A Field Study of Scale Economies in Software Maintenance

Rajiv D. Banker • Sandra A. Slaughter
School of Management, The University of Texas at Dallas, Richardson, Texas 75083-0688
Graduate School of Industrial Administration, Carnegie Mellon University, Pittsburgh, Pennsylvania 15213

Software maintenance is a major concern for organizations. Productivity gains in software maintenance can enable redeployment of Information Systems resources to other activities. Thus, it is important to understand how software maintenance productivity can be improved. In this study, we investigate the relationship between project size and software maintenance productivity. We explore scale economies in software maintenance by examining a number of software enhancement projects at a large financial services organization. We use Data Envelopment Analysis (DEA) to estimate the functional relationship between maintenance inputs and outputs and employ DEA-based statistical tests to evaluate returns to scale for the projects. Our results indicate the presence of significant scale economies in software maintenance, and are robust to a number of sensitivity checks. For our sample of projects, there is the potential to reduce software maintenance costs 36% by batching smaller modification projects into larger planned releases. We conclude by rationalizing why the software managers at our research site do not take advantage of scale economies in software maintenance. Our analysis considers the opportunity costs of delaying projects to batch them into larger size projects as a potential explanation for the managers' behavior.
(Software Maintenance; Data Envelopment Analysis; Software Productivity; Software Economics; Software Engineering; Management of Computing and Information Systems)

1. Introduction

Software maintenance represents a significant cost for organizations. The IEEE estimates that in the United States alone, companies spend more than $70 billion annually on software maintenance (Sutherland 1995). A report by Forrester Research, Inc., describes software maintenance as a "black hole" that consumes over 75% of the Fortune 1000's Information Systems (IS) budget (Eastwood 1993). Thus, there is increased motivation to manage software maintenance more effectively (Hanna 1993). Yet, perhaps because software maintenance tends to be viewed as a necessary evil, many technological and process improvements are directed toward software development rather than maintenance (Swanson and Beath 1989). There is a growing realization, however, that productivity improvements in software maintenance can enable redeployment of IS resources to other activities.

In this study, we investigate the influence of project size on software maintenance productivity. Much of the work in software maintenance arises as relatively small requests from the users for modifications to existing software systems. Many organizations handle these requests by simply processing them as they are submitted.[1] This approach ignores benefits from scale economies that may arise from proactive renovation strategies such as a release control concept (Branch et al. 1985,

[1] A survey of software maintenance practices by Nosek and Palvia (1990, p. 170), for example, found that while 89% of the responding organizations reported *logging* user requests for changes, less than 33% reported *batching* change requests.

0025-1909/97/4312/1709$05.00

McNeil 1979) in which user requests for modifications to installed applications are grouped into a package and implemented in a release.

Although there has been considerable interest in investigating project scale economies for software *development* (e.g., Banker et al. 1994, Byrnes et al. 1993, Banker and Kemerer 1989), very little examination of this issue has been done for software *maintenance*. Empirical studies of software maintenance suggest that maintenance is a different economic process than development, with significant potential to control project size (Swanson and Beath 1989). However, performance incentives in software maintenance generally do not favor maximizing project scale efficiency. This is due, in part, to the low precision of information signals for performance evaluation in software maintenance (Banker and Kemerer 1992).

Software maintenance includes work of a short term nature (e.g., fixing critical problems) as well as of a long term nature (e.g., a major enhancement to an existing system). However, maintenance performance is typically evaluated from the short term perspective, i.e., all maintenance work is tacitly deemed essential in keeping software systems operational on a day-to-day basis (Jones 1991). From this perspective, even if scale economies are present, managers are penalized for delaying projects in order to group them into an efficient size. As a result, there may be little incentive to take advantage of scale economies in the maintenance production process. Thus, it is important to empirically examine scale economies in software maintenance. Field studies such as this can provide *evidence* to justify altering suboptimal performance-related incentive schemes in a manner that would improve software maintenance practice.

We explore the relationship between enhancement project size, project team inexperience, project tools and techniques, and productivity for 129 software enhancement projects completed over a three and one-half year timeframe at a large financial services organization. In the context of our study, we define an enhancement project to include the activities necessary to make and implement the software modifications for a particular user request. We use the nonparametric Data Envelopment Analysis (DEA) methodology (Banker et al. 1984) to estimate the functional relationship between maintenance inputs and outputs. We also employ DEA-based statistical tests to evaluate returns to scale in software maintenance (Banker and Chang 1995).

Our results indicate that significant scale economies are present in the enhancement projects at our research site. We confirm the robustness of our results by various approaches, including sensitivity analysis to assess the influence of potential outliers and conducting tests for returns to scale under alternative formulations of our basic model. In all cases, the results confirm the presence of significant scale economies in our sample of enhancement projects. Using the median project size as an example, we find that if the median project were scale efficient, there would be almost 4.5 times improvement in the use of labor hours to modify software functionality.

We rationalize why the software managers at our research site do not take advantage of scale economies in software maintenance despite the potential for labor efficiency benefits. Our analysis considers organizational incentives such as the opportunity costs of delay due to batching enhancement projects as a potential explanation for the behavior of the software managers. For example, for the median project size, we find that to take advantage of scale economies, the software manager would need to wait 24 times as long to batch up to the efficient scale size. The rational manager will not delay beyond the batch size for which the cost of delaying further exceeds the additional benefit from improved efficiency in the use of labor hours. Thus, we conclude that the behavior of the software managers is consistent with the incentives reflected in the optimal tradeoff between delay costs and scale economies benefits.

In the following sections of the paper, we describe our microeconomic model of production in the software maintenance context, outline our research methodology, and present an analysis of our empirical data. We conclude with a discussion of the implications of our results for software maintenance management and future research.

2. An Economic View of Software Maintenance Productivity

Software maintenance refers to the modification of software after implementation to correct errors, to improve performance, or to adapt to a changed environment

(Schneidewind 1987, Swanson 1976). In this study, we focus on projects that adapt or enhance software functionality for existing systems.[2] From an economic perspective, software maintenance can be conceptualized as an economic production process (Banker et al. 1991, Kemerer 1987, Kriebel and Raviv 1980). In this sense, software enhancement projects convert inputs (the effort of maintenance professionals) into outputs (modified software). The maintenance production function represents the efficient relationship between effort and modified software for enhancement projects, with the less efficient software enhancement projects lying further below the production frontier (Varian 1992, Aigner and Chu 1968).

Although there has been little examination of scale efficiencies in software *maintenance*, there have been several studies of scale economies in software *development* (Banker et al. 1994, Byrnes et al. 1993, Banker and Kemerer 1989, Vessey 1986, Jeffrey and Lawrence 1979, Walston and Felix 1977). An empirical analysis of returns to scale in software development for eight publicly available data sets demonstrated the presence of both scale economies and diseconomies (Banker and Kemerer 1989). The authors suggest that, in software development, productivity increases on larger projects arise from spreading fixed project management overhead over a larger base, and from greater use of specialized personnel and tools (Boehm 1981). However, eventually, larger project size tends to increase the complexity of interface requirements, the number of inter- and intra-project communication paths, and the requirements for documentation (Brooks 1995, Conte et al. 1986). Thus, average productivity of the project team is likely to decline beyond the most productive scale size of the project (Banker 1984). These findings are supported by an analysis of eleven software development datasets (Banker et al. 1994), which demonstrated the presence of both economies and diseconomies of scale in software development.

[2] In industry, software enhancements represent the largest category of IS maintenance expenditures (Nosek and Palvia 1990, Hale and Haworth 1988, Lientz and Swanson 1980). Corrective and perfective software maintenance represent a smaller proportion of maintenance expenditures, accounting for about 20% and 25% of expenses, respectively.

We hypothesize that, in contrast to software development, software maintenance is primarily characterized by scale economies. Although there are similarities between software development and maintenance, software maintenance is a different production process because the programming team must spend a significant amount of time attempting to understand the purpose and construction of the programs to be modified (Banker et al. 1997, Littman et al. 1987, Fjeldstad and Hamlen 1983). In addition, the team must integrate modifications into an existing technical and user environment. Thus, while economies and diseconomies of scale arise for similar reasons as in software development, there are additional sources contributing to scale economies in software maintenance. For example, efficiency gains in maintenance occur where requests for a particular system are batched, so that the maintenance team can spread learning curve effects from becoming familiar with that system over several requests. In addition, it is more productive to make several changes to a system and then perform a single system test prior to reinstallation, instead of a series of tests. Migrating program, data library, and other technical changes from a test to production environment is more efficient if done for a batch of changes, rather than for each individual small change, depending upon the nature of the IS administrative procedures for compiling into production. There are also efficiencies realized from accomplishing and implementing documentation updates from a batch of changes, rather than continually modifying documentation as small changes are made.

Even if scale economies are present in software maintenance, there could be organizational resistance against a proactive release control process. The nature of enhancement project requests tends to differ from software development in that the requests are generally smaller, incremental to existing systems, and have a more immediate impact on the user's work. Thus, although "true" emergency repairs account for only a small portion of maintenance work (Lientz and Swanson 1980), there may be institutional pressures to complete all maintenance requests upon demand. In describing the implementation of a System Release Discipline (SRD) approach to maintenance, McNeil (1979, p. 112) notes some of the difficulties encountered:

> ". . . some old habits may die hard. Software staff may feel they are giving up control of 'their' system, and indeed they must sacrifice the freedom to bomb it at will. Users, for their part, may object to being unable to demand a new feature for the system on an overnight basis. Their payoff must wait, but at least when the feature is implemented it is more likely to work properly. Operations staff may have been getting blamed for every system problem anyhow, so they can be expected to favor any change in procedures which makes their lives less hectic."

These sentiments are emphasized by software maintenance professionals participating in a study by Dekleva (1992, p. 17):

> ". . . due to the length of maintenance tasks, new requests frequently come along before the ongoing task is completed. There is always something more critical or another user with a problem. A great deal of time is wasted on stopping and starting maintenance tasks."

In software maintenance, organizational incentives can perpetuate scale inefficiencies. To the users in the organization, support for existing systems is viewed as necessary to task accomplishment, with immediate service response and minimization of downtime as key measures of performance (Grady and Caswell 1987). Due to the large backlog of IS work, users can place a high priority on every modification request, with the notion that requests of lower priority would not likely be addressed until much later, if at all (Martin and McClure 1983). Thus, because all maintenance is viewed as essential to system operation, maintenance managers could be penalized for delaying modifications, even if it is more efficient to batch the modifications into larger projects. These pressures are reflected in a remark by a maintenance manager at our research site:

> ". . . it would be nice if I could plan (maintenance) requests, if I had extended time to plan and schedule. For example . . . there are so many small projects in the (specific application) area which I'd like to batch if I could. I'd like to use a release method like software vendors . . . but the users wouldn't go for it. It's too dynamic here." [Interview transcript, January 25, 1993]

Organizational incentives and pressures could thus promote an inefficient process in which scale economies are left unexploited in favor of completing each modification as it is requested.

To justify change in maintenance practice, and to support the development of more efficient economic performance evaluation mechanisms in software maintenance, it is critical to provide *evidence* of the potential benefits to be gained from managerial innovations that take advantage of scale economies in accomplishing software maintenance tasks. Our study makes a contribution in this area by empirically documenting the existence and extent of scale economies in software enhancement projects at our research site and by providing insight into why software managers fail to take advantage of scale economies.

3. Methodology

3.1. Research Site

Data for this study were collected at a large financial services organization. The IS department for the organization is located at company headquarters and supports all centralized computer processing activities for the company. The organization has a large investment in applications written in the COBOL programming language that run on IBM mainframe computers. The study of software maintenance in this kind of environment is important. According to the Gartner Group, there are more than 240 billion lines of code in all computer languages, about 80 percent of which is COBOL (McFarland 1995). A large portion of business computing expenditures remains devoted to the maintenance of COBOL software (Hoffman 1996). At the research site, approximately two-thirds of the application portfolio was in COBOL at the time of the study. The majority of maintenance expenditures (80–90%) was spent on modifications to these COBOL applications. Most of the application software in the firm was written in the 1970s and 1980s, with the oldest system written in 1969.

The IS department supports both in-house developed and packaged software applications that have been written employing a variety of tools and techniques. At the end of 1993, the size of the application portfolio was estimated at 366,137 function points (approximately 73.3 million lines of code). Types of maintenance work include user support (nonprogramming activities such as responding to user queries), repairs (corrections of application defects), and enhancements (addition or modification of application functionality). In 1993, more than 50% of total IS expenditures was spent on enhancements, 3% was new development, and the

remainder included a variety of activities such as user support, feasibility studies, and business process redesign projects. Enhancement projects accounted for similar proportions of total IS expenditures from 1990 to 1992. Thus, we chose to examine software enhancement projects as they represented the major kind of IS work at this site.

In this organization, project teams do not utilize a formal change management program in software maintenance, i.e., requests for all types of maintenance work are processed individually. There is a strong incentive to deliver each enhancement project by the date requested by the user; application managers and project leaders are evaluated according to their ability to deliver projects on time. In addition, the IS department often competes with consultants and contractors for project work requested by the organization's business units, with the effect that project dates are set aggressively. As one project leader commented,

> "There's this feeling that if we can't do this (project), the business will find someone else who can . . . upper management is always looking over their shoulders for something better out there." [Interview transcript, July 13, 1994, Application Project Leader]

Thus, although a program to encourage scale efficiency may be beneficial, there are no organizational and individual performance incentives in place to support such a program. That the current approach to managing enhancement projects is unsatisfactory is reflected in the remarks of one of the application managers at the site:

> "It would be nice to have a change management program . . . to install changes on a schedule. But we try to be more flexible to meet what the users want, and then we lose control . . . entirely. It's a zoo. We need to get control." [Interview transcript, June 15, 1994, Application Manager]

3.2. Data Collection

Our general strategy was to collect data retrospectively for completed software enhancement projects. This approach was chosen because it was not feasible to start a process to collect project data contemporaneously and wait for a sufficient mass of data to be available. We verified the accuracy and reliability of the data by checking for unusual values, by interviews with project team personnel and by comparing against the original project documentation and other company databases. In addition, to increase the validity of the project data, we imposed a number of selection criteria: recency of completion, similarity of project type, and similarity of programming language. Personnel turnover and lack of documentation retention make accurate data collection impossible for projects that are less recent. In addition, technology and personnel involved are more similar for projects completed in a shorter and more recent timeframe; this enables cross-project comparisons. Similarity of project type is important because this improves homogeneity of the projects analyzed. We included only projects that modified functionality for the application systems; thus, projects such as package installations or conversions to new operating systems were not compared with projects that modify functionality. Our final criterion was similarity of programming language. All projects included modifications to COBOL programs; therefore, the results of our analysis are not confounded by the effects of multiple languages.

After discussions with IS staff at the research site, only enhancement projects completed within a 3.5-year timeframe (between July 1990 and December 1993) were considered for inclusion in the study. This is because archival data for projects completed prior to mid-1990 were incomplete or nonexistent. Of the 144 projects initially considered for inclusion in the study, 15 were eliminated because they were either non-COBOL or nonmodification projects. Our final data set included data for 129 software enhancement projects that modified 34 different application systems.

3.3. Empirical Analysis

To investigate the relationship between enhancement project size and productivity, we employ Data Envelopment Analysis (DEA), a nonparametric methodology for production frontier estimation developed by Charnes et al. (1981) and extended to a formal production economics framework by Banker et al. (1984).[3] We employ DEA rather than a parametric model as our primary methodology to estimate the

[3] We refer to the DEA model developed by Charnes et al. (1981) as the CCR model, and by Banker et al. (1984) as the BCC model. For an intuitive discussion of the use of DEA to estimate the software maintenance production function and evaluate returns to scale, see Banker and Slaughter (1994).

production relationship between enhancement project inputs and outputs. Because DEA does not impose a specific form on the production function and maintains relatively few assumptions, its estimates are likely to be more robust than those obtained from parametric models that postulate a certain structure like a linear or quadratic form for the maintenance production function (Banker and Maindiratta 1988).

We specify the following model to describe the production relationship between maintenance inputs and outputs: $x_j = f(y_j, Z_j)*\theta_j$. In this model, x_j and y_j are the input and output quantities, Z_j is a vector of project contextual variables, and θ_j is the inefficiency of project j. Specifically, x_j represents *project team labor*. Project team labor is assessed in terms of enhancement project hours. Labor hours is a critical input because IS personnel staff time is an expensive and scarce resource in software maintenance (Moad 1993). Project team labor is measured by the total number of labor hours logged to the project by the maintenance team, and was obtained from the organization's project time tracking system.

Enhancement project size is represented by y_j. Enhancement project size is assessed in terms of modified software functionality. The size of modified software functionality is measured by the number of function points added, changed and deleted by the project. A function point corresponds to a user business function such as *sales order entry* or *add customer*. The function point metric counts the number of unique inputs, outputs, logical files, external interface files, and external queries modified in a software project (IFPUG 1993). Each count is weighted for difficulty depending upon a number of environmental factors such as whether the software application runs in a distributed environment, or whether the application is held to above average performance standards (Albrecht and Gaffney 1983). Thus, function points provide an estimate of the number of files, reports and screens updated by the enhancement project, adjusted for environmental complexity. In a survey by the Quality Assurance Institute, function points were rated as the best available software size measure (Perry 1986), and were considered superior to the alternative measure of software lines of code. Function points have been demonstrated to be a reliable estimate of software size (Kemerer 1993). For this study, project function points were obtained from the organization's Metrics Group, which is responsible for measuring software projects and applications.

Z_j is a vector of contextual variables that affect the production relation. These are project factors, including characteristics of the project team as well as project tools and techniques. Specifically, our model includes *project team inexperience*, the *number of hardware platforms impacted by the project*, the *number of design techniques employed in the project*, *the number of development tools required in the project*, and the *number of testing tools and techniques required by the project team*. We include project team inexperience because prior empirical research has demonstrated that the experience of the project team with the software application has a significant influence on maintenance task performance (Oman et al. 1989, Vessey 1989, Littman et al. 1987, Pennington 1987, Curtis 1981). Project team inexperience is subjectively assessed on a five-point scale by software application managers at the research site in response to a project questionnaire, with one representing low inexperience and five high inexperience.

The number of hardware platforms impacted by the enhancement project as well as the diversity of project tools and techniques employed for design, development, and testing should also influence project efficiency. The evidence from manufacturing environments suggests that production of dissimilar products where there is little sharing of inputs and production processes reduces focus and results in lower performance (Skinner 1974). In economics, it is argued that there are cost complementarities or economies of scope in sharing common inputs and processes among various products with commonalities in production, and diseconomies of scope when inputs and processes differ (Panzar and Willig 1977 and 1981). In the software context, diversity of tools, techniques and platforms may require increased programmer labor in software maintenance for several reasons. Switching costs are incurred due to multiple, varied process flows and change over in processes required when modifying software that runs on different hardware platforms or when using different methodologies and tools. Diversity in platforms, tools and techniques can also increase the difficulty of software quality control, testing, and verification. For example, to change an application that runs on multiple hardware platforms, the project team must switch

Table 1 Descriptive Statistics for Variables in Model

Variable	Mean	Median	Std Dev	Min	Max	Q1	Q3	*n*
Project Function Points	424.82	108.00	865.91	4.00	5156.00	18.00	320.50	129
Project Hours	607.02	328.75	1038.30	16.00	10250.10	169.37	661.55	129
# Hardware Platforms	1.44	1.00	0.85	1.00	4.00	1.00	2.00	129
# Design Tools	0.67	1.00	1.05	0.00	5.00	0.00	1.00	129
# Development Tools	7.50	7.00	3.26	1.00	18.00	5.00	9.00	129
# Testing Tools	9.68	10.00	5.33	0.00	21.00	5.00	14.00	129
Project Team Inexperience	1.71	2.00	0.76	1.00	5.00	1.00	2.00	129
Average Project Team Size	2.92	2.00	2.22	1.00	15.00	1.50	4.00	77
Elapsed Days	96.70	50.00	232.88	1.00	1992.00	19.50	95.50	77

between tools and libraries that are appropriate to each platform. To test the changes, the project team must conduct an integration test that is not necessary in a single-platform environment. The number of hardware platforms impacted by the enhancement project, and the number of different tools and techniques used in enhancement project design, development and testing are obtained from project questionnaires and verified against project documentation. Table 1 provides descriptive statistics for the variables included in our model.

4. Empirical Results

4.1. BCC and CCR Efficiency Ratings for the Enhancement Projects

To assess the technical inefficiency $\theta_0^B \equiv \theta^B(x_0; y_0, Z_0)$ of each enhancement project $(x_0; y_0, Z_0)$, we estimate the following BCC model:

$$\theta^B(x_0; y_0, Z_0) \equiv \text{Min } \theta \tag{4.a}$$

subject to:

$$\sum_{j=1}^{129} \lambda_j x_j \le x_0\theta, \tag{4.b}$$

$$\sum_{j=1}^{129} \lambda_j y_j \ge y_0, \tag{4.c}$$

$$\sum_{j=1}^{129} \lambda_j z_{kj} \le z_{k0}, \quad k = 1, \ldots, 5, \tag{4.d}$$

$$\sum_{j=1}^{129} \lambda_j = 1, \tag{4.e}$$

$$\theta, \lambda_j \ge 0, \tag{4.f}$$

where j = project observation number, x = project labor hours, y = project function points, z_k = contextual variables, $k = 1, \ldots, 5$, θ^B = inefficiency variable, BCC model, and λ = weights on referent observations.

As shown by Banker (1993), the DEA estimator of θ is statistically consistent, and the asymptotic empirical distribution of the DEA estimates retrieves the true distribution of θ under the maintained assumptions embodied in the DEA postulates of convexity, monotonicity, envelopment and likelihood of efficient performance. These assumptions are consistent with both increasing and decreasing returns to scale, and do not impose constant returns to scale. For our sample of enhancement projects, 17 projects are identified as efficient using the BCC model.

Estimates of aggregate technical and scale efficiency under the CCR model are obtained by solving the above linear program, except that the objective function in (4.a) is maximized subject only to constraints (4.b), (4.c), (4.d), and (4.f). The CCR estimator is also statistically consistent under the maintained DEA assumptions, with the addition of a postulate for constant returns to scale. We refer to the CCR inefficiency estimate as θ^C. For our sample, nine projects are identified as technical and scale efficient using the CCR model.

4.2. Tests for Returns to Scale

Since the DEA estimator is statistically consistent, under the null hypothesis of constant returns to scale, the asymptotic empirical distributions of the DEA estimators of θ^B and θ^C are identical, each recovering the true distribu-

tion of the inefficiency variable θ (Banker 1993). This motivates the development of two semiparametric statistical tests of returns to scale (Banker and Chang 1995). The two tests correspond to different maintained assumptions about the distribution (exponential versus half-normal) of $\psi = \theta - 1 \geq 0$. If ψ is distributed exponentially, then the statistic to test for constant returns to scale is:

$$\sum_{j=1}^{J} (\theta_j^C - 1) \Big/ \sum_{j=1}^{J} (\theta_j^B - 1).$$

This statistic is evaluated asymptotically against the F-distribution with $(2N, 2N)$ degrees of freedom. If ψ is distributed half-normally as $|N(0, \sigma^2)|$, the statistic to test for constant returns to scale is:

$$\sum_{j=1}^{J} (\theta_j^C - 1)^2 \Big/ \sum_{j=1}^{J} (\theta_j^B - 1)^2.$$

This statistic is evaluated asymptotically against the F-distribution with (N, N) degrees of freedom. These tests have been used to evaluate increasing or decreasing returns to scale for software development projects (Banker et al. 1994). The robustness of similar DEA-based tests has been demonstrated for different underlying production functions and inefficiency distributions as well as small sample sizes (Banker and Chang 1995).

Table 2 summarizes the hypotheses, test statistics, and results. Under all tests, the null hypothesis of constant returns to scale is rejected at the 5% level of significance. This suggests that variable returns to scale are present. We also reject the hypotheses of decreasing returns to scale and non-increasing returns to scale. Given this result, we conclude that this data set is characterized primarily by increasing returns to scale, and that the enhancement projects are too small for maximum average productivity.

4.3. Robustness of Results

To assess the validity and robustness of our results, we conducted a number of sensitivity analyses. First, we reestimated our original model, substituting project cost for project labor hours as the input variable. This substitution reflects the conjecture that differences in project team skill levels are reflected more accurately in project cost rather than in project hours. Our results are presented in Table 3. The results are consistent with our original model: the same projects are identified as efficient, and the enhancement projects exhibit increasing returns to scale.

As a second sensitivity check, following Richmond (1974), we iteratively removed efficient projects from our data set and recomputed our test statistics to

Table 2 Returns to Scale DEA Tests for Basic Model

DEA Tests Based On: Null Hyp.	Alt. Hyp.	Exponential Distribution Assumption: Test Statistic	Exponential: F Value	Half-Normal Distribution Assumption: Test Statistic	Half-Normal: F Value	Result
CRS	VRS	$\sum_{i=1}^{129} (\theta_i^C - 1)/\sum_{i=1}^{129} (\theta_i^B - 1)$	7.79*	$\sum_{i=1}^{129} (\theta_i^C - 1)^2/\sum_{i=1}^{129} (\theta_i^B - 1)^2$	77.09*	Reject CRS
NDRS	DRS	$\sum_{i=1}^{129} (\theta_i^C - 1)/\sum_{i=1}^{129} (\theta_i^D - 1)$	1.00	$\sum_{i=1}^{129} (\theta_i^C - 1)^2/\sum_{i=1}^{129} (\theta_i^D - 1)^2$	1.00	Fail to Reject NDRS
NIRS	IRS	$\sum_{i=1}^{129} (\theta_i^C - 1)/\sum_{i=1}^{129} (\theta_i^E - 1)$	7.79*	$\sum_{i=1}^{129} (\theta_i^C - 1)^2/\sum_{i=1}^{129} (\theta_i^E - 1)^2$	77.09*	Reject NIRS

* Indicates significance at 5% level.

Key to Tables 2–6:

CRS = constant returns to scale, VRS = variable returns to scale, NDRS = nondecreasing returns to scale, DRS = decreasing returns to scale, NIRS = non-increasing returns to scale, IRS = increasing returns to scale, θ^C = CCR inefficiency variable, θ^B = BCC inefficiency variable, θ^D = NIRS inefficiency variable, θ^E = NDRS inefficiency variable.

Table 3 Sensitivity Analysis Using Project Cost

DEA Tests Based On: Null Hyp.	Alt. Hyp.	Exponential Distribution Assumption: Test Statistic	*F* Value	Half-Normal Distribution Assumption: Test Statistic	*F* Value	Result
CRS	VRS	$\sum_{i=1}^{129}(\theta_i^C-1)/\sum_{i=1}^{129}(\theta_i^B-1)$	7.42*	$\sum_{i=1}^{129}(\theta_i^C-1)^2/\sum_{i=1}^{129}(\theta_i^B-1)^2$	72.44*	Reject CRS
NDRS	DRS	$\sum_{i=1}^{129}(\theta_i^C-1)/\sum_{i=1}^{129}(\theta_i^D-1)$	1.00	$\sum_{i=1}^{129}(\theta_i^C-1)^2/\sum_{i=1}^{129}(\theta_i^D-1)^2$	1.00	Fail to Reject NDRS
NIRS	IRS	$\sum_{i=1}^{129}(\theta_i^C-1)/\sum_{i=1}^{129}(\theta_i^E-1)$	7.42*	$\sum_{i=1}^{129}(\theta_i^C-1)^2/\sum_{i=1}^{129}(\theta_i^E-1)^2$	72.44*	Reject NIRS

* Indicates significance at 5% level.

assess the extent of the influence of efficient projects on our results. We deleted the 17 projects identified as efficient using the BCC model. In the next iteration, we then deleted the 22 projects identified as efficient using the BCC model for the remaining projects. After each deletion, we recomputed the

Table 4 Sensitivity Analysis With Efficient Projects Removed

Iteration #1: 17 Projects Removed, Analysis of Remaining 112 Projects

DEA Tests Based On: Null Hyp.	Alt. Hyp.	Exponential Distribution Assumption: Test Statistic	*F* Value	Half-Normal Distribution Assumption: Test Statistic	*F* Value	Result
CRS	VRS	$\sum_{i=1}^{129}(\theta_i^C-1)/\sum_{i=1}^{129}(\theta_i^B-1)$	6.15*	$\sum_{i=1}^{129}(\theta_i^C-1)^2/\sum_{i=1}^{129}(\theta_i^B-1)^2$	39.73*	Reject CRS
NDRS	DRS	$\sum_{i=1}^{129}(\theta_i^C-1)/\sum_{i=1}^{129}(\theta_i^D-1)$	1.00	$\sum_{i=1}^{129}(\theta_i^C-1)^2/\sum_{i=1}^{129}(\theta_i^D-1)^2$	1.00	Fail to Reject NDRS
NIRS	IRS	$\sum_{i=1}^{129}(\theta_i^C-1)/\sum_{i=1}^{129}(\theta_i^E-1)$	6.15*	$\sum_{i=1}^{129}(\theta_i^C-1)^2/\sum_{i=1}^{129}(\theta_i^E-1)^2$	39.73*	Reject NIRS

* Indicates significance at 5% level.

Iteration #2: 22 Additional Projects Removed, Analysis of Remaining 90 Projects

DEA Tests Based On: Null Hyp.	Alt. Hyp.	Exponential Distribution Assumption: Test Statistic	*F* Value	Half-Normal Distribution Assumption: Test Statistic	*F* Value	Result
CRS	VRS	$\sum_{i=1}^{129}(\theta_i^C-1)/\sum_{i=1}^{129}(\theta_i^B-1)$	7.36*	$\sum_{i=1}^{129}(\theta_i^C-1)^2/\sum_{i=1}^{129}(\theta_i^B-1)^2$	56.85*	Reject CRS
NDRS	DRS	$\sum_{i=1}^{129}(\theta_i^C-1)/\sum_{i=1}^{129}(\theta_i^D-1)$	1.00	$\sum_{i=1}^{129}(\theta_i^C-1)^2/\sum_{i=1}^{129}(\theta_i^D-1)^2$	1.01	Fail to Reject NDRS
NIRS	IRS	$\sum_{i=1}^{129}(\theta_i^C-1)/\sum_{i=1}^{129}(\theta_i^E-1)$	7.36*	$\sum_{i=1}^{129}(\theta_i^C-1)^2/\sum_{i=1}^{129}(\theta_i^E-1)^2$	56.84*	Reject NIRS

* Indicates significance at 5% level.

Table 5 Sensitivity Analysis with Additional *Z* Variables

DEA Tests Based On: Null Hyp.	Alt. Hyp.	Exponential Distribution Assumption: Test Statistic	*F* Value	Half-Normal Distribution Assumption: Test Statistic	*F* Value	Result
CRS	VRS	$\sum_{j=1}^{129}(\theta_j^C-1)/\sum_{j=1}^{129}(\theta_j^B-1)$	17.55*	$\sum_{j=1}^{129}(\theta_j^C-1)^2/\sum_{j=1}^{129}(\theta_j^B-1)^2$	294.38*	Reject CRS
NDRS	DRS	$\sum_{j=1}^{129}(\theta_j^C-1)/\sum_{j=1}^{129}(\theta_j^D-1)$	1.00	$\sum_{j=1}^{129}(\theta_j^C-1)^2/\sum_{j=1}^{129}(\theta_j^D-1)^2$	1.00	Fail to Reject NDRS
NIRS	IRS	$\sum_{j=1}^{129}(\theta_j^C-1)/\sum_{j=1}^{129}(\theta_j^E-1)$	17.55*	$\sum_{j=1}^{129}(\theta_j^C-1)^2/\sum_{j=1}^{129}(\theta_j^E-1)^2$	294.38*	Reject NIRS

* Indicates significance at 5% level.

DEA test statistics for the remaining projects. Our results are presented in Table 4. These results indicate that, for each iteration, the sample of remaining enhancement projects exhibits increasing returns to scale.[4]

As a third sensitivity check, we examined the effect on scale economies of including two additional project variables in our vector of contextual variables Z_j: *average project team size* and *elapsed project days*.[5] We estimated our model for the subset of 77 projects for which these variables were available and re-computed the DEA test statistics. Our results are presented in Table 5, and indicate that these enhancement projects exhibit increasing returns to scale.

Finally, we examined alternative nonparametric formulations for our model to determine whether a different production relationship is exhibited for enhancement projects that *add* versus *modify* or *delete* software functionality. Our original model includes a single output—the total functionality added, changed, and deleted by the project. We considered the possibility that a different production relationship may exist for projects that add functionality versus change functionality. Thus, we examined alternative models with multiple outputs. One of the models includes three outputs for functionality added, functionality changed, and functionality deleted (note that the sum of these three output measures equals the single output measure we used in our original model). As only 18 projects in our sample deleted functionality, we also formed a model with two outputs: one for functionality added and one for functionality changed and deleted. Results of estimating these models are presented in Table 6. For both alternative models, DEA tests confirm the presence of significant scale economies.

In all cases, our sensitivity analyses demonstrate the robustness of our result that enhancement projects at our research site are characterized by increasing returns to scale. That is, most of the projects are too small for maximum average productivity, with the implication that significant scale economies could potentially be exploited by increasing enhancement project size.

5. Discussion

Our results provide evidence that scale economies are present in software maintenance at our research site.

[4] We also considered the possibility of measurement error in the data by estimating a stochastic DEA model (Banker et al. 1991). While formal statistical tests are not available for this model, we find the correlations between the efficiency variables for different weights for measurement errors to be very high.

[5] *Average project team size* indicates the mean size of the project team over the life of the project. *Elapsed project days* includes the number of calendar days from project start to project finish. Both variables were available for 77 of the 129 projects. As not all of the application managers wished to collect these variables, they were available only for enhancement projects that modified certain applications. A comparison of means for the variables included in our original model between the 77 projects and remaining 52 projects indicates that the means are not significantly different for any of the variables. This suggests that the 77 projects are representative of our larger sample.

Table 6 Sensitivity Analysis with a Two Output and a Three Output Model

Two Output Model:

DEA Tests Based On: Null Hyp.	Alt. Hyp.	Exponential Distribution Assumption: Test Statistic	*F* Value	Half-Normal Distribution Assumption: Test Statistic	*F* Value	Result
CRS	VRS	$\sum_{i=1}^{129}(\theta_i^C-1)/\sum_{i=1}^{129}(\theta_i^B-1)$	7.52*	$\sum_{i=1}^{129}(\theta_i^C-1)^2/\sum_{i=1}^{129}(\theta_i^B-1)^2$	70.42*	Reject CRS
NDRS	DRS	$\sum_{i=1}^{129}(\theta_i^C-1)/\sum_{i=1}^{129}(\theta_i^D-1)$	1.00	$\sum_{i=1}^{129}(\theta_i^C-1)^2/\sum_{i=1}^{129}(\theta_i^D-1)^2$	1.00	Fail to Reject NDRS
NIRS	IRS	$\sum_{i=1}^{129}(\theta_i^C-1)/\sum_{i=1}^{129}(\theta_i^E-1)$	7.52*	$\sum_{i=1}^{129}(\theta_i^C-1)^2/\sum_{i=1}^{129}(\theta_i^E-1)^2$	70.42*	Reject NIRS

* Indicates significance at 5% level.

Three Output Model:

DEA Tests Based On: Null Hyp.	Alt. Hyp.	Exponential Distribution Assumption: Test Statistic	*F* Value	Half-Normal Distribution Assumption: Test Statistic	*F* Value	Result
CRS	VRS	$\sum_{i=1}^{129}(\theta_i^C-1)/\sum_{i=1}^{129}(\theta_i^B-1)$	7.89*	$\sum_{i=1}^{129}(\theta_i^C-1)^2/\sum_{i=1}^{129}(\theta_i^B-1)^2$	78.08*	Reject CRS
NDRS	DRS	$\sum_{i=1}^{129}(\theta_i^C-1)/\sum_{i=1}^{129}(\theta_i^D-1)$	1.00	$\sum_{i=1}^{129}(\theta_i^C-1)^2/\sum_{i=1}^{129}(\theta_i^D-1)^2$	1.00	Fail to Reject NDRS
NIRS	IRS	$\sum_{i=1}^{129}(\theta_i^C-1)/\sum_{i=1}^{129}(\theta_i^E-1)$	7.91*	$\sum_{i=1}^{129}(\theta_i^C-1)^2/\sum_{i=1}^{129}(\theta_i^E-1)^2$	78.09*	Reject NIRS

* Indicates significance at 5% level.

Note: The three outputs include Project Function Points Added, Project Function Points Changed, and Project Function Points Deleted. The two outputs include Project Function Points Added, and Project Function Points Changed + Deleted. Total Project Function Points = Project Function Points Added + Project Function Points Changed + Project Function Points Deleted.

The presence or absence of scale economies at a given enhancement project size is important for software maintenance management because of the potential for reducing maintenance costs. We calculate the potential percentage cost reduction from exploiting scale economies in this set of projects using the actual project hours and the scale efficiency measure $e^{SCALE} = \theta^B/\theta^C \leq 1$ (Banker et al. 1984, p. 1089), which assesses the divergence from the most productive scale size for each project. The potential percentage cost reduction = Σ (actual project hours − scale efficient project hours)/Σ (actual project hours) = $1 - \Sigma\ [e^{SCALE} * (\text{actual project hours})/(\text{total actual project hours})] = 1 -$ weighted average scale efficiency. For our set of projects, the weighted average scale efficiency is 64.22%, which implies that if all of the projects were scale efficient, the potential percentage cost reduction would be 1 − .6422 or 35.78%. This translates into a potential reduction in total project hours of 28,018 for the set of projects, a dollar cost reduction of $1,821,170 (at the organization's standard IS wage rate of $65 per hour). The potential cost savings are $1,070,924 if only the worst half of the projects become scale efficient, and $595,523 if only the worst one-fourth of the projects become scale efficient.

An intriguing question emerges, then, from our study: *given the economies of scale in the enhancement projects, why did the software managers fail to take advantage of efficiency gains from batching the projects*? The

remarks of software managers at the research site, such as, "It would be nice to have a change management program," and "There are so many small projects . . . that I would like to batch if I could," indicate that, although the managers may not have known the specific scale sizes to optimize maintenance productivity, they were not unaware of the benefits of grouping small maintenance modifications into larger projects. It is also evident from their comments that there were institutional pressures (and in some cases, performance penalties) for delaying work on projects so that the projects could be batched. Performance requirements such as system availability and minimization of downtime also encouraged a short-term focus. That the software managers adjusted their behavior to conform to institutional pressures is evident in their remarks such as "I'd like to use a release method like software vendors, but users wouldn't go for it," or "We try to be flexible to meet what the users want and then we lose control."

From an economic perspective, the observed behavior of the software managers could be rationalized by examining the tradeoff between the opportunity cost of delay to batch projects versus the benefit of scale economies that could be exploited. To gain insight into this tradeoff, we estimated the scale elasticities for our sample of enhancement projects. Using the DEA BCC model, scale elasticity $\hat{s}$ (Banker et al. 1984, p. 1090) is estimated by:

$$\hat{s}_j = v^* x_j / \theta_j^* (u^* y_j + \sum w_k^* z_{kj}),$$

where $k = 1, \ldots, 5$ contextual variables; v^*, u^*, w^* are optimal values of the dual variables to constraints (4.b), (4.c), and (4.d), respectively; and θ^* is the optimal solution to the linear program in (4) for each of the $j = 1, \ldots, 129$ projects. Scale elasticity measures the percentage change in output corresponding to a one percent change in input. Thus, the larger the value of $\hat{s} > 1$, the greater the extent of increasing returns of scale. For our sample, 126 of the 129 values for $\hat{s}$ are greater than one, the mean value of $\hat{s}$ is 2.77, the median value is 2.24, the maximum value is 23.94, and the upper and lower quartile values are 3.30 and 1.52, all of which indicate increasing returns to scale. We also estimated a parametric function, $x = \alpha y^\beta \prod_{k=1}^5 z_k^{\gamma k}$. Table 7 shows the estimates. We find that β is significantly less than zero, and $\hat{s} = 1/\beta = 3.74$, which suggests increasing returns to scale (Panzar and Willig 1977), consistent with our nonparametric results.

We then estimated the optimal batch size for enhancement projects at our research site, adapting the standard production lot sizing model (Hax and Candea 1984, p. 134) to evaluate the tradeoff between the costs of not exploiting scale economies and the costs of delay due to batching. We optimize over a period of t days and use the following notation: n = the total number of function points of enhancement requests for an application during the period, y = project batch size in function points, x = labor hours required for a project of y function points with contextual variables z_k, w = the average hourly wage rate, and c = the opportunity cost per day of delaying a request beyond the normal production time, i.e., the cost of batching.[6] Assuming a production function $x = \alpha y^\beta \prod_{k=1}^5 z_k^{\gamma k}$ with $\beta < 1$ to reflect increasing returns to scale, the costs of production can be expressed as: $(n/y)w\alpha y^\beta \prod_{k=1}^5 z_k^{\gamma k}$, and the costs of delaying work on projects in order to batch can be expressed as: $tcy/2$.

Therefore, to minimize total costs = production costs + costs of delay = $(n/y)w\alpha y^\beta \prod_{k=1}^5 z_k^{\gamma k} + tcy/2$ with respect to y, we set the first derivative equal to zero:

$$(\beta - 1)wn\alpha y^{\beta-2} \prod_{k=1}^{5} z_k^{\gamma k} + ct/2 = 0.$$

Solving for the optimal project batch size y, we have

$$y^* = \left[(2/ct)(1-\beta)\left(wn\alpha \prod_{k=1}^{5} z_k^{\gamma k} \right) \right]^{1/(2-\beta)}.^7$$

However, since it is difficult to estimate a precise value

[6] Note that the assumptions behind this model include known demand for project function points, uniform arrival of function points, work is done sequentially, and batching is not possible across applications. In the context of software maintenance at our research site, it is reasonable to assume that demand for function points is known and that only within-application batching is possible because of the nature of the enhancement projects and the organizational structure (individual application managers respond to the needs of specific sets of users). Whether work is done sequentially (i.e., one project at a time) depends on how work is assigned to the project teams within an application. The arrival of function points depends on the timing of the requests submitted by the business users.

[7] Second-order conditions indicate that the solution from the first-order condition provides a global minimum when $\beta < 1$. We employ a

Table 7 Estimates for Parametric Model

Variables	Intercept	Project Function Points	# Hardware Platforms	# Design Tools	# Development Tools	# Test Tools	Team Inexperience	Average Size	Elapsed Days	R^2	Adjusted R^2	*F*-Test (Model)
Coefficient (predicted sign)	α	β (+)	γ_1 (+)	γ_2 (+)	γ_3 (+)	γ_4 (+)	γ_5 (+)	γ_6 (+)	γ_7 (+)	–	–	–
Coefficient	−.1336	.2673*	.3454	.2634	.2772	1.4375*	.5202*	–	–	0.5845	0.5206	9.1438*
Estimated	1.0110	.0899	.2480	.2702	.3175	.4624	.2933					
Standard Error	(−.132)	(2.971)	(1.393)	(.975)	(.873)	(3.109)	(1.774)					
(*t*-value)												
$n = 129$												
Coefficient	2.8673*	.2956*	.1662	.2229	.5183*	.1758	.4510	–	–	0.4949	0.4516	11.430*
Estimated	.4871	.0492	.2018	.3114	.1911	.1654	.2116					
Standard Error	(5.886)	(5.999)	(.824)	(.716)	(2.712)	(1.063)	(2.132)					
(*t*-value)												
$n = 77$												
Coefficient	2.3879*	.2533*	.0909	.2100	.5396*	.1588	.3989*	.3742*	.0946	0.5528	0.5002	10.505*
Estimated	.5024	.0509	.1944	.3000	.1848	.1581	.2029	.1350	.0754			
Standard Error	(4.753)	(4.976)	(.468)	(.700)	(2.920)	(1.004)	(1.966)	(2.772)	(1.254)			
(*t*-value)												
$n = 77$												

* Indicates significance at 5% level. Significance of *t*-tests in one-tailed for parameter coefficients. *F*-tests are two-tailed for the model.

Table 8 **Delay Costs Necessary to Rationalize Various Batch Sizes**

Delay Cost $= c^* = (n/t)(\hat{s} - 1)(1/\hat{s})2w\alpha y^{1/(\hat{s}-2)} \prod_{k=1}^{5} z_k^{\gamma^k} = [(1608.63/1217)(\hat{s} - 1)(1/\hat{s})(2)(65)(.8749)(y^{1/(\hat{s}-2)})(1^{.3454})(1^{.2634})(7^{.2772})(10^{1.4375})(2^{.5202})]$

For various values of y and $\hat{s}$, $c^* =$

Project Function Points (y)	Delay Cost $\hat{s}$ = 23.94	Delay Cost at $\hat{s}$ = 10.00	Delay Cost at $\hat{s}$ = 3.74	Delay Cost at $\hat{s}$ = 3.30	Delay Cost at $\hat{s}$ = 2.24	Delay Cost at $\hat{s}$ = 1.52
5,500	0.00046	0.00071	0.00245	0.00316	0.00861	0.03302
5,000	0.00055	0.00085	0.00289	0.00372	0.00999	0.03753
4,500	0.00068	0.00104	0.00347	0.00445	0.01176	0.04323
4,000	0.00086	0.00131	0.00426	0.00543	0.01413	0.05063
3,500	0.00111	0.00168	0.00536	0.00681	0.01739	0.06057
3,000	0.00151	0.00225	0.00701	0.00885	0.02209	0.07450
2,500	0.00215	0.00319	0.00961	0.01206	0.02933	0.09516
2,000	0.00333	0.00487	0.01415	0.01762	0.04150	0.12840
1,500	0.00585	0.00842	0.02329	0.02871	0.06490	0.18892
1,000	0.01295	0.01818	0.04702	0.05713	0.12188	0.32559
750	0.02274	0.03141	0.07740	0.09310	0.19061	0.47906
500	0.05031	0.06786	0.15626	0.18528	0.35800	0.82563
250	0.19551	0.25327	0.51932	0.60086	1.05151	2.09366
200	0.30265	0.38700	0.76446	0.87753	1.48750	2.82490
150	0.53161	0.66849	1.25845	1.42995	2.32630	4.15649
100	1.17605	1.44433	2.54067	2.84581	4.36909	7.16342
75	2.06578	2.49489	4.18244	4.63733	6.83285	10.54010
50	4.56996	5.39044	8.44391	9.22896	12.83294	18.16514
25	17.75827	20.11782	28.06329	29.92949	37.69303	46.06349
20	27.48991	30.74043	41.30995	43.71066	53.32152	62.15182
15	48.28728	53.09988	68.00428	71.22776	83.38983	91.44880
10	106.82207	114.72734	137.29348	141.75349	156.61651	157.60571
5	415.09616	428.17758	456.29434	459.70606	460.01565	399.65942
3	1,128.70609	1,130.15112	1,105.71403	1,094.03236	1,017.71137	793.43471
2	2,496.94578	2,441.79914	2,232.32001	2,177.28199	1,911.38901	1,367.43012
1	9,702.79377	9,113.11561	7,419.10687	7,060.91691	5,614.15130	3,467.55407
In particular, for batch size:						
425 Mean	0.06923	0.09249	0.20725	0.24433	0.46122	1.02758
4 Min	642.57139	654.26390	671.67819	671.37973	650.74977	539.24614
18 Q1	33.78915	37.55340	49.58368	52.27003	62.81019	71.59473
108 Median	1.01152	1.24785	2.22349	2.49733	3.87646	6.46027
321 Q3	0.11983	0.15751	0.33676	0.39310	0.71294	1.49679
5,156 Max	0.00052	0.00081	0.00274	0.00353	0.00952	0.03601

practices. Finally, future studies could employ our methodology to assess the performance of organizations that utilize a proactive change management program in software maintenance.[9]

[9] We gratefully acknowledge research support from the Quality Leadership Center at the University of Minnesota. The cooperation and assistance of managers and staff at our data site was invaluable. Helpful suggestions from three anonymous reviewers, the associate editor, and the editors are also gratefully acknowledged.

References

Aigner, D. J. and S. F. Chu, "On Estimating the Industry Production Function," *American Economic Rev.*, 58 (1968), 826–839.

Albrecht, A. J. and J. Gaffney, "Software Function, Source Lines of

for c, we find it more convenient to pose the question in terms of the value of

$$c^* = (n/t)(1 - \beta)2w\alpha y^{\beta-2} \prod_{k=1}^{5} z_k^{\gamma k}$$

that will rationalize an observed project size y.[8]

Our results from this analysis are presented in Table 8. The analysis suggests that for $\hat{s} = 3.74$, the cost of delay required to rationalize the mean project (425 function points) at the research site is \$0.21 per function point per day; to rationalize the median project (108 function points) is \$2.22 per function point per day; and to rationalize the minimum project (4 function points) is \$671.68 per function point per day.

We examine the tradeoff between batching efficiencies and costs using the median project as an example. The median size project of 108 function points is not scale efficient. The average productivity for this project is 1.72 function points per hour. The average productivity for the scale efficient project size of 2,485 function points is 7.69 function points per hour. Thus, if the median project were scale efficient, there would be almost 4.5 times improvement in the use of labor hours to produce function points. On the other hand, to take advantage of scale economies, the software manager must wait 235 days to achieve efficient size versus 10 days to achieve the median size. This suggests that the manager must wait 24 times as long to batch up to the efficient scale size versus the median size. Therefore, the rational manager will not delay beyond the batch size for which the cost of delaying further exceeds the additional benefit from improved efficiency in the use of labor hours.

We conclude that the observed behavior of the software managers is generally consistent with the incentives reflected in the optimal tradeoff between the cost of batching delays and benefits of scale economies in the production of these projects. This implies that for the managers at our research site to have acted rationally, they must have perceived estimates of the cost of delay to be no less than \$0.21 per function point per day on average for the common enhancement requests.

6. Concluding Remarks

In this paper, we explore the relationship between project size and productivity for software enhancement projects in a field setting. Our results indicate the presence of significant scale economies in the software enhancement projects. We examine why the software managers at our research site do not take advantage of these scale economies. Our analysis of the tradeoff between the benefits of scale economies and opportunity costs of batching indicates that the behavior of the software managers can be rationalized given the extent of scale elasticities for these projects, as well as the performance penalties for delays due to batching of projects.

Our findings have important implications for software maintenance research. A primary implication is that researchers interested in software productivity should consider project size as an important influence on maintenance productivity. Another implication is that organizational incentives can discourage practices that improve productivity. There are several possible extensions to this study. We have examined software enhancement projects in a commercial, profit-oriented environment and expect that our findings would generalize to similar settings. It would be interesting to assess the tradeoff between the delay costs of batching and scale efficiencies in a different environment, such as a nonprofit-oriented environment like the government. In government environments, software maintenance is often contracted at lowest fixed annual price with less pressure to implement by certain dates. Under such an arrangement, incentives to bid the lowest amount could encourage use of batching to lower maintenance costs, and delay cost penalties to batch could be relatively low. Thus, we would expect to see greater exploitation of scale economies from batching in government environments. This conjecture could be empirically tested in future studies. Another possibility for further research is to examine the impact of different organizational incentives on the use of alternative software maintenance

parametric form here because the nonparametric DEA production function does not yield a closed form expression for the optimal project size y^*, and its estimation is computationally demanding. Analysis for a limited number of cases yielded results similar to those reported here.

[8] Note that, because c^* is linear in n, our results for c^* are robust across the portfolio of applications, if we assume that all applications have the same production function. On the other hand, because y^* is concave in n, y^* may be proportionately smaller for applications with a greater demand for enhancement.

Code, and Development Effort Prediction: a Software Science Validation," *IEEE Trans. on Software Engineering*, SE-9(6) (1983), 639–648.

Banker, R. D., "Maximum Likelihood, Consistency and Data Envelopment Analysis: A Statistical Foundation," *Management Sci.*, 39, 10 (1993), 1265–1273.

——, "Estimating Most Productive Scale Size Using Data Envelopment Analysis," *European J. Oper. Res.*, 17, 1 (1984), 35–44.

—— and H. Chang, "A Simulation Study of Hypotheses Tests for Differences in Efficiencies," *International J. Production Economics*, 39, 1 (1995), 37–54.

——, ——, and C. F. Kemerer, "Evidence on Economies of Scale in Software Development," *Information and Software Technology*, 36, 5 (1994), 275–282.

—— and C. F. Kemerer, "Performance Evaluation Metrics for Information Systems Development: A Principal-Agent Model," *Information Systems Res.*, 3, 4 (1992), 379–400.

—— and ——, "Scale Economies in New Software Development," *IEEE Trans. on Software Engineering*, 15, 10 (1989), 1199–1205.

—— and A. Maindiratta, "Nonparametric Analysis and Allocative Efficiencies in Production," *Econometrica*, 56, 6 (1988), 1315–1332.

——, A. Charnes, and W. W. Cooper, "Some Models for Estimating Technical and Scale Inefficiencies in DEA," *Management Sci.*, 30, 9 (1984), 1078–1092.

——, S. Datar, and C. F. Kemerer, "A Model to Evaluate Variables Impacting the Productivity of Software Maintenance Projects," *Management Sci.*, 37, 1 (1991), 1–18.

——, G. Davis, and S. A. Slaughter, "Software Development Practices, Software Complexity, and Software Maintenance Performance: A Field Study," forthcoming in *Management Sci.*

—— and S. A. Slaughter, "Project Size and Software Maintenance Productivity: Empirical Evidence on Economies of Scale in Software Maintenance," *Proc. Fifteenth International Conf. on Information Systems* (1994), 279–290.

Boehm, B. W., *Software Engineering Economics*, Prentice-Hall, Englewood Cliffs, NJ, 1981.

Branch, M. A., M. C. Jackson, M. C. Laviolette, and E. C. Frankel, "Software Maintenance Management," *Conf. on Software Maintenance* (1985), 62–68.

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

Byrnes, P. E., T. P. Frazier, and T. R. Gulledge, "Returns-to-Scale in Software Production: A Comparison of Approaches," in T. R. Gulledge and W. P. Hutz (Eds.), *Analytical Methods in Software Engineering Economics*, Springer-Verlag, New York, 1993.

Charnes, A., W. W. Cooper, and E. Rhodes, "Evaluating Program and Managerial Efficiency: An Application of Data Envelopment Analysis to Program Follow Through," *Management Sci.*, 27, 6 (1981), 668–697.

Conte, S., H. Dunsmore, and V. Shen, *Software Engineering Metrics and Models*, Benjamin Cummings, Reading, MA, 1986.

Curtis, B., "Substantiating Programmer Variability," *IEEE Proceedings*, 69, 7 (1981), p. 846.

Dekleva, S., "Delphi Study of Software Maintenance Problems," *Conf. on Software Maintenance* (1992), 10–17.

Eastwood, A., "Firm Fires Shots at Legacy Systems," *Computing Canada*, 19, 2 (1993), p. 17.

Fjeldstad, R. K. and W. T. Hamlen, "Application Program Maintenance Study—Report to our Respondents," in G. Parikh and H. Zvegintzov (Eds.), *Tutorial on Software Maintenance*, IEEE Computer Society Press, Silver Spring, MD, 1983.

Grady, R. B. and D. L. Caswell, *Software Metrics: Establishing a Company-Wide Program*, Prentice-Hall, Englewood Cliffs, NJ, 1987.

Hale, D. P. and D. A. Haworth, "Software Maintenance: a Profile of Past Empirical Research," *Proc. 4th Conf. on Software Maintenance* (1988), 236–240.

Hanna, M., "Maintenance Burden Begging for Remedy," *Software Magazine*, 13, 6 (1993), 53–63.

Hax, A. C. and D. Candea, *Production and Inventory Management*, Prentice-Hall, Englewood Cliffs, NJ, 1984.

Hoffman, T., "Timing is Right for COBOL Programmers," *Computerworld*, 30, 11 (1996), 10.

IFPUG, *Function Point Counting Practices Manual*, International Function Points User's Group, J. Sprouls (Ed.), AT&T, Piscataway, NJ, 1993.

Jeffrey, D. R. and M. J. Lawrence, "An Inter-Organisational Comparison of Programming Productivity," in *Proc. 4th International Conf. on Software Engineering* (1979), 369–377.

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

Kemerer, C. F., "Reliability of Function Points Measurement," *Comm. ACM*, 36 (February 1993), 85–97.

——, "Measurement of Software Development Productivity," unpublished Ph.D. Dissertation, UMI #8905252, Carnegie Mellon University, Pittsburgh, Pennsylvania, 1987.

Kriebel, C. and A. Raviv, "An Economics Approach to Modeling the Productivity of Computer Systems," *Management Sci.*, 26, 3 (1980), 297–311.

Lientz, B. P. and E. B. Swanson, *Software Maintenance Management*, Addison-Wesley, Reading, MA, 1980.

Littman, D. C., J. Pinto, S. Letovsky, and E. Soloway, "Mental Models and Software Maintenance," *J. Systems and Software*, 7 (1987), 341–355.

Martin, J. and C. McClure, *Software Maintenance: The Problem and its Solution*, Prentice-Hall, Englewood Cliffs, NJ, 1983.

McFarland, D., "COBOL Forges Ahead," *Computerworld*, 29, 23 (1995), p. 38.

McNeil, D. H., "Adopting a System Release Discipline," *Datamation*, January (1979), 110–117.

Moad, J., "Tight Budgets: It's How You Cut 'em," *Datamation*, 39, 9 (1993), 30–34.

Nosek, J. T. and P. Palvia, "Software Maintenance Management: Changes in the Last Decade," *J. Software Maintenance*, 2, 3 (1990), 157–174.

Oman, P. W., C. R. Cook, and M. Nanja, "Effects of Programming Experience in Debugging Semantic Errors," *J. Systems and Software*, 9 (1989), 192–207.

Panzar, J. C. and R. D. Willig, "Economies of Scope," *American Economic Rev.*, 71, 2 (1981), 268–272.

—— and ——, "Economies of Scale in Multi-output Production," *Quarterly J. Economics*, 91 (1977), 481–493.

Pennington, N., "Stimulus Structures and Mental Representations in Expert Comprehension of Computer Programs," *Cognitive Psychology*, 19 (1987), 295–341.

Perry, W. E., "The Best Measures for Measuring Data Processing Quality and Productivity," *Quality Assurance Institute Technical Report*, 1986.

Richmond, J., "Estimating the Efficiency of Production," *International Economic Rev.*, 15 (June 1974), 515–521.

Schneidewind, N. F., "The State of Software Maintenance," *IEEE Trans. on Software Engineering*, SE-13(3), March (1987), 303–310.

Skinner, W., "The Focused Factory," *Harvard Business Rev.*, 52, 3 (1974), 113–121.

Sutherland, J., "Business Objects in Corporate Information Systems," *ACM Computing Surveys*, 27, 2 (1995), 274–276.

Swanson, E. B., "The Dimensions of Maintenance," *Proc. Second International Conf. on Software Engineering*, San Francisco, CA, October 13–15 (1976), 492–496.

—— and C. M. Beath, *Maintaining Information Systems in Organizations*, John Wiley and Sons, New York, 1989.

Varian, H. R., *Microeconomic Analysis*, 2nd Edition, Norton, New York, 1992.

Vessey, I., "Toward a Theory of Computer Program Bugs: an Empirical Test," *International J. Man-Machine Studies*, 30, 3 (1989).

——, "On Program Development Effort and Productivity," *Information and Management*, 10 (1986), 255–266.

Walston, C. E. and C. P. Felix, "A Method of Programming Measurement and Estimation," *IBM Systems J.*, 16, 1 (1977), 54–73.