is replaced.

Second, effort is needed to maintain the new software system. Similarly, the total effort required for the maintenance of the new software system is

$$\int_{T_R}^{T} \lambda\theta_m p_1\big(F_1(t), Q_1(t)\big)dt. \tag{2.11}$$

It is the integral sum of all the effort required for fulfilling all the maintenance requests from the time the design specification is 'frozen' up to the end of the planning horizon, T. It should be noted that the maintenance period for the new software system is $T - T_R$ and not $T - T_N$. This is to account for any enhancement requests arrived during the rewriting period.

The final effort to be expended is the rewriting effort for the new software system. Rewriting a software system often requires a dedicated team of staff working over an extended period of time. A general formulation of this effort is

$$L(T_N - T_R), \tag{2.12}$$

where L is the size of the development team that is employed to rewrite the software system (measured in person-hours per month), and $(T_N - T_R)$ is the rewriting period. The rewriting effort is related to the functional complexity of the existing software system at the time of rewriting. From (2.2), the expected functional complexity of the existing software system at time T_R, $F_0(T_R)$ is given by

$$F_0(T_R) = \theta_0 + \lambda T_R \theta_m. \tag{2.13}$$

Similar to [3], we conceptualize software development as a function points *production* process. That is, the rewriting effort is related to the functional complexity of the software system at time T_R by the following equation:

$$F_0(T_R) = S(L)(T_N - T_R), \tag{2.14}$$

where $S(L)$ represents the *speed* of function points delivery by a development team of size L.

The total cumulative effort of maintaining the application over the planning horizon T, $E(T_R, T_N)$, is the sum of the three components, (2.10), (2.11), and (2.12):

$$E(T_R, T_N) = \int_0^{T_N} \lambda\theta_m p_0\big(F_0(t), Q_0(t)\big)dt + \int_{T_R}^{T} \lambda\theta_m p_1\big(F_1(t), Q_1(t)\big)dt + L(T_N - T_R) \tag{2.15}$$

Table 1 summarizes the variables, functions, and parameters used in our model. The optimization problem of interest here is:

$$[G] \quad \min_{T_R, T_N} \quad E(T_R, T_N)$$
$$\text{subject to} \quad F_0(T_R) = S(L)(T_N - T_R),$$
$$\text{and} \quad T_R <= T_N. \tag{2.16}$$

13. It is likely that β and γ are also affected by the technology platform. However, we believe that this effect should be less critical compared to programmers' familiarity with the software system.

14. The productivity figures in [19] are reported in units of person-month. As in [7], we multiply the number of person-months by 152 to obtain the corresponding number of person-hours.

TABLE 1
MODEL VARIABLES, FUNCTIONS, AND PARAMETERS

TABLE 1a
MODEL VARIABLES

Variables	Definition	Dimension
T_R	The time when rewriting starts.	month
T_N	The time when the existing software system is replaced.	month
L	The size of the rewriting team.	$\frac{person-hours}{month}$

TABLE 1b
MODEL FUNCTIONS

Functions	Definition	Dimension
$S(L)$	The speed of the rewriting team.	$\frac{FP}{month}$
$F_0(t)$	Functional complexity of the existing software system at time t.	FP
$F_1(t)$	Functional complexity of the new software system at at time t.	FP
$Q_0(t)$	Code quality of the existing software system at time t.	dimensionless
$Q_1(t)$	Code quality of the new software system at time t.	dimensionless
$p_0(F_0(t), Q_0(t))$	Maintenance productivity on the existing software system at time t.	$\frac{person-hours}{FP}$
$p_1(F_1(t), Q_1(t))$	Maintenance productivity on the new software system at time t.	$\frac{person-hours}{FP}$

TABLE 1c
MODEL PARAMETERS

Category	Parameters	Definition	Dimension
Technology Platform	α_0	Effort required to develop a function point equivalent of code with the existing technology platform; it reflects the structuredness of the existing technology platform.	$\frac{person-hours}{FP}$
	α_1	Effort required to develop a function point equivalent of code with the new technology platform; it reflects the structuredness of the new technology platform.	$\frac{person-hours}{FP}$
Software Familiarity	β_0	Marginal effort required to deal with the functional complexity of the existing software system; it reflects staff familiarity with the existing software system.	$\frac{person-hours}{FP}$
	γ_0	Marginal effort required to deal with the deteriorating code quality of the existing software system; it reflects staff familiarity with the existing software system.	person-hours
	β_1	Marginal effort required to deal with the functional complexity of the new software system; it reflects staff familiarity with the new software system.	$\frac{person-hours}{FP}$
	γ_1	Marginal effort required to deal with the deteriorating code quality of the new software system; it reflects staff familiarity with the new software system.	person-hours
Development Quality	q_0	Code quality of the existing software system when it became operational; it reflects the control imposed on code quality during the development of the existing software system.	dimensionless
	q_1	Code quality of the new software system when it becomes operational; it reflects the control imposed on code quality during the development of the new software system.	dimensionless
Maintenance Quality	δ_0	Deterioration rate of code quality of the existing software system; it reflects the control imposed on code quality during the maintenance of the existing software system.	$\frac{1}{request}$
	δ_1	Deterioration rate of code quality on the new software system; it reflects the control imposed on code quality during the maintenance of the new software system.	$\frac{1}{request}$
User Environment	θ_0	Functional complexity of the existing software system when it became operational; it reflects the complexity of the functional domain of the software system.	FP
	θ_m	Average complexity of each maintenance request.	FP
	λ	Average rate of arrival of requests; it reflects the volatility of the business environment.	$\frac{requests}{month}$

In the following sections, we solve the optimization problem [G] and characterize the optimal software replacement policies. In addition, we perform sensitivity analyses to obtain qualitative insights on the impact of technology platform, software familiarity, user environment, development quality, maintenance quality, and rewriting effectiveness on the optimal policies. In Section 3, we assume that $S(L)$ is a linear function of L. In Section 4, $S(L)$ is assumed to be a concave function of L.

3 LINEAR REWRITING SPEED

In this section, we consider a software development scenario in which $S(L)$ could be expressed as a linear function of L as follows:

TABLE 2
SENSITIVITY ANALYSIS (LINEAR REWRITING SPEED)

x	Tech. Platform		Staff Familiarity				Develop. Quality		Maint. Quality		User Environment			Rewriting Effect.	
	α_0	α_1	β_0	γ_0	β_1	γ_1	q_0	q_1	δ_0	δ_1	θ_0	θ_m	λ	c	m
$\frac{dT_R^*}{dx}$	0	+	0	0	+	D^2	0	−	0	+	+	D^3	D^4	−	+
$\frac{dT_N^*}{dx}$	−	0	−	D^1	0	0	+	0	−	0	−	−	−	+	−
$\frac{d(T_N^*-T_R^*)}{dx}$	−	−	−	D^1	−	D^2	+	+	−	−	−	D^3	−	+	−
$\frac{dL^*}{dx}$	+	+	+	D^1	+	D^2	−	−	+	+	+	D^5	+	−	+

CONDITIONS:

D^1 *If* $\beta_0\theta_m q_0 + \alpha_0\delta_0 + \beta_0\theta_0\delta_0 > \frac{c\delta_0}{m\lambda\theta_m}$, *then* $T_N^*, T_N^* - T_R^*$ *increases and* L^* *decreases with* γ_0, *else* $T_N^*, T_N^* - T_R^*$ *decreases and* L^* *increases with* γ_0.

D^2 *If* $\beta_1\theta_m q_1\lambda + \frac{\delta_1\lambda}{m} + \frac{c}{m\theta_m}\delta_1 > \beta_1\theta_m\delta_1\lambda^2 T + \alpha_1\delta_1\lambda + \beta_1\theta_0\delta_1\lambda$, *then* T_R^* *increases;* $T_N^* - T_R^*$ *decreases and* L^* *increases with* γ_1, *else* T_R^* *decreases;* $T_N^* - T_R^*$ *increases and* L^* *decreases with* γ_1.

D^3 *If* $(m\lambda^2\beta_1\gamma_1\delta_1 T + m\lambda\alpha_1\beta_1 + m\lambda\beta_1^2\theta_0 - m\lambda\beta_1\gamma_1 q_1 - \lambda\beta_1)\theta_m^2 - 2c\beta_1\theta_m + c\gamma_1\delta_1 > 0$, *then* T_R^* *increases;* $T_N^* - T_R^*$ *decreases with* θ_m, *else* T_R^* *decreases;* $T_N^* - T_R^*$ *increases with* θ_m.

D^4 *If* $\lambda < \frac{2c}{\theta_m[m(\alpha_1+\beta_1\theta_0-\gamma_1 q_1)-1]}$, *then* T_R^* *increases with* λ, *else* T_R^* *decreases with* λ.

D^5 *If* $(m\lambda^2\beta_1\gamma_1\delta_1 T + m\lambda\alpha_1\beta_1 + m\lambda\beta_1^2\theta_0 - m\lambda\beta_1\gamma_1 q_1 - \lambda\beta_1)\theta_m^2 - 2c\beta_1\theta_m + c\gamma_1\delta_1 > 0$, *then* L^* *increases with* θ_m.

$$S(L) = c + mL, \tag{3.1}$$

where m and c are constants. The linear rewriting function can be considered as a linear approximation of the more general concave rewriting function (discussed in Section 4). This approximation will be good if the size of the development team can vary only over a small range. Given a concave rewriting speed curve, a higher m is associated with a smaller c and vice versa. A higher m (and therefore smaller c) imply that the speed of rewriting, $S(L)$ would increase more with an increase in L. This is especially the case if a superior technology platform is adopted for rewriting the software system and the development process is well managed so that the inefficiency associated with a larger team size is smaller [8]. This linear rewriting speed case is interesting because many IS departments have very limited freedom in expanding the size of the development team. Empirically, c and m can be derived by regressing the speed of rewriting (or software development) against the size of the development team.[15]

3.1 Optimal Software Replacement Policies

Solving [G] gives us the following proposition.

PROPOSITION 1. *If* $\gamma_1\delta_1 > \beta_1\theta_m$, $E(T_R, T_N)$ *is strictly convex in* T_R *and* T_N. *If*

$$\left(\frac{c}{\beta_0\theta_m + \gamma_0\delta_0} + \frac{\theta_m\lambda + c}{\gamma_1\delta_1 - \beta_1\theta_m}\right)\frac{1}{m\lambda\theta_m} > \frac{\gamma_1\delta_1 T\lambda + \alpha_1 + \beta_1\theta_0 - \gamma_1 q_1}{\gamma_1\delta_1 - \beta_1\theta_m} + \frac{\alpha_0 + \beta_0\theta_0 - \gamma_0 q_0}{\beta_1\theta_m + \gamma_0\delta_0},$$

then the optimal replacement policies T_R^* *and* T_N^* *are given by*

$$T_R^* = \frac{\gamma_1\delta_1 T}{(\gamma_1\delta_1 - \beta_1\theta_m)} + \frac{(\alpha_1 + \beta_1\theta_0 - \gamma_1 q_1)}{(\gamma_1\delta_1 - \beta_1\theta_m)\lambda} - \frac{\theta_m\lambda + c}{m(\gamma_1\delta_1 - \beta_1\theta_m)\lambda^2\theta_m}, \tag{3.2}$$

$$T_N^* = \frac{c}{m(\beta_0\theta_m + \gamma_0\delta_0)\lambda^2\theta_m} - \frac{\alpha_0 + \beta_0\theta_0 - \gamma_0 q_0}{(\beta_0\theta_m + \gamma_0\delta_0)\lambda}. \tag{3.3}$$

Otherwise,

$$T_R^* = T_N^* = \frac{\gamma_1\delta_1\lambda T - \left[(\alpha_0 + \beta_0\theta_0 - \gamma_0 q_0) - (\alpha_1 + \beta_1\theta_0 - \gamma_1 q_1)\right] - \frac{1}{m}}{\left[(\beta_0\theta_m + \gamma_0\delta_0) + (\gamma_1\delta_1 - \beta_1\theta_m)\right]\lambda}.$$

PROOF. See Appendix. □

Equations (3.2) and (3.3) provide closed-form solutions for the optimal timings of rewriting and replacement when

$$\left(\frac{c}{\beta_0\theta_m + \gamma_0\delta_0} + \frac{\theta_m\lambda + c}{\gamma_1\delta_1 - \beta_1\theta_m}\right)\frac{1}{m\lambda\theta_m} > \frac{\gamma_1\delta_1 T\lambda + \alpha_1 + \beta_1\theta_0 - \gamma_1 q_1}{\gamma_1\delta_1 - \beta_1\theta_m} + \frac{\alpha_0 + \beta_0\theta_0 - \gamma_0 q_0}{\beta_1\theta_m + \gamma_0\delta_0}.$$

Using these equations, we analyze the sensitivity of the variables $T_R^*, T_N^*, T_N^* - T_R^*$, and L^* with respect to each of the problem parameters, x. The results are summarized in Table 2. Their proof can be found in the Appendix (under Implications 1-15).

We divide our insights into six categories, according to the factors that impact the software replacement policies.

3.1.1 Technology Platform

The platform of technology affects the productivity of maintenance significantly. Enhancements can be made more quickly if a superior technology platform is adopted [19]. Our results show that the technology platform can also impact the replacement policy. We show that an inferior

15. Regressing $S(L)$ against L using a limited data set in [19 p. 143, Table 3.11] yields an adjusted R^2 of 0.88, suggesting that the linear function is a reasonable approximation.

existing technology platform (i.e., higher α_0) leads to an earlier software replacement or a shorter operational life for the existing software system. The reduction in the operational life will reduce the expensive maintenance effort (due to high α_0) of the existing software system.

Our results also suggest that an existing software system should be rewritten earlier if a superior new technology platform (i.e., lower α_1) is available. This is done to exploit the benefits of the superior technology platform earlier. An inferior technology platform, either existing or new, necessitates a compressed development schedule to reduce the period of maintenance overlap, which is prohibitively expensive when an inferior technology platform is involved. Such a compressed strategy can be achieved by assigning more programmers to the rewriting project.

3.1.2 Software Familiarity

We also show that if the maintenance staff are not familiar with the existing software system (i.e., higher β_0), its operational life should be shorter. The reduction in the operational life will reduce the expensive maintenance effort (due to high β_0) of the existing software system. We also show that an existing software system should be rewritten earlier if the staff are familiar with the new software system (i.e., lower β_1). In general, software familiarity reduces the need for a compressed development schedule. This implies that if maintenance jobs are assigned based on staff availability rather than familiarity, then if an existing software system must be replaced, it must be done with a shorter development schedule.

3.1.3 User Environment

The greater the initial functional complexity of the software system, (i.e., higher θ_0), the later should rewriting begin and the earlier should replacement be done. This is so because θ_0 increases both the maintenance effort of the existing and the new software systems. This makes any duplication in maintenance effort much more costly. However, by delaying the timing of rewriting and bringing forward the timing of replacement, the maintenance overlap would be reduced. The size of the development team however must be increased to handle the more complex rewriting project. Note that this is true only if the rewriting team is significantly more productive than the maintenance team. In practice, this is often the case [19].

The timing of rewriting increases with the rate of maintenance requests (λ) if it is not too high.[16] The timing of replacement decreases, however, with the rate of maintenance requests. Overall, a more volatile business environment implies that a more compressed development schedule is necessary for reducing the maintenance overlap so that less maintenance requests are being serviced for both the existing and the new software systems. Similar insights are obtained for the complexity of the request θ_m. In general, we observe that the firm in a volatile business environment should staff for its rewriting project and strive to complete the project as early as possible.

It is useful to note that the above results are consistent with the situations we observed at a field site [9]. The IS department at this field site is currently planning to replace three of its software systems. These software systems are large-each having more than 2,000 FPs. In addition, the volume of demand for modifications to these software systems is high, accounting for almost 90% of all the requests received by the IS department per month. Despite these factors, the IS department plans to accomplish the rewriting of *all* the three software systems within a short schedule of one and a half year. Apparently, the reason cited for the proposed compressed schedule is to reduce maintenance overlap.

3.1.4 Development Quality

Based on the effort function for the existing software system, we can observe that its initial quality, q_0, affects the maintenance effort per request not only initially when the existing software system becomes operational, but actually throughout the entire operational period of the software system. Our sensitivity analysis shows that, with a better initial quality, the existing software system can be replaced later. This is because with better initial quality, the cumulative maintenance effort for the existing software system would be lower, therefore making it less costly to replace the software later. By delaying the replacement of the existing software system, we can afford a more relaxed rewriting schedule. Similar benefits can be derived if the new software system has better initial quality (i.e., higher q_1). Rewriting can start earlier so that a lower maintenance effort per request for the new software system can be enjoyed earlier. It also means a lower rewriting effort since a less complex software system needs to rewritten and the rewriting schedule is less compressed. This insight emphasizes the importance of the quality of development. A higher quality software system not only reduces maintenance effort out front, it also reduces the effort throughout the entire operational life of the software system.

3.1.5 Maintenance Quality

As our model suggests, the rate at which the quality of the software system degrades determines how quickly the maintenance effort (per request) rises. A higher δ_0 implies that the software quality will degrade quickly and that the maintenance effort for the existing software system will be much higher in the later part of its operational life. We have shown that this increase can be counteracted by replacing the existing software system earlier. Similarly, if δ_1 is higher, the operational life of the new software system should begin later by starting the rewriting later. In addition, with poor maintenance quality (i.e., higher δ), a system manager will need to plan for a short rewriting schedule by assigning as many programmers as possible to rewrite the software system.

3.1.6 Rewriting Effectiveness

Our sensitivity analysis shows that when c is higher, that is, when increasing the team size is less efficient, the rewriting schedule should be extended by rewriting earlier and replacing later so that a smaller team size is required for rewriting the software system. Our analysis also shows that if m is higher, then a more compressed schedule is feasible. The application manager can now start rewriting later while at the same time replacing the existing software system earlier. This implies a shorter operational life for both the existing and the new software systems and thus a lower maintenance effort.

16. The threshold value for λ in condition D^4 is rather high in general.

3.2 Potential Saving from Software Replacement

PROPOSITION 2. *The potential saving in effort from replacing the existing software system is given by*

$$\begin{aligned} sav = {} & \frac{\theta_m}{2(\beta_0\theta_m + \gamma_0\delta_0)}\left[\frac{c}{m\lambda\theta_m} - (\alpha_0 + \beta_0\theta_0 - \gamma_0 q_0)\right]^2 \\ & + \frac{\lambda^2\theta_m}{2(\gamma_1\delta_1 - \beta_1\theta_m)}\left[\gamma_1\delta_1 T + \frac{(\alpha_1 + \beta_1\theta_0 - \gamma_1 q_1)}{\lambda} - \frac{\theta_m\lambda + c}{m\lambda^2\theta_m}\right]^2 \\ & + \left[(\alpha_0 - \alpha_1) + (\beta_0 - \beta_1)\theta_0 - (\gamma_0 q_0 - \gamma_1 q_1)\right]\lambda\theta_m T \\ & + \left[(\beta_0 - \beta_1)\theta_m + (\gamma_0\delta_0 - \gamma_1\delta_1)\right]\frac{\lambda^2\theta_m}{2}T^2 \\ & - \frac{\theta_0}{m}. \end{aligned} \tag{3.4}$$

PROOF. See Appendix. □

Note that the first two terms in the (3.4) are always positive. Thus, whether *sav* is positive or negative depends on the last three terms in Proposition 2. The third and the fourth terms are expressed as the *difference* between the problem parameters of the existing and the new software systems to show the impact of the technology platform, software familiarity, development quality, and maintenance quality on the potential saving derivable from an optimal software replacement policy. The last term relates to the initial functional complexity of the software system and the effectiveness of rewriting. We make the following observations.

- The third term of *sav* indicates that the potential savings from rewriting will be higher if the new software system is developed with a superior technology platform, if programmers' familiarity with the new software system is higher, and if the software system is rewritten with a higher initial quality. This term will only be positive if $(\alpha_1 + \beta_1\theta_0 - \gamma_1 q_1) < (\alpha_0 + \beta_0\theta_0 - \gamma_0 q_0)$. This implies that replacing a software system with a better technology platform and quality (i.e., lower α_1 and higher q_1) alone may not be sufficient to justify replacement. The maintenance staff for the new software system should also be sufficiently familiar with the new software system (i.e., low β_1 and γ_1). Thus, IS managers should avoid assigning inexperienced programmers to maintain the new software system if they want to maximize the return from rewriting. A high β_1 and γ_1 may reduce or even eliminate the intended saving from software replacement.
- The fourth term indicates that to increase potential savings, stringent maintenance procedure must also be implemented for the new software system (to achieve low δ_1). Since the multiplier of this term ($\frac{\lambda^2 T^2}{2}$) is greater than that of the third term (λT), the effect of δ_1 cannot be over-emphasized. Thus, a formal procedure for controlling software quality during maintenance is critical. A plan for software replacement should therefore be accompanied by an appropriate quality control plan for the maintenance of the new software system.
- The last term suggests that rewriting may not be economical for a large software system when the planning horizon is short or the rate of arrival is low.[17] A large θ_0 may render *sav* negative and thus make rewriting uneconomical. Even if the initial software size is not large, a short planning horizon or infrequent maintenance request arrivals may also make rewriting an unattractive option. Note that the rewriting effort includes not only the effort to redevelop the past enhancements, but also the effort to redevelop the initial functions of the software system. This makes rewriting quite expensive.

4 CONCAVE REWRITING SPEED

In many practical situations, the speed of rewriting, $S(L)$, is a concave function of the team size, L. That is, there is a diminishing return to labor input: an increase in input, L, results in less than proportionate increase in output (the number of FPs that could be produced per unit time). This is so in situations where the development task requires constant communications among the team members [7], [8]. The need for communication implies that the potential increase in output due to an additional programmer is discounted by a decrease in output from other members of the team who need to spend otherwise productive effort communicating with the new programmer. For these cases, the number of function points produced per period can be represented as a log-linear form of the development team size. That is,

$$S(L) = KL^{\omega}, \tag{4.1}$$

where K measures the structuredness of the development technology platform used and ω measures the productivity of the rewriting team. The log-linear form is similar to the schedule and effort equations in [7, pp. 75]. A high K implies a highly structured development technology platform. A high ω implies a rewriting team whose members are very experienced and are familiar with the software system under maintenance. It also involves good project management that employs supporting methods, such as configuration management, to support communications among the programmers.[18] In most situations, $\omega < 1$, which implies a diminishing return to labor input [6].

With concave rewriting speed given in (4.1), the rewriting effort is of the form

$$L(T_N - T_R) = \left[\frac{\theta_0 + \theta_m\lambda T_R}{K}\right]^{\frac{1}{\omega}}(T_N - T_R)^{1-\frac{1}{\omega}} \tag{4.2}$$

Unlike the linear case, the rewriting effort in this case is not separable in T_R and T_N. The total cumulative effort for maintaining the application over the planning horizon T, is given as

$$\begin{aligned} E(T_R, T_N) = {} & (\alpha_0 + \beta_0\theta_0 - \gamma_0 q_0)\lambda\theta_m T_N + \frac{(\beta_0\theta_m + \gamma_0\delta_0)\lambda^2\theta_m T_N^2}{2} \\ & + (\alpha_1 + \beta_1\theta_0 - \gamma_1 q_1)\lambda\theta_m(T - T_R) - \gamma_1\delta_1\lambda^2\theta_m T_R(T - T_R) \\ & + \frac{(\beta_1\theta_m + \gamma_1\delta_1)\lambda^2\theta_m(T^2 - T_R^2)}{2} \end{aligned}$$

17. Note that θ_0 also appears in the first three terms. Our extensive numerical experiment in Section 4 shows that the impact of θ_0s on these terms is relatively small compared to its impact on the last term.

18. We thank an anonymous reviewer for highlighting this point to us.

TABLE 3
SENSITIVITY ANALYSIS (CONCAVE REWRITING SPEED)

x	Tech. Platform		Software Familiarity				Develop. Quality		Maint. Quality		User Environment			Rewrit. Effect.	
	α_0	α_1	β_0	γ_0	β_1	γ_1	q_0	q_1	δ_0	δ_1	θ_0	θ_m	λ	K	ω
$\underline{x}$	15	10	$\frac{1}{10^3}$	200	$\frac{0.5}{10^3}$	100	−0.01	−0.005	$\frac{1}{10^3}$	$\frac{0.5}{10^3}$	500	1	3	0.05	0.8
$\overline{x}$	20	12	$\frac{5}{10^3}$	300	$\frac{1}{10^3}$	200	-0.02	-0.01	$\frac{0.125}{10^4}$	$\frac{1}{10^3}$	700	5	5	0.1	0.9
$\frac{dT_R^*}{dx}$	−	+	−	−	+	+	+	−	−	+	−	−	+	+	+
$\frac{dT_N^*}{dx}$	−	+	−	−	+	+	+	−	−	+	+	−	+	+	+
$\frac{d(T_N^*-T_R^*)}{dx}$	−	+	−	−	+	+	+	−	−	+	+	−	−	−	−
$\frac{dL^*}{dx}$	−	+	−	−	+	+	+	−	−	+	+	+	+	−	−

$$+\left[\frac{\theta_0+\theta_m\lambda T_R}{K}\right]^{\frac{1}{\omega}}\left(T_N-T_R\right)^{1-\frac{1}{\omega}}. \qquad (4.3)$$

PROPOSITION 3. *If $\gamma_1\delta_1 > \beta_1\theta_m$, $E(T_R, T_N)$ is strictly convex in T_R and T_N. The optimal replacement policies T_R^* and T_N^* are characterized by the first-order conditions as follows:*

$$(\gamma_1\delta_1-\beta_1\theta_m)\lambda^2\theta_m T_R^* = -\frac{1}{\omega}\left[\frac{\theta_0+\theta_m\lambda T_R^*}{K}\right]^{\frac{1}{\omega}-1}\frac{\theta_m\lambda}{K}\left(T_N^*-T_R^*\right)^{1-\frac{1}{\omega}}$$

$$-\left(\frac{1}{\omega}-1\right)\left[\frac{\theta_0+\theta_m\lambda T_R^*}{K}\right]^{\frac{1}{\omega}}\left(T_N^*-T_R^*\right)^{-\frac{1}{\omega}}$$

$$+\left(\alpha_1+\beta_1\theta_0-\gamma_1 q_1\right)\lambda\theta_m+\gamma_1\delta_1\lambda^2\theta_m T, \qquad (4.4)$$

and

$$(\beta_0\theta_m+\gamma_0\delta_0)\lambda^2\theta_m T_N^* = \left(\frac{1}{\omega}-1\right)\left[\frac{\theta_0+\theta_m\lambda T_R^*}{K\left(T_N^*-T_R^*\right)}\right]^{\frac{1}{\omega}}$$

$$-\left(\alpha_0+\beta_0\theta_0-\gamma_0 q_0\right)\lambda\theta_m. \qquad (4.5)$$

PROOF. See Appendix. □

We can make use of the convexity of $E(T_R, T_N)$ to efficiently compute the optimal replacement policies. In an extensive numerical experiment to study the sensitivity of the optimal replacement policies with respect to the problem parameters, we make use of this property (explained below). The results of the simulation are summarized in Table 3:

There are altogether 15 problem parameters. Like before, we have classified them into six categories. Each parameter is varied at two levels: $\underline{x}$ and $\overline{x}$. The first two rows show the low and high values for each parameter used in the simulation. The values of the parameters are carefully chosen to satisfy two criteria:

1) they yield values that are consistent with the literature and
2) rewriting is always economical.[19]

For example, the values of α_0, which represent the number of hours required to code a function point worth of new code, are based on the data reported in [19]. The values of ω and K are selected to reflect the empirical values reported in [7] and [19]. Proposition 2 has provided us with guidelines in the choice of the parameters for the new technology platform and software familiarity. For instance, α_1 is assigned a value which is smaller than α_0 so that the new maintenance effort is a certain fraction of the existing maintenance effort.

There were altogether 2^{15} cases (since the number of parameters is 15). For each parametric combination, we determine the optimal values of T_R, T_N, $T_N - T_R$, and L according to this 'greedy' algorithm designed based on the convexity of $E(T_R, T_N)$ in Proposition 3.

```
MINEFFORT := a_large_real_number;
for Tr := 0 to T do
  MINFOUND := false;
  Tn := Tr+1
  PREVIOUS := a_large_real_number;
  repeat
    determine the total effort for maintaining the
            existing software system
        from 0 up to Tn, OLD;
    determine the total effort for maintaining the
            new software system
        from Tr up to T, NEW;
    determine the total effort for rewriting the
            software system
        with schedule (Tn - Tr), REWRITE;
    TOTAL := OLD + NEW + REWRITE;
    if TOTAL < PREVIOUS then
      PREVIOUS := TOTAL
    else begin
      MINFOUND := true;
      if PREVIOUS < MINEFFORT then
      begin
        Tr*    := Tr;
        Tn*    := Tn - 1;
        L*     := REWRITE/(Tn* - Tr*);
        MINEFFORT := PREVIOUS;
      end;
    end;
    Tn := Tn + 1;
until MINFOUND or Tn > T;
end;
keep statistics for this combination of parameters;
```

Since $E(T_R, T_N)$ is convex in T_R and T_N, it is also convex in T_N for a given T_R. Algorithmically, it means that, for a fixed T_R, it is not necessary to search through the entire feasible space of T_N (the feasible region of T_N is (T_R, T)). The searching for the optimal T_N is only carried out from

19. The second criterion allows us to examine the sensitivity of the replacement policies with respect to the problem parameters properly. Thus the high and low values for each parameter are chosen such that the potential saving is always greater than zero (see Proposition 2), i.e., replacement is always superior to no replacement.

$T_{R_{fixed}} + 1$ to the optimal T_N. That is, the terminating condition for the repeat-loop is $E(T_{R_{fixed}}, T_N) < E(T_{R_{fixed}}, T_N + 1)$. This knowledge has greatly saved computation time from 10 hours to 0.5 hour.

To determine the sensitivity of a particular decision, such as, T_R^*, to a parameter, say, α_0, we find the average T_R^*s over all parametric combinations (2^{14}) for "low" and "high" values of α_0. We compare the two average T_R^* values. If $T^*_{R(\underline{\alpha_0})} < T^*_{R(\overline{\alpha_0})}$, then we conclude that T_R^* increases with α_0 (indicated as + in our table). If they are equal, then it means T_R^* is not affected by α_0 (indicated as 0). Otherwise, it implies that T_R^* decreases with α_0 (indicated as –).

The results from this sensitivity analysis are quite consistent with those in the case of linear rewriting speed. The differences are due to the less productive rewriting team captured by the concavity in the rewriting speed. This lower productivity tends to lead to expanded rewriting schedule and smaller team size. The following observations are new insights. Since these results are derived by simulation, they should be generalized with caution.

4.1 Technology Platform, Software Familiarity, Development, and Maintenance Quality

Similar to the case of linear rewriting speed, software replacement here occurs earlier if the existing software system requires more effort to maintain due to either an inferior technology platform (higher α_0), lower software familiarity (higher β_0 and γ_0), lower development quality (lower q_0), or lower maintenance quality (higher δ_0). The earlier replacement shortens rewriting schedule and reduces maintenance overlap. Unlike the linear case where the timing of rewriting is not affected by these parameters, the timing of rewriting here becomes earlier when the existing software system requires more effort to maintain. This implies that a smaller software system will be rewritten (indicated by a smaller T_R^*) when the rewriting speed is concave. The smaller software size results in a smaller overall rewriting effort. Consequently, a smaller team size will be needed. However, the system should not be rewritten too early; it should strike a proper balance between the rewriting effort and the maintenance effort for the new software system.

Our simulation results also indicate that rewriting should begin later if the new software system has become more expensive to maintain due to any of the four factors. This implies that a more complex software system must be developed. In the case of linear rewriting speed, the complex software system is developed by assigning more people without having to lengthen the rewriting schedule. In this case, however, it may be uneconomical to assign proportionately more people because of the diminishing return to scale. A more economical solution is therefore to expand both the rewriting schedule and the team size.

4.2 User Environment

Unlike Table 2, Table 3 indicates that T_R^* decreases while T_N^* increases with the initial software size θ_0. In addition, both the rewriting schedule and development team size increase with θ_0. Here, the impact of increasing L^* to cope with a larger θ_0 is less efficient than in the case of linear rewriting speed because of diminishing return to scale. Thus, the schedule should be expanded in order to cope with a more complex software system. It is useful to note that our result on T_N^* is consistent with existing empirical evidence in [19], [32], and [34]. They found that a larger software system gets replaced later (i.e., larger T_N^*). Our result suggests further that the rewriting should begin earlier or the effort involves in the rewriting may be too costly for the replacement to be economical.

The results for θ_m and λ are consistent with those of the linear rewriting speed case: the overlap between the maintenance of the existing and the new software systems is reduced by assigning more people to rewriting when these parameters are larger. Note that we have an earlier rewriting and replacement timing when θ_m increases, and a later rewriting and replacement timing when λ increases. This result appears counterintuitive. We would expect θ_m and λ to change the optimal replacement policies in the same way. This could be attributed to the fact that an increase in λ will speed up the quality degradation process but an increase in θ_m will not.

4.3 Rewriting Effectiveness

An increase in the superiority of the rewriting technology platform or productivity of the rewriting team implies that a more compressed schedule can be achieved more efficiently. This compression in schedule is exploited to increase the potential saving from rewriting. The simulation results show that this is done by starting the rewriting and replacement later.

5 An Illustrative Example

This section illustrates the application of our model to derive the optimal software replacement policies for a Cobol software system. The software system has an initial functional complexity of 500 FPs ($\theta_0 = 500$) and the IS manager is interested in minimizing the total application maintenance effort over a planning horizon of 20 years ($T = 240$ months). When the software system was first installed, it had an initial quality of –0.01 ($q_0 = -0.01$). It deteriorates in a step of 0.001 ($\delta_0 = 0.001$) for each request performed on the software system. The user environment generates an average of four requests per month ($\lambda = 4$); and each of the requests is assumed to entail a functional complexity of 1.2 FP ($\theta_m = 1.2$).

The software system is currently maintained in a technology platform with $\alpha_0 = 20.3$ person-hours/FP. The level of familiarity of the maintenance staff with the software system is specified by $\beta_0 = 0.005$ person-hours/FP and $\gamma_0 = 200$ person-hours. As discussed in Section 2.2, these values are representative of the sample of applications in [19]. Without replacement, the software system will require a cumulative maintenance effort of 142,479 person-hours over the planning horizon.

TABLE 4
EXAMPLE PARAMETER VALUES

Tech. Platform		Staff Familiarity				Develop. Quality		Maint. Quality		User Environment			Rewrite Effect. (Linear)		Rewrite Effect. (Concave)	
α_0	α_1	β_0	γ_0	β_1	γ_1	q_0	q_1	δ_0	δ_1	θ_0	θ_m	λ	c	m	K	ω
20.3	12	0.005	200	0.005	200	−0.01	−0.005	0.001	0.0005	500	1.2	4	31.5	0.083	0.083	0.9

TABLE 5
EXAMPLE OPTIMAL SOFTWARE REPLACEMENT POLICIES

	Without Replacement	Replacement with Linear Rewriting Speed	Replacement with Concave Rewriting Speed
Total Cumulative Effort (person-hours)	142,479	71,264	84,759
T_R^* (month)	-	54	43
T_N^* (month)	-	66	50
L^* (person-hours/month)	-	383	2677
Saving (person-hours)	-	71,215	57,720

The IS manager can potentially reduce this total maintenance effort by rewriting the software system with a superior technology platform such as a 4GL. Jones found that the development productivity in a 4GL software development environment ranges from eight person-hours/FP to 16 person-hours/FP [19]. Since the average value in this range is approximately 12 person-hours/FP, we set α_1 to this value. This productivity value is also used to compute the values of m and K. They are 0.083 FP/person-hour. The IS manager could also use this opportunity to impose strict quality control during the rewriting to ensure a high development quality for the new software system such that q_1 is –0.005. The quality of the new software system will deteriorate at a rate of 0.0005 (δ_1 = 0.0005). The programming staff are assumed to be equally familiar with the new software system so that β_1 and γ_1 have the same values as β_0 and γ_0 (i.e., β_1 = 0.005 and γ_1 = 200). Finally, c takes the value 31.5 FP/month, which we obtained in regressing $S(L)$ against L using a data set in [19] and ω takes a value of 0.9 [7].

The parameter values are summarized in Table 4.

Using this set of parameters, we derive the optimal timing to start rewriting (T_R^*), the optimal timing to replace the old software system (T_N^*), the optimal size of the rewriting team (L^*), and the potential saving derivable from the replacement using Propositions 1-3. These results are summarized in Table 5.

Note that the timings of replacement in both the linear and concave rewriting speed cases (66th and 50th month respectively) are consistent with the empirical observations made in [19], [25], and [33] that a software system is generally replaced after five to seven years.

Fig. 2 depicts the growth pattern of the cumulative maintenance effort of the software system with replacement (for both linear and concave rewriting speed) against that without replacement. This example clearly illustrates that a substantial investment is required for rewriting the software system and the payoff for doing so may take a long time to realize. In the linear case, it takes about 20 months after replacement to "break-even" the additional effort spent in rewriting. In concave case, it takes 65 months to do so. This implies that an IS manager may inappropriately choose to continue maintaining the existing software system over replacing it if he or she does not plan sufficiently further ahead. However, as our model shows, the potential saving in effort derivable from an optimal replacement policy could be highly substantial over the entire planning horizon (50% in the linear case and 40.5% in the concave case).

6 DISCUSSION

As the trend of high maintenance cost is likely to continue, it is imperative that more research effort be directed at understanding and analyzing the means to control this cost. Gode, Barua, and Mukhopadhyay's model [16] represents a first attempt to formalize the tradeoffs between software maintenance and replacement. We have extended their research in [10] by providing a more realistic model that took into account the user environment and the schedule of rewriting.

This paper attempts to contribute to this line of research by explicitly modeling the software degradation process and by considering a more general problem scenario. By modeling the software degradation process, we show that rewriting a software system with a superior technology platform alone may not be able to reduce maintenance cost sufficiently to make it economical. It should be done in conjunction with proper control over the quality of the new software system both during development and during maintenance to harness the full benefits of software replacement.

By investigating a more general problem setting, we are able to derive the manpower staffing requirements for rewriting and the optimal rewriting schedule. These extensions allow us to develop a deeper understanding of the complicated tradeoffs underlying software replacement and maintenance. For instance, previous research shows that an inferior existing technology platform should lead to an earlier replacement. Our results show that this should be accompanied by a later rewrite, if the rewriting team is productive, so that the rewriting schedule is as small as possible.

In summary, our model yields the following managerial implications.

1) *Avoid complete rewrite when the application concerned is large.* We have shown that it may not be economical to rewrite a large application because much of the effort will be expended on redeveloping the initial software functionality.[20] In this case, an IS manager may con-

20. The reader should be cautioned, however, that there are circumstances where complete rewrite is necessary despite the software size. For example, a software system written in an Assembly language may need to be rewritten entirely if it is to be done in conjunction with the installation of a new hardware system that supports a different Assembly language.

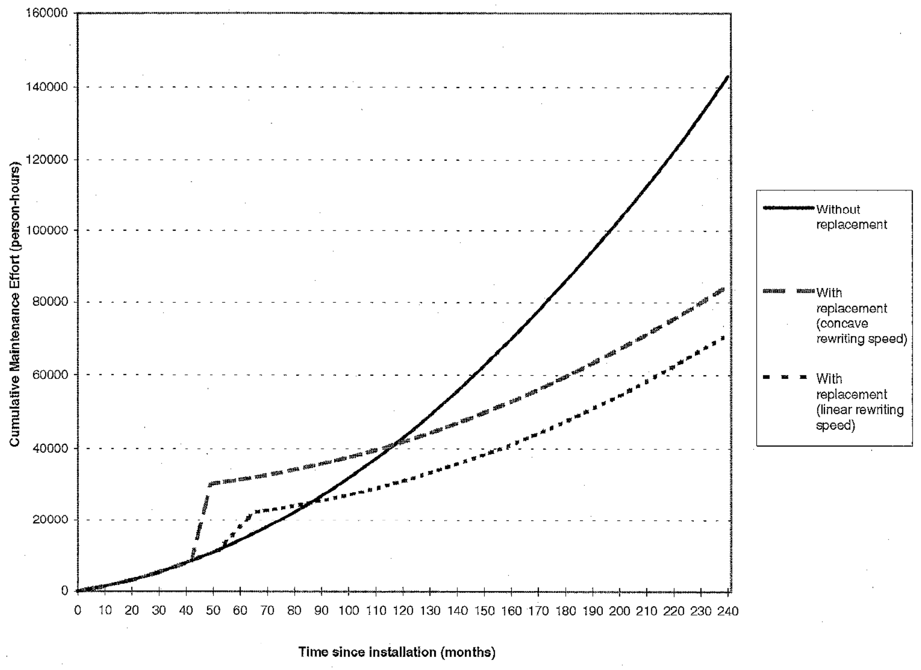


Fig. 2. Cumulative maintenance effort over time of a software system without replacement vs. with replacement under two rewriting scenarios (linear and concave writing speed).

sider other means, such as downsizing (or offloading) the software system so that it is possible to partially rewrite the software system in the future (see future research direction below). Results from case studies suggest that this is a promising avenue for reducing maintenance cost [30].

2) *Organize programming staff by application.* As explained earlier, rewriting a software system with better technology platform and tighter quality control alone are insufficient to obtain the maximal gain from rewriting. It is equally important to ensure that the maintenance staff is familiar with the new software system so that the effort per maintenance request on the new software system is as small as possible. One possible way to accomplish this is to assign the same programmers who rewrite the software system to its maintenance. This application-oriented approach has been adopted by some companies [33].
3) *Compress the rewriting schedule as much as possible.* Our model indicates that the rewriting schedule should be as compressed as possible in order to reduce the duplication of maintenance during rewriting. This should be accomplished without sacrificing the software quality by assigning more productive staff to rewriting (instead of pressuring the staff to complete as soon as possible, for example). Thus the rewriting team should be composed of staff who are familiar with the software system to be rewritten. Another way is to adopt proven technology platform which will ease the effort of rewriting.
4) *Impose strict quality control in maintenance.* We have shown that a large deterioration rate in code quality of the new software system (i.e., high δ_1) may eliminate the intended savings from rewriting because the maintenance effort for the new software system may increase so quickly that its cumulative effort may be greater than expected. Thus, it is important for an application manager to lay down systematic maintenance procedures for the new software system while planning for the replacement of the existing software system.

In summary, when planning for software replacement, an application manager must consider both the short-term issues such as the technology platform to be adopted, the composition of the rewriting team, and the quality of the new software system as well as long-term issues such as plans and procedures for controlling the quality of maintenance over the planning horizon.

Our model has opened up several new research possibilities. The first two are currently being pursued by the authors.

1) *Study the effect of maintenance backlogs.* Our analysis has assumed that all maintenance requests are fulfilled. In many situations, not all the maintenance requests can be satisfied. Some maintenance requests must be shelved for more urgent ones. In addition, some requests are not serviceable due to technological constraints [25]. Our model indicates that deteriorating quality may not be enough to justify software replacement. Maintenance backlogs is another important reason that an aged software system is replaced. When an aged software system is rewritten, not only the quality of the software system is improved, all the backlogs can also be incorporated into the new software system. Our model can be extended for studying the effect of backlogs by breaking the

expected number of maintenance requests, λ, into fulfilled and unfulfilled ones and assigning a penalty cost to each unfulfilled request. By applying similar optimization methods, we can study how backlogs affect the software replacement policies.

2) *Model an application as a collection of submodules to study the economic impact of partial rewriting.* We have suggested that partially rewriting a software system may be a more viable alternative to controlling the escalating maintenance effort of a large application. As many existing applications are complex, economic analysis of this strategy is imperative. We plan to develop a model of maintenance of an application as a collection of subsystems. We will model each incoming request as having certain probabilities of affecting the different subsystems, and modeling each subsystem as having different functional complexity and quality and thus require different effort per maintenance request. The timings of rewriting and replacement for each submodule can then be determined.
3) *Study the trade-off between development effort and initial software quality.* We have modeled the initial quality of a software system explicitly, and analyzed how it may save the maintenance effort of a software system throughout its operational period. However, it is well recognized that greater effort must be expended to develop a higher quality software system. Therefore, a more general model for the economics of software maintenance should explicitly model the tradeoff between the saving in effort due to better quality software system and the development effort that is required to achieve a high level of quality.
4) *Study interactions among applications in a portfolio.* We have assumed a single application in our model. In many real-life situations, however, a system manager has to wrestle with a portfolio of applications [33]. In these situations, the manager must decide how to allocate a common pool of programmers to different applications. The scarcity of programmer resources gives rise to backlogs and frequent switching of programmers from a system to another (which in turn affects their familiarity with the software system). These represent interesting extensions to our model.

APPENDIX

Proof of Proposition 1. The cumulative effort of maintenance can easily shown to be:

$$E(T_R, T_N) = (\alpha_0 + \beta_0\theta_0 - \gamma_0 q_0)\lambda\theta_m T_N + \frac{(\beta_0\theta_m + \gamma_0\delta_0)\lambda^2\theta_m {T_N}^2}{2}$$
$$+ (\alpha_1 + \beta_1\theta_0 - \gamma_1 q_1)\lambda\theta_m(T - T_R) - \gamma_1\delta_1\lambda^2\theta_m T_R(T - T_R)$$
$$+ \frac{(\beta_1\theta_m + \gamma_1\delta_1)\lambda^2\theta_m(T^2 - {T_R}^2)}{2}$$
$$+ \frac{\theta_0 + \theta_m\lambda T_R}{m} - \frac{c}{m}(T_N - T_R).$$

Since if $\gamma_1\delta_1 > \beta_1\theta_m$

$$\frac{\partial^2 E(T_R, T_N)}{\partial T_R^2} = (\gamma_1\delta_1 - \beta_1\theta_m)\lambda^2\theta_m > 0,$$

and

$$\frac{\partial^2 E(T_R, T_N)}{\partial T_N^2} = (\alpha_0\theta_m + \gamma_0\delta_0)\lambda^2\theta_m > 0.$$

Also

$$\frac{\partial^2 E(T_R, T_N)}{\partial T_R \partial T_N} = 0.$$

Thus, $E(T_R, T_N)$ is convex in T_R and T_N, and the first order conditions are necessary and sufficient.

The K-T conditions for optimality are given by

$$(\gamma_1\delta_1 - \beta_1\theta_m)\lambda^2\theta_m T_R^* - (\alpha_1 + \beta_1\theta_0 - \gamma_1 q_1)\lambda\theta_m$$
$$- \gamma_1\delta_1\lambda^2\theta_m T + \frac{\theta_m}{m}\lambda + \frac{c}{m} + \rho = 0;$$
$$(\beta_0\theta_m + \gamma_0\delta_0)\lambda^2\theta_m T_N^* + (\alpha_0 + \beta_0\theta_0 - \gamma_0 q_0)\lambda\theta_m - \frac{c}{m} - \rho = 0;$$
$$\rho(T_N^* - T_R^*) = 0,$$

where ρ is a Lagrangean constant. If

$$\left(\frac{c}{\beta_0\theta_m + \gamma_0\delta_0} + \frac{\theta_m\lambda + c}{\gamma_1\delta_1 - \beta_1\theta_m}\right)\frac{1}{m\lambda\theta_m} >$$
$$\frac{\gamma_1\delta_1 T\lambda + \alpha_1 + \beta_1\theta_0 - \gamma_1 q_1}{\gamma_1\delta_1 - \beta_1\theta_m} + \frac{\alpha_0 + \beta_0\theta_0 - \gamma_0 q_0}{\beta_1\theta_m + \gamma_0\delta_0},$$

$\rho = 0$, $T_N^* > T_R^*$, and

$$T_R^* = \frac{\gamma_1\delta_1 T}{(\gamma_1\delta_1 - \beta_1\theta_m)} + \frac{(\alpha_1 + \beta_1\theta_0 - \gamma_1 q_1)}{(\gamma_1\delta_1 - \beta_1\theta_m)\lambda} - \frac{\theta_m\lambda + c}{m(\gamma_1\delta_1 - \beta_1\theta_m)\lambda^2\theta_m},$$

and

$$T_N^* = \frac{c}{m(\beta_0\theta_m + \gamma_0\delta_0)\lambda^2\theta_m} - \frac{\alpha_0 + \beta_0\theta_0 - \gamma_0 q_0}{(\beta_0\theta_m + \gamma_0\delta_0)\lambda}.$$

If $\rho > 0$ then $T_R^* = T_N^*$ and

$$T_R^* = T_N^*$$
$$= \frac{\gamma_1\delta_1\lambda T - \left[(\alpha_0 + \beta_0\theta_0 - \gamma_0 q_0) - (\alpha_1 + \beta_1\theta_0 - \gamma_1 q_1)\right] - \frac{1}{m}}{\left[(\beta_0\theta_m + \gamma_0\delta_0) + (\gamma_1\delta_1 - \beta_1\theta_m)\right]\lambda}.$$

Table A shows the results of sensitivity analysis of the variables $T_R^*, T_N^*, T_N^* - T_R^*$, and L^* with respect to the various parameters, x.

Conditions:

D^1 If $\beta_0\theta_m q_0 + \alpha_0\delta_0 + \beta_0 q_0\delta_0 > \frac{c\delta_0}{m\lambda\theta_m}$, then $T_N^*, T_N^* - T_R^*$ increases and L^* decreases with γ_0, else $T_N^*, T_N^* - T_R^*$ decreases and L^* increases with γ_0.

D^2 If $\beta_1\theta_m q_1\lambda + \frac{\delta_1\lambda}{m} + \frac{c}{m\theta_m}\delta_1 > \beta_1\theta_m\delta_1\lambda^2 T + \alpha_1\delta_1\lambda + \beta_1\theta_0\delta_1\lambda$, then T_R^* increases; $T_N^* - T_R^*$ decreases and L^* increases with γ_1, else T_R^* decreases; $T_N^* - T_R^*$ increases and L^* decreases with γ_1.

TABLE A

x	Tech. Platform		Staff Familiarity				Develop. Quality		Maint. Quality		User Environment			Rewriting Effect.	
	α_0	α_1	β_0	γ_0	β_1	γ_1	q_0	q_1	δ_0	δ_1	θ_0	θ_m	λ	c	m
$\frac{dT_R^*}{dx}$	0	+	0	0	+	D^2	0	–	0	+	+	D^3	D^4	–	+
$\frac{dT_N^*}{dx}$	–	0	–	D^1	0	0	+	0	–	0	–	–	–	+	–
$\frac{d(T_N^*-T_R^*)}{dx}$	–	–	–	D^1	–	D^2	+	+	–	–	–	D^3	–	+	–
$\frac{dL^*}{dx}$	+	+	+	D^1	+	D^2	–	–	+	+	+	D^5	+	–	+

D^3 If $(m\lambda^2\beta_1\gamma_1\delta_1 T + m\lambda\alpha_1\beta_1 + m\lambda\beta_1^2\theta_0 - m\lambda\beta_1\gamma_1 q_1 - \lambda\beta_1)\theta_m^2$ $-2c\beta_1\theta_m + c\gamma_1\delta_1 > 0$, then T_R^* increases; $T_N^* - T_R^*$ decreases with θ_m, else T_R^* decreases; $T_N^* - T_R^*$ increases with θ_m.

D^4 If $\lambda < \frac{2c}{\theta_m[m(\alpha_1+\beta_1\theta_0-\gamma_1 q_1)-1]}$, then T_R^* increases with λ, else T_R^* decreases with λ.

D^5 If $(m\lambda^2\beta_1\gamma_1\delta_1 T + m\lambda\alpha_1\beta_1 + m\lambda\beta_1^2\theta_0 - m\lambda\beta_1\gamma_1 q_1 - \lambda\beta_1)\theta_m^2$ $-2c\beta_1\theta_m + c\gamma_1\delta_1 > 0$, then L^* increases with θ_m.

Based on the expressions for T_R^*, T_N^*, and L^*, the results presented in the entries of the sensitivity analysis table can be proven as follows:

IMPLICATION 1. $T_N^*, T_N^* - T_R^*$ decreases and L^* increases with α_0 and T_R^* is independent of α_0.

PROOF. Obvious from the expressions for T_R^*, $T_N^*, T_N^* - T_R^*$, and L^*.

IMPLICATION 2. T_R^* increases, $T_N^* - T_R^*$ decreases and L^* increases with α_1 and T_N^* is independent of α_1.

PROOF. Obvious from the expressions for T_R^*, $T_N^*, T_N^* - T_R^*$ and L^*.

IMPLICATION 3. $T_N^*, T_N^* - T_R^*$ decreases and L^* increases with β_0 and T_R^* is independent of β_0.

PROOF. Obvious from the expressions for T_R^*, $T_N^*, T_N^* - T_R^*$, and L^*.

IMPLICATION 4. T_R^* is independent of γ_0. If $\beta_0\theta_m q_0 + \alpha_0\delta_0$ $+\beta_0\theta_0\delta_0 > \frac{c\delta_0}{m\lambda\theta_m}$, then $T_N^*, T_N^* - T_R^*$ increases and L^* decreases with γ_0, else $T_N^*, T_N^* - T_R^*$ decreases and L^* increases with γ_0.

PROOF. It is obvious that T_R^* is independent of γ_0 from the expression of T_R^*. For T_N^*, since

$$\frac{dT_N^*}{d\gamma_0} =$$

$$\frac{1}{\lambda}\left[-\frac{c\delta_0}{m\lambda\theta_m(\beta_0\theta_m + \gamma_0\delta_0)^2} + \frac{q_0(\beta_0\theta_m + \gamma_0\delta_0) + (\alpha_0 + \beta_0\theta_0 - \gamma_0 q_0)\delta_0}{(\beta_0\theta_m + \gamma_0\delta_0)^2}\right]$$

$$= \frac{1}{\lambda(\beta_0\theta_m + \gamma_0\delta_0)^2}\left[\beta_0\theta_m q_0 + \alpha_0\delta_0 + \beta_0\theta_0\delta_0 - \frac{c\delta_0}{m\lambda\theta_m}\right].$$

Therefore if

$$\beta_0\theta_m q_0 + \alpha_0\delta_0 + \beta_0\theta_0\delta_0 > \frac{c\delta_0}{m\lambda\theta_m},$$

$\frac{dT_N^*}{d\gamma_0} > 0$ and T_N^* increases with γ_0. It is obvious that $T_N^* - T_R^*$ also increases but L^* decreases with γ_0.

IMPLICATION 5. T_R^* increases, $T_N^* - T_R^*$ decreases and L^* increases with β_1 and T_N^* is independent of β_1.

PROOF. Obvious from the expressions for T_R^*, T_N^*, $T_N^* - T_R^*$, and L^*.

IMPLICATION 6. T_N^* is independent of γ_1. If $\beta_1\theta_m q_1\lambda + \frac{\delta_1\lambda}{m}$ $+\frac{c\delta_1}{m\theta_m} > \beta_1\theta_m\delta_1\lambda^2 T + \alpha_1\delta_1\lambda + \beta_1\theta_0\delta_1\lambda$, then T_R^* increases; $T_N^* - T_R^*$ decreases and L^* increases with γ_1, else T_R^* decreases; $T_N^* - T_R^*$ increases and L^* decreases with γ_1.

PROOF.

$$\frac{dT_R^*}{d\gamma_1} = \frac{-\beta_1\delta_1\theta_m\lambda^2 T + \beta_1\theta_m q_1\lambda - \alpha_1\delta_1\lambda - \beta_1\theta_0\delta_1\lambda + \frac{\delta_1\lambda}{m} + \frac{c\delta_1}{m\theta_m}}{(\gamma_1\delta_1 - \beta_1\theta_m)^2\lambda^2}.$$

Therefore, if $\beta_1\theta_m q_1\lambda + \frac{\delta_1\lambda}{m} + \frac{c}{m}\delta_1 > \beta_1\theta_m\delta_1\lambda^2 T +$ $\alpha_1\delta_1\lambda + \beta_1\theta_0\delta_1\lambda, \frac{dT_R^*}{d\gamma_1} > 0$ and the results follow.

IMPLICATION 7. T_N^* increases, $T_N^* - T_R^*$ increases and L^* decreases with q_0 and T_R^* is independent of q_0.

PROOF. Obvious from the expressions for T_R^*, T_N^*, $T_N^* - T_R^*$, and L^*.

IMPLICATION 8. T_R^* decreases, $T_N^* - T_R^*$ increases and L^* decreases with q_1 and T_N^* is independent of q_1.

PROOF. Obvious from the expressions for T_R^*, T_N^*, $T_N^* - T_R^*$, and L^*.

IMPLICATION 9. T_N^* decreases, $T_N^* - T_R^*$ decreases and L^* increases with δ_0 and T_R^* is independent of δ_0.

PROOF. Obvious from the expressions for T_R^*, T_N^*, $T_N^* - T_R^*$, and L^*.

IMPLICATION 10. T_R^* increases, $T_N^* - T_R^*$ decreases and L^* ıcreases with δ_1 and T_N^* is independent of δ_1.

PROOF. From the optimality condition,

$$-(\alpha_1 + \beta_1\theta_0 - \gamma_1 q_1)\lambda\theta_m - \gamma_1\delta_1\lambda^2\theta_m T + (\gamma_1\delta_1 - \beta_1\theta_m)\lambda^2\theta_m T_R^* + \frac{\theta_m}{m}\lambda + \frac{c}{m} = 0.$$

Differentiate w.r.t. δ_1 gives

$$-\gamma_1\lambda^2\theta_m T + \gamma_1\lambda^2\theta_m T_R^* + (\gamma_1\delta_1 - \beta_1\theta_m)\lambda^2\theta_m \frac{dT_R^*}{d\delta_1} = 0,$$

or

$$\frac{dT_R^*}{d\delta_1} = \frac{\gamma_1(T - T_R^*)}{(\gamma_1\delta_1 - \beta_1\theta_m)} > 0.$$

Therefore the results follow.

IMPLICATION 11. T_N^* decreases, T_R^* increases, $T_N^* - T_R^*$ decreases and L^* increases with θ_0.

PROOF. It is obvious from the expressions for T_R^*, T_N^*, $T_N^* - T_R^*$. For L^*

$$\frac{dL^*}{d\theta_0} = \frac{\left(1 + \theta_m\lambda\frac{dT_R^*}{d\theta_0}\right)(T_N^* - T_R^*) - (\theta_0 + \theta_m\lambda T_R^*)\frac{d(T_N^* - T_R^*)}{d\theta_0}}{m(T_N^* - T_R^*)^2}.$$

Since $\frac{dT_R^*}{d\theta_0} > 0$ and $\frac{d(T_N^*-T_R^*)}{d\theta_0} < 0$, therefore $\frac{dL^*}{d\theta_0} > 0$ and L^* increases with θ_0.

IMPLICATION 12. T_N^* decreases with θ_m. If $(m\lambda^2\beta_1\gamma_1\delta_1 T + m\lambda\alpha_1\beta_1 + m\lambda\beta_1^2 - m\beta_1\gamma_1 q_1 - \lambda\beta_1)\theta_m^2 - 2c\beta_1\theta_m + c\gamma_1\delta_1 > 0$, then T_R^* increases; $T_N^* - T_R^*$ decreases with θ_m, else T_R^* decreases; $T_N^* - T_R^*$ increases with θ_m.

PROOF. Differentiate the optimality condition for T_N^* w.r.t. θ_m gives

$$(2\beta_0\theta_m + \gamma_0\delta_0)\lambda^2 T_N^* + (\beta_0\theta_m + \gamma_0\delta_0)\lambda^2\theta_m \frac{dT_N^*}{d\theta_m} + (\alpha_0 + \beta_0\theta_0 - \gamma_0 q_0)\lambda = 0,$$

or

$$\frac{dT_N^*}{d\theta_m} = -\frac{(2\beta_0\theta_m + \gamma_0\delta_0)\lambda T_N^* + (\alpha_0 + \beta_0\theta_0 - \gamma_0 q_0)}{(\beta_0\theta_m + \gamma_0\delta_0)\lambda\theta_m}$$

which is negative and so T_N^* decreases with θ_m.

For T_R^*, since

$$\frac{dT_R^*}{d\theta_m} = \frac{1}{(\alpha_1\delta_1 - \beta_1\theta_m)^2}\left[\beta_1\gamma_1\delta_1 T + \frac{\beta_1(\alpha_1 + \beta_1\theta_0 - \gamma_1 q_1)}{\lambda} - \frac{\beta_1}{m\lambda} + \frac{c(\gamma_1\delta_1 - 2\beta_1\theta_m)}{m\lambda^2\theta_m^2}\right]$$

$$= \frac{1}{(\alpha_1\delta_1 - \beta_1\theta_m)^2 m\lambda^2\theta_m^2}\left[(m\lambda^2\beta_1\gamma_1\delta_1 T + m\lambda\alpha_1\beta_1 + m\lambda\beta_1^2\theta_0 - m\lambda\beta_1\gamma_1 q_1 - \lambda\beta_1)\theta_m^2 - 2c\beta_1\theta_m + c\gamma_1\delta_1\right]$$

therefore, $\frac{dT_R^*}{d\theta_m} > 0$ if

$$(m\lambda^2\beta_1\gamma_1\delta_1 T + m\lambda\alpha_1\beta_1 + m\lambda\beta_1^2\theta_0 - m\lambda\beta_1\gamma_1 q_1 - \lambda\beta_1)\theta_m^2 - 2c\beta_1\theta_m + c\gamma_1\delta_1 > 0.$$

The result of $T_N^* - T_R^*$ follows from the results of T_N^* and T_R^*.

For L^*, since

$$\frac{dL^*}{d\theta_m} = \frac{\left(\lambda T_R^* + \theta_m\lambda\frac{dT_R^*}{d\theta_m}\right)(T_N^* - T_R^*) - (\theta_0 + \theta_m\lambda T_R^*)\frac{d(T_N^* - T_R^*)}{d\theta_m}}{m(T_N^* - T_R^*)^2}$$

$$= \frac{\lambda T_R^*(T_N^* - T_R^*) - \theta_0\frac{d(T_N^* - T_R^*)}{d\theta_m} + \theta_m\lambda\left(T_N^*\frac{dT_R^*}{d\theta_m} - T_R^*\frac{dT_N^*}{d\theta_m}\right)}{m(T_N^* - T_R^*)^2}$$

is positive if $\frac{dT_R^*}{d\theta_m} > 0$, $\frac{dT_N^*}{d\theta_m} < 0$, and $\frac{d(T_N^*-T_R^*)}{d\theta_m} < 0$.

IMPLICATION 13. T_N^*, $(T_N^* - T_R^*)$ decreases and L^* increases with λ. If $\lambda < \frac{2c}{\theta_m[m(\alpha_1+\beta_1\theta_0-\gamma_1 q_1)-1]}$, T_R^* increases with λ, else T_R^* decreases with λ.

PROOF. From the optimality condition for T_N^*,

$$2(\beta_0\theta_m + \gamma_0\delta_0)\lambda\theta_m T_N^* + (\beta_0\theta_m + \gamma_0\delta_0)\lambda^2\theta_m\frac{dT_N^*}{d\lambda} + (\alpha_0 + \beta_0\theta_0 - \gamma_0 q_0)\theta_m = 0,$$

or

$$2(\beta_0\theta_m + \gamma_0\delta_0)\lambda\theta_m T_N^* + (\beta_0\theta_m + \gamma_0\delta_0)\lambda^2\theta_m\frac{dT_N^*}{d\lambda} + \frac{c}{m\lambda} - (\beta_0\theta_m + \gamma_0\delta_0)\lambda\theta_m T_N^* = 0,$$

i.e.,

$$\lambda\frac{dT_N^*}{d\lambda} = -T_N^* - \frac{c}{m(\beta_0\theta_m + \gamma_0\delta_0)\lambda^2\theta_m} < 0.$$

Similarly,

$$\lambda\frac{dT_R^*}{d\lambda} = -T_R^* + \frac{c}{m(\gamma_1\delta_1 - \beta_1\theta_m)\lambda^2\theta_m} + \frac{\gamma_1\delta_1 T}{(\gamma_1\delta_1 - \beta_1\theta_m)},$$

and hence $\frac{d(T_N^*-T_R^*)}{d\lambda} < 0$.

For L^*,

$$\frac{dL^*}{d\lambda} = \frac{\theta_m\left(T_R^* + \lambda\frac{dT_R^*}{d\lambda}\right)(T_N^* - T_R^*) - (\theta_0 + \theta_m\lambda T_R^*)\frac{d(T_N^* - T_R^*)}{d\lambda}}{m(T_N^* - T_R^*)^2}$$

$$= \frac{\theta_m T_R^*(T_N^* - T_R^*) - \theta_0\frac{d(T_N^* - T_R^*)}{d\lambda} + \theta_m\lambda\left(T_N^*\frac{dT_R^*}{d\lambda} - T_R^*\frac{dT_N^*}{d\lambda}\right)}{m(T_N^* - T_R^*)^2}.$$

Since $\frac{d(T_N^*-T_R^*)}{d\lambda} < 0$ and

$$\lambda\left[T_N^*\frac{dT_R^*}{d\lambda}-T_R^*\frac{T_N^*}{d\lambda}\right]=$$
$$-\,T_N^*T_R^*+T_N^*\left[\frac{c}{m(\gamma_1\delta_1-\beta_1\theta_m)\lambda^2\theta_m}+\frac{\gamma_1\delta_1T}{(\gamma_1\delta_1-\beta_1\theta_m)}\right]$$
$$+\,T_R^*T_N^*+T_R^*\left[\frac{c}{m(\beta_0\theta_m+\gamma_0\delta_0)\lambda^2\theta_m}\right]$$
$$>0,$$

therefore $\frac{dL^*}{d\lambda}>0$.

For T_R^*, since

$$\frac{dT_R^*}{d\lambda}=\frac{1}{(\gamma_1\delta_1-\beta_1\theta_m)\lambda^2}\left[-(\alpha_1+\beta_1\theta_0-\gamma_1q_1)+\frac{1}{m}+\frac{2c}{m\lambda\theta_m}\right],$$

therefore if $\lambda<\frac{2c}{\theta_m[m(\alpha_1+\beta_1\theta_0-\gamma_1q_1)-1]}$, then T_R^* increases with λ, else T_R^* decreases with λ.

IMPLICATION 14. T_N^*, $T_N^*-T_R^*$ increase with c. T_R^*, L^* decrease with c.

PROOF It is obvious from the expressions for T_R^*, T_N^*, $T_N^*-T_R^*$, and L^*.

IMPLICATION 15. T_N^*, $T_N^*-T_R^*$ decrease with m. T_R^*, L^* increase with m.

PROOF. From the expressions of T_R^* and T_N^*, it is obvious that T_R^* increases and T_N^* decreases with m.

Since T_N^* decreases and T_R^* increases, therefore $T_N^*-T_R^*$ decreases with m.

For L^*,

$$\frac{dL^*}{dm}=$$
$$\frac{\theta_m\lambda\frac{dT_R^*}{dm}m(T_N^*-T_R^*)-(\theta_0+\theta_m\lambda T_R^*)\left[(T_N^*-T_R^*)+m\frac{d(T_N^*-T_R^*)}{dm}\right]}{m^2(T_N^*-T_R^*)^2}$$
$$+\frac{c}{m^2}.$$

From the optimality conditions

$$m\frac{dT_N^*}{dm}=-\frac{c}{(\beta_0\theta_m+\gamma_0\delta_0)m\lambda^2\theta_m};$$
$$m\frac{dT_R^*}{dm}=\frac{\theta_m\lambda+c}{(\gamma_1\delta_1-\beta_1\theta_m)m\lambda^2\theta_m}.$$

Therefore

$$(T_N^*-T_R^*)+m\frac{d(T_N^*-T_R^*)}{dm}=T_N^*+m\frac{dT_N^*}{dm}-\left(T_R^*+m\frac{dT_R^*}{dm}\right)$$
$$=-\frac{\alpha_0+\beta_0\theta_0-\gamma_0q_0}{(\beta_0\theta_m+\gamma_0\delta_0)\lambda}-\frac{\gamma_1\delta_1\lambda T+(\alpha_1+\beta_1\theta_0-\gamma_1q_1)}{(\gamma_1\delta_1-\beta_1\theta_m)\lambda}<0.$$

Since $\frac{dT_R^*}{dm}>0$, therefore $\frac{dL^*}{dm}>0$. □

PROOF OF PROPOSITION 2. Since

$$E(T_R^*,T_N^*)=(\alpha_0+\beta_0\theta_0-\gamma_0q_0)\lambda\theta_mT_N^*+\frac{(\beta_0\theta_m+\gamma_0\delta_0)\lambda^2\theta_mT_N^{*2}}{2}$$
$$+(\alpha_1+\beta_1\theta_0-\gamma_1q_1)\lambda\theta_m(T-T_R^*)-\gamma_1\delta_1\lambda^2\theta_mT_R^*(T-T_R^*)$$
$$+\frac{(\beta_1\theta_m+\gamma_1\delta_1)\lambda^2\theta_m\left(T^2-T_R^{*2}\right)}{2}$$
$$+\frac{\theta_0+\theta_m\lambda T_R^*}{m}-\frac{c}{m}(T_N^*-T_R^*),$$

and after some simplifications, becomes

$$E(T_R^*,T_N^*)=-\frac{(\beta_0\theta_m+\gamma_0\delta_0)\lambda^2\theta_m}{2}T_N^{*2}$$
$$-\frac{(\gamma_1\delta_1-\beta_1\theta_m)\lambda^2\theta_m}{2}(T-T_R^*)^2$$
$$+\frac{\theta_m\lambda+c}{m}T+\frac{\theta_0}{m}.$$

The maintenance effort throughout the whole planning horizon without rewriting is given by

$$C_N(T)=\int_0^T\lambda\theta_mp_0(F_0(t),Q_0(t))dt$$
$$=(\alpha_0+\beta_0\theta_0-\gamma_0q_0)\lambda\theta_mT+\frac{(\beta_0\theta_m+\gamma_0\delta_0)\lambda^2\theta_m}{2}T^2.$$

Therefore, the potential saving for rewriting the software is

$$sav=C_N(T)-E(T_R^*,T_N^*)$$
$$=(\alpha_0+\beta_0\theta_0-\gamma_0q_0)\lambda\theta_mT+\frac{(\beta_0\theta_m+\gamma_0\delta_0)\lambda^2\theta_m}{2}T^2$$
$$+\frac{(\beta_0\theta_m+\gamma_0\delta_0)\lambda^2\theta_m}{2}T_N^{*2}$$
$$+\frac{(\gamma_1\delta_1-\beta_1\theta_m)\lambda^2\theta_m}{2}(T-T_R^*)^2$$
$$-\frac{\theta_m\lambda+c}{m}T-\frac{\theta_0}{m}$$
$$=\frac{\theta_m}{2(\beta_0\theta_m+\gamma_0\delta_0)}\left[\frac{c}{m\lambda\theta_m}-(\alpha_0+\beta_0\theta_0-\gamma_0q_0)\right]^2$$
$$+\frac{\lambda^2\theta_m}{2(\gamma_1\delta_1-\beta_1\theta_m)}\left[\gamma_1\delta_1T+\frac{(\alpha_1+\beta_1\theta_0-\gamma_1q_1)}{\lambda}-\frac{\theta_m\lambda+c}{m\lambda^2\theta_m}\right]^2$$
$$+\left[(\alpha_0-\alpha_1)+(\beta_0-\beta_1)\theta_0-(\gamma_1q_0-\gamma_1q_1)\right]\lambda\theta_mT$$
$$+\left[(\beta_0-\beta_1)\theta_m+(\gamma_0\delta_0-\gamma_1\delta_1)\right]\frac{\lambda^2\theta_m}{2}T^2$$
$$-\frac{\theta_0}{m}.$$

□

PROOF OF PROPOSITION 3. We can treat $E(T_R,T_N)$ as the sum of the three functions:

$$E(T_N) = (\alpha_0 + \beta_0\theta_0 - \gamma_0 q_0)\lambda\theta_m T_N + \frac{(\beta_0\theta_m + \gamma_0\delta_0)\lambda^2\theta_m T_N^2}{2},$$

$$E(T_R) = (\alpha_1 + \beta_1\theta_0 - \gamma_1 q_1)\lambda\theta_m (T - T_R) - \gamma_1\delta_1\lambda^2\theta_m T_R (T - T_R) + \frac{(\beta_1\theta_m + \gamma_1\delta_1)\lambda^2\theta_m (T^2 - T_R^2)}{2},$$

$$R(T_R, T_n) = \left[\frac{\theta_0 + \theta_m\lambda T_R}{K}\right]^{\frac{1}{\omega}} (T_N - T_R)^{1-\frac{1}{\omega}},$$

i.e., $E(T_R, T_N) = E(T_R) + E(T_N) + R(T_R, T_N)$.

The second partial derivatives are

$$\frac{\partial^2 E(T_R)}{\partial T_R^2} = (\gamma_1\delta_1 - \beta_1\theta_m)\lambda^2,$$

$$\frac{\partial^2 E(T_N)}{\partial T_N^2} = (\beta_0\theta_m + \gamma_0\delta_0)\lambda^2,$$

$$\frac{\partial^2 R(T_R, T_N)}{\partial T_R^2} = \frac{1}{\omega}\left(\frac{1}{\omega} - 1\right)\left[\frac{\theta_0 + \theta_m\lambda T_R}{K}\right]^{-2+\frac{1}{\omega}} \left(\frac{\theta_m\lambda}{K}\right)^2 (T_N - T_R)^{1-\frac{1}{\omega}} + 2\cdot\frac{1}{\omega}\left(\frac{1}{\omega} - 1\right)\left[\frac{\theta_0 + \theta_m\lambda T_R}{K}\right]^{-1+\frac{1}{\omega}} \frac{\theta_m\lambda}{K}(T_N - T_R)^{-\frac{1}{\omega}} + \frac{1}{\omega}\left(\frac{1}{\omega} - 1\right)\left[\frac{\theta_0 + \theta_m\lambda T_R}{K}\right]^{\frac{1}{\omega}} (T_N - T_R)^{-1-\frac{1}{\omega}},$$

$$\frac{\partial^2 R(T_R, T_N)}{\partial T_N^2} = \frac{1}{\omega}\left(\frac{1}{\omega} - 1\right)\left[\frac{\theta_0 + \theta_m\lambda T_R}{K}\right]^{\frac{1}{\omega}} (T_N - T_R)^{1-\frac{1}{\omega}},$$

$$\frac{\partial^2 R(T_R, T_N)}{\partial T_R \partial T_N} = -\frac{1}{\omega}\left(\frac{1}{\omega} - 1\right)\left[\frac{\theta_0 + \theta_m\lambda T_R}{K}\right]^{\frac{1}{\omega}} (T_N - T_R)^{-1-\frac{1}{\omega}} - \frac{1}{\omega}\left(\frac{1}{\omega} - 1\right)\left[\frac{\theta_0 + \theta_m\lambda T_R}{K}\right]^{-1-\frac{1}{\omega}} \frac{\theta_m\lambda}{K}(T_N - T_R)^{-\frac{1}{\omega}}$$

Since $\gamma_1\delta_1 - \beta_1\theta_m > 0$, therefore $\frac{\partial^2 E(T_R)}{\partial T_R^2} > 0$ and it is obvious that $\frac{\partial^2 E(T_N)}{\partial T_N^2}$, $\frac{\partial^2 R(T_R, T_N)}{\partial T_R^2}$, $\frac{\partial^2 R(T_R, T_N)}{\partial T_N^2}$ are all positive.

Let $Det(H(E(T_R, T_N)))$ be the determinant of the hessian of $E(T_R, T_N)$. Since

$$Det(H(E(T_R, T_N))) = \left(\frac{\partial^2 E(T_R)}{\partial T_R^2} + \frac{\partial^2 R(T_R, T_N)}{\partial T_R^2}\right)\cdot\left(\frac{\partial^2 E(T_N)}{\partial T_N^2} + \frac{\partial^2 R(T_R, T_N)}{\partial T_N^2}\right) - \left(\frac{\partial^2 R(T_R, T_N)}{\partial T_R \partial T_N}\right)^2$$

$$= \frac{\partial^2 E(T_R)}{\partial T_R^2}\cdot\frac{\partial^2 E(T_N)}{\partial T_N^2} + \frac{\partial^2 E(T_R)}{\partial T_R^2}\cdot\frac{\partial^2 R(T_R, T_N)}{\partial T_N^2} + \frac{\partial^2 R(T_R, T_N)}{\partial T_R^2}\cdot\frac{\partial^2 E(T_N)}{\partial T_N^2} + \frac{\partial^2 R(T_R, T_N)}{\partial T_R^2}\cdot\frac{\partial^2 R(T_R, T_N)}{\partial T_N^2} - \left(\frac{\partial^2 R(T_R, T_N)}{\partial T_R \partial T_N}\right)^2.$$

Since $\frac{\partial^2 E(T_R)}{\partial T_R^2}$, $\frac{\partial^2 E(T_N)}{\partial T_N^2}$, $\frac{\partial^2 R(T_R, T_N)}{\partial T_R^2}$, and $\frac{\partial^2 R(T_R, T_N)}{\partial T_N^2}$ are all positive and

$$\frac{\partial^2 R(T_R, T_N)}{\partial T_R^2}\cdot\frac{\partial^2 R(T_R, T_N)}{\partial T_N^2} - \left(\frac{\partial^2 R(T_R, T_N)}{\partial T_R \partial T_N}\right)^2 = 0.$$

Therefore $Det(H(E(T_R, T_N))) > 0$ which shows that $E(T_R, T_N)$ is convex and the solution to [G] exists and is unique.

Let ρ be a Lagrangean constant. The K-T conditions for [G] are given by

$$(\alpha_1\delta_1 - \beta_1\theta_m)\lambda^2\theta_m T_R^* - (\alpha_1 + \beta_1\theta_0 - \gamma_1 q_1)\lambda\theta_m - \gamma_1\delta_1\lambda^2\theta_m T - \frac{1}{\omega}\left[\frac{\theta_0 + \theta_m\lambda T_R^*}{K}\right]^{\frac{1}{\omega}-1} \frac{\theta_m\lambda}{K}(T_N^* - T_R^*)^{1-\frac{1}{\omega}} - \left(1 - \frac{1}{\omega}\right)\left[\frac{\theta_0 + \theta_m\lambda T_R^*}{K}\right]^{\frac{1}{\omega}} (T_N^* - T_R^*)^{-\frac{1}{\omega}} + \rho = 0;$$

$$(\alpha_0 + \beta_0\theta_0 - \gamma_0 q_0)\lambda\theta_m + (\beta_0\theta_m + \gamma_0\delta_0)\lambda^2\theta_m T_N^* + \left(1 - \frac{1}{\omega}\right)\left[\frac{\theta_0 + \theta_m\lambda T_R^*}{K(T_N^* - T_R^*)}\right]^{\frac{1}{\omega}} - \rho = 0;$$

$$\rho(T_R^* - T_N^*) = 0.$$

Consider the case when $T_R^* = T_N^*$. We can see that for $\omega < 1$

$$\lim_{T_R^* = T_N^*} R(T_R^*, T_N^*) \to \infty$$

Therefore, optimal solution cannot occur when $T_R^* = T_N^*$, thus the optimality conditions correspond to $\rho = 0$ and are given by:

$$(\beta_0\theta_m + \gamma_0\delta_0)\lambda^2\theta_m T_N^* = \left(\frac{1}{\omega} - 1\right)\left[\frac{\theta_0 + \theta_m\lambda T_R^*}{K(T_N^* - T_R^*)}\right]^{\frac{1}{\omega}} - (\alpha_0 + \beta_0\theta_0 - \gamma_0 q_0)\lambda\theta_m,$$

and

$$(\gamma_1\delta_1 - \beta_1\theta_m)\lambda^2\theta_m T_R^* = -\frac{1}{\omega}\left[\frac{\theta_0 + \theta_m\lambda T_R^*}{K}\right]^{\frac{1}{\omega}-1} \frac{\theta_m\lambda}{K}(T_N^* - T_R^*)^{1-\frac{1}{\omega}} - \left(\frac{1}{\omega} - 1\right)\left[\frac{\theta_0 + \theta_m\lambda T_R^*}{K}\right]^{\frac{1}{\omega}} (T_N^* - T_R^*)^{-\frac{1}{\omega}} + (\alpha_1 + \beta_1\theta_0 - \gamma_1 q_1)\lambda\theta_m + \gamma_1\delta_1\lambda^2\theta_m T. \quad \square$$

ACKNOWLEDGMENTS

We would like to thank the editor, Prof. Hausi Muller, and three anonymous reviewers for their helpful comments and constructive suggestions on earlier versions of the paper. Taizan Chan gratefully acknowledges partial financial support from the Reginald H. Jones Research Center of the Wharton School, University of Pennsylvania. This research was partially undertaken while Taizan Chan was a visiting scholar at the Wharton School of the University of Pennsylvania. Teck Hua Ho undertook this research while he was in Singapore in 1994.

REFERENCES

[1] A.J. Albrecht and J.E. Gaffney, "Software Function, Source Lines of Code, and Development Effort Prediction: A Software Science Validation," *IEEE Trans. Software Eng.*, vol. 9, pp. 639-648, Nov. 1983.

[2] R.S. Arnold and D.A. Parker, "The Dimensions of Healthy Maintenance," *Proc. Sixth Int'l Conf. Software Eng.*, pp. 10-27, Sept. 1982.

[3] R.D. Banker, S.M. Datar, and C.F. Kemerer, "A Model to Evaluate Variables Impacting the Productivity of Software Maintenance Projects," *Management Science*, vol. 37, pp. 1-18, Jan. 1991.

[4] R.D. Banker, S.M. Datar, C.F. Kemerer, and D. Zweig, "Software Complexity and Maintenance Costs," *Comm. ACM*, vol. 36, pp. 81-94, Nov. 1993.

[5] G.M. Berns, "Assessing Software Maintainability," *Comm. ACM*, vol. 27, pp. 14-23, Jan. 1984.

[6] G.D. Bergland, "A Guided Tour of Program Design Methodologies," *Computer*, vol. 14, no. 10, pp. 13-37, Oct. 1981.

[7] B.W. Boehm, *Software Engineering Economics*. Englewood Cliffs, N.J.: Prentice Hall, 1981.

[8] F.P. Brooks Jr., *The Mythical Man-Month (20th Anniversary Edition)*. Reading, Mass.: Addison-Wesley, 1995.

[9] T. Chan and T.H. Ho, "An Empirical Analysis of the Arrival and the Service Processes in Software Maintenance," Working Paper, Dept. of Information Systems and Computer Science, Nat'l Univ. of Singapore, 1996.

[10] T. Chan, S.L. Chung, and T.H. Ho, "Timing of Software Replacement," *Proc. 15th Int'l Conf. Information Systems*, pp. 291-307, Dec. 1994.

[11] D. Coleman, D. Ash, B. Lowther, and P. Oman, "Using Metrics to Evaluate Software System Maintainability," *Computer*, vol. 27, no. 8, pp. 44-49, Aug. 1994.

[12] B. Curtis, S.B. Sheppard, P. Milliman, M.A. Borst, and T. Love, "Measuring the Psychological Complexity of Software Maintenance Tasks with the Halstead and McCabe Matrics," *IEEE Trans. Software Eng.*, vol. 5, pp. 96-104, Feb. 1979.

[13] T. DeMarco, *Controlling Software Projects*. New York: Yourdon, 1982.

[14] J.B. Dreger, *Function Point Analysis*. Englewood Cliffs, N.J.: Prentice Hall, 1989.

[15] V.R. Gibson and J.A. Senn, "System Structure and Software Maintenance Performance," *Comm. ACM*, vol. 32, pp. 347-358, Mar. 1989.

[16] D.K. Gode, A. Barua, and T. Mukhopadhyay, "On the Economics of the Software Replacement Problem," *Proc. 11th Int'l Conf. Information Systems*, pp. 159-170, Dec. 1990.

[17] T. Ho and T. Chan, "Estimating Service Time in Software Maintenance: Application-Specific and General-Purpose Human Capital," UCLA Working Paper, 1996.

[18] C. Jones, "Software Enhancement Modelling," *J. Software Maintenance*, vol. 1, pp. 91-100, 1989.

[19] C. Jones, *Applied Software Measurement: Assuring Productivity and Quality*. New York: McGraw-Hill, 1991.

[20] D. Kafura and G. Reddy, "The Use of Software Complexity Metrics in Software Maintenance," *IEEE Trans. Software Eng.*, vol. 13, no. 3, pp. 335-343, Mar. 1987.

[21] C. Kemerer, "An Empirical Validation of Software Cost Estimation Models," *Comm. ACM*, vol. 30, pp. 416-429, May 1987.

[22] M.M. Lehman and L.A. Belady, *Program Evolution: Processes of Software Change*. London: Academic Press, 1985.

[23] J.R. Lyle and K.B. Gallagher, "Using Program Decomposition to Guide Modification," *Proc. Conf. Software Maintenance*, pp. 265-269, Oct. 1988.

[24] B.P. Lientz and E.B. Swanson, *Software Maintenance Management*. Reading, Mass.: Addison-Wesley, 1980.

[25] J. Martin and C. McClure, *Software Maintenance: The Problem and its Solution*. Englewood Cliffs, N.J.: Prentice Hall, 1983.

[26] P. Oman, J. Hagemeister, and D. Ash, "A Definition and Taxonomy for Software Maintainability," SETL Report #91-08-TR, Univ. of Idaho, 1991.

[27] P. Oman and J. Hagemeister, "Metrics for Assessing Software System Maintainability," *Proc. Conf. Software Maintenance 1992*, pp. 337-344, Nov. 1992.

[28] W.P. Pierskalla and J.A. Voelker, "A Survey of Maintenance Models: The Control and Surveillance of Deteriorating Systems," *Naval Research Logistics*, vol. 23, pp. 353-388, Sept. 1976.

[29] H. Sneed, "Planning the Reengineering of Legacy Systems," *IEEE Software*, vol. 12, no. 1, pp. 24-34, Jan. 1995.

[30] H.M. Sneed and E. Nyary, "Downsizing Large Application Programs," *Proc. Conf. Software Maintenance 1993*, pp. 110-119, Sept. 1993.

[31] E.B. Swanson, "The Dimensions of Maintenance," *Proc. Second Int'l Conf. Software Eng.*, pp. 492-497, 1976.

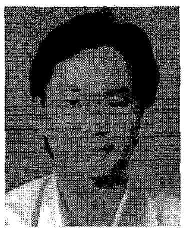
[32] E.B. Swanson, "System Maintenance and Expected Life," Information Systems Working Paper #9-95, John E. Anderson Graduate School of Management, UCLA, Sept. 1995.

[33] E.B. Swanson and C.M. Beath, *Maintaining Information Systems in Organizations*. New Work: John Wiley and Sons, 1989.

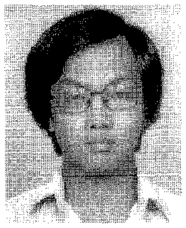
[34] T. Tamai and Y. Torimitsu, "Software Lifetime and Its Evolution Process over Generations," *Proc. Conf. Software Maintenance*, pp. 63-69, Nov. 1992.

[35] C. Valdez-Flores and R. Feldman, "A Survey of Preventive Maintenance Models for Stochastically Deteriorating Single Unit Systems," *Naval Research Logistics*, vol. 36, pp. 419-446, Sept. 1989.

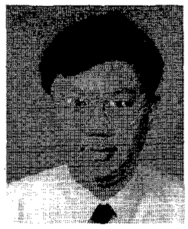
[36] C. Woodside, "A Mathematical Model for the Evolution of Software," *Program Evolution: Processes of Software Change*, M.M. Lehman and L.A. Belady, eds. London: Academic Press, 1985.



Taizan Chan is currently a PhD candidate in the Department of Information Systems and Computer Science at the National University of Singapore. He was a visiting scholar at the Wharton School at the University of Pennsylvania from 1995-1996. His research interests include economics of software maintenance and electronic commerce.



Siu Leung Chung obtained his PhD in computer science from the University of Illinois at Chicago. He was a lecturer in the Department of Information Systems and Computer Science at the National University of Singapore. His research interests include quantity methods in information systems, computer security, legal and policy issues in information systems security, and uses of IT in distance education.



Teck Hua Ho obtained his PhD in operations and information management from the Wharton School at the University of Pennsylvania in 1993. His dissertation received honorable mention for the prestigious George Dantzig Dissertation Award in 1994. He is an assistant professor of operations and technology management at the Anderson Graduate School of Management at the University of California at Los Angeles. His research interests include marketing-operations coordination, new product management, individual and group decision making, and competitive strategy. Dr. Ho has articles and research papers published or to be published in *Communications of the ACM*, *Management Science*, *Journal of Risk and Uncertainty*, *Journal of Economic Dynamics & Control*, and *Journal of Marketing Research*.