Effects of Process Maturity on Quality, Cycle Time, and Effort in Software Product Development

Donald E. Harter • Mayuram S. Krishnan • Sandra A. Slaughter
University of Michigan Business School, Ann Arbor, Michigan 48109-1234
University of Michigan Business School, Ann Arbor, Michigan 48109-1234
Graduate School of Industrial Administration, Carnegie Mellon University, Pittsburgh, Pennsylvania 15213
harter@umich.edu • mskrish@umich.edu • sandras@andrew.cmu.edu

The information technology (IT) industry is characterized by rapid innovation and intense competition. To survive, IT firms must develop high quality software products on time and at low cost. A key issue is whether high levels of quality can be achieved without adversely impacting cycle time and effort. Conventional beliefs hold that processes to improve software quality can be implemented only at the expense of longer cycle times and greater development effort. However, an alternate view is that quality improvement, faster cycle time, and effort reduction can be simultaneously attained by reducing defects and rework. In this study, we empirically investigate the relationship between process maturity, quality, cycle time, and effort for the development of 30 software products by a major IT firm. We find that higher levels of process maturity as assessed by the Software Engineering Institute's Capability Maturity Model™ are associated with higher product quality, but also with increases in development effort. However, our findings indicate that the reductions in cycle time and effort due to improved quality outweigh the increases from achieving higher levels of process maturity. Thus, the net effect of process maturity is reduced cycle time and development effort.

(*Software Process Improvement; Software Economics; Software Productivity; Software Quality; Software Costs; Software Cycle Time; Capability Maturity Model*)

1. Introduction

Over the past decades, effective deployment of computer software has emerged as one of the most important determinants of success in the business world. Firms are investing heavily in software as information technology (IT) infiltrates and plays a critical role in all aspects of the value chain. In consequence, the IT industry has experienced more than 500% growth worldwide over the past decade (*Wall Street Journal* 1997, Mowrey 1996). As the number of software development firms increases, competition intensifies. To survive, IT firms must accelerate the time to market for their software products. However, reduced cycle times cannot be achieved at the expense of low quality and high development costs.

In an effort to keep costs within budget and deliver quality products to customers, IT firms adopt various quality practices in their software development processes. However, IT firms may not consistently follow these practices because of the dynamic nature of the commercial software development environment. Customers frequently change their requirements after

0025-1909/00/4604/0451$05.00
1526-5501 electronic ISSN

product design has begun, but they expect the software to be delivered without delay. As a consequence, to satisfy customer needs under severe schedule pressure and yet control development costs, IT firms may deviate from disciplined practices in their development process and cut corners by shipping the products without adequate testing (Kemerer 1997). Such behavior can result in a higher number of defects at the customer site that delay the acceptance of the product, and the cost of fixing these defects can be significant. These dynamics suggest the importance of delivering software products on time without compromising quality and cost.

In manufacturing, reduction of product development cycle time has become the focal point of competition in many industries. The fast pace of technological change and the intensity of international competition place a premium on a firm's ability to respond quickly to customer demands. Many firms have adopted time-based competition strategies to reduce product development time and deliver higher quality products and services to their respective customers at lower cost. Under time-based competition, firms strive to streamline and constantly improve the reliability and capability of their manufacturing processes. A key premise underlying process improvement in manufacturing is the elimination of waste and rework in manufacturing activities by reducing product defects (Bockerstette and Shell 1993).

In software development, an important issue is whether process improvement pays off in terms of higher quality, reduced cycle time, and lower cost. Under the conventional paradigm, higher quality can be achieved only at the expense of increased development expenditures and longer cycle times. From this perspective, effort expended to improve software development processes varies with the level of quality attained. A typical view of software managers operating from this paradigm is: "I'd rather have it wrong than have it late. We can always fix it later." (Paulk et al. 1995, p. 4)

An alternative view from manufacturing is that quality, cost, and cycle time are complementary, i.e., improvements in quality directly relate to improved cycle time and productivity (Crosby 1979, Deming 1986, Nandakumar et al. 1993). This view is also espoused by advocates for software process improvement (Humphrey 1995). These advocates have provided frameworks for software firms to characterize the capability of their software development practices (El Emam and Goldenson 1996, Paulk et al. 1995). One of the widely adopted frameworks is the Capability Maturity Model™ (CMM™) developed by the Software Engineering Institute (SEI) at Carnegie Mellon University (Paulk et al. 1995). Based on the specific software practices adopted, the CMM classifies the software process into five maturity levels. Analogous to concepts of time-based competition in manufacturing, the basic premise of this framework is that improvements in cycle time, cost, and quality can be simultaneously attained by improving software process capability. These improvements are thought to arise from reduced defects and rework in a mature software development process. However, it is important to provide empirical evidence to substantiate these beliefs.

In this study, we empirically investigate the relationships between process maturity measured on the CMM maturity scale, product quality, development cycle time, and effort for 30 software products created by a major IT firm over a period of 12 years. Size and design complexity are reported as additional significant variables that explain quality in the delivered software products.

We find that improvements in process maturity lead to higher quality but also increased effort in our sample of software products. However, higher quality in turn leads to reduced cycle time and development effort in the software products. The net effect of improvement in process maturity on development cycle time and effort is negative. At the average values for process maturity and software quality, a 1% improvement in process maturity leads to a 0.32% net *reduction* in cycle time, and a 0.17% net *reduction* in development effort (taking into account the positive direct effects and the negative indirect effects through quality). These findings provide empirical support in the context of software production for theories of the cycle time and cost benefits of improved quality deriving from process improvement.

The rest of our paper is organized as follows. A brief review of the relevant literature is presented in §2. Section 3 contains the development of our model and theory. Section 4 provides details about the research site and data collection. Model estimation and results are presented in §5. In the final sections, we discuss the managerial implications of our results and provide directions for future research.

2. Prior Literature

The increasing dependence on software and the severe consequences of software failures have mandated IT firms to deliver higher quality products. In addition, with the increased competition in the global IT industry, attention has turned more recently to problems of economically delivering defect-free products to customers in a shortened development cycle time. The software quality literature has approached these problems from the perspective of predicting and preventing software defects. A primary focus has been the development of models to analyze and prevent defects and to predict the reliability of software products (e.g., Lyu 1996, Basili and Perricone 1984).

The cycle time and cost implications of product quality have been addressed in manufacturing research. Kaplan (1986), for example, discusses the impact of product quality on the direct costs of labor and materials. Nandakumar et al. (1993) argue that poor quality in manufacturing systems not only leads to higher defect rates but also affects product development time. Hence, they emphasize that it is important to include both direct costs and indirect opportunity costs of the decrease in demand and price resulting from delayed product delivery in the cost of quality analysis. Their analytical model captures both the direct and indirect (through delayed product development time) costs of poor quality, and indicates that costs of poor quality are a convex and increasing function of defects.

In software engineering research, the life-cycle cost impact of quality in software products has been examined by Krishnan et al. (2000). They find that improved conformance quality in system software products leads to significant improvement in life-cycle productivity, and that factors such as deployment of resources in the initial stages of product design drive increased quality. However, this study does not consider the cycle time impact of quality improvement.

New technologies are emerging at an increasing rate in the software industry, rapidly rendering the existing products obsolete. As a consequence, product life-cycles are getting shorter, and cycle time reduction is imperative for software firms. Although a number of commercial computer-aided software engineering (CASE) tools have been introduced to facilitate cycle time reduction, from a software engineering research perspective the quality impacts of delays in product development are not well understood. The effect of development cycle time on total effort has been indirectly addressed in software cost models through the effect of schedule pressure in projects (Boehm 1981, Abdel-Hamid and Madnick 1991).

The streams of research on software cost, quality, and cycle time have often considered one factor in the absence of one or more of the other factors. That is, many cost models ignore the quality or time to market of the delivered product, and quality models often ignore cycle time or the cost incurred in product development. In the absence of an understanding of the interrelationships between these factors, software managers eliminate important process steps such as front-end inspections in the erroneous belief that they are saving time. However, this can result in major delays in product development resulting from the additional time required to fix errors detected during customer acceptance testing (Humphrey 1995). With the focus in the IT industry on simultaneously reducing product cycle time, cost, and defects, it is important to examine the interrelationships among *all* these factors.

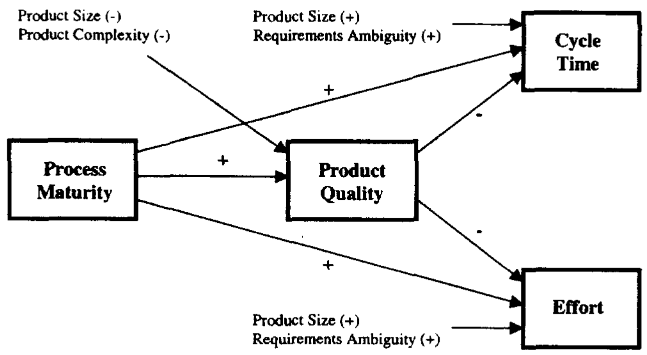
The quality, cycle time, and cost problems associated with software are partly a result of the nature of software and partly of the practices adopted in software development (Brooks 1995). As noted earlier, software process is viewed as an important determinant of product quality, effort, and cycle time. The importance of process maturity has been demonstrated in manufacturing contexts. For example, Bohn (1995) has reported a field study that provides evidence for the significance of process maturity and controlling process variability in enhancing process yield and product quality. Even if a manufacturer has a low-quality supplier, it has been shown that the cost impact of higher defects can be controlled by improvement in the manufacturing process (Tagaras and Lee 1996).

The disciplined methods and practices described in the CMM and various other software process models are believed to provide various benefits to software firms, such as improved quality and reduced cycle time and cost. Informal studies such as anecdotal case reports and surveys (Herbsleb et al. 1997) suggest positive impacts of software process improvement. In light of the significant investment required to improve software processes, it is important to rigorously study and quantify empirically the benefits of process maturity for quality, cycle time, and effort.

3. Research Models and Hypotheses

The conceptual framework for our study (Figure 1) integrates three models that interrelate process maturity, quality, cycle time, and effort for software product development. The first model relates the maturity of the development process to product quality, controlling for the size and design complexity of the software products. In the second and third models, cycle time and development effort are specified as a function of product quality and process maturity, controlling for the size of the product and the ambiguity of user requirements. A detailed explanation of each model and our hypotheses follow.

Figure 1 Conceptual Model



3.1. Product Quality

In our first model (1), we relate process maturity to product quality, controlling for the size and design complexity of the software product:

$$Product\text{-}Quality = \text{Function}\,(Process\text{-}Maturity,\; Product\text{-}Size,\; Product\text{-}Design\text{-}Complexity). \quad (1)$$

In the life-cycle perspective of software development, there are several phases of testing that are conducted to discern the quality of software products. Unit tests are conducted separately for each module in the software product, with the goal of detecting and removing syntactical errors. System tests examine the functioning of the product as a whole to determine whether discrete modules will function together as planned and whether discrepancies exist between the way the product actually works and the way it was designed. Acceptance tests are conducted by the customers of the product to ensure that it meets their specifications.

Our definition of product quality is based on errors uncovered in system and acceptance testing. These errors are deviations from the initial customer specifications and do not include enhancements. Specifically, we define *product-quality* as the lines of source code in the software product divided by the number of errors reported during system testing and customer acceptance testing. This definition reflects the size of the software per defect, with larger values implying a lower frequency of errors per unit of software, i.e., higher product quality.

Process-maturity measured on the CMM maturity scale reflects the firm's level of investment to improve software process capabilities. The CMM framework includes 18 key process areas such as quality assurance, configuration management, defect prevention, peer review, and training (Paulk et al. 1995). In all these process areas, the CMM identifies various disciplined software development practices. A software process is assigned the highest maturity level if the practices in the 18 key process areas of the CMM are adopted. As defined in the CMM, an effective and mature software process must include the interactions among employee skill and morale, tools, and methods used in all the tasks and clearly defined metrics and methods of products and process. The CMM practices aid in reducing defect injection and in early identification of defects. As a consequence, the number of errors in system and acceptance testing will be lower for software products developed with a mature process. This implies:

HYPOTHESIS 1 (PROCESS MATURITY AND PRODUCT QUALITY). *Higher levels of process-maturity lead to higher product-quality in software products.*

We control in this model for the effect of product size (*product-size*) and product design complexity (*product-design-complexity*). *Product-size* (defined in terms of thousand lines of source code in the product) is a factor that we expect to be inversely related to product quality. Prior studies of software productivity and quality have identified product size measured in lines of code as a primary determinant of product defects (Basili and Musa 1991). Because of the volume of code and increased product functionality, larger products provide more opportunity to introduce errors. In addition, larger products may include more modules and interactions between modules, thus further increasing the possibility of defects. *Product-design-complexity* in our model captures three important dimensions of complexity: domain, data, and decision complexity. Domain complexity reflects the difficulty of the functionality, specifically the algorithms and calculations that will be accomplished in a product. Data complexity refers to the complexity of the data structures and relationships in the software product. Decision complexity refers to the complexity of decision paths and structures in the software product. The more complex the product design on these dimensions, the higher the likelihood that errors will be injected in development and uncovered in system and acceptance testing when live data rather than test cases are used (Munson 1996).

3.2. Cycle Time

In our second model (2), we examine the link between process maturity, quality, and development cycle time in software products, controlling for the effects of product size and the ambiguity of user requirements.

$$\begin{aligned}\textit{Cycle-Time} = \text{Function}\ (&\textit{Process-Maturity},\\ &\textit{Product-Quality},\ \textit{Product-Size},\\ &\textit{Requirements-Ambiguity}).\end{aligned} \tag{2}$$

Cycle-time is the time to develop the product, i.e., the elapsed time in days from the start of design on the product until its final acceptance by the customer. The relationship between cycle time, process maturity, and quality has been viewed from two different perspectives. One view is that cycle time must be traded off against improvements in quality. That is, increased investment in process improvement requires additional time for testing, code inspections, etc., that add to the elapsed time required to develop the product and delay its introduction.

In contrast, an alternate view is that improvements in quality can lead to an overall reduction in cycle time. The rationale behind this is that by adopting disciplined practices as specified in the CMM framework and improving process maturity, the time spent on preventing and uncovering defects early in the development can be more than recovered by avoiding rework to correct defects detected at the later stages of product development (Jones 1997). In the context of software production, several studies have indicated that more time is needed to fix defects uncovered at later stages in software development than in earlier stages (e.g., Swanson et al. 1991, Abdel-Hamid and Madnick 1991). Following this view, we hypothesize that increased investment in process improvement activities (as reflected in process maturity) can increase cycle time. However, higher process maturity leads to higher product quality, and as products exhibit fewer defects, there is less rework, thereby reducing cycle time. Therefore:

HYPOTHESIS 2 (PROCESS MATURITY AND CYCLE TIME). *Higher levels of process-maturity are associated with increased cycle-time in software products.*

HYPOTHESIS 3 (PRODUCT QUALITY AND CYCLE TIME). *Higher product-quality is associated with lower cycle-time in software products.*

In the cycle time model, we control for the effects of product size (*product-size*) and the ambiguity of user requirements (*requirements-ambiguity*). We expect that product size will be positively related to cycle time, *ceteris paribus,* because larger products involve more work and should take longer to develop in the absence of variations in schedule pressure. Products for which user requirements are not clearly specified in design should also take longer to develop because additional time is required to elicit precise requirements from the users in a series of meetings during the development process (Gopal et al. forthcoming). It may be argued that the average team size could determine the cycle time. Although our research site had predefined rules for budgeting personnel in each phase of development, because of differences in specific circumstances the actual loading profile of individual projects often deviated from the established norm. This variation in actual loading profile makes average team size a less meaningful variable at our research site.

3.3. Development Effort

In our third model (3) we examine the link between process maturity, quality, and development effort in software products, controlling for the effects of product size and the ambiguity of user requirements:

$$\begin{aligned}&\mathit{Effort} = \text{Function}\,(\mathit{Process\text{-}Maturity},\\ &\quad \mathit{Product\text{-}Quality}, \mathit{Product\text{-}Size},\\ &\quad \mathit{Requirements\text{-}Ambiguity}). \qquad (3)\end{aligned}$$

Effort refers to the person-months required to develop the software product. The relationship between development effort, process maturity, and product quality has also been discussed from two different viewpoints. The conventional school of thought asserts that effort and quality improvement must be counterbalanced. That is, increased effort is required to follow disciplined practices, use quality tools, and conduct rigorous testing and code reviews in attaining higher quality levels. A different school of thought views development effort and quality improvement as complementary. This stems from the argument that defects detected at the later stages of product development lead to substantial rework, and the cost of this rework could be significantly higher than investments in quality improvement at the early stages of product development (Boehm 1981). This implies that there is an inverse relationship between quality and effort. Along these lines, we expect that improved quality is associated with lower development effort. However, higher levels of process maturity would directly increase development effort as a result of the additional activities for process improvement. Thus:

HYPOTHESIS 4 (PROCESS MATURITY AND DEVELOPMENT EFFORT. *Higher levels of process-maturity are associated with increased development-effort in software products.*

HYPOTHESIS 5 (PRODUCT QUALITY AND DEVELOPMENT EFFORT). *Higher product-quality is associated with lower development-effort in software products.*

We control in this model for the effect of product size (*product-size*) and the ambiguity of user requirements (*requirements-ambiguity*). Software size has been identified as the most significant factor that explains development effort (Kemerer 1997). Accordingly, we expect that product size will be positively related to development effort. Products for which user requirements are ambiguous in design should also take more effort to develop as additional work is required to elicit precise requirements from the users in development. The importance of effective requirements determination for software productivity has been emphasized by Jones (1996) and others.

4. Research Design and Methodology

4.1. Research Setting

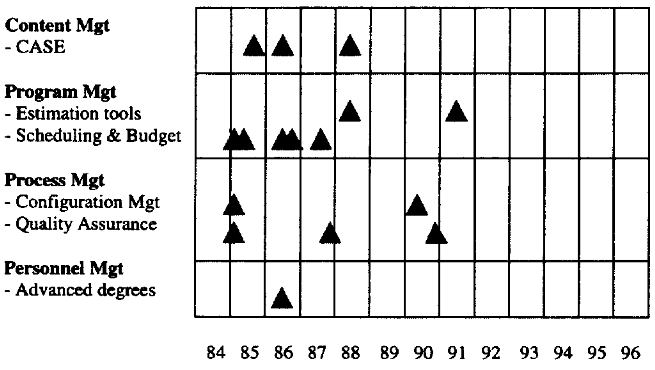
We examine data collected on 30 software products created by the systems integration division of a $1 billion per year IT firm. The products comprise approximately 3.3 million lines of COBOL code and were created from 1984 to 1996. The products were developed as part of a $230 million effort to build a material requirements planning (MRP) information system to manage spare parts acquisition. All of the MRP software is designed to satisfy a stringent five-second response time for all queries, regardless of geographic location or complexity of retrieval.

This IT firm provides an interesting research site to study software development because the firm focused on improvements to development processes and quality during the time frame in which the products were developed, and advanced in process maturity. A contractual arrangement using incentive targets motivated the IT firm to continually search for techniques to improve the process and reduce cycle time and costs. Costs exceeding a fixed ceiling were borne by the firm, but savings resulting from process improvement were shared jointly by the firm and customer. The risk of cost overruns and the opportunity for increased profits provided significant motivation for improving process capabilities. Incremental process improvements included adoption of advanced development tools including CASE tools, integration of documentation with CASE, and automated configuration management tools. In addition, techniques such as Pareto Analysis were adopted to detect, manage, and reduce software errors. Human resource practices were modified by changing recruitment policies to encourage improved skills and qualifications in IT personnel. Figure 2 summarizes the process improvements adopted by the IT firm.

4.2. Data Collection Methods

Process improvement data were collected by external divisions and by government agencies to provide independent assessments of the IT development firm's software development processes. Government auditors and senior corporate personnel in divisions outside of systems integration performed five software process maturity assessments during the 12-year period. These independent groups used the SEI's CMM (Paulk et al. 1995) to assess the maturity of software development and supporting activities.

Figure 2 Process Improvements at the Research Site



All other data were collected by the IT development firm and were audited by the customer to ensure accuracy and accountability. Within the IT firm, the departments responsible for data collection included configuration management, program control, and engineering. The researchers extracted data used in this study from the electronic and paper files maintained by these divisions and departments.

Configuration management (CM) maintained a database on software errors identified during the IT firm's system level test and during the customer's acceptance test. An independent test team within the IT firm (different from the development team) developed a formal test plan to test the functionality of the system based on the product requirements. Upon completion of development, the software was migrated to the independent test team, where the product was tested using the detailed test plan procedures. This test team recorded development errors via the Software Problem Report. Upon completion of development testing and resolution of all errors found, the software was formally migrated to the customer test region. The customer's acceptance test team used the test plan to test the functionality of the system, performed stress testing with larger data bases and user populations, and used the Software Problem Report to document acceptance errors. Both test teams followed a comprehensive and consistent approach to software testing. All errors were fixed regardless of the severity of the error. The IT firm's quality assurance department and an independent customer audit team reviewed both types of error reports to ensure accuracy and completeness.

The IT firm's program control department was responsible for maintaining audited records of cycle time and effort. Cycle time data were stored using the Artemis© scheduling system. Effort data were tracked by the corporate time reporting system. The effort data were entered into the standard corporate payroll system and summarized by software development product. The customer audited both schedule and effort data to ensure accuracy.

Engineering was responsible for estimating product schedules and costs based on the user requirements specified in design in order to allocate resources and manage the development process. Domain, data, and decision complexity are three variables that were critical to the estimation process and were principal inputs to the Software Productivity Quality and Reliability (SPQR20©) methodology for projecting schedules and costs. Based on user requirements specification documents, engineering estimated the domain, decision, and data complexity of the design in order to use SPQR20© to develop initial product development estimates. Engineering also evaluated the ambiguity of user requirements to assess the potential effect of unclear or incomplete specifications. Products that implemented familiar functionality tended to have low requirements ambiguity; products that implemented new policies and procedures that had never before been used by the customer tended to have high requirements ambiguity. Engineering estimated the level of requirements ambiguity based on historical understanding of the processes, the relative newness of policies, and the clarity with which the specification described them.

4.3. Construct Measurement

We operationalize the variables used in our models as follows. *Process maturity* reflects the level of discipline and sophistication of the software development and supporting processes. We measure process maturity using the SEI's CMM level of maturity. Maturity levels were associated with a software product based on the maturity level of the IT firm at the beginning of a product's design. The maturity level of a product that benefited from process improvements later in the product's life-cycle stages (e.g., coding stage) was assigned a commensurate increase in maturity level. *Product quality* assesses the total defects in the product prior to customer release. We measure this variable as the number of lines of source code in the product divided by the sum of the defects found in system and acceptance testing. The inverse of this measure, i.e., defect density, has been used in many prior quality studies (Basili and Musa 1991, Fenton and Pfleeger 1997).

Cycle time measures the number of calendar days elapsed from the first day of design to final customer acceptance of the product. Life-cycle stages included in cycle time are high-level design, detailed design, coding, unit testing, system-level test, and customer acceptance test. *Development effort* is the total number of person-months logged by the development team in all the stages of product development, starting from initial design through final product acceptance testing. Our measures for development cycle time and effort are consistent with those used in prior research of software engineering (e.g., Boehm 1981).

Product size is measured in terms of KLOC (thousand lines of source code). Lines of code is a primary metric for assessing product size in many empirical software productivity studies (e.g., Boehm 1981, Conte et al. 1986). The main shortcoming of this measure stems from the inaccurate and inconsistent definition of "a line of code" across various programming languages (Jones 1986). However, in this study, this problem is not salient because all the products were developed in a single language (COBOL), and the lines of code were counted in a consistent manner using the same tool.

As noted earlier, the *product design complexity* variable in our analysis captures three important underlying dimensions of complexity: *domain, data,* and *decision complexity* (Jones 1996). *Domain complexity* measures the level of functional complexity as determined by engineering from the user requirements specification. Although the products are part of the same application domain (i.e., MRP), the development difficulty of these products may significantly differ because of differences in the tasks implemented in the products. *Data complexity* measures the anticipated level of difficulty in developing the system because of complicated data structures and data base relationships. *Decision complexity* captures the degree of difficulty in the decision paths and structures within the product. All three complexity constructs are assessed subjectively on a scale of 1 to 5 (low to high). The complexity constructs exhibited high intercorrelation. An exploratory factor analysis using the principal components method revealed a single factor that explained 60% of the variance among the three variables. For ease of interpretation, scores for the *product design complexity* variable were computed by taking an average of the three complexity constructs.

Requirements-ambiguity is a subjective, ordinal measure that assesses the degree to which user requirements are clearly defined for a product. It is measured

Table 1 Summary Statistics

Variable	Mean	Std Deviation	Median	Minimum	Maximum
Product Quality	659.53	531.02	400.77	105.16	2024.00
Cycle Time	759.27	457.30	663.00	62.00	1692.00
Development Effort	196.88	211.53	130.89	3.38	829.78
Process Maturity	2.02	0.64	2.00	1.00	3.00
Product Size	109.82	130.79	78.02	4.05	605.68
Product Design Complexity	1.95	0.41	1.95	1.23	2.85
Requirements Ambiguity	2.68	1.08	2.93	1.00	5.00

Table 2 Correlation Matrix

	ln(Product Quality)	ln(Cycle Time)	ln(Effort)	ln(Process Maturity)	ln(Product Size)	ln(Product Design Complexity)	ln(Reqmts Ambiguity)
ln(Product Quality)	1.000						
ln(Cycle Time)	−0.342	1.000					
	(0.065)						
ln(Effort)	−0.220	0.861	1.000				
	(0.243)	(0.000)					
ln(Process Maturity)	0.454	−0.124	−0.073	1.000			
	(0.012)	(0.516)	(0.702)				
ln(Product Size)	0.036	0.624	0.843	−0.115	1.000		
	(0.850)	(0.000)	(0.000)	(0.545)			
ln(Product Design Complexity)	−0.131	0.459	0.533	0.363	0.480	1.000	
	(0.489)	(0.011)	(0.002)	(0.049)	(0.007)		
ln(Reqmts Ambiguity)	−0.043	−0.043	0.457	0.141	0.541	0.687	1.000
	(0.823)	(0.082)	(0.011)	(0.458)	(0.002)	(0.000)	

Note. Pearson correlation coefficients with *p* values in parentheses.

on a scale of 1 to 5, with 5 reflecting the highest degree of ambiguity in user requirements specification, and 1 indicating that user requirements are clear and unambiguous. Descriptive statistics for all variables are in Table 1, and a correlation matrix is in Table 2.

5. Analysis and Results

A linear specification of the three models implies that the effects of process maturity, product size, product design complexity, and ambiguity of user requirements on product quality, cycle time, and development effort are additively separable and linear. However, in the case of software development, the effects of size and other factors are not linear. Both economies and diseconomies of scale (i.e., size) of the software product have been observed by researchers for cost (Banker and Slaughter 1997, Banker et al. 1994) and quality (Newfelder 1993). In addition, for the data sample in this study, statistical tests rejected standard assumptions of the linear model such as normality of error terms. Thus, a generic multiplicative specification for our models is adopted through log transformation of the variables (Kmenta 1986). The specification test for nonnested models (the *J*-test) supported the log-linear models specified below over linear models for each of our equations (Davidson and MacKinnon 1995).

$$\ln(\mathit{Product\text{-}Quality}) = \beta_{01} + \beta_{11} * \ln(\mathit{Process\text{-}Maturity}) + \beta_{21} * \ln(\mathit{Product\text{-}Size}) + \beta_{31} * \ln(\mathit{Product\text{-}Design\text{-}Complexity}) + \epsilon_{01} \quad (1)$$

$$\ln(\mathit{Cycle\text{-}Time}) = \beta_{02} + \beta_{12} * \ln(\mathit{Product\text{-}Quality})$$

Table 3 Model 1 Parameter Estimates (n = 30)

Variable	Parameter	OLS Estimate	SURE Estimate	2SLS Estimate	Rank Regression Estimate
Intercept	β_{01}	5.597	5.602	5.597	5.611
	s.e	0.464	0.470	0.464	3.814
	t	12.059	11.930	12.059	1.471
	p	0.000	0.000	0.000	0.076
ln(Process-Maturity)	β_{11}	1.589	1.594	1.589	0.733
	s.e	0.386	0.391	0.386	0.179
	t	4.116	4.081	4.116	4.095
	p	0.000	0.000	0.000	0.000
ln(Product-Size)	β_{21}	0.234	0.236	0.234	0.369
	s.e	0.108	0.109	0.108	0.177
	t	2.160	2.152	2.160	2.080
	p	0.020	0.017	0.020	0.024
ln(Product-Design-Complexity)	β_{31}	−2.111	−2.137	−2.111	−0.464
	s.e	0.712	0.720	0.712	0.195
	t	−2.963	−2.969	−2.963	−2.378
	p	0.003	0.002	0.003	0.012
R^2		0.412	0.413	0.412	0.399
R^2 (adj)		0.345		0.345	0.330
F Model	$F_{3,26}$	6.080	6.010	6.080	5.750
	p	0.003	0.001	0.003	0.004

Note. (one-tailed p values)
ln(*Product-Quality*) = β_{01} + β_{11}* ln(*Process-Maturity*) + β_{21}* ln(*Product-Size*) + β_{31}* ln(*Product-Design-Complexity*) + ϵ_{01}.

$$+ \beta_{22} * \ln(\textit{Process-Maturity})$$
$$+ \beta_{32} * \ln(\textit{Product-Size})$$
$$+ \beta_{42} * \ln(\textit{Requirements-Ambiguity}) + \epsilon_{02} \qquad (2)$$
$$\ln(\textit{Development-Effort}) = \beta_{03}$$
$$+ \beta_{13} * \ln(\textit{Product-Quality})$$
$$+ \beta_{23} * \ln(\textit{Process-Maturity})$$
$$+ \beta_{33} * \ln(\textit{Product-Size})$$
$$+ \beta_{43} * \ln(\textit{Requirements-Ambiguity}) + \epsilon_{03} \qquad (3)$$

The parameters of the individual equations were initially estimated using ordinary least squares (OLS) (column three in Tables 3 through 5). Because all the products in our sample are from the same company, it may be possible that the error terms are correlated as a result of a common effect. Hence, we also estimated the seemingly unrelated regression (SURE) parameters using a feasible generalized least squares (FGLS) procedure that allows for correlation of disturbances across equations (Greene 1997). The FGLS estimates were very similar in sign, magnitude, and significance to the OLS estimates (column four in Tables 3 through 5), indicating the absence of any correlation across the error terms.

It may be argued that quality, cycle time, and effort are codetermined. We checked for endogeneity of quality, cycle time, and effort in our models using the Hausman specification test (Hausman 1978) and the Durbin-Wu-Hausman test (Davidson and MacKinnon 1993). Although these tests did not suggest endogeneity, in small samples the precision of these tests is not certain. We thus estimated a two-stage least squares (2SLS) model to correct for possible inconsistency in the OLS estimators. The 2SLS estimates of the parameters for all models (column five in Tables 3 through 5) were very similar in sign, significance, and magnitude to the OLS estimates.

Table 4 Model 2 Parameter Estimates (n = 30)

Variable	Parameter	OLS Estimate	SURE Estimate	2SLS Estimate	Rank Regression Estimate
Intercept	β_{02}	7.399	7.660	8.553	7.999
	s.e	0.912	0.904	1.847	4.022
	t	8.116	8.470	4.631	1.989
	p	0.000	0.000	0.000	0.058
ln(Product-Quality)	β_{12}	−0.454	−0.503	−0.674	−0.395
	s.e	0.155	0.154	0.343	0.167
	t	−2.920	−3.267	−1.966	−2.369
	p	0.004	0.001	0.030	0.013
ln(Process-Maturity)	β_{22}	0.403	0.461	0.655	0.239
	s.e	0.360	0.357	0.510	0.168
	t	1.120	1.292	1.285	1.421
	p	0.137	0.100	0.106	0.084
ln(Product-Size)	β_{32}	0.424	0.432	0.453	0.699
	s.e	0.099	0.098	0.110	0.150
	t	4.285	4.397	4.110	4.670
	p	0.000	0.000	0.000	0.000
ln(Reqmts-Ambiguity)	β_{42}	−0.170	−0.194	−0.249	−0.060
	s.e	0.259	0.257	0.291	0.150
	t	−0.656	−0.756	−0.858	−0.399
	p	0.259	0.226	0.200	0.347
R^2		0.546	0.546	0.510	0.498
R^2 (adj)		0.474		0.431	0.418
F Model	$F_{4,25}$	7.520	8.130	5.960	6.210
	p	0.000	0.000	0.002	0.001

Note. (one-tailed p values).
ln(*Cycle-Time*) = β_{02} + β_{12}* ln(*Product-Quality*) + β_{22}* ln(*Process-Maturity*) + β_{32}* ln(*Product-Size*) + β_{42}* ln(*Requirements-Ambiguity*) + ϵ_{02}.

As a robustness check, we estimated a rank regression model for each equation (Iman and Conover 1979). The parameter estimates from the rank regressions (column six in Tables 3 through 5) are similar in sign, magnitude, and significance to those from the OLS regressions, suggesting the robustness of the OLS estimates.

Thus, we used the OLS estimates for the interpretation of our results. Standard assumptions of the OLS estimators were tested. The assumption of normality is not rejected for any of the models at the 5% significance level using the Shapiro-Wilk test (Shapiro and Wilk 1965). The presence of heteroskedasticity in all the models was tested using White's (1980) test, and no evidence of it was found. The effect of multicollinearity was examined using conditions specified in Belsley et al. (1980). The condition index for all models is less than 30, within the acceptable limit. We did not detect any influential outliers in the equations using Cook's distance (Cook and Weisberg 1982) and the guidelines specified by Belsley et al. (1980). Because the products were developed over a period of 12 years, we tested for serial correlation between products using both the Durbin-Watson test (Durbin and Watson 1971) and the Breusch-Godfrey test (Breusch and Godfrey 1981) but did not find any evidence of it. The calculated values of all four models' F-statistics exceeded the critical values at the 5% significance level.

6. Discussion

6.1. Product-Quality Model

In model one, we find as expected that higher levels of process maturity are associated with higher product

Table 5 Model 3 Parameter Estimates ($n = 30$)

Variable	Parameter	OLS Estimate	SURE Estimate	2SLS Estimate	Rank Regression Estimate
Intercept	β_{03}	4.280	4.479	5.065	2.274
	s.e	1.049	1.041	2.073	2.538
	t	4.081	4.302	2.433	0.896
	p	0.000	0.000	0.011	0.190
ln(Product-Quality)	β_{13}	−0.611	−0.649	−0.760	−0.223
	s.e	0.179	0.177	0.385	0.105
	t	−3.416	−3.657	−1.977	−2.119
	p	0.001	0.000	0.029	0.022
ln(Process-Maturity)	β_{23}	0.799	0.844	0.971	0.211
	s.e	0.414	0.411	0.572	0.106
	t	1.932	2.055	1.697	1.986
	p	0.033	0.021	0.051	0.029
ln(Product-Size)	β_{33}	0.954	0.960	0.973	0.930
	s.e	0.114	0.113	0.124	0.094
	t	8.375	8.486	7.869	9.839
	p	0.000	0.000	0.000	0.000
ln(Reqmts-Ambiguity)	β_{43}	−0.235	−0.253	−0.289	−0.064
	s.e	0.298	0.296	0.326	0.095
	t	−0.787	−0.856	−0.885	−0.681
	p	0.220	0.197	0.193	0.251
R^2		0.803	0.803	0.798	0.800
R^2 (adj)		0.772		0.766	0.768
F Model	$F_{4,25}$	25.530	26.260	22.970	25.040
	p	0.000	0.000	0.000	0.000

Note. (one-tailed p values)
ln(*Development-Effort*) = β_{03} + β_{13}* ln(*Product-Quality*) + β_{23}* ln(*Process-Maturity*) + β_{33}* ln(*Product-Size*) + β_{43}* ln(*Requirements-Ambiguity*) + ϵ_{03}.

quality (i.e., fewer defects in system and acceptance testing, $\beta_{11} = 1.589$, $p < 0.001$). The value of the coefficient for process maturity implies that, holding the other variables in the equation constant at their mean levels, a 1% improvement in process maturity is associated with a 1.589% *increase* in product quality. The value of the process maturity coefficient also suggests that the quality benefits increase with higher levels of process maturity. As noted earlier, if the development process is not disciplined and mature, there will be deviations from the standard process such as absence of requirements tracking, lack of rigorous code reviews and inspections, and myopic design choices. As a consequence, both design and coding errors will be injected during the development process.

Our results indicate that larger products exhibit higher quality. Although larger products may have more defects, in our sample the rate of increase in defects resulting from size is lower for larger products. Hence the coefficient of size in our model is positive ($\beta_{21} = 0.234$, $p = 0.020$). Also as we expected, products with a more complex design have lower quality.

6.2. Cycle-Time Model

Our findings in this model indicate that product quality is significantly and negatively associated with cycle time ($\beta_{12} = -0.454$, $p = 0.004$). That is, higher quality products exhibit significant reductions in cycle time. The value of the coefficient for product quality implies that, holding the other variables in the equation constant at their mean levels, a 1% improvement in product quality is associated with a 0.454% *decrease* in cycle time. Because the model specification is multiplicative, the value for the coefficient of product quality implies that, at higher levels of quality, the payoff in development cycle time savings decreases. Our results also indicate that controlling for product size and requirements ambiguity, the direct effect of process maturity on cycle time is positive. However, in our sample this effect is not statistically significant at the 5% level ($\beta_{22} = 0.403$, $p = 0.137$).

An interesting finding in our model is that the net effect of process maturity on cycle time is negative. Although the direct effect of process maturity on cycle time is positive, when we include the positive effect on quality and the consequent reduction in cycle time, the net marginal effect of process maturity on cycle time is negative (-0.318). As expected, we find that larger products are associated with longer cycle times. The coefficient for requirements ambiguity is not statistically significant in this model.

6.3. Development-Effort Model

Similar to our findings in the cycle time model, our results in the effort model indicate that the direct effect of process maturity on development effort is positive ($\beta_{23} = 0.799$, $p = 0.033$). This effect is statistically significant, indicating that higher development effort is associated with higher levels of process maturity. We find that the quality of the product is significantly and negatively associated with effort ($\beta_{13} = -0.611$, $p = 0.001$). That is, higher quality products exhibit lower development effort. The value of the coefficient for product quality in our multiplicative specification implies that, holding the other variables in the equation constant at their mean levels, a 1% improvement in quality is associated with a 0.611% *decrease* in development effort. Also, the value for the coefficient of product quality implies that, at higher levels of quality, the payoff in development cost savings decreases. Our results on the relationship between development cost and product quality are consistent with findings from the study of life-cycle cost and quality by Krishnan (1996).

Similar to the cycle time model, we find that the net effect of process maturity on effort is negative. Although the direct effect of process maturity on development effort is positive, when we include the consequent reduction in effort because of improved quality, the net effect is negative (-0.175). As expected, we find that larger products are associated with greater effort. The coefficient for requirements ambiguity is not statistically significant in this model.

6.4. Marginal Analysis

The value of the coefficient for process maturity in the product quality model suggests that the marginal benefits from improved process maturity increase with higher levels of process maturity (Figure 3). This implies that from the perspective of quality improvement, it is beneficial to advance to the higher levels of process maturity, all other things being equal. It should be noted, however, that this relationship is valid only within the range of process maturity present within our data set. We also observe that the values of the coefficients for product quality in the development effort and cycle time models imply that there may exist a threshold quality level beyond which any process improvements would not be justified in terms of development effort and cycle time savings. As shown in Figures 4 and 5, the marginal benefits of quality improvement for cycle time and effort decrease with higher levels of quality.

To illustrate the application of our models from the perspective of justifying investments in process maturity, we compare the predicted effect of an improvement in process maturity on quality, cycle time and effort. Based on our sample, the predicted benefit of

Figure 3 **Marginal Effects of Process Maturity on Product Quality**

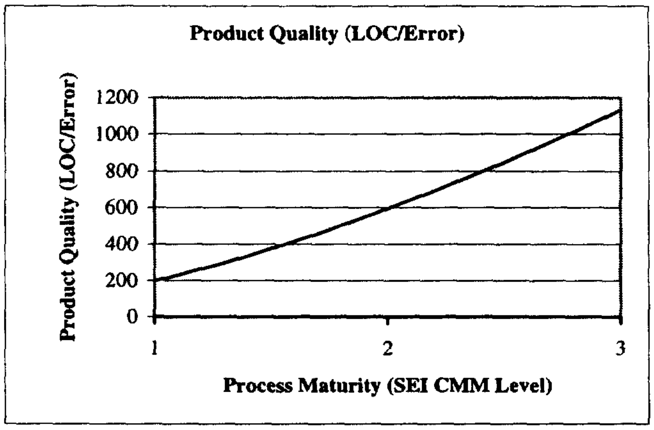
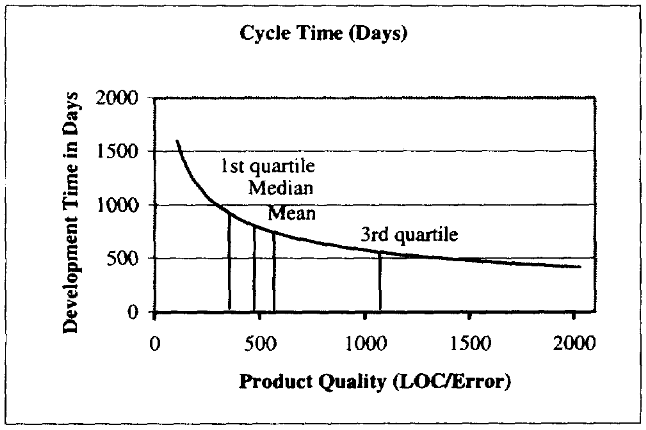


Figure 4 Marginal Effects of Product Quality on Cycle Time



increasing from CMM level 1 to level 2 for the average product is a reduction in cycle time of 183 calendar days and in development effort of 23 person-months (Table 6). By comparison, increasing from CMM level 2 to level 3 would yield an additional 90-calendar-days reduction in cycle time, and a 12-person-months reduction in development effort.

7. Conclusion

We have studied empirically the relationships between process maturity, product quality, cycle time, and development effort. Our analysis suggests that higher levels of process maturity as assessed by the SEI's CMM are associated with significantly higher

Figure 5 Marginal Effects of Product Quality on Development Effort

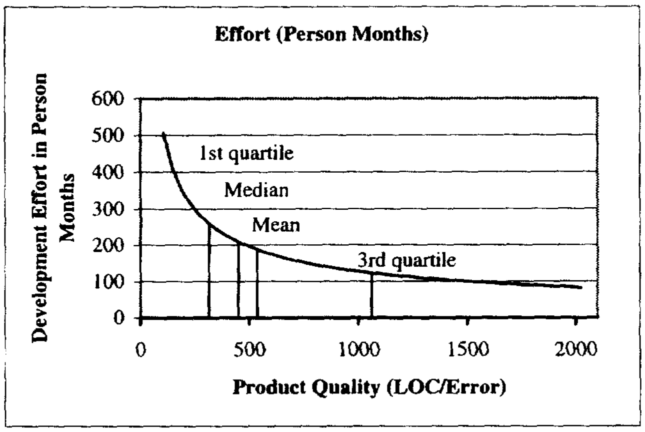


Table 6 Estimated Benefits of Process Maturity

SEI CMM Maturity Level	Quality (lines of code/error)	Cycle Time (calendar days)	Effort (person-months)
1	195.96	926.92	202.41
2	593.21	743.31	179.32
3	1133.97	653.26	167.05

product quality, but also with increases in cycle time and development effort. However, the marginal reductions in cycle time and effort resulting from improved quality outweigh the marginal increases from achieving higher levels of process maturity. Thus, the net marginal effect of process maturity is reduced cycle time and development effort in our sample of software products.

It should be noted that the relationships between process maturity, quality, cycle time, and development effort are valid only in the ranges observed in this application domain (custom software development of an algorithmically intense system in a COBOL, mainframe development environment). It is possible that the effort and cycle time incurred to achieve considerably higher quality levels in a mission-critical software product could be significantly higher. This is because additional time and resources must be deployed for rigorous testing, inspection or checking at every stage of product development and for several rounds of regression testing of the product. Hence, the investments required to achieve extremely high levels of product quality may not be recovered by reductions in cycle time and effort.

We expect that our models, methodology, and approach for evaluating the effects of process maturity may be applied usefully in other software development contexts. However, variables such as product size may need to be operationalized differently to reflect other environments such as object-oriented design where, for example, size may be measured using object points (Pfleeger 1998) rather than lines of code. Other independent variables may need to be included to reflect the unique aspects of a particular firm's development environment. It would be useful to determine and evaluate the cost of quality improvement across different development environments, team size, and compositions. Such comparisons could yield insights into the variations in the quality practices used in the various software development domains.

Our study makes several significant contributions to software engineering research and practice. A primary contribution is the rigorous examination of the interrelationships between quality, cycle time, and effort in software product development. As noted earlier, prior studies have modeled quality, cycle time, or effort in the absence of one or more of the other variables. Another contribution is the examination of the direct and indirect effects of improved process maturity on quality, cycle time, and development effort. These effects are not well understood and are generally not rigorously studied in the software engineering literature. Further research could explore reengineering the software life cycle through leaner processes, enabling greater reduction in cycle time from quality improvement. Also, future research could examine the tradeoff between cycle time and effort, controlling for team loading and schedule pressure. It would be instructive to explore the effects of process improvement in different phases of the software life cycle. Finally, the impact of process improvement on the nonengineering activities that support software development could be examined.

For software engineering practice, our results quantify the benefits of process maturity. Software managers may be reluctant to invest in quality improvement practices without knowledge of the return on that investment. For managers of software projects, our analysis provides a methodology for quantifying the time and cost savings from quality improvement. These results provide useful insights for planning and assessing the return on investment from quality improvement in the development process.[1]

[1] The authors gratefully acknowledge research support from the Graduate School of Industrial Administration at Carnegie Mellon University and the University of Michigan Business School. The assistance of managers and staff at our data site were invaluable. Helpful comments were provided by participants in research seminars at Carnegie Mellon University, the University of Michigan, the University of Pennsylvania, the University of Minnesota, the University of Texas at Austin, and the Workshop on Information Systems and Economics (WISE '97).

References

Abdel-Hamid, T., S. Madnick. 1991. *Software Project Dynamics: An Integrated Approach.* Prentice-Hall, Englewood Cliffs, NJ.

Banker, R. D., H. Chang, C. Kemerer. 1994. Evidence on economies of scale in software development. *Inform. Software Tech.* **36**(5) 275–282.

——, S. Slaughter. 1997. A field study of scale economies in software maintenance. *Management Sci.* **43**(12) 1709–1725.

Basili, V. R., J. D. Musa. 1991. The future engineering of software: A management perspective. *IEEE Comput.* **20**(4) 90–96.

——, B. Perricone. 1984. Software errors and complexity: An empirical investigation. *Comm. ACM* **27**(1) 42–52.

Belsley, D. A., E. Kuh, R. E. Welsch. 1980. *Regression Diagnostics: Identifying Influential Data and Sources of Collinearity.* Wiley and Sons, New York.

Bockerstette, J., R. Shell. 1993. *Time Based Manufacturing.* McGraw-Hill, New York.

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

Bohn, R. E. 1995. Noise and learning in semiconductor manufacturing. *Management Sci.* **41**(1) 31–42.

Breusch, T., L. Godfrey. 1981. A review of recent work on testing for autocorrelation in dynamic simultaneous models. D. Currie, R. Nobay, D. Peel, eds. *Macroeconomic Analysis: Essays in Macroeconomics and Econometrics.* Croon Helm, London, England. 63–105.

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

Conte, S. D., H. E. Dunsmore, V. Y. Shen. 1986. *Software Engineering Metrics and Models.* Benjamin/Cummings Publication Company, Menlo Park, CA.

Cook, R. D., S. Weisberg. 1982. *Residuals and Influence in Regression.* Chapman & Hall, London, England.

Crosby, P. B. 1979. *Quality Is Free.* McGraw-Hill, New York.

Davidson, R., J. MacKinnon. 1995. Several tests for model specifications in the presence of multiple alternatives. *Econometrica* **49** 781–793.

——, ——. 1993. *Estimation and Inference in Econometrics.* Oxford University Press, New York.

Deming, W. E. 1986. *Out of the Crisis.* MIT Center for Advanced Engineering Study, Cambridge, MA.

Durbin, J., G. Watson. 1971. Testing for serial correlation in least squares regression III. *Biometrica* **58** 1–42.

El Emam, K., D. R. Goldenson. 1996. Some initial results from the international SPICE trials. *Software Process Newsletter, Technical Council on Software Engineering* **6** (Spring) IEEE Computer Society.

Fenton, N. E., S. L. Pfleeger. 1997. *Software Metrics: A Rigorous and Practical Approach.* International Thompson Computer Press, London, England.

Gopal, A., T. Mukhopadhyay, M. S. Krishnan. The role of software process and communication in offshore software development. *Comm. ACM.* Forthcoming.

Greene, W. H. 1997. *Econometric Analysis.* 3rd ed., MacMillan Publishing Company, New York.

Hausman, J. 1978. Specification tests in econometrics. *Econometrica* **46** 1251–1271.

Herbsleb, J., D. Zubrow, D. Goldenson, W. Hayes, M. Paulk. 1997. Software quality and the capability maturity model. *Comm. ACM* **40**(6) 30–40.

Humphrey, W. S. 1995. *A Discipline for Software Engineering.* Addison-Wesley, Reading, MA.

Iman, R., W. Conover. 1979. The use of the rank transform in regression. *Technometrics* **21**(4) 499–509.

Jones, C. 1986. How not to measure programmer productivity. *Computerworld* **20**(2) 65–76.

—— 1996. *Applied Software Measurement: Assuring Productivity and Quality.* McGraw-Hill, New York.

—— 1997. *Software Quality: Analysis and Guidelines for Success.* ITP Press, London, UK.

Kaplan, R. 1986. Must CIM be justified by faith alone? *Harvard Bus. Rev.* **64**(2) 87–95.

Kemerer, C. F. 1997. *Software Project Management Readings and Cases.* McGraw-Hill, New York.

Kmenta, J. 1986. *Elements of Econometrics.* Macmillan, New York.

Krishnan, M. S. 1996. Cost and quality considerations in software product management. Ph.D. Dissertation, Graduate School of Industrial Administration, Carnegie Mellon University, Pittsburgh, PA.

——, S. Kekre, C. H. Kriebel, T. Mukhopadhyay. 2000. An empirical analysis of productivity and quality in software products. *Management Sci.* Forthcoming.

This paper was accepted by Haim Mendelson and Seungjin Whang.

Lyu, M. R. 1996. *Handbook of Software Reliability Engineering.* McGraw-Hill, New York.

Mowrey, D. C. 1996. *The International Computer Software Industry: A Comparative Study of Industrial Evolution and Structure.* Oxford University Press, Oxford, U.K.

Munson, J. 1996. Software faults, software failures, and software reliability modeling. *Inform. Software Tech.* **38**(11) 687–699.

Nandakumar, P., S. M. Datar, R. Akella. 1993. Models for measuring and accounting for cost of conformance quality. *Management Sci.* **39**(1) 1–16.

Newfelder, A. M. 1993. *Ensuring Software Reliability.* Marcel Dekker, Inc., New York.

Paulk, M. C., C. V. Weber, B. Curtis, M. B. Chrissis. 1995. *The Capability Maturity Model: Guidelines for Improving the Software Process.* Addison-Wesley Publishing Company, Reading, MA.

Pfleeger, S. 1998. *Software Engineering: Theory and Practice.* Prentice-Hall, NJ.

Shapiro, S., M. Wilk. 1965. An analysis of variance test for normality. *Biometrika* **52** 591–612.

Swanson, K., D. McComb, J. Smith, D. McCubbrey. 1991. The application software factory: Applying total quality techniques to systems development. *MIS Quart.* **15**(4) 566–580.

Tagaras, G., H. Lee. 1996. Economic models of vendor evaluation with quality cost analysis. *Management Sci.* **42**(11) 1531–1543.

The Wall Street Journal. 1997. The spoils of war. September 11.

White, H. 1980. Heteroskedasticity-consistent covariance matrix estimator and a direct test for heteroskedasticity. *Econometrica* **48**(5) 817–838.