ON THE ECONOMICS OF THE-SOFTWARE REPLACEMENT PROBLEM

Dhananjay K. Gode
Carnegie Mellon University

Anitesh Barua
Carnegie Mellon University

Tridas Mukhopadhyay
Carnegie Mellon University

ON THE ECONOMICS OF THE SOFTWARE REPLACEMENT PROBLEM

Dhananjay K. Gode
Anitesh Barua
Tridas Mukhopadhyay
Graduate School of Industrial Administration
Carnegie Mellon University

ABSTRACT

Software maintenance constitutes a significant fraction of the software budget. The cost of maintaining old applications has been escalating and this trend is likely to continue in the foreseeable future. The study of software maintenance strategies has become important to both researchers and practitioners in Information Systems. While there is a rich literature on the technical aspects of software maintenance, research on the economics of maintenance is in its infancy. In particular, the tradeoffs between maintaining and rewriting old software have not been investigated from a theoretical standpoint. In this paper, we present an economic model of the software replacement problem. Based on available empirical evidence, we hypothesize that, with frequent modifications and enhancements, the complexity of software increases rapidly. This deterioration of the code leads to a sharp increase in the maintenance cost. Thus, there may exist a time when it is optimal (in an economic sense) to rewrite the system, which reduces the system complexity and the subsequent maintenance cost. The proposed model allows us to compare the economics of various rewriting strategies and to determine the optimal rewriting point(s). Some interesting results with implications for the systems manager are obtained from the analysis. These include the impacts of system size, structuredness of the underlying technology, and the availability of superior technologies upon the rewriting point(s) and life cycle costs. A numerical example is provided to demonstrate the applicability of the model.

1. INTRODUCTION

The study of software maintenance and replacement strategies is an important topic for both researchers and practitioners in Information Systems. The cost of maintaining software has been escalating rapidly (Kemerer 1987; Banker, Datar and Zweig 1989) and the trend is likely to continue in the foreseeable future. Empirical evidence (Lientz, Swanson and Tompkins 1978; Swanson and Beath 1989) indicates that a major fraction (between 50 and 80 percent) of the software budget is allocated to the maintenance of existing software. With an estimated annual expenditure of about $100 billion on software in the United States alone (Boehm 1987), it is imperative that additional effort is directed to study the underlying economics of software maintenance and to analyze ways of reducing this staggering cost.

According to the classification scheme proposed by Swanson (1976), maintenance activities are termed as adaptive, perfective and corrective. It has been estimated that as much as 75 percent of maintenance work is devoted to adaptive and perfective maintenance (Lientz, Swanson and Tompkins 1978). Several factors contribute to the magnitude of this maintenance problem. Rapidly changing business environments and user needs necessitate continuous modifications and enhancements of the application software. With modification/enhancements, the number of control flows and inter-module interactions in the system increase over time, leading to higher system complexity. Often, the changes made to the system are not integrated well in the overall design, leading to a loss of clarity. Sometimes the changes made are not adequately documented and naming conventions are not strictly followed. Additionally, small changes in the task environment may require significant modifications in the system structure. As a result, the initial correspondence between the system environment and structure is adversely affected due to maintenance. System modifications may also have ripple effects which make it difficult to determine all possible impacts on the existing code. Since the system testing after a "small" change may not be as rigorous as it is at the time of initial development, the maintenance performed on a system may introduce errors in the system. In summary, there is a degradation in the system structure because of continuous modification/enhancements. The maintainability of software decreases rapidly due to this deterioration of system structure (Curtis et al. 1979; Kafura and Reddy 1987), leading to a sharp increase in maintenance cost.

Based on the notion that structured code is cheaper to modify and enhance, structured programming is often suggested as a means of reducing software maintenance cost. While there is some evidence that structured programs are easier to maintain, it is estimated that approximately 75 percent of the existing software base is not structured (Schneidewind 1987). Moreover, as a result of frequent modifications and enhancements, it is very difficult and excessively costly to maintain the initial structure. As a result, regardless of the initial structure, programs are expected to become more and more com-2.plex due to changes in user needs and business environments (Barua and Mukhopadhyay 1989).

We present an economic model for analyzing the trade-offs between maintaining and rewriting old software. The thesis of our paper is that there may exist a time when it is optimal (in an economic sense) to rewrite the system, which may result in a reduction of complexity and subsequent maintenance cost. The questions addressed by our research include the following: Can maintenance cost be reduced by rewriting the software? Is it optimal to rewrite more than once within a specified planning horizon? Under what conditions is it better to switch to a "superior" technology for the system rewrite?[1]

There is a rich body of literature (for a summary, see Kemerer 1987; Banker, Datar and Zweig 1989) on the technical aspects of software maintenance (such as software metrics or determinants of complexity). However, relatively few studies have focused on the economics of software maintenance. While cost estimation for software maintenance has received some attention (for example, see Boehm 1981), the economic ramifications of various maintenance and replacement strategies have not been investigated. A recent survey by Swanson and Beath (1989) indicates that replacement of old systems is a significant activity, thereby emphasizing the need to study policies that can guide the replacement process. This paper is an attempt toward the development of a formal theory of software replacement with a view to generating empirically testable propositions.

Barua and Mukhopadhyay (1989) developed a simple deterministic model of software maintenance. Their model is restrictive due to the assumption of specific functional forms for various cost components and a known constant arrival rate of maintenance requests. Additionally, the impact of system size upon the rewriting time(s) was not considered in the model. Another limitation is the assumption of homogeneous maintenance requirements. In this paper, we generalize and extend the basic model proposed by Barua and Mukhopadhyay. Starting with a convex maintenance cost function[2] and any arrival pattern, we derive the optimal rewriting schedule over the planning horizon. The effect of initial system size on rewriting is addressed; we find that the operational life of the original system decreases with an increase in the initial system size. We also investigate the impact of the structuredness of the technology upon the optimal rewriting time(s). For multiple rewrites of the software, we show that each rewriting point occurs earlier for more structured technologies. Next we study the problem of switching to new ("superior") technologies, and compare the rewriting times for old and new technologies.

The paper is organized as follows: The characteristics of the problem setting to be analyzed are described in Section 2. In Section 3, we develop the analytical model of software replacement, and derive rewriting strategies under various conditions. An application of the model is illustrated through a numerical example in 3.4. We summarize our plans for future research in Section 4. Section 5 contains concluding comments.

2. CHARACTERISTICS OF THE PROBLEM SETTING

In this section, we describe the elements of our software replacement model.

2.1 Technology Characterization

In our model, the term technology is used to denote the programming languages and environment used to develop and maintain software. According to Bergland (1981), although the technology affects the program structure, development process and development support tools, its impact on program structure is the most important determinant of the life-cycle costs. Thus, in our model, technology is primarily characterized by its "structuredness."

To analyze the role of structuredness in maintenance effort, it is important to understand the maintenance process itself. According to Yao and Collofello (1980), maintenance activities may be divided into four phases:

1. Understanding the existing code: The effort required is dependent on complexity of the code, documentation and self-descriptiveness.

2. Implementing the desired change: The extensibility of the program is the critical attribute in this phase.

3. Analyzing the impact of the change on other parts of the program: The stability of the system, i.e., resistance to the amplification of changes through ripple effects, is important in this phase. Programs with low coupling have higher stability.

4. Testing the system for reliability.

Thus, there is substantial support in favor of the argument that structured code is easier to maintain. However, the degree to which code can be structured depends critically on the inherent structuredness of the technology. The impact of the structuredness of the technology can be studied at the code, module and system levels (Bergland 1981).

Code-level concepts involve abstraction, communication, clarity and control-flow. Abstraction refers to the ability of a technology to make the low-level hardware and system software details transparent to the users. Thus, Fortran has a higher level of abstraction than an assembly language. Programs written using a higher level language require fewer changes for functional modifications. Moreover, the changes are easier to implement. If a technology permits easier documentation, and if programs developed using the technology are self-descriptive (possibly because of syntax and naming conventions), then it is said to have higher communication abilities. Clarity and control flow aspects of structuredness involve various program control structures available and the ease with which they can be used and understood. For example, programs developed in Pascal do not need to use the "GoTo" statements as compared to several other languages and are thus easier to understand.

Module-level constructs deal with issues of cohesion, coupling, complexity, correspondence with the task environment and correctness. More structured technologies permit earlier modularization by allowing independent procedures, local variables for the procedures and parameterization. In some programming environments, it is possible to test each module for correctness separately. This makes the task of testing large programs much simpler with structured technologies.

The system-level concepts include consistency, connectivity, optimization and packaging. If the programs developed using a particular technology have a consistent structure even if they have been developed by different programmers, then it is easier to coordinate the efforts of the team and maintain the program structure in spite of personnel turnover. If the technology encourages functional partitioning, then the system has lower connectivity and hence the ripple effects are expected to be less.

Thus, structuredness covers various important aspects of technology and has a significant impact on software costs. For two technologies A and B, we consider A as *superior* to B if a system developed with A is more structured than the one developed with B.[3] A superior technology results in more structured systems and is expected to reduce the maintenance cost. However, no *a priori* assumptions are made regarding the initial acquisition and learning costs associated with switching to superior technologies. Such costs may make it infeasible to rewrite the system with a new technology.

2.2 Software Costs

The total software cost incurred during a given time period has two components:

(i) Cost of maintaining the software.

(ii) Rewriting cost (if the system is rewritten during the period of interest).

As discussed earlier, the system structure decreases due to frequent modifications/enhancements over time (Gurbaxani and Mendelson 1987; Lehman 1982). The system structure depends on the structuredness of the technology used in developing the system; for a given number of enhancements, the system structure is expected to deteriorate less for a system written with a more structured technology (Bergland 1981).

The deterioration of the code makes it progressively difficult and costly to maintain the system (Curtis et al. 1979; Kafura and Reddy 1987). Thus, the cumulative cost of maintaining the software may be considered as a function of the number of modifications/enhancements and the structuredness of the technology. This cost is represented by $C(m,s)$, where m is the number of units of maintenance performed on the system,[4] and s represents the structuredness of the technology.

The marginal maintenance cost is positive and increasing, because the system complexity gets compounded rapidly due to patchup maintenance; i.e.,

$$\frac{\partial C(m,s)}{\partial m} > 0, \ \frac{\partial^2 C(m,s)}{\partial m^2} > 0$$

(the maintenance cost is convex in m)

As mentioned above, increased structuredness of the technology used should lead to less deterioration in system structure and hence reduce the marginal maintenance cost and convexity. These effects are stated as

$$\frac{\partial}{\partial s}\left(\frac{\partial C(m,s)}{\partial m}\right) < 0 \text{ (decreasing marginal maintenance cost)}$$
$$\frac{\partial}{\partial s}\left(\frac{\partial^2 C(m,s)}{\partial m^2}\right) < 0 \text{ (decreasing convexity)}$$

In Figure 1, the cumulative maintenance cost is shown as a function of the amount of modifications/enhancements and the structuredness of the technology in Figure 1.

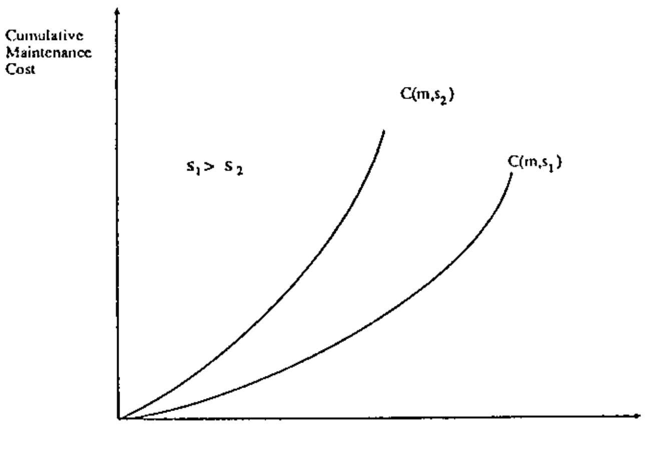

Figure 1. Cumulative Maintenance Cost C as a Function of Maintenance Units (*m*) and Structuredness of Technology (*s*)

The rewriting cost depends on the current system size and the technology used to rewrite the system. It is expected to decrease with structuredness of the rewriting technology (Bergland 1981) and increase with the current size of the system. However, the current size is itself a function of the initial size, ϕ, and the number of the modifications/enhancements, *m*. Therefore, the rewriting cost is denoted by $R(m,s,\phi)$, with

$$\frac{\partial R(m,s,\phi)}{\partial s} < 0, \quad \frac{\partial R(m,s,\phi)}{\partial \phi} > 0, \quad \frac{\partial R(m,s,\Phi)}{\partial m} > 0$$

We note that maintenance, *m*, affects $C(m,s)$ and $R(m,s,\phi)$ quite differently. The increase in maintenance cost with *m* is much greater than the corresponding increase in rewriting cost. This is explained as follows: Since modification/enhancements are performed within the constraints of the existing code, the effort required is critically dependent on the system structure at the time of change. As discussed earlier, there is a deterioration in structure with each modification/enhancement. Thus the cumulative effect leads to a sharp increase in maintenance cost over time.

On the other hand, rewriting takes place in a relatively unconstrained environment, where the designer can consider the overall functional requirements and integrate all the features in a structured code.[5] The rewriting effort is primarily determined by task requirements. The rewritten system must incorporate all the features that have been added as a result of enhancements since the beginning of the planning horizon. Since rewriting does not involve modifications of the existing code, the increase in rewriting cost is dependent on the net increase in requirements, and not the total number of maintenance jobs performed on the system. Thus the rewriting cost does not increase as sharply with maintenance *m* as compared to the maintenance cost. This is shown in Figure 2 and stated as

$$\frac{\partial C(m,s)}{\partial m} \gg \frac{\partial R(m,s,\phi)}{\partial m}$$

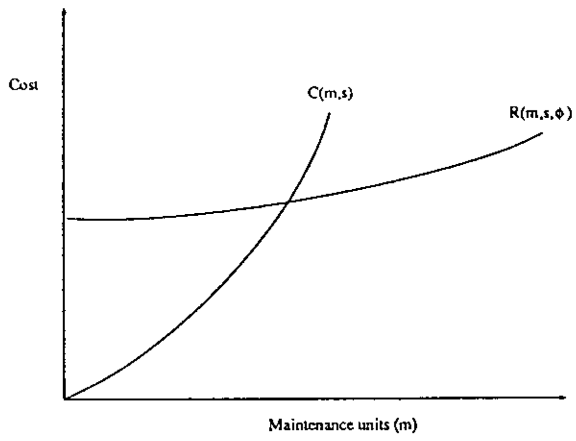

Figure 2. Differential Impacts of Maintenance Units Upon Maintenance Cost C(m,s) and Rewriting Cost R(m,s,ϕ)

2.3 Planning Horizon

The uncertainty and the rapid changes in the technological and business environments make it infeasible to develop a technological plan for an infinite time horizon. With each new generation of information technology, there have been dramatic improvements in the cost-performance ratio. These improvements have changed the basic design of information systems in many instances. Thus, it becomes impossible for a system manager to plan for an indefinite period.

A second motivation for a finite horizon is related to the dynamic nature of the required system functionality. Intense competition in business environments places rapidly changing demands on information systems, thereby making it difficult to formulate very long term system plans. Yet another potential reason for a finite planning horizon is the expected duration of the system manager's involvement in the decision making process. In short, there is ample justification for doing the cost analysis for a finite period of time, say, $[0,T]$.

While the system manager is likely to plan for a finite time horizon, the software cost is a function of the amount of maintenance performed on the system. This necessitates a mapping between time and cumulative maintenance. The mapping is dependent on whether the arrival of modification/enhancement requests is deterministic or random. If the arrival is deterministic, say, λ per unit time, then the time horizon T can be mapped into a maintenance horizon $M = \lambda T$, where M is measured in maintenance units such as function points. When the requests arrive randomly, say, according to a distribution with mean, λ, we use the same mapping, with the new interpretation that M is the expected number of arrivals in time T. In this paper, we derive the optimal rewriting point, m^*, in terms of M, with the understanding that the optimal rewriting time, τ^* can be obtained from the inverse mapping (as demonstrated in the numerical example in 3.4).

3. A MODEL OF SOFTWARE REPLACEMENT

In this section, we develop an economic model for analyzing the tradeoff between maintaining and rewriting existing software. In particular, we consider two rewriting strategies involving new (superior) and old (existing) technologies, and study their economic consequences. We start with an analysis of rewriting the software using the same technology with which it was originally developed.

3.1 Optimal Rewriting Point

To derive the optimal rewriting point (if any), we write the total cost as a function of the rewriting point. If the system is rewritten at m, then the total cost, denoted by $F(m)$, consists of three components: the maintenance cost from 0 to m, the rewriting cost at m, and the maintenance cost from m to M. The cost components and the total cost function are shown in Figures 3 and 4 respectively. Note that the choice of a rewriting point involves a tradeoff between the costs before and after the rewrite. Rewriting the system earlier decreases the rewriting and the maintenance costs in the first period. However, since in that case the system must be maintained for a longer duration in the second period, the corresponding maintenance cost is higher.

The total cost is given by

$$F(m) = C(m,s) + R(m,s,\phi) + C(M-m,s)$$

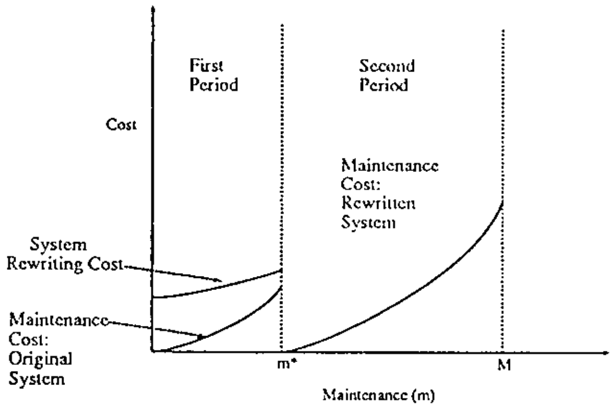

Figure 3. Cost Functions for Original and Rewritten Systems

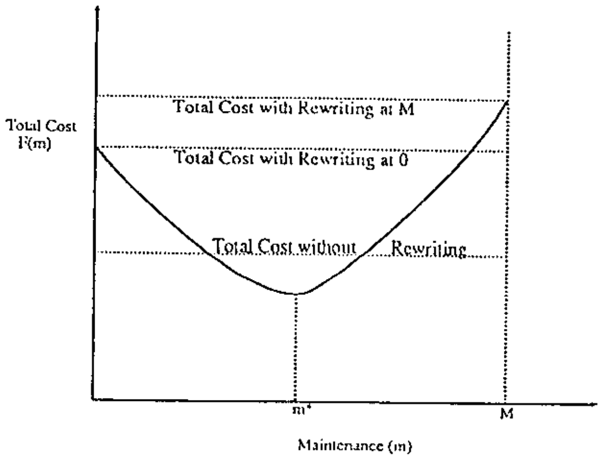

Figure 4. Total Maintenance Cost F(m) as a Function of Rewriting Point (m*)

Note that in this model, the initial system size, ϕ, the technology, s, and the planning horizon, M, are assumed to be exogenously given. Endogenous choice of M and s are considered as an extension of the current model. In Figure 4 above, if the system is rewritten at $m=0$, then $F(0)$ is given by $R(0,s,\phi) + C(M,s)$. The total cost for a rewrite at M is $F(M) = C(M,s) + R(M,s,\phi)$. Clearly $m=0$ or M are not candidates for a rewriting point, since the total cost without rewriting $(C(M,s))$ is less than the cost in either of the two cases. The cost difference at $m=0$ and M is small (since R is relatively insensitive to m), and is equal to $R(M,s,\phi) - R(0,s,\phi)$. We also note that there may not exist an optimal time to rewrite the software for a given horizon $[0,M]$. This is explained as follows: Let m^* be the value of m for which $F(m^*)$ is a minimum. This can be obtained from the first order condition. However, this marginal analysis does not consider the "fixed" component of the rewriting cost, $R(0,s,\phi)$. It is possible (for relatively short horizons) that with this fixed component, the total cost with a rewrite at m^* will exceed the cost without rewriting. If $F(m^*) < C(M,s)$, then an optimal rewriting point exists. Next we derive a proposition relating the optimal rewriting point (when it exists) and the planning horizon M.

Proposition 1: *If it is optimal to rewrite the system after m^* modifications/enhancements, then m^* is less than $M/2$.*

Proof: Let m^* be the optimal rewriting point. Differentiating $F(m)$ with respect to m we get the following first order condition:

$$\text{Let } X(m,s,\phi) = \frac{\partial C(m,s)}{\partial m} + \frac{\partial R(m,s,\phi)}{\partial m} - \frac{\partial C(M-m,s)}{\partial m} \qquad \text{(1a)}$$

At $m=m^*$, $X(m^*,s,\phi) = 0$

$$\text{or } \frac{\partial C(M-m,s)}{\partial m}\Big|_{m=m^*} - \frac{\partial C(m,s)}{\partial m}\Big|_{m=m^*} = \frac{\partial R(m,s,\phi)}{\partial m}\Big|_{m=m^*} \qquad \text{(1b)}$$

A solution to the above equation exists because the marginal maintenance and rewriting costs (with respect to m) are positive, and the former is greater than the latter.

To show that m^* is a minimum, we take the second derivative:

$$\frac{\partial X(m,s,\phi)}{\partial m}\Big|_{m=m^*} = \frac{\partial^2 C(m,s)}{\partial m^2}\Big|_{m=m^*} + \frac{\partial^2 R(m,s,\phi)}{\partial m^2}\Big|_{m=m^*} + \frac{\partial^2 C(M-m,s)}{\partial m^2}\Big|_{m=m^*} \qquad \text{(1c)}$$

The convexity of the software costs in m implies that m^* is a minimum, and $M\text{-}m^* > m^*$.

$$\text{Therefore, } m* < \frac{M}{2} \text{ .}$$

This proposition indicates whether it is too early or late for rewriting a given system. The intuition behind the result is obtained from Equation (1b): At the optimal point, the marginal rewriting cost is equal to the difference of the marginal costs of maintaining the rewritten and the original systems. The difference must therefore be positive, which requires m^* to be less than $M/2$. A straightforward corollary to this proposition is that when the marginal rewriting cost is zero, the system is rewritten at $M/2$.

The direct dependence of m^* upon M may seem restrictive. As stated above, there may not exist an optimal rewriting point for short horizons. However, this sensitivity to the length of the planning period highlights an important real-world aspect of software maintenance. As noted earlier, maintenance accounts for a very high percentage of the software budget. This is indicative of systems being maintained for too long without being rewritten. A possible explanation for this phenomenon is the short-term planning on the part of the software manager, whereby a system rewrite becomes economically infeasible.

Next we study the impact of software technologies on system rewrite. We address the following question: How do differences in the structuredness of the technology affect the rewriting point? A related proposition is stated below.

Proposition 2: *The number of maintenance jobs performed on the system before it is written decreases with an increase in the structuredness of the technology.*

Proof: From equation (1a), $X(m^*,s,\phi) = 0$.

From the implicit function theorem,

$$\frac{\partial m*}{\partial s} = - \frac{\partial X(m*,s,\phi)/\partial s}{\partial X(m*,s,\phi)/\partial m*} \qquad \text{(2a)}$$

from Equation (1c) $\partial X(m^*,s,\phi)\partial m^* > 0$ (2b)

Since the marginal maintenance cost is much greater than the marginal rewrite cost, and since $(M\text{-}m^*) > m^*$ (from Proposition 1), we have,

$\partial X(m^*,s,\phi)\partial m^* > 0$ (2c)

This is obtained from (1a), where

$$\frac{\partial^2 C(m*,s)}{\partial s \partial m} - \frac{\partial^2 C(M-m*,s)}{\partial s \partial m} > 0, \text{ and } \frac{\partial R(m*,s,\phi)}{\partial s \partial m}$$

is small.

Therefore, $\frac{\partial m*}{\partial s} < 0$ •

According to this proposition, a more structured technology results in a shorter operational life of a system. This result seems counter-intuitive because a system written with a more structured technology is expected to be less complex for a given number of modifications/enhancements, and warrant rewriting later. However, with an increase in structuredness, the reduction in the marginal maintenance cost of the rewritten system is more than the corresponding increase in the marginal cost of the original system. Therefore, the manager can take the advantage of an overall cost reduction with an earlier rewrite of the system. It should be noted that the impact of structuredness on the marginal rewriting cost is small, relative to the marginal maintenance cost.

3.2 Impact of Initial System Size

The rewrite cost is significantly dependent on the initial size of the system, since the rewritten system must incorporate a number of functions equal to or higher than that of the original system at $m=0$.[6] We study the effect of initial system size on the rewriting point with the following proposition.

Proposition 3: *The operational life of the system decreases with an increase in the initial size of the system.*

Proof: From Equation (1a), $X(m^*,s,\phi) = 0$.

From the implicit function theorem.

$$\frac{\partial m*}{\partial \phi} = - \frac{\partial X(m*,s,\phi)/\partial \phi}{\partial X(m*,s,\phi)/\partial m*}$$

From (1c), $\partial X(m^*,s,\phi)/\partial m^* > 0$

Since rewrite cost increases with system size, we have $\partial X(m^*,s,\phi)/\partial \phi > 0$.

Therefore, $\frac{\partial m*}{\partial \phi} < 0$. • (2f)

With an increase in initial system size, the marginal rewriting cost increases. This causes the costs in the first period to increase. Therefore, the rewrite must occur earlier to reduce this cost.

This proposition questions the commonly observed phenomenon of larger systems being maintained for a longer period of time. While the software manager is possibly reluctant to undertake the risk of replacing large and critical systems, from the perspective of the rewriting cost, such systems should be replaced earlier than smaller systems. However, we note that this analysis applies to situations where optimal rewriting points exist. For a large system, there may be a high fixed cost associated with the rewrite, which may make replacement infeasible.

3.3 Rewriting with a Superior Technology

To this point we have considered the strategy of using the original technology for rewriting the system. However, with the availability of new software technologies, it is important to consider the option of switching to such a technology during the rewrite. Studies (Martin 1985, Rudolph 1984) indicate that advanced technologies like 4GLs reduce software development cost significantly. In fact, because of their inherent structuredness, 4GLs have been suggested as appropriate rewriting tools for easier system maintenance (Sprague and McNurlin 1986). However, as indicated by Swanson and Beath (1989), assessing the impacts of these new technologies on the maintenance function requires a detailed investigation.

From the manager's perspective, it is important to know the direction and magnitude of the shift in the rewriting point when the software is rewritten with a superior technology. A comparison of the rewriting points with different technologies is examined below.

Proposition 4: *Let a system be originally written with a technology of structuredness* $s_2(s_2>s_1)$ *be available for rewriting. Let* $m^*_1 > m^*_2$ *be the optimal rewriting points when the system is rewritten using the same technology and the superior technology respectively.* $m^*_1 > m^*_2$.

Proof: From the first order conditions for the two rewriting strategies, we get

$$\frac{\partial C(m_1,s_1)}{\partial m_1}\Big|_{m_1=m_1^*} + \frac{\partial R(m_1,s_1,\phi)}{\partial m_1}\Big|_{m_1=m_1^*} - \frac{\partial C(M-m_1,s_1)}{\partial m_1}\Big|_{m_1=m_1^*} = 0 \quad (4a)$$

$$\frac{\partial C(m_2,s_1)}{\partial m_2}\Big|_{m_2=m_2^*} + \frac{\partial R(m_2,s_2,\phi)}{\partial m_2}\Big|_{m_2=m_2^*} - \frac{\partial C(M-m_2,s_2)}{\partial m_2}\Big|_{m_2=m_2^*} = 0 \quad (4b)$$

Suppose $m_1^* < m_2^*$

Subtracting equation (4a) from equation (4b) we get

$$\frac{\partial C(m_2,s_1)}{\partial m_2}\Big|_{m_2=m_2^*} - \frac{\partial C(m_1,s_1)}{\partial m_1}\Big|_{m_1=m_1^*}$$
$$+ \frac{\partial R(m_2,s_2,\phi)}{\partial m_2}\Big|_{m_2=m_2^*} - \frac{\partial R(m_1,s_1,\phi)}{\partial m_1}\Big|_{m_1=m_1^*}$$

$$= \frac{\partial C(M-m_2,s_2,\phi)}{\partial m}\Big|_{m=m_2^*} - \frac{\partial C(M-m_1,s_1)}{\partial m}\Big|_{m=1}$$

The convexity of the maintenance cost and $m_1^* < m_2^*$ imply that

$$\frac{\partial C(m_2,s_1)}{\partial m_2}\Big|_{m_2=m_2^*} - \frac{\partial C(m_1,s_1)}{\partial m_1}\Big|_{m_1=m_1^*} > 0 \qquad (4c)$$

Since the maintenance performed on the existing system does not affect the rewrite cost significantly,

$$\frac{\partial R(m_2,s_2,\phi)}{\partial m}\Big|_{m=m_2^*} - \frac{\partial R(m_1,s_1,\phi)}{\partial m}\Big|_{m=m_1^*}$$

is relatively small.

By hypothesis, $s_1 < s_2$, *and* $m^*_1 < m^*_2$. Moreover the marginal maintenance cost is decreasing in structuredness and increasing in m. Combining these we get,

$$\frac{\partial C(M-m_2,s_2)}{\partial m_2}\Big|_{m_2=m_2^*} - \frac{\partial C(M-m_1,s_1)}{\partial m_1}\Big|_{m_1=m_1^*} < 0 \qquad (4d)$$

Thus, the R.H.S. is negative and L.H.S. is positive. This is a contradiction. Therefore, $m^*_1 \geq m^*_2$. We arrive at a similar contradiction by supposing that m^*_1 is equal to m^*_2. •

This proposition indicates the sensitivity of the optimal rewriting point to the rewriting technology. It suggests that rewriting should occur earlier as the rewriting technology becomes more structured. Equation (4b) provides the intuitive justification for this result. With an increase in the structuredness of the rewriting technology, the marginal maintenance cost of the rewritten system decreases considerably. However, the maintenance cost in the first period remains unchanged. Therefore, the optimal rewriting point must occur earlier to take an advantage of the decrease in the marginal maintenance cost of the rewritten system.

In the above proposition, we studied the differential impacts of two rewriting technologies while keeping the original technology unchanged. Next we compare two initial technologies which are to be replaced by a given superior technology.

Proposition 5: *We compare two systems originally written with technologies 1 and 2, with structuredness* s_1 and s_2 *respectively* ($s_2 > s_1$). *Both the systems are rewritten using a third technology of structuredness* s_3, *which is superior to both 1 and 2, i.e.,* $s_1 > s_2 < s_3$. *The inferior system is replaced earlier, i.e.,* $m^*_1 < m^*_2$.

Proof: From the first order conditions for technologies 1 and 2, we get

$$\frac{\partial C(m_1,s_1)}{\partial m_1}\Big|_{m_1=m_1^*} + \frac{\partial R(m_1,s_3,\phi)}{\partial m_1}\Big|_{m_1=m_1^*}$$
$$- \frac{\partial C(M-m_1,s_3)}{\partial m_1}\Big|_{m_1=m_1^*} = 0 \qquad (5a)$$

$$\frac{\partial C(m_2,s_2)}{\partial m_2}\Big|_{m_2=m_2^*} + \frac{\partial R(m_2,s_3,\phi)}{\partial m_2}\Big|_{m_2=m_2^*}$$
$$- \frac{\partial C(m-m_2,s_3)}{\partial m_2}\Big|_{m_2=m_2^*} = 0 \qquad (5b)$$

Suppose $m^*_1 < m^*_2$. Subtracting Equation (5a) from equation (5b) we get

$$\frac{\partial C(m_2,s_2)}{\partial m_2}\Big|_{m_2=m_2^*} - \frac{\partial C(m_1,s_1)}{\partial m_1}\Big|_{m_1=m_1^*}$$
$$+ \frac{\partial R(m_2,s_3,\phi)}{\partial m_2}\Big|_{m_2=m_2^*} - \frac{\partial R(m_1,s_3,\phi)}{\partial m_1}\Big|_{m_1=m_1^*}$$

$$= \frac{\partial C(M-m_2,s_3)}{\partial m_2}\Big|_{m_2=m_2^*} - \frac{\partial C(M-m_1,s_3)}{\partial m_1}\Big|_{m_1=m_1^*}$$

$$\frac{\partial C(m_2,s_2)}{\partial m_2}\Big|_{m_2=m_2^*} - \frac{\partial C(m_1,s_1)}{\partial m_1}\Big|_{m_1=m_1^*} < 0$$ (similar to (4d)

with $m_1^* > m_2^*$)

Convexity of R in m and $m^*_1 > m^*_2$ implies that

$$\frac{\partial R(m_2,s_3,\phi)}{\partial m_2}\Big|_{m_2=m_2^*} - \frac{\partial R(m_1,s_3,\phi)}{\partial m_1}\Big|_{m_1=m_1^*} < 0$$

$$\frac{\partial C(M-m_2,s_3)}{\partial m_2}\Big|_{m_2=m_2^*} - \frac{\partial C(M-m_1,s_3)}{\partial m_1}\Big|_{m_1=m_1^*} > 0$$

(similar to (4c) with $m_1^* > m_2^*$)

Thus, the R.H.S. is negative, while the L.H.S. is positive. This is a contradiction. Hence $m_1^* < m_2^*$.●

Intuitively, systems developed with less structured (inferior) technologies are expected to be rewritten earlier; however, Proposition 2 presented the opposite view, suggesting that such systems should be maintained longer, when they are to be rewritten with the same technology. This proposition considers the option of using a superior technology for the rewrite, and shows that inferior initial technologies are indeed replaced earlier with the availability of better technologies.

Combining Propositions 4 and 5 we note that the optimal rewriting point is a function of the structuredness of both the original and rewriting technologies. An interesting corollary emerges from these propositions. As the difference between the structuredness of the original and rewriting technologies increases, it is better to rewrite the system earlier (see Figure 5) because of an increasing relative cost advantage in the second period.

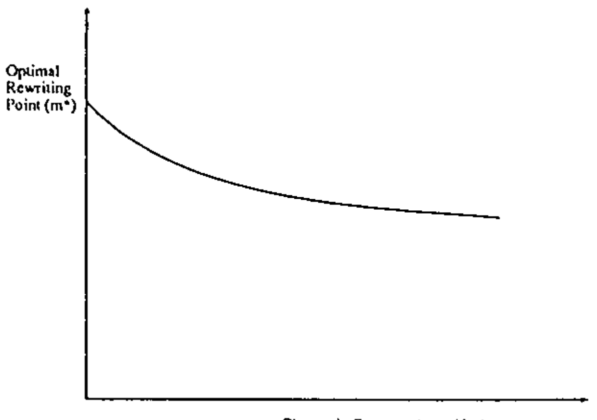

Figure 5. Impact of the Difference Between the Structuredness of Original and Rewriting Technologies (Δs) Upon the Optimal Rewriting Point (m*)

3.4 Multiple Rewritings of Code

For long time horizons and frequent maintenance requests, multiple rewrites may be more economical than a single rewrite of the system. In this section, we determine the optimal rewriting points for the multiple rewrite case, and analyze the impact of structuredness on the rewriting points.

Proposition 6: *The number of maintenance jobs performed on the system before it gets rewritten increases progressively with the number of rewrites. If $m_1, m_2, m_3, ..m_{n-1}, m_n$ are the optimal rewriting points, then this proposition implies that $m_1^* < m_2^*-m_1^* < ... m_n^*-m_{n-1}^* < M-m_n^*$.*

Proof: From the first order conditions we get:

$$\frac{\partial C(m_1,s)}{\partial m_1}\Big|_{m_1=m_1^*} + \frac{\partial R(m_1,s,\phi)}{\partial m_1}\Big|_{m_1=m_1^*} - \frac{\partial C(m_2^*-m_1,s)}{\partial m_1}\Big|_{m_1=m_1^*} = 0 \qquad (6a)$$

$$\frac{\partial C(m_2-m_1^*,s)}{\partial m_2}\Big|_{m_2=m_2^*} + \frac{\partial R(m_2,s)}{\partial m_2}\Big|_{m_2=m_2^*} - \frac{\partial C(m_3^*-m_2,s)}{\partial m_2}\Big|_{m_2=m_2^*} = 0 \qquad (6b)$$

•
•
•

$$\frac{\partial C(m_n-m_{n-1}^*,s)}{\partial m_n}\Big|_{m_n=m_n^*} + \frac{\partial R(m_n,s,\phi)}{\partial m_n}\Big|_{m_n=m_n^*} - \frac{\partial C(M-m_n,s)}{\partial m_n}\Big|_{m_n=m_n^*} = 0 \qquad (6n)$$

Consider Equation (6a). By comparing with Equation (1a) we can conclude that a solution exists and $m_2^*-m_1^* > m_1^*$. Similarly from equation (6b), $m_3^*-m_2^* > m_2^*$.

From Equation (6n), $M-m_n^* > m_n^*-m_{n-1}^*$.

Thus the operational life is increasing progressively. •

This result is a generalization of Proposition 1 *($m^* < M/2$)* where a single rewrite was considered. An important observation is that if the rewriting cost is independent of *m*, then the rewrites are equally spaced. Each system is then operational for *M(n+1)* maintenance jobs, where *n* is the number of rewrites. This situation may be encountered in an environment where maintenance primarily consists of modifications rather than enhancements. In such a situation, the size of the system remains approximately equal to its initial value, making the rewriting cost independent of the cumulative maintenance activities.

Proposition 7: *All rewriting points occur earlier with an increase in the structuredness of the technology.*

Proof: From Equation (6a), $X(m_1^*,s,\phi) = 0$

Also, $$\frac{\partial m_1^*}{\partial s} = - \frac{\partial X(m_1^*,s,\phi)/\partial s}{\partial X(m_1^*,s,\phi)/\partial m_1}$$

By comparing with Equation (2b), we have $\partial X((m_1^*,s,\phi)/\partial m_1^* > 0$

By comparing with Equation (2c) and using Proposition 6, we have $\partial X(m_1^*,s,\phi)\partial s > 0$

This implies that $\dfrac{\partial m_1^*}{\partial s} < 0.$

Similarly it can be shown that

$$\frac{\partial m_2^*}{\partial s} < 0, ,..., \frac{\partial m_n^*}{\partial s} < 0 \bullet$$

Since the optimal rewriting points occur earlier with an increase in structuredness, there is a possibility of an increase in the number of rewrites.

3.5 Application of the Model: A Numerical Example

To this point, we have developed a theoretical analysis of the software replacement problem. In this section, we use some parameter values suggested by earlier empirical studies to illustrate the applicability of the model.

Let Z_n denote the size of the system (initial size Z_0) after it has been maintained for n years. We take the typical initial system size to be 30 KSLOC, the median size reported by Swanson and Beath (1989) in a study of more than 500 systems across twelve organizations. They also state that systems tend to grow at an average rate of 10 percent per year, i.e., Z_n/Z_{n-1} (denoted by r) is equal to 1.1. This increase in size is due to enhancements performed on the system; such enhancements constitute around 40 percent of the overall maintenance activity (Swanson and Beath). For this illustration, we may then take the number of lines added/modified per year to be approximately equal to 25 percent of the system size. Therefore, for a system of initial size Z_0, if m is the number of lines modified/added during n years, then the size of the resulting system can be shown to be equal to $r[m(r-1)/.25 + Z_0]$.

We take the planning horizon (T) to be 15 years, as many systems are maintained for this duration. Let M be the total number of lines modified/added during this period. We get $M=210$ KSLOC using the expression below:

$$M=.25\ Z_0\frac{r^{T-1}-1}{r-1}$$

The cost of maintenance (C) is assumed to be of the form $k_1m^{\beta}_1$, where $\beta_1 > 1$, and m is the number of lines added or modified. β_1 decreases as the structuredness of the underlying technology increases. The rewriting cost (R) is taken to be $k_2Z^{\beta}_2$, $\beta_2 > 1$. For a given technology, we expect k_1 and β_1 to be higher than k_2 and β_2 respectively, since maintenance activities for old systems should require more effort per line of code than development. Note that the functional forms are taken from the COCOMO (basic) model (Boehm 1981), and have the characteristics of the maintenance and rewriting costs proposed earlier in this paper.

The optimal m must satisfy the condition

$$k_1\beta_1 m^{\beta_1-1} + \frac{k_2\beta_2 r^{\beta_2(r-1)}}{.25}[m(r-1)/.25 + Z_0]^{\beta_2-1} - k_1\beta_1(M-m)^{\beta_1-1} = 0$$

From the above equation, we get the optimal rewriting point for given values of k_1,k_2,β_1, and β_2. For example, with $k_1 = 3.6$, $k_2 = 2.4$, $\beta_1 = 1.2$, and $\beta_2 = 1.05$, we obtain m^* as 73 KSLOC, and the corresponding replacement time as approximately 7 years. Note that the values of k's and β's are taken from the COCOMO parameters (Boehm 1981).

Within this simple framework, our example suggests that the rewriting should occur earlier than what would be typically observed for a similar system. However, we are aware of other factors (such as managerial risk aversion and myopic planning) that can delay the replacement. These issues are discussed in the following section.

4. FUTURE RESEARCH

An economic model for analyzing the tradeoffs between software maintenance and rewriting costs has been presented. While this paper generalizes and extends an earlier model by Barua and Mukhopadhyay (1989), incorporating certain features would significantly enhance the current model. A plan for our future research is summarized below.

A relatively straightforward but useful extension examined in this research concerns the partial rewriting of software, when a complete rewrite proves to be too costly. This feature can be incorporated in the model by hypothesizing a relationship between the fraction of the system rewritten and the corresponding reduction in the subsequent maintenance cost.

An important aspect not considered in this research is the maintenance backlog and associated delay (opportunity) cost incurred by the system users. With fixed software maintenance resources, the backlog of user requests and the delay cost are expected to increase rapidly due to a continuous deterioration of the software over time. How can the backlog be controlled over the planning horizon? We hypothesize that the backlog can be reduced with the deployment of additional maintenance resources such as manpower. Our ongoing research is focusing on the derivation of the optimal schedule for resource addition to minimize the total (backlog plus resource) cost. A related issue involves the resource capacity constraints and the maintenance service level to be provided. Including these features will make the model more realistic.

We considered the initial system size, the technology and the planning horizon to be exogenously given. Interesting results may be obtained from a model where the technology and the planning horizon are chosen endogenously in a sequential mode.

Yet another issue to be addressed is the risk attitude of the software manager. A widely observed fact is that software tends to be maintained for too long without being rewritten. While it may be very costly to maintain the system beyond a certain time, we argue that the decision to continue with the system need not constitute an irrational behavior on part of the software manager. We have assumed a risk neutral manager taking the rewriting decision. More realistically, the manager may be risk averse when it comes to replacing the system. We are currently investigating the effect of risk aversion on the delay in rewriting the system. A related issue is the adoption of a myopic decision rule by the manager, whereby he/she periodically considers a short time horizon in determining whether or not to replace the system. We conjecture that the joint effect of risk aversion and the myopic rule may be such that the manager never replaces the system.

5. CONCLUSION

Software maintenance constitutes a significant part of the software budget and is expected to remain so in the foreseeable future. Thus the study of various software maintenance strategies and their economic consequences is of prime importance to researchers in Information Systems. There exists a considerable body of well-researched literature on the technical aspects of maintenance (such as determinants of software complexity). However, it is only recently that researchers (e.g., Banker, Datar and Kemerer 1988; Banker, Datar and Zweig 1989) have begun to address the economics of software maintenance. In particular, the tradeoffs between maintaining and rewriting old software have not been investigated earlier. Our aim is to bridge this gap through the development of a theory-based model of software replacement. We have utilized concepts from economics and software engineering to develop such a model.

The model allows us to compare the economics of alternative rewriting strategies and to determine the optimal rewriting point(s) when they exist. Several interesting results (propositions) with managerial implications emerged from the analysis. These include the impacts of structuredness, initial system size and switching to new technologies upon the rewriting point(s).

The limitations of the model and our plans for future research have been summarized above. Currently we are in the theoretical phase of our research, deducing the implications of our mathematical model. The next step will involve empirical testing of some of the results obtained in this phase. Our findings from the theoretical model will help specify the exact data requirements for such empirical work.

6. REFERENCES

Banker, R. D.; Datar, S. M.; and Kemerer, C. F. "A Model to Evaluate Factors Impacting the Productivity of Software Maintenance Projects." Working paper #2093-88, Massachusetts Institute of Technology Sloan School of Management, 1988.

Banker, R. D.; Datar. S. M.; and Zweig, D. "Software Complexity and Maintainability." In J. I. DeGross, J. C. Henderson, and B. R. Konsynski (eds.), *Proceedings of the Tenth International Conference on Information Systems*, Boston, December 1989, pp. 247-255.

Barua, A., and Mukhopadhyay, T. "A Cost Analysis of the Software Dilemma: To Maintain or to Replace." *Proceedings of the Twenty-Second Hawaii International Conference on Systems Sciences*, January 1989.

Bergland, G. D. "A Guided Tour of Program Design Methodologies." *IEEE Computer*, October 1981, pp. 13-37.

Boehm, B. *Software Engineering Economics*. Englewood Cliffs, New Jersey: Prentice Hall, 1981.

Boehm, B. "Improving Software Productivity." *IEEE Computer*, September 1987, pp. 43-57.

Curtis, B.; Sheppard, S. B.; Milliman, P.; Borst, M. A.; and Love, T. "Measuring the Psychological Complexity of Software Maintenance Tasks with the Halstead and McCabe Metrics." *IEEE Transactions on Software Engineering*, Volume SE-5, Number 2, March 1979, pp. 96-104.

Gurbaxani, V., and Mendelson, H. "Software and Hardware in Data Processing Budgets." *IEEE Transactions on Software Engineering*, Volume SE-13, Number 3, March 1987, pp. 1010-1017.

Kafura, D., and Reddy, G. "The Use of Software Complexity Metrics in Software Maintenance." *IEEE Transactions on Software Engineering*, Volume SE-13, Number 3, March 1987, pp. 335-343.

Kemerer, C. F. *Measurement of Software Development Productivity*. Unpublished Ph.D. Dissertation, Graduate School of Industrial Administration, Carnegie Mellon University, Pittsburgh, Pennsylvania, April 1987.

Lehman, M. "Programs, Life Cycles, and Laws of Software Evolution." In G. Parikh and N. Zvegintzov (eds.), *Tutorial on Software Maintenance*, Los Angeles: IEEE Computer Society, 1982, pp. 199-215.

Lientz, B. P.; Swanson, E. B.; and Tompkins, G. E. "Characteristics of Application Software Maintenance." *Communications of ACM*, Volume 21, Number 6, June 1978, pp. 466-471.

Martin, J. *Fourth Generation Languages Volume I, Principles*. Englewood Cliffs, New Jersey: Prentice Hall, Inc., 1985.

Rudolph, E. E. "Productivity in Computer Application Development." University of Auckland, New Zealand, 1984.

Schneidewind, N. "The State of Software Maintenance." *IEEE Transactions on Software Engineering*, Volume SE-13, Number 3, March 1987, pp. 303-310.

Sprague, R. H., and McNurlin, B. C. (Editors) *Information Systems Management in Practice*. Englewood Cliffs, New Jersey: Prentice Hall, 1986.

Swanson, E. B. "The Dimensions of Maintenance." *Proceedings of the Second International Conference on Software Engineering*, 1976, pp. 492-497.

Swanson, E. B., and Beath, C. M. *Maintaining Information Systems in Organizations*. New York: John Wiley and Sons, 1979.

Yao, S. S., and Collofello, J. "Some Stability Measures for Software Maintainers." *IEEE Transactions on Software Engineering*, Volume SE-6, Number 6, November 1980.

7. ENDNOTES

1. Advanced technologies such as 4GLs have been found to make programming less time consuming (Martin 1985; Rudolph 1984), although the economic issues such as initial acquisition and learning costs have not been investigated.

2. Without assuming a restrictive functional form, we only utilize the signs of the first and second derivatives.

3. It is assumed that both of the systems are developed in a structured manner within the limits of technologies A and B.

4. For example, the number of function points or source lines of code modified or added.

5. In this paper, we consider a complete rewrite of the software. On the other hand, a partial rewrite will be affected more directly by the state of the system. An analysis of partial rewriting is suggested as an extension in Section 4.

6. It will be equal if maintenance involves only modifications. Enhancements to the system increase the functionality.