9-1991

Reuse and Productivity in Integrated Computer-Aided Software Engineering: An Empirical Study

Rajiv D. BANKER
University of Minnesota - Twin Cities

Robert J. Kauffman
Singapore Management University, rkauffman@smu.edu.sg

Reuse and Productivity in Integrated Computer-Aided Software Engineering: An Empirical Study

By: Rajiv D. Banker
Carlson School of Management
University of Minnesota
Minneapolis, Minnesota 55455

Robert J. Kauffman
Stern School of Business
New York University
New York, New York 10006

Abstract

Growing competition in the investment banking industry has given rise to increasing demand for high functionality software applications that can be developed in a short period of time. Yet delivering such applications creates a bottleneck in software development activities. This dilemma can be addressed when firms shift to development methods that emphasize software reusability. This article examines the productivity implications of object and repository-based integrated computer-aided software engineering (ICASE) software development in the context of a major investment bank's information systems strategy. The strategy emphasizes software reusability. Our empirical results, based on data from 20 projects that delivered software for the bank's New Trades Processing Architecture (NTPA), indicate an order of magnitude gain in software development productivity and the importance of reuse as a driver in realizing this result. In addition, results are presented on the extent of the learning that occurred over a two-year period after ICASE was introduced, and on the influence of the link between application characteristics and the ICASE tool set in achieving development performance. This work demonstrates the viability of the firm's IS strategy and offers new ideas for code reuse and software development productivity measurement that can be applied in development environments that emphasize reuse.

Keywords: CASE, ICASE, productivity measurement, reuse, software development, software economics, software engineering

ACM Categories: D.2.8, D.2.9, D.2.m, K.6.0, K.6.1, K.6.3

Introduction

In 1988, Boehm and Papaccio estimated that by the early 1990s, firms would be spending in excess of $125 billion per year on software in their efforts to remain competitive. To control these spiralling costs, many firms are turning to computer-aided software engineering (CASE) products in hopes of realizing improvements in productivity and system quality. Although expenditures on CASE were reported to be rapidly rising (*Software Magazine*, 1988), and firms continue to view CASE as holding out considerable promise for delivering gains (Loh and Nelson, 1989), investment in CASE has tended to be a leap of faith: the performance gains have been very difficult to identify and measure (Davis, 1988; Voelckner, 1988), and unsubstantiated claims of productivity improvements are widespread. For example, IBM has claimed 20 percent to 30 percent gains for its AD/Cycle (Sperling, et al., 1989), and Sony claims to have achieved 600 percent gains in a limited range of applications (Gabel, 1989). Popular press estimates top out at around 10,000 percent in gains (Breidenbach, 1989; Clemons, 1991), but the impact of such high-end claims is offset by others who report little or no gains at all (McGuff, 1989). Moreover, in a recent survey of 196 CASE-using firms conducted by *Software Magazine*, 74 percent responded that they did not have a productivity measurement program in place (Knight, 1989). Clearly, many firms have not yet begun to even measure the impact of CASE.

CASE technologies involve significant automation of the software development life cycle. Currently available tools fall into three categories:

lower CASE, upper CASE, and integrated CASE. *Lower CASE* provides support for the later life cycle stages, especially code construction and testing. *Upper CASE* offers assistance during the early life cycle stages of analysis and design. *Integrated CASE* (ICASE) tools, by contrast, support both the earlier and later stages of the life cycle. Depending upon the specific ICASE tool that is considered, developers will have access to a variety of automated software engineering facilities. Some of these include specialized report painting and user screen design tools, code reuse search facilities, automated documentation preparation facilities, multiple code generators, code debugging tools, links to existing 3GL code libraries, entity-relationship (E/R) and data flow diagrammers (DFD), and software version distribution control facilities.

At the heart of an investment program in ICASE is management's desire to improve development productivity and software maintainability. In this article, some empirical evidence is presented about the viability of implementing an information systems (IS) strategy that enables a firm to produce high functionality software that could not have been produced in a cost-effective manner with traditional development methods. The cornerstone of this strategy was the deployment of an ICASE tool that emphasizes software reusability. The benefits of constructing reusable software were recently discussed by Kim and Stohr (1991) and Nunamaker and Chen (1989a). Apte, et al. (1990) present a case study of a large commercial bank's experiences with this software reusability-based approach. However, there is a lack of generally accepted methods for creating and implementing reusable software (Lenz, et al., 1987), and little research has been done to document the gains that a reusability approach can produce (Biggerstaff and Richter, 1987; Parker and Hendley, 1988). Moreover, little work has been done to determine the leverage that object-oriented development methods may provide in improving development efficiency.

The empirical results presented in this article are intended to provide some initial evidence about the gains from reuse. They are based on an analysis of data collected in a two-year field study of software development at the First Boston Corporation, a large investment bank in New York City. The firm deployed an ICASE tool called High Productivity Systems (HPS) that was meant to support and integrate activities involved in the development of multi-platform, cooperative processing or client-server architecture applications. Such applications achieve levels of functionality beyond what can be produced using traditional development approaches. This architecture is increasingly seen in the industry as a prerequisite to manage growing hardware costs while optimizing processing performance for local decision making, trades processing, and large-scale database searches and financial optimization at the mainframe level.[1]

Twenty-one software development projects were studied in depth (though one project was later omitted from the formal analysis as an outlier). The applications that resulted now form First Boston Corporation's new trades processing architecture (NTPA), a set of core software applications that provide much of the bank's trades processing, securities inventory management, trades commission processing, real-time pricing of financial instruments, and general ledger accounting capabilities.

The principal contributions of this research are:

1. Description of an IS strategy that relies on software reusability so that high functionality, cooperative processing software can be produced in a cost-effective manner.
2. Documentation of an order of magnitude gain in software development productivity that appears to be associated with the deployment of an ICASE tool that supports reusability in software development.
3. Demonstration of the use of a new reuse measurement approach for software development productivity that is generalizable to other organizations that deploy ICASE with the objective of supporting reusability.

HPS at First Boston: An IS Strategy Involving ICASE and Reusable Software

This section discusses the rationale for the IS strategy pursued by the First Boston Corporation,

[1] For additional background information on cooperative processing and the client-server architecture, refer to Desmond (1989) and Edelstein (1989).

the characteristics of the ICASE tool, and the software development environment in which it was implemented. Also presented are the research questions that later sections of this article address through empirical analysis.

The need for a new IS strategy

The 1980s were characterized by rapid technological change and increasing competition for major American investment banks. The First Boston Corporation's investment banking business required more sophisticated software applications and growing computer hardware power for high-speed securities and money market transactions processing, immediate access to and processing of large mainframe databases for use with real-time financial analytics, local access and customized analysis of distributed databases for financial market traders, and management and control of the firm's cash balances and securities inventory. Similar to those of its competitors, First Boston's systems increasingly were required to operate seamlessly 24 hours a day across three platforms—microcomputers, minicomputers and mainframes—in support of global investment banking and money market trading activities.

The trend in the investment banking industry has been in the direction of applications software that achieves a higher level of functionality for the user than in prior generations. Much of this is aimed at giving traders added capabilities to realize a profit in highly competitive markets. For example, at First Boston *high functionality software* was expected to offer the trader the ability to:

1. Consolidate multiple digital feeds of market information into a single trader workstation;
2. Support real-time, computer-based financial optimization analytics for trading decisions with respect to existing (e.g., index arbitrage and option pricing) and newly created (e.g., synthetic options, and multi-instrument hedging) financial instruments;
3. Provide a user-friendly, windowing interface that traders could customize for their own needs; and,
4. Deliver consolidated and unbundled information on customer accounts and trader positions for risk management purposes.

Gene Bedell, the firm's CIO at the time, believed that First Boston's strategic necessity was to deliver systems that could attain such high levels of functionality. Because bringing high functionality systems into production rapidly was not possible with traditional development methods, maintaining the status quo of traditional software development methods would be a losing strategy (Clemons, 1991). In the absence of more productive development methods, the IS operations would blunt the firm's ability to deliver and support innovative financing products in a timely manner.

Characteristics of the IS strategy

Bedell recognized that First Boston's IS operations needed to build software in a way that growing system complexity and system size would not lead to increasingly uncontrolled growth in development and maintenance costs. Bedell also recognized that increasingly complex interfaces were needed between cooperative processing platforms and that this would create major development bottlenecks. First, the functionality that had to be built was substantial, and the larger the system the more cost prohibitive it would be to deliver (Conte, et al., 1986). Second, the firm would need to retain three sets of highly skilled and highly paid developers—one set for each of the three processing platforms. The only way to avoid this "software trap" was to consider automating software development (Feder, 1988).

Bedell's next move was just that: to recommend that First Boston consider CASE as the major component of its IS strategy. But after a range of vendor-supplied lower and upper CASE tools available on the market were tested, Bedell's technical staff concluded that none would offer the right mix of power and flexibility to build the high functionality, cooperative processing systems that were needed to take the bank into the mid-1990s. Piece-meal application of specific CASE tools to individual phases of the software development life cycle would make it difficult to effectively link the phase-by-phase products of development, and there would likely be little positive impact on productivity.

Bedell's alternative strategy to cope with this "functionality risk" was to build an ICASE tool *in-house* (Clemons, 1991). Although the investment posed a major risk to the firm, First Boston

subsequently committed $65 million for a new software development methodology and a new architecture of investment banking software applications in late 1986, one year before the stock market crashed.[2] This investment would lay the foundation for High Productivity Systems (HPS), the firm's ICASE tool, and a set of core applications for First Boston called the New Trades Processing Architecture. (Figure 1 expands on the business strategy/IS strategy link that led to First Boston's investment in HPS.)

The technical vision behind the IS strategy

In early 1986, the First Boston Corporation gathered together a team of senior IS and user managers to make a set of recommendations regarding the firm's technology and systems. They reached an important conclusion: the firm would implement a *reusability approach* to rebuild and upgrade the capabilities of the existing information systems architecture in a way

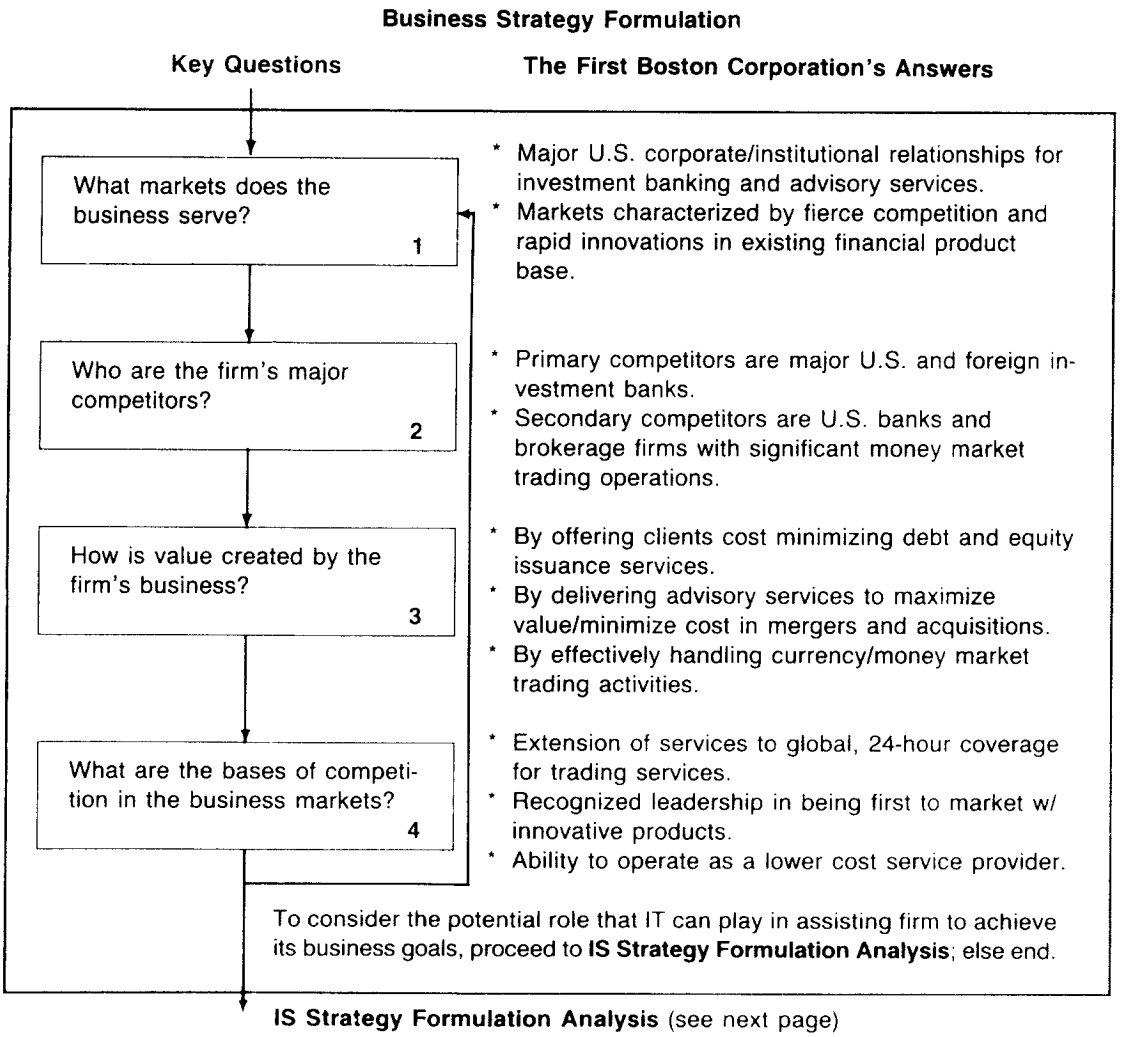


Figure 1. An Analysis of the Business Strategy/IS Strategy Link for the First Boston Corporation

[2] The costs would later rise as high as $100 million (Schwartz, 1990).

that their basic building blocks—objects and modules—could be reused repeatedly. Vivek Wadhwa, later to become the firm's top technical specialist, was hired to implement and expand on this technical vision, which also was meant to reduce the bank's reliance on costly language-specialized programmers by making it possible to develop software that could run on

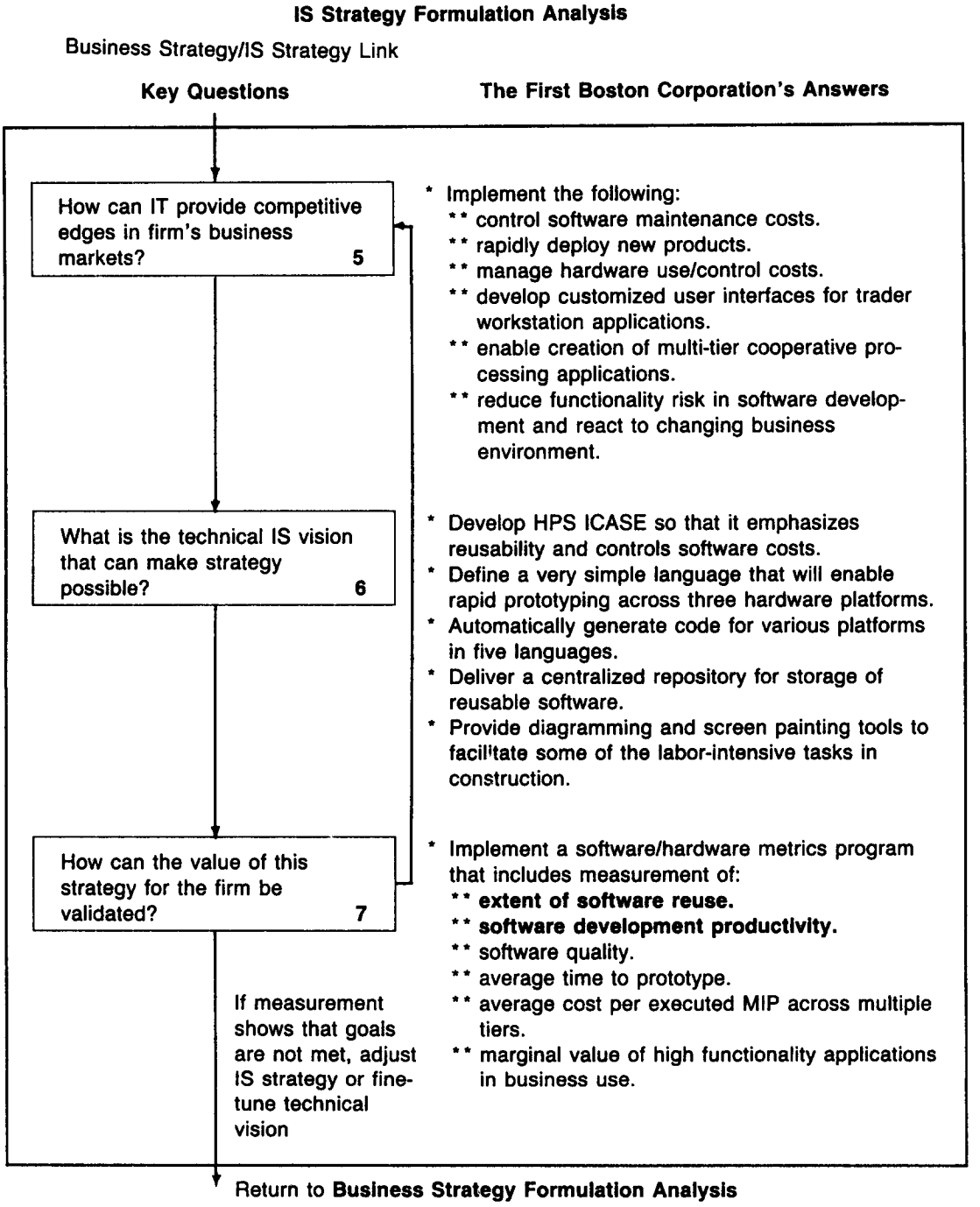


Figure 1. Continued

any of the three platforms with a single "rules language." This rules language would be defined within HPS. Code generators would then process this HPS code so that run-time COBOL, PL/1, and C and other code would result for each of the three major development platforms, effectively screening developers from the complexity of the development environment. Wadhwa believed that this approach, when combined with special facilities to provide the platform-to-platform communications links called "middleware," would lead to the development of NTPA systems at an affordable cost. (For a description of some representative NTPA systems, refer to Table 1.)

HPS: An ICASE tool to support the IS strategy

HPS supports reusability because it is an *object-based* ICASE tool. The object types include programs, rules, output screens, user reports, data fields, and 3GL components, among others.[3] In order to make the reusability approach possible, HPS development was coupled with a centralized repository for reusable objects. The structure of the data stored in the repository is based on an implementation of the entity-relationship attribute model, originally developed for database design (Chen, 1976; Chen and Sibley, 1991; Teory, et al., 1986). Specifications for the objects used to construct an application are stored in a central repository where they become available to other developers. The repository includes all the definitions of the data and objects that make up the organization's business. The motivation for having a single repository for all such objects is similar to that for having a single database for all data: all objects need only be written once, no matter how many times they are used. When they are used and reused in various combinations, repository objects form the functionality that represents the information systems processing capability of the firm.[4]

In addition, HPS provides diagramming and "painting" facilities for enterprise modeling and analysis and design; it provides code generators for five development languages; and it also offers tools for debugging code and managing versions of the same application. The coverage of the HPS tool set across the systems development life cycle is depicted in Appendix 1.

Software development performance at First Boston Corporation

Prior to 1987, First Boston did not have a formal program in place for tracking software development productivity. However, First Boston's senior IS managers informed us that some IS consulting firms had reported productivity of about eight to 10 function points per person month at large financial institutions. (*Function points* is a popular productivity metric for software development that is discussed in more detail later in the article. For example, Capers Jones estimates function point productivity levels in the range of eight function points per person month for MIS business applications (Bouldin, 1989).) In addition, several project managers that we interviewed had worked in the IS development areas of other large financial institutions. They indicated that First Boston's traditional software development performance prior to the deployment of HPS was similar to those other firms. They also believed the IS consulting firms' estimates were applicable to the pre-HPS deployment environment at First Boston.

To obtain additional background information on the extent to which ICASE enabled the firm's reusability approach to deliver productivity benefits, interviews were conducted with seven managers who had overseen NTPA development projects at First Boston. When the development

[3] This term is used to distinguish development environments like HPS and should not be confused with the object-oriented approach. The primary differences are that object-based development does not allow for instances of object classes to be "classes" themselves, nor would objects in object-based development have any special "inheritance properties." (See Booch, 1989, for additional details on the distinctions, and Meyer, 1988, for a discussion of the object-oriented paradigm of software construction.) Thus, in HPS development object types, such as a "screen" or a "rule," parallel abstract data types or classes in the object-oriented paradigm. Instances of HPS object types might include a "CLIENT_INPUT_SCREEN" or a "COMMISSION_CALCULATION_RULE." These parallel "instances" of object classes in object-oriented development; however, in HPS an instance of an object is unable to act as a class and would not include any "inheritance" capabilities.

[4] For additional background on repositories, see the following: Banker and Kauffman (1991), Chen and Sibley (1991), Fisher (1990), and Hazzah (1989).

Table 1. Representative Applications From First Boston Corporation's New Trades Processing Architecture (NTPA)

Application Name	Description
Dealer's Clearance	Designed to improve operational and treasury management productivity by automating settlement, providing online, real-time display of clearances, and projected end-of-day securities and cash balance positions.
General Ledger Interface	A table-driven, self-balancing system that automatically posts entries from every transaction processing system included in NTPA. As a result, manual reconciliations are never required.
Firm Inventory/ Foreign Securities & Currencies	Maintains information for firm-wide management of foreign securities and currencies. Tracks individual trade lots and can determine profit and loss using various trading accounting bases.
Floor Broker	Manages fee and discount information for all brokers used by the firm. The system maintains payment histories linked to exchange, broker, and trading volume.
Product Master	This system supports identification of financial products across business areas. It enables each business group to classify and process securities according to its own business requirements, and it allows trading areas to establish new product types in the process of conducting business.
Real-Time Firm Inventory	Trading management uses this system to monitor trading positions, exposures, and intraday profit and loss by product, account, desk, department, or the entire organization. This system also enables traders to set up and monitor a strategy by linking several positions.

of multi-tiered (cooperative processing), high functionality applications using traditional tools was compared with using HPS, the interviewees estimated that:

1. Full life-cycle traditional development would take 115 percent longer on average (std. dev. = 73.4%) to complete;
2. The construction phase alone would take 121 percent longer on average (std. dev. = 101.9%); and,
3 Maintenance and enhancement also would take 121 percent longer on average (std. dev. = 71.6%).

The project managers indicated that reusable software was essential for such productivity gains to be possible. Yet the firm had not measured development productivity in an HPS environment, so their estimates were not confirmed.

Because high productivity gains from software reusability were believed to be essential to First Boston's IS strategy, this article addresses three principal research questions:

1. What was the level of reuse observed in software development projects?
2. Did reusability lead to any significant productivity gains during the first two years of the deployment of the HPS ICASE tool?
3. Did the evolving features of the HPS tool set and the organization's adaptation to the new development environment influence reuse and productivity gains?

Systematic evidence documenting product gains from software reusability is the essential first step in this research because this source of productivity gains forms the very foundation of the IS strategy at the research site. In this first study, however, we did not test other important aspects of the firm's IS strategy—for example, the value of the high functionality user interface offered to the trader and the importance of real-time connections across multiple processing platforms. Nor did we attempt to verify claims as to whether downstream maintenance costs would be lower, whether higher quality would be created, or whether using HPS would drive down the skill level at which a developer becomes productive.

The answers to these research questions regarding First Boston's experience with ICASE should be of general interest to practitioners and the IS research community alike. At present, there is little descriptive information available about reuse levels in ICASE environments. Nor is there much evidence to assist our understanding of the extent of the leverage that reuse creates in making software development operations more efficient. And, because every vendor's software development tools are evolving at a rapid rate, we believe that it is important for all firms that deploy CASE tools to gauge how development performance will shape up in the presence of a changing and imperfect tool set.

A Descriptive Model for Reusability and ICASE Development Productivity

This section presents a model relating development productivity to the deployment of an ICASE methodology that emphasizes reuse. Specifically discussed are the motivation for this model, the constructs it involves, and the measures for the constructs. This model does not attempt to capture all aspects of the deployment of ICASE: it focuses on the leverage that reuse creates on productivity.

Background for model development

A series of semi-structured interviews were initially conducted with five senior managers and seven project managers who managed the development of various aspects of the First Boston Corporation's New Trades Processing Architecture. The interviews probed how the firm's ICASE tool affected development productivity under a variety of development scenarios. Project managers reported that:

1. The single most important feature of the ICASE tool was its ability to store reusable objects in a centralized repository.
2. Software reuse under HPS required experienced developers to learn new skills; however, relatively inexperienced developers seemed to learn about reuse quite rapidly. Project managers reported that it took only two months to become productive as an HPS developer, and after about six months the learning curve flattened out.
3. The reliable performance of the development tools offered within HPS and the stability of the overall development environment over a longer period of time were believed to be more important than the skill levels of individual developers.
4. During the first year after ICASE was deployed, not all of the key pieces were in place, and this meant that not all kinds of application projects were equally well supported. For example, batch systems tended to be heavy in one object type called "components." These are 3GL library routines whose integration was not well handled early in version 1 of HPS. In addition, the relatively early availability of an HPS screen-painting facility within version 1 tended to enhance development productivity for online, real-time applications. These contained more user screens than applications with mostly batch processing functions.

Overall, reuse was believed to be the major factor affecting development productivity. These observations are supported by a growing literature on productivity and software reuse (Apte, et al., 1990; Moad, 1990; Nunamaker and Chen, 1989a; 1989b; Norman and Nunamaker, 1989; Scacchi, 1989).

The interview findings led us to build into our model characteristics that differ from traditional software development productivity models in several ways. The *extent of observed reuse* should be explicitly incorporated to capture the leverage on development labor consumed in boosting out-

put functionality. Because the ICASE tool that was deployed at the First Boston Corporation is object-based, reuse occurs as reuse of *repository objects* (e.g., rules, screens, and reports, and 3GL components—which can also be used as though they were repository objects, though technically they are not) that were created for other applications or earlier for the same application.

The *maturity of the ICASE tool set* also should be incorporated as a predictor of the extent of reuse and software development productivity, especially when major changes in its capabilities change with the deployment of new versions of the tool. In addition, the initial deployment of ICASE is as likely to result in reduced productivity as it is to deliver on the promise of productivity gains. Initially, it is unlikely that all of the bugs in the tool will be worked out nor that all of the ICASE tool's planned capabilities will be in place. In addition, development staff will need to be retrained, and project managers will need to refine elements of their management tactics to effect project control. Finally, with the deployment of a new ICASE tool—especially one that is repository-based—it will be necessary to establish a base of reusable code in the repository.

Although the firm's software development staff varied in its levels of traditional software development experience, most had little exposure to HPS when the construction of the NTPA applications was initiated. The maximum amount of experience that anyone had with HPS was about one year. All of the projects examined were managed under a common management structure; most project personnel worked on multiple projects to broaden their experience, and the HPS rules language was constant across projects. Interviews with project managers and team members also indicated that experience and skills did not vary significantly across projects.

The descriptive model

A descriptive model was proposed for evaluating the impact of reuse in ICASE development that utilizes five constructs: software development labor consumed, the application functionality produced, the extent of reuse, the nature of the ICASE tool set at the time an application was developed, and the characteristics of the application under development. (See Figure 2.)

New Objects Built

The descriptive model employs an *object reuse* metric that can be applied to a range of object-based and object-oriented development environments and can be readily measured.[5] Our metric fits First Boston's ICASE environment well because software development essentially involves constructing and reusing objects, thereby reducing the amount of new software that must be written. Both repository objects and 3GL components (which are also considered to be objects in this environment) are reusable within applications and across their boundaries. The metric we employed is defined as:

$$\text{NEW OBJECT PCT} = \frac{\text{NUMBER UNIQUE OBJECTS BUILT FOR APPLICATION}}{\text{TOTAL NUMBER OF OBJECTS COMPRISING APPLICATION}}$$

NEW_OBJECT_PCT (new object percentage) provides a measure of the portion of the total number of objects that comprise an application that must be built for the first time.[6] The total number of objects in an application is based on the number of different situations in which a program uses an object. As a result, multiple executions of the same object in the same situation are not counted. The computation of NEW_OBJECT_PCT is illustrated in Appendix 3.

The ICASE Object Reuse Model

The extent of new code that must be constructed (NEW_OBJECT_PCT) is likely to be influenced by the stability and length of time the ICASE tools have been in place (MATURITY) and the features of the ICASE tool that offer support for building systems with different characteristics (APPLICATION). Reuse may be limited for applications with certain characteristics, especially when an object with highly specialized functionality is required that is not already stored in the repository. These relationships are depicted in the ICASE Object Reuse Model in Figure 2.

[5] For a more comprehensive discussion of software reuse metrics, see Banker, et al. (1990a).

[6] An alternative is to weight each object by counting the number of function points it represents. However, this approach is not readily applied in practice because functionality spans multiple objects.

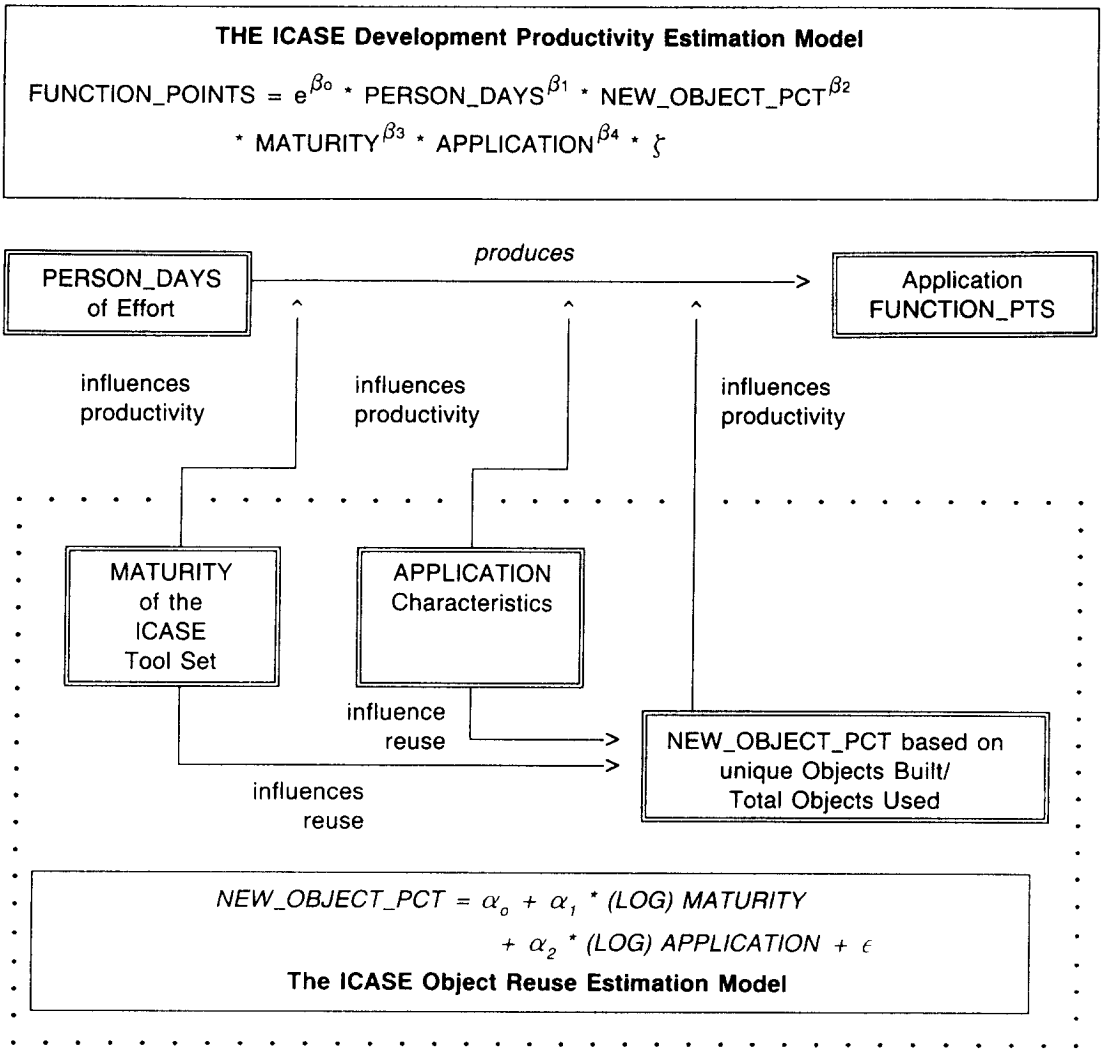


Figure 2. Conceptual and Estimation Models

The ICASE Development Productivity Model

Software development effort, operationalized as PERSON_DAYS of effort, is the primary driver of the overall amount of functionality that is delivered in a software development project, operationalized in terms of FUNCTION_POINTS. Functionality delivered by the software development process is influenced by three project factors:

1. ICASE tool set MATURITY, a binary variable indicating whether a project was one of 13 built in the first year (and primarily built with HPS version 1) or one of seven built in the second year of the tools' deployment (and mostly built with HPS version 2, which expanded on the capabilities of the earlier version);[7]

[7] The reader should recognize that the operationalization we selected is in some sense the best surrogate for MATURITY that was available. While we expect that CASE tools mature gradually over time, as new capabilities are deployed and come into increasing use among developers, normally an upgrade from an old version to a new one involves a significant number of changes. When First Boston's software development managers characterized projects as "Year 1" or "Year 2" development efforts, they not only were indicating that a Year 2 project had started at a certain point in time, they were also indicating that the composition of the tool set used to build the project had changed from Year 1.

2. NEW_OBJECT_PCT, a variable representing the portion of new code that had to be developed in a project;
3. APPLICATION type, a binary variable indicating a batch processing or an online, real-time application. Although we chose to focus on only two application types in this research, this variable more generally is meant to capture how the characteristics of an application interact with the available development tools to result in code reuse and the production of software functionality.

The variables are defined in more detail in Table 2. The relationships among these variables are labeled as the ICASE Development Productivity Model in Figure 2. A more general model, based on additional experience with other organizations that have deployed CASE, could employ other operationalizations of the variables we selected and may also involve additional variables not considered here.

Software Development Productivity Research from an ICASE Perspective

This section discusses two aspects of the prior literature on software development performance evaluation that guided our development of the descriptive model ICASE productivity and reuse: metrics in software development productivity research, and measurement of code reuse.

Metrics for software development performance assessment

Much has been written in recent years regarding the modeling and measurement of software development productivity (Banker and Kemerer, 1989; Banker, et al., 1991; Davis, 1988; Gaffney, 1986; Grammas and Klein, 1985; Kang and Levy, 1989; Nunamaker and Chen, 1989b; Scacchi, 1989). Evaluation of development productivity involves measuring inputs consumed in the process of planning, designing, documenting, building, testing, and implementing, as well as comparing the resulting functionality. The standard approach is to gauge project development productivity across the entire life cycle. This approach is often represented as a "black box" production process, in which consumption of labor results in developed code delivering the functionality of a software application.

Software development labor is the primary input that drives functionality delivered in a software project. Software development productivity is often measured using the input-output ratio:

$$\text{PRODUCTIVITY} = \frac{\text{SIZE OF APPLICATION DEVELOPED}}{\text{LABOR CONSUMED DURING DEVELOPMENT}}$$

Two popular ways of measuring the size of the output of a software development project have emerged over the years: counting *source lines of code*[8] (SLOC) (Jones, 1986; 1988) and performing *function point analysis* (Albrecht and Gaffney, 1983; Low and Jeffrey, 1990; Sprouls, 1990; Symons, 1988). SLOC is not an appropriate metric for this study because HPS generates code in five different languages, making the resultant measures incomparable across applications and with manually written SLOC.

The function point analysis methodology gauges the size of an application in terms of its functionality. The metric that results—function points—incorporates two intermediate measures: function counts (FC) and a complexity multiplier (CM). Function counts aggregate the number and relative complexity of data input types, output types, file types, external interface types, and external inquiry types. Function counts are then adjusted using a complexity multiplier representing the impact of 14 dimensions (FACTORS) of the application's implementation environment. This yields function points. (Additional introductory details on function point analysis are provided in Appendix 2.)

The function points method offers an attractive metric because it abstracts from the programming languages used in different development projects. Firms that use function points for productivity comparisons are also supported by the existence of national and international user groups that define the standards for implementing the methodology to ensure that software development productivity results are comparable

[8] SLOC can be thought of in terms of the code that forms the procedure division of a Cobol program because this is what creates the functionality of an application developed using this language.

across organizations. As a result, function points currently enjoy support among *Fortune 500* firms (Bouldin, 1989; Dreger, 1989; IFPUG, 1988; Sprouls, 1990) and receive continued attention by the academic research community (Kemerer, 1990; Symons, 1988). There is currently no other common metric that emphasizes functionality as much as function points or supports better software development comparisons, and so we have chosen to adopt it in this research.

On the input side, labor is clearly the primary ingredient; all other capital inputs, such as the ICASE tool or the hardware used, are invariant across projects. The measures most often employed for labor are days or months of effort.

Measurement of software reusability

Prior research has investigated numerous project factors that influence software development

Table 2. Definitions of Variables in the Estimation Models

Variable Name	Definition
Object Reuse Estimation Model	
NEW_OBJECT_PCT	Number of unique ojects built for the application divided by the total number of objects comprising the application (UNIQUE OBJECTS/OBJECTS USED).
MATURITY	A binary variable that takes the value e if a project is a "Year 2/HPS Version 2 Project" and 1 if a project is a "Year 1/HPS Version 1 Project."
APPLICATION	A binary variable that takes the value e if a project results in an online, real-time application, and 1 if it is a batch application.
α_o , α_1 , α_2	Model parameters to be estimated.
ϵ	A normally distributed error term.
ICASE Development Productivity Estimation Model	
FUNCTION_POINTS	An output metric for the size of the software product that is delivered.
PERSON_DAYS	An input metric for development effort in person days.
OBJECT_REUSE	Number of unique objects built for the application divided by the total number of objects comprising the application (UNIQUE OBJECTS/OBJECT USED).
MATURITY	A binary variable that takes the value e if a project is a "Year 2/HPS Version 2 Project" and 1 if a project is a "Year 1/HPS Version 1 Project."
β_o , β_1 , β_2 , β_3	Model parameters to be estimated.
ζ	A log-normally distributed error term.

Note: Taking logs of the MATURITY and APPLICATION variables results in the values 1 and 0, respectively, when these variables have the observed values of e and 1. The usual interpretation of a binary variable is maintained.

productivity (Banker, et al., 1991). The relevant ones include project management practices, application type, development staff experience, the programming language used, and the stability of the development platform and user requirements (e.g., in terms of project size or function points). In software projects developed using HPS, the *level of reuse* (conceivably a project management practice) was believed to be the major factor deserving attention (Gaffney and Durek, 1989; Jones, 1984). If extensive reuse can increase productivity by an order of magnitude or more, as has been reported in the popular press, it would yield significant cost reductions in software development operations.

Reused code is also present to some degree in traditionally developed projects. For example, it is common for a developer to identify a piece of code that has similar functionality and then adapt or "template" it to meet a specific need. Successful reuse programs have been stymied in many organizations, however, because of weaknesses in the methodologies used (Horowitz and Munson, 1984; Mathis, 1986), problems with training and top management support (Bigger-staff and Richter, 1987; Tracz, 1987), and motivational reasons when developers and project managers feel that a methodology change endangers jobs (Kemerer, 1989; Nezlek and Leitheiser, 1991; Wong, 1987). Assessing the level of reuse in a 3GL programming environment is also difficult. Although certain types of explicit reuse (for example, modules from code libraries) are easy to identify, most of the code that might be reused is hidden within programs where it cannot be readily identified.

Unfortunately, prior research on reuse provides little guidance as to how to construct a relevant metric for reuse. The bulk of the work has focused on how to exploit the available technology to increase the level of reuse in 3GL and 4GL environments. (Some representative references include Horowitz and Munson, 1984; Jones, 1984; Kernighan, 1984; Lanergan and Grasso, 1984; and Matsumoto, 1984.) We identified just two studies that made concrete suggestions regarding the measurement of reuse. Standish (1984) proposed that reuse should be measured at the line of code level. Neighbors (1984) argued that reuse should be abstracted from the level of source code into some meta-language that relates to the problem. This idea fits better with the ICASE tool development environment examined in this field study because the repository objects can be thought of as the elements of the meta-language. In fact, in the HPS environment systems developers reuse entire objects rather than specific lines of code, and such reuse is more appropriately referred to as *object reuse*.

An Estimation Model for Object Reuse and ICASE Development Productivity

Data collection

Data were obtained on person months from First Boston's software development labor charge-out records for 21 projects. Labor was charged out by PERSON_DAYS to the various phases of a software development project, and a normal person month was conservatively viewed as having 18 PERSON_DAYS. Interviews were conducted with project managers wherever possible to review project charge-out data and examine their project worksheets. When a project manager had left the bank, the labor data were unavailable or too sketchy to give a picture of the overall level of effort, or the documentation was not in order, the project was eliminated from further consideration. Also investigated, in addition to the 21 projects, was the feasiblity of obtaining data on other NTPA projects. However, one project for which other information was available could not be evaluated because of a lack of records about the amount of labor consumed; six more were not documented in a manner that would enable us to measure functionality.

Nearly all the information needed to analyze FUNCTION_POINTS for an application was obtained from the functional and technical specification documentation stored in the central repository. HPS also provided a facility for printing out a hierarchically decomposed diagram of application modules. This greatly assisted function point and object reuse analysis because the documentation we examined was effectively standardized.

Manual collection and cross-checking of function points data is very costly. The collection of function points was coordinated by a single person who was on site at First Boston for three months.

He initially trained and provided feedback on function point analyses conducted by two other members of the research team, who were also involved in performing object reuse analysis and conducting project manager interviews. Function point complexity measures were obtained directly from the project managers who had built and implemented the projects.[9] Function point estimates for the projects were checked by performing a recount. Any discrepancies were later resolved in discussion between project members. Our findings were consistent with the recent results of Kemerer (1990), who found that function point estimation tends to vary little more than plus or minus 10 percent when different people do the counting.

Our final sample of 20 projects excluded one project among the initial 21 that was believed to be an outlier. This project exhibited a very low level of productivity that was attributed to changing functional specifications and interruption of the development schedule. The data set examined in this research compare favorably in size with other studies that used data sets employing function points as the output metric (for example, Albrecht and Gaffney, 1983 (24 obs.); Behrens, 1983 (22 obs.); and Kemerer, 1987 (17 obs.)). Detailed data and summary statistics for object reuse and productivity for the 20 projects are shown in Table 3.

Projects ranged in size from a minimum of about 98 to a maximum of 5,876 FUNCTION_POINTS. The PERSON_DAYS of labor expended on these projects also varied, with a low of 85 and a high of 2,193 days, respectively. The value of NEW_OBJECT_PCT ranged from 100 percent (indicating there was no reuse because each newly built object was used just once) to 16.1 percent (indicating the project involved 83.9 percent reuse).

An econometric model for object reuse and CASE productivity estimation

The estimation model we employed to represent the descriptive model is shown below.

EQUATION 1. $NEW\_OBJECT\_PCT = \alpha_o + \alpha_1 * LOG\ (MATURITY) + \alpha_2 * LOG\ (APPLICATION) + \epsilon$

EQUATION 2. $LOG\ (FUNCTION\_POINTS) = \beta_o + \beta_1 * LOG\ (PERSON\_DAYS) + \beta_2 * LOG\ (NEW\_OBJECT\_PCT) + \beta_3 * LOG\ (MATURITY) + \beta_4 *$ LOG (APPLICATION) + LOG (ζ)

In EQUATION 1, NEW_OBJECT_PCT is estimated as a function of the logarithms of the MATURITY of the ICASE tool in the development environment and the type of APPLICATION being developed. EQUATION 2, presented in its loglinear estimation form, extends prior multiplicative models of software development productivity (e.g., Albrecht and Gaffney, 1983; Bailey and Basili, 1981; Banker and Kemerer, 1989; Behrens, 1983; Belady and Lehman, 1979; Boehm, 1981; DeMarco, 1981; Kemerer, 1987; Walston and Felix, 1977; Wingfield, 1982) to incorporate software reusability. It captures the intuition expressed in the descriptive model.

The error terms of the two models are likely to be correlated (i.e., COV(ϵ,log(ζ)) $\neq$ 0), as they relate to the same projects. For this reason, we employed *seemingly unrelated regression* (SUR) to estimate the joint model. Estimation of the two equations together as a SUR model, using an iterative method devised by Zellner (1962), results in statistical efficiency gains.[10]

[9] The median value of the sum of the raw application complexity scores (FACTORS) was 36, implying that on average FUNCTION_COUNTS would be adjusted by a complexity multiplier (CM) of:

CM = .65 + (.01 * FACTORS) = .65 + (.01 * 36) = 1.01

to arrive at FUNCTION_POINTS. (See Appendix 2 for details.) In addition:

* The upper quartile value of FACTORS = 55 (a CM of 1.20); the lower quartile value of FACTORS = 29 (CM of 0.94).

*The median value of CM for online applications = 1.15; the median value of CM for batch processing applications = 0.91.

The results are robust with respect to this relatively subjective measure. When the model was estimated substituting FUNCTION_COUNTS for FUNCTION_POINTS, the results did not change much.

[10] SUR would not result in statistical efficiency gains if the set of dependent variables in EQUATION 2 was identical to, or a linear transformation of, the set of dependent variables in EQUATION 1. Note, however, that PERSON_DAYS appears only in the right hand side of EQUATION 2. It is also worthwhile to point out that the presence of NEW_OBJECT_PCT as a dependent variable in EQUATION 1 and as an independent variable in EQUATION 2 does not eliminate the statistical efficiency gains of the joint estimation or render the use of SUR inappropriate. For additional details, see Judge, et al. (1985), pp. 466–471.

Table 3. An Overview of the Data Set

Project #	Person Days Labor (A)	Function Points (B)	Function Points/ Person Month (B/A)*	Online or Batch Application Type	New Object Pct	Year 1 or Year 2 Project? (Maturity)
1	1068	2250.08	37.93	ONLINE	23.2%	YEAR 1
2	737	170.56	4.17	BATCH	100.0%	YEAR 1
3	492	300.14	10.98	BATCH	54.3%	YEAR 1
4	2193	632.96	5.20	BATCH	71.9%	YEAR 1
5	520	264.60	9.16	ONLINE	35.1%	YEAR 1
6	1294	1273.70	17.71	BATCH	61.0%	YEAR 1
7	295	352.50	21.51	BATCH	49.3%	YEAR 1
8	471	494.08	18.88	ONLINE	48.1%	YEAR 1
9	136	97.92	12.96	BATCH	93.1%	YEAR 1
10	426	148.41	6.27	BATCH	96.2%	YEAR 1
11	862	385.14	8.04	BATCH	69.0%	YEAR 1
12	147	1092.00	133.71	ONLINE	44.8%	YEAR 1
13	230	241.82	18.93	ONLINE	45.7%	YEAR 1
14	686	3812.40	100.03	ONLINE	26.6%	YEAR 2
15	376	1772.40	84.85	ONLINE	34.7%	YEAR 2
16	469	3475.20	133.38	ONLINE	29.2%	YEAR 2
17	85	135.00	28.59	ONLINE	78.1%	YEAR 2
18	648	5876.25	163.23	ONLINE	23.1%	YEAR 2
19	233	3712.80	286.83	ONLINE	16.1%	YEAR 2
20	416	886.58	38.38	BATCH	32.8%	YEAR 2
Descriptive Statistics						
MEAN	589.2	1368.7	57.0**	——	51.6%**	——
STDDEV	489.3	1630.0	73.4	——	25.7%	——
MAXIMUM	5876.0	2193.0	286.8	——	100.0%	——
MINIMUM	97.9	85.0	4.2	——	16.1%	——
COUNT	——	——	——	BATCH: 9	——	YEAR 1:13
				ONLINE: 11	——	YEAR 2:7

Correlation Matrix

	PERSON DAYS	FUNCTION POINTS	APPLICATION TYPE	NEW OBJECT PCT	MATURITY
PERSON DAYS	1.000				
FUNCTION POINTS	.043	1.000			
APPLICATION TYPE	−.326	.511	1.000		
NEW OBJECT PCT	.096	−.683	−.653	1.000	
MATURITY	−.336	.576	. 533	−.515	1.000

* We used a conservative estimate of 18 PERSON_DAYS in a PERSON_MONTH of software development to determine function point productivity; i.e.:
FUNCTION_POINTS / PERSON_MONTH = FUNCTION_POINTS / PERSON_DAY * 18.

** The means for FUNCTION_POINTS/PERSON_MONTH and NEW_OBJECT_PCT reported in this table are simple averages. We report more meaningful *project size-weighted average values for the means* of FUNCTION_POINTS/PERSON_MONTH and NEW_OBJECT_PCT in Figures 3 and 4.

Discussion of Empirical Results

Results and interpretation of the estimation model

The results of the estimation of the ICASE object reuse and development productivity models for 20 projects are shown below.

EQUATION 1. NEW_OBJECT_PCT =

α_o + α_1 * *MATURITY* + α_2

0.71 ***−0.12*** ***−0.27***
.001 *.11* *.003*

* *APPLICATION*

EQUATION 2. LOG (FUNCTION_POINTS) =

β_o + β_1 * *LOG (PERSON_DAYS)*
1.45 ***0.55***
.10 *.001*

+ β_2 * *LOG (NEW_OBJECT_PCT)* + β_3
−1.92 ***0.40***
.001 *.11*

* *LOG (MATURITY)* + β_4 * *LOG (APPLICATION)*
−0.04
.46

Note: The one-tailed significance level is shown below the coefficient estimate, which is shown in bold. The correlation between the estimates for ϵ and LOG(ζ) was 62.4 percent, supporting the need for our SUR estimation approach.

EQUATION 1. The negative estimated coefficient of the variable APPLICATION (α_2 = −0.27, .003 level) indicates that batch processing applications development required more new code to be developed, and thus less reuse, on average than online, real-time projects. In addition, the coefficient for MATURITY (α_1 = −0.12, .11 level) was also negative, suggesting that the development of new application functionality required increasingly less new code in Year 2. The estimated value of the intercept is also noteworthy (α_o = 0.71, .001 level). The intercept is significantly less than 1.00, providing evidence that HPS has a beneficial effect overall in reducing the proportion of new objects that must be built in application development.

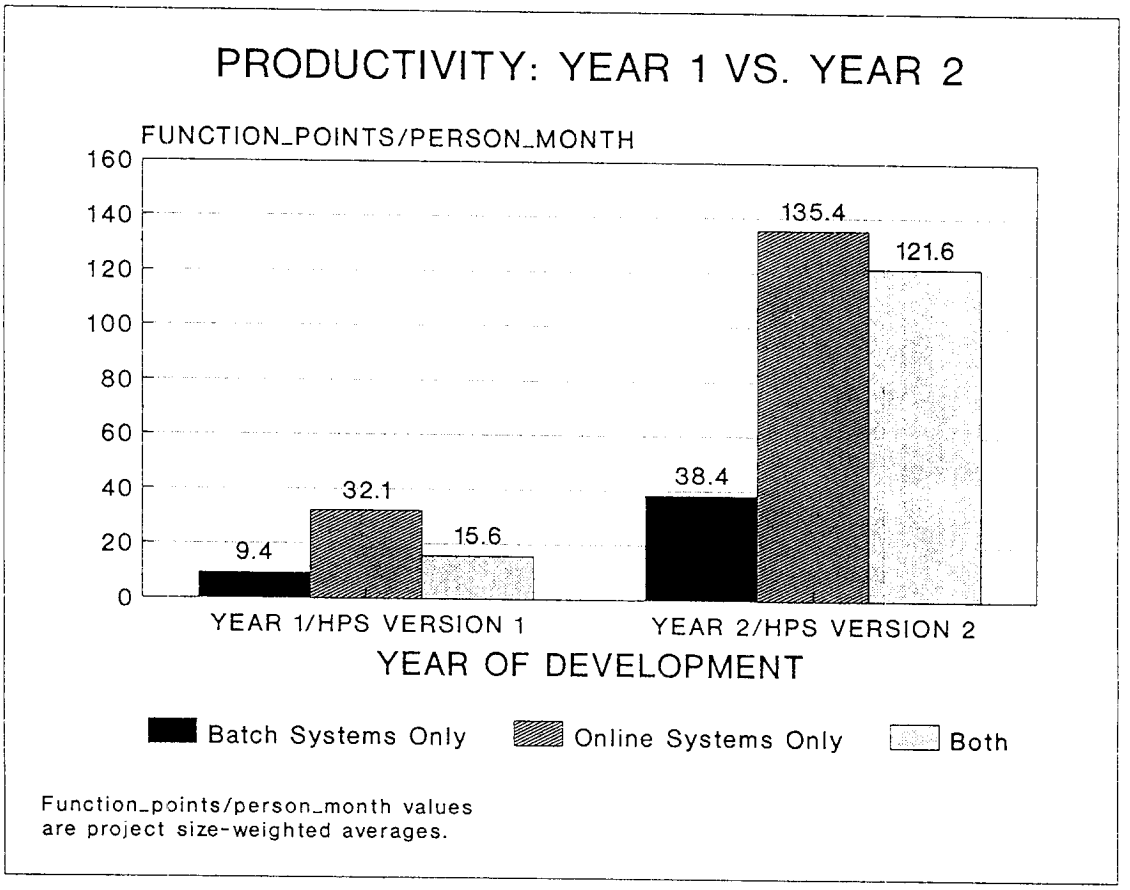
EQUATION 2. The results of the estimation of EQUATION 2 show that PERSON_DAYS of effort (β_1 = 0.55, .001 level) are substantially leveraged by object reuse (β_2 = −1.92, .001 level) and the MATURITY of the ICASE environment (β_3 = 0.40, .11 level) in the production of FUNCTION_POINTS. Again, a substantial amount of learning was occurring between Year 1 and Year 2 in the use of HPS. The variable for APPLICATION type (β_4 = −0.04, .46 level) was not significant at conventional levels, suggesting that if there is an effect on productivity, it occurs largely through reuse.[11] The coefficient for NEW_OBJECT_PCT (β_2 = −1.92, .001 level) has a very straightforward interpretation in the original multiplicative model: it can be interpreted as a 1.92 percent productivity gain associated with a 1 percent decrease in the value of NEW_OBJECT_PCT (or a 1 percent increase in reuse) in the range of project sizes that we observed. The β_3 parameter (β_3 = 0.40, .11 level) estimate can be interpreted in a similar way. The impact of an additional year of organizational experience with the CASE tool is to amplify the productivity of development labor by a factor of $e^{.40}$ = 1.49 times, because the value of the binary MATURITY variable is e to indicate Year 2.

The presence of diseconomies of scale for development labor (β_1 = 0.55, .001 level) suggests that higher productivity can be achieved with smaller-size projects. (Note: In our multiplicative model, a coefficient estimate of greater than 1 indicates the presence of scale economies, an estimate of 1.0 indicates constant returns to scale, and an estimate of less than 1 indicates diseconomies of scale.) This is probably due to the considerable complexity of managing large software projects in a new development environment.

Was the IS strategy delivering software development performance gains?

The average productivity and object reuse by application type and year are presented in Figures 3 and 4. Their implications for the performance

[11] When we estimated this model without the APPLICATION type variable in EQUATION 2, we found that the estimated coefficient of NEW_OBJECT_PCT rose to about −1.57, yet the estimated coefficients of the other variables were largely unaffected. This suggests that APPLICATION type affected productivity through reuse.



F'gure 3. Year 1 and Year 2 Productivity Comparisons

of the firm's IS strategy are discussed in the remainder of this section.[12]

As can be seen in Table 3, it appears there is greater productivity in the development of online, real-time systems, with a range of between 9.1 and 287.8 function points per person month. The proportion of new objects built (NEW_OBJECT_PCT) ranged from 16.1 percent to 48.1 percent, equivalent to reuse levels of 83.1 percent and 51.9 percent respectively. Batch system development productivity, by contrast, ranged between 4.2 and 38.4 function points per person month. One application, Project #2, exhibited no reuse at all. And the smallest proportion of new objects built for a batch application was observed for Project #20. In this instance, NEW_OBJECT_PCT was 32.8 percent, consistent with a 67.2 percent level of reuse.

Differences between Year 1 and Year 2 project productivity performance are also evident from the weighted averages (see Figure 3, which presents averages weighted for project size). Online application project development productivity grew from 32.1 to 135.4 function points per person month, and batch development productivity also grew from 9.4 to 38.4 function points per person month. The combined averages for online and batch projects evidenced an order of magnitude of growth, from 15.6 to 121.6 function

[12] A potential approach to the analysis of these data would have been to conduct simple t-tests to gauge whether *significant differences* exist between the means for the productivity results by application type in Years 1 and 2, and for the NEW_OBJECT_PCT numbers as well. However, such a test would not fairly represent the characteristics of our structural model, especially because of the existence of scale diseconomies and correlated error terms.

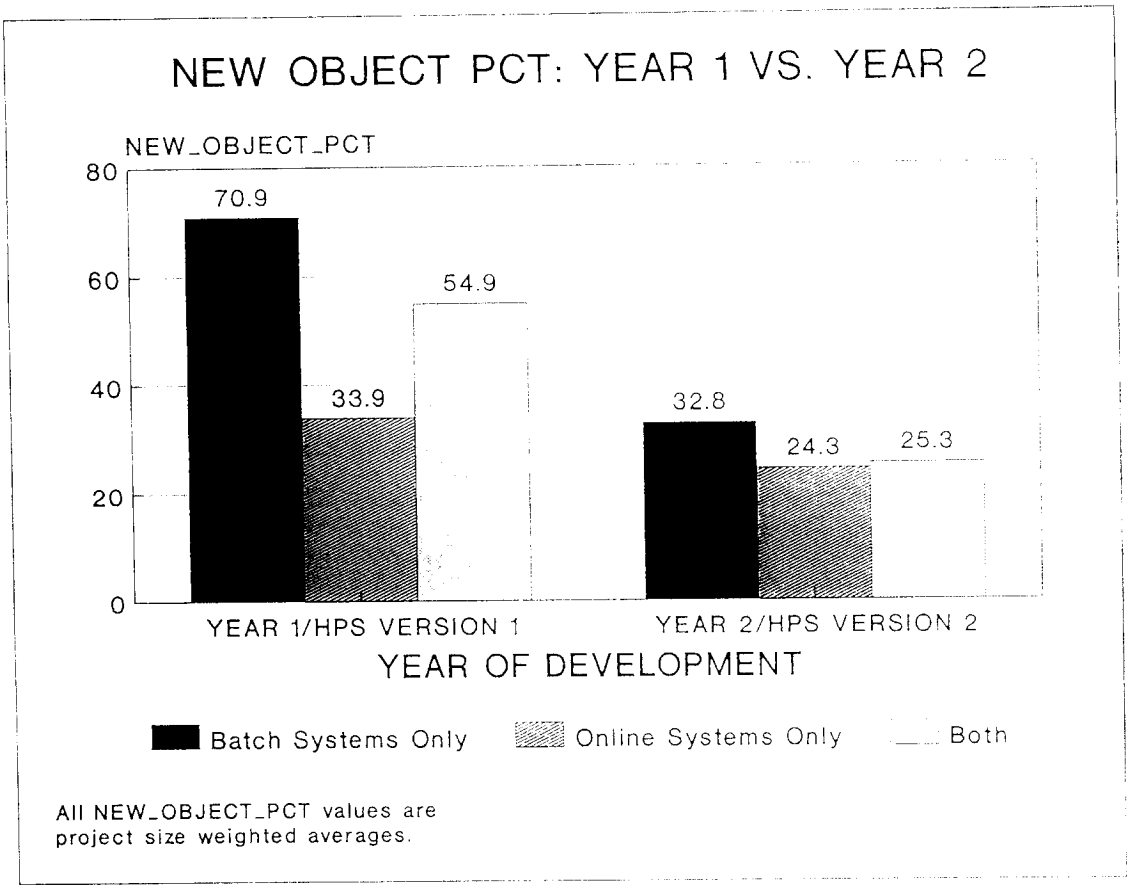


Figure 4. Year 1 and Year 2 Reuse Comparisons

points per person month between Years 1 and 2. If the estimated figure of 8 function points per person month is used for traditional development prior to the deployment of HPS, there appear to have been efficiency gains that were made during the first year of development.

The overall weighted average for proportion of new objects built (NEW_OBJECT_PCT) for the 20 projects was 39.8 percent. The weighted average NEW_OBJECT_PCT among Year 1 projects was 54.9 percent, which fell to 25.3 percent for Year 2 projects. The construction of new code was minimized among online projects at 24.3 percent of the total application objects in Year 2. The big potential for additional gains appears to have been in the creation of batch processing projects, however. Although our data set only includes one observation from Year 2 batch development, NEW_OBJECT_PCT fell from 70.9 percent to 32.8 percent. This suggests that potential efficiency gains are available from the HPS version 2 tool set for batch development.

Conclusion

This section reviews the major results of our research and discusses their generalizability, as well as the generalizability of the model employed to obtain them.

Major findings at the First Boston Corporation

This article reports on the productivity gains that can result from the implementation of an IS strategy that was consciously adopted to promote the development and delivery of customizable

and rapidly deployable, high functionality cooperative processing software applications, while controlling software development costs. The First Boston Corporation's investment in HPS was the solution to this problem, and the technical vision behind it involved three important related elements:

1. Emphasis on software reusability and automated code generation;
2. Storage of reusable software in a centralized object repository;
3. Integration of the CASE tool set across the development life cycle.

The evidence we presented suggests that the deployment of HPS was an essential first step in achieving the goals that senior management had set, and that effective software reuse was a precondition for this success. The levels of productivity reported represent an order of magnitude gain over the eight to 10 function points per person month productivity estimates for prior development at First Boston and for Capers Jones' national sample of MIS business applications (Bouldin, 1989). Further, the results of our economic model of ICASE reuse and productivity are of special interest: they offer evidence to confirm the importance of software reusability in the achievement of improved productivity, despite the increasing functionality and complexity of the software applications that were built.

The productivity gains that we observed were indeed substantial—high enough, in fact, to provide First Boston's senior software development managers with confirmation that the technical vision they implemented to support the firm's IS strategy was working well. First Boston's software development performance also exceeded Capers Jones' estimates for development productivity of 15 function points per person month prior to the maturation of a newly deployed CASE tool (Bouldin, 1989). In fact, the firm is likely to be achieving an even higher level of software development productivity than initially estimated by software development project managers. This kind of feedback reinforces the importance of implementing a software metrics program to track reusability and productivity, the final step we recommend in our analysis of the link between IS strategy and business strategy (see Figure 1).

First Boston's investment in HPS was undertaken with the intent of obtaining competitive advantage in operations with high functionality software, while at the same time controlling software development costs. Interestingly, First Boston Corporation sold its NTPA software in 1988 to Kidder Peabody (Arend, 1988), a large investment banking competitor, thus creating an opportunity for senior management to generate additional cash flows to defray the costs of implementing its IS strategy.

On the generalizability of the model and results

One of the most important questions for practitioners who read this article is: How generalizable is the modeling approach employed by this research? In addition, to what extent is First Boston's experience with a software reusability strategy, improved productivity, and ICASE tools likely to be transferrable to other organizations? And, overall is CASE necessary to obtain the kind of productivity gains reported in this article?

Although prior literature on software development productivity was utilized for deciding on how to model what we saw and how to derive results from the descriptive model that emerged, we emphasize one of the limitations of this study: it investigates a model of software development productivity that is specific to what was learned about ICASE and reuse *at First Boston*. Thus, as a next step it would be valuable to examine additional data from other sites that have deployed HPS to determine whether reuse, application type, and the maturity of the ICASE tool set have similar productivity effects. But even this effort would only yield information about a single tool. Some preliminary insights are presented below.

The Case of Carter Hawley Hale, Inc. Carter Hawley Hale (CHH), a large Los Angeles-based retailing firm, is one such site from which additional conclusions might be drawn about our modeling approach and the efficacy of a reusability strategy involving HPS. The firm purchased HPS to support a sotware reusability strategy in its corporate systems development activities. Although we are unable to offer any quantitative information on development productivity or reuse levels, this model would be testable in the CHH environment, because the HPS tool set continues

to be upgraded over time, and learning effects are still observed as developers' use of the tool set matures. In addition, reuse at CHH is primarily object-based; CHH bought only the tool, not the NTPA systems as in Kidder Peabody's case (below). There also is preliminary evidence that the reusability strategy is working well. CHH, recently struggling with financial difficulties, filed for protection from creditors under Chapter 11 early in 1991. Although HPS had only recently been implemented at the firm, management decided to use it to enable deployment of a crucial, high functionality creditor claims management system in a short period of time, while under great pressure to cut software development costs.

The Case of Kidder Peabody. Additional conclusions can be drawn about the generalizability of our model from Kidder Peabody's experience with HPS. Kidder Peabody is a firm that has taken the reusability strategy even farther. Its purchase of NTPA was made with the idea that NTPA would be customized from First Boston's reusable object building blocks and repository models of core investment banking industry systems for use in a different organizational environment with different business requirements. In this case, not only would reuse be object-based, it would also be *model-based*,[13] leading to additional gains in productivity that were not possible in First Boston's primary development activities, and which would not be adequately addressed by the model we proposed. Because this kind of development is similar to system enhancement in traditional software engineering, it points out the need for a model that can meaningfully measure performance when reuse extends to the design phase (Lanergan and Grasso, 1984) or the maintenance/enhancement phase of development (Basili, 1990; Rombach, 1991).

Application Type and ICASE Tool Set Feature Evaluation. Because some shortcomings of HPS' handling of batch development were eliminated with the release of version 2, examining batch versus online systems development now would only be of interest to determine how well the capabilities of the ICASE tool set have developed. Management, however, also may want to examine other aspects of the applications that they build to determine how well the ICASE tools support development. We expect that no ICASE tool will support all tasks equally well, and inclusion of a relevant operationalization of the application type variable could help to confirm other strengths or weaknesses. Thus, with a more general treatment, the application-type variable should continue to be of interest to those who do single tool/single site, single tool/multi-site, or multi-tool/single site research on CASE productivity.

The Reuse Model. One of the most generalizable aspects of the model that we present is the inclusion of reuse as a driver for productivity. Equally important, however, is the realization that high levels of reuse are not automatic when repository and object-based ICASE tools are deployed. One researcher recently pointed out that "[r]euse is enabled by some development process and both reusable components and the reuse process employed need to be tailored to and integrated into that development process model" (Rombach, 1991, p. 89). Thus, a useful model for CASE productivity evaluation is likely to require a model for software reuse evaluation embedded within it.

Although we identified application type and ICASE maturity as drivers for reuse in the First Boston model, a larger set of factors must be considered in a more general model to evaluate reuse (Banker, et al., 1990). The factors would include:

1. Technical qualities of the tool (e.g., ease of reusable object identification and retrieval, and functionality of reusable objects);
2. Development team characteristics (e.g., size of the team, expert-novice composition, and relative knowledge of the repository);
3. Organizational factors (e.g., an incentive system to promote reuse by developers, and project reuse level targets); and,
4. Architectural factors (e.g., the extent to which reusability is leveraged in systems planning, and whether the development environment integrates reusability across life cycle phases).

Note that each of these classes of variables offers management an opportunity to redesign the systems development environment to promote reuse in a cost-effective manner (Barnes and Bollinger, 1991; Bollinger and Pfleeger, 1990). Considering such variables will become increas-

[13] Personal communication with Vivek Wadhwa, Seer Technologies, July 10, 1991.

ingly important in the future as firms move forward from merely implementing ICASE to maximizing its effectiveness.

The Reuse Metric. The reuse metric that we have presented, NEW_OBJECT_PCT, is readily generalized. It can represent the percentage of new code developed in a variety of software engineering environments. For example, this metric could be operationalized as "new lines of CASE code," "new objects built" in other entity relationship attribute object-based tools (such as IEF), or "new objects classes" built in object-development (Rubin, 1990). We should also point out that a more immediate metric for reuse is given by 1 - NEW_CODE_PCT, and that this metric can also be employed where management's focus is on the level of reuse, rather than the level of new code built.

Clearly, this research is only a first step in developing an understanding of reuse and productivity in ICASE environments. Additional work at other sites is needed to probe the impacts of other tools that are based on entity relationship attribute data modeling, as well as those that are based on object-oriented data modeling (Chen and Sibley, 1991). We make no claim that the model presented in this article is a general model, though we argued that it can be generalized along a number of dimensions. Nor have we shown that ICASE is a prerequisite for reuse, since high levels of reuse could be obtained from CASE tools that do not integrate the entire life cycle. (In fact, a repository may be more important.) However, the capabilities that ICASE provides to integrate phase-specific outputs for use across the life cycle may well be necessary to translate the potential that a reusability strategy offers into meaningful software development productivity gains.

Acknowledgements

We wish to thank Mark Baric, Gene Bedell, Tom Lewis, and Vivek Wadhwa of Seer Technologies for providing us access to data on software development projects and offering their own and their managers' time. We also thank Charles Wright, Eric Fisher, and Vannevar Yu for assistance with the collection of the software project data. We acknowledge four anonymous reviewers, Macedonio Alanis, Cynthia Beath, Eric Clemons, Rosann Collins, Gordon Davis, Gordon Everest, Jean Kauffman, Chris Kemerer, Rachna Kumar, Dani Zweig, and the participants of research seminars at the University of Minnesota, New York University, the University of Pennsylvania, the University of Rochester, and the University of California, Irvine for their helpful comments and suggestions. Finally, we thank the National Science Foundation (grant #SES-8709044) for partial funding of data collection.

References

Albrecht, A.J. and Gaffney, J.E. "Software Function, Source Lines of Code, and Development Effort Prediction: A Software Science Validation," *IEEE Transactions on Software Engineering* (9:6), November 1983, pp. 639–647.

Apte, U., Sankar, C. S., Thakur, M., and Turner, J. "Reusability Strategy for Development of Information Systems: Implementation Experience of a Bank," *MIS Quarterly* (14:4), December 1990, pp. 421–431.

Arend, M. "Kidder, First Boston Get on Each Other's CASE," *Wall Street Computer Review,* September 1988, pp. 86–90.

Bailey, J. W. and Basili, V. R. "A Meta-model for Software Development Resource Expenditures," *Proceedings of the 5th International Conference on Software Engineering*, 1981, pp. 107–116.

Banker, R. D. and Kauffman, R. J. "Automated Software Metrics, Repository Evaluation and the Software Asset Management Perspective," working paper, Center for Information Systems, Stern School of Business, New York University, New York, NY, 1991.

Banker, R. D. and Kemerer, C. F. "Scale Economies in New Software Development," *IEEE Transactions on Software Engineering* (15:10), October 1989, pp. 1199–1205.

Banker, R. D., Fisher, E., Kauffman, R. J., Wright, C., and Zweig, D. "Automating Software Development Productivity Metrics," working paper, Center for Research on Information Systems, Stern School of Business, New York University, New York, NY, 1990a.

Banker, R. D., Kauffman, R. J., and Morey, R. C. "Measuring Gains in Operational Productivity from Information Technology: A Study of the Positran Deployment at Hardee's Inc.,"

Journal of Management Information Systems, Fall 1990b.

Banker, R. D., Datar, S., and Kemerer, C. "A Model to Evaluate Variables Impacting the Productivity of Software Maintenance," *Management Science* (37:1), January 1991, pp. 1-18.

Barnes, H. B. and Bollinger, T. "Making Software Reuse Cost Effective," *IEEE Software* (8:1), January 1991.

Basili, V. "Viewing Maintenance as Reuse-Oriented Software Development," *IEEE Software* (7:1), January 1990, pp. 19–25.

Behrens, C. A. "Measuring the Productivity of Computer Systems Development Activities with Function Points," *IEEE Transactions on Software Engineering* (9:6), November 1983, pp. 648–651.

Belady, L. A. and Lehman, M. M. "The Characteristics of Large Systems," in *Research Directions in Software Technology*, P. Wegner (ed.), MIT Press, Cambridge, MA, 1979, pp. 106–138.

Biggerstaff, T. and Richter, C. "Reusability Framework, Assessment and Directions," *IEEE Software* (4:2), March 1987, pp. 41–49.

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

Boehm, B. and Papaccio, P. N. "Understanding and Controlling Software Costs," *IEEE Transactions on Software Engineering* (14:10), October 1988, pp. 1462–1477.

Bollinger, T. B. and Pfleeger, S. L. "Economics of Reuse: Issues and Alternatives," *Information and Software Technology* (32:10), December 1990.

Booch, G. "What Is and What Isn't Object-Oriented Design," *Ed Yourdon's Software Journal* (2:7-8), Summer 1989, pp. 14–21.

Bouldin, B. M. "CASE: Measuring Productivity—What Are You Measuring? Why Are You Measuring It?" *Software Magazine* (9:10), August 1989, pp. 30–39.

Breidenbach, S. "Developers Dump Hosts for PC LANs," *Network World*, March 20, 1989.

Chen, M. and Sibley, E. H. "Using a CASE-Based Repository for Systems Integration," *Proceedings of the 1991 Hawaii International Conference on Systems Sciences*, IEEE, January 1991, pp. 578-587.

Chen, P. P. "The ER Model Toward a Unified View of Data," *ACM Transaction on Database Systems* (1:1), 1976, pp. 9–36.

Clemons, E. "Evaluating Investments in Strategic Information Technologies," *Communications of the ACM*, January 1991.

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

Davis, G. B. "Commentary on Information Systems: Productivity Gains from Computer-Aided Software Engineering," *Accounting Horizons* (2:2), June 1988, pp. 90–93.

DeMarco, T. "Yourdon Project Survey: Final Report," Yourdon Inc., Technical Report, 1981.

Desmond, J. "Tools Are Needed for Race to the Desktop," *Software Magazine* (9:9), July 1989.

Dreger, J. B. *Function Point Analysis*, Prentice-Hall, Englewood Cliffs, NJ, 1989.

Edelstein, H. "Cooperative Processing Applications Expanding in Number," *Software Magazine* (11:2), December 1989, pp. 39–45.

Feder, B. "The Software Trap: Automate—Or Else," *Business Week*, May 9, 1988.

Fisher, J. T. "IBM's Repository: Can Big Blue Establish OS/2 EE as the Professional Programmer's Front End?" *DBMS*, January 1990, pp. 42–49.

Gabel. "A Yen for Just-in-time Decisions Aids Sony's Drive for Coprocessing," *Computerworld*, April 10, 1989.

Gaffney, J. E., Jr. "Estimation of Software Code Size Based on Quantitative Aspects of Function (with Application of Expert Systems Technology)," working paper, IBM Federal Systems Division, Advanced Technology Department, Gaithersburg, MD, 1986.

Gaffney, J. E., Jr. and Durek, T. A. "Software Reuse—Key to Enhanced Productivity: Some Quantitative Models," *Information and Software Technology* (31:5), June 1989, pp. 258–267.

Grammas, G. W. and Klein, J. R. "Software Productivity as a Strategic Variable," *Interfaces* (15:3), May-June 1985, pp. 116–126.

Hazzah, A. "Making Ends Meet: Repository Manager," *Software Magazine*, December 1989, pp. 59–72.

Horowitz, E. and Munson, J. "An Expansive View of Reusable Software," *IEEE Transactions on Software Engineering* (SE-10:3), September 1984, pp. 477–487.

IFPUG. *Proceedings of the International Function Points Users Group*, International Func-

tion Point Users' Group, 1988.

Jones, T. C. "Reusability in Programming: A Survey of the State of the Art," *IEEE Transactions on Software Engineering* (SE-10:5), September 1984, pp. 484–494.

Jones, T. C. *Programming Productivity*, McGraw-Hill, New York, NY, 1986.

Jones, T. C. "A New Look At Languages," *Computerworld*, November 7, 1988.

Judge, G. G., Griffiths, W. E., Hill, R. C., Lutkepohl, H., and Lee, T.-C. *The Theory and Practice of Econometrics, Second Edition*, John Wiley and Sons, New York, NY, 1985.

Kang, K. C. and Levy, L. S. "Software Methodology in the Harsh Light of Economics," *Information and Software Technology* (31:5), June 1989, pp. 239–249.

Kemerer, C. F. "An Empirical Valuation of Software Cost Estimation Models," *Communications of the ACM*, May 1987.

Kemerer, C. F. "An Agenda For Research in the Managerial Evaluation of Computer-Aided Software Engineering (CASE) Tool Impacts," *Proceedings of the 22nd Hawaii International Conference on Systems Sciences*, Kailuo-Kona, HI, January 1989.

Kemerer, C. F. "Reliability of Function Points Measurement: A Field Experiment," working paper, Sloan School of Management, MIT, Boston, MA, December 1990.

Kernighan, B. W. "The UNIX System and Software Reusability," *IEEE Transactions on Software Engineering* (SE-10:5), September 1984, pp. 513–518.

Kim, Y. and Stohr, E. A. "Software Reuse: Issues and Research Directions," working paper, Center for Research on Information Systems, Stern School of Business, New York University, New York, NY, June 1991.

Knight, J. "CASE Up On MIS Agenda," *Software Magazine*, August 1989, pp. 32–36.

Lanergan, R. G. and Grasso, C. A. "Software Engineering with Reusable Designs and Code," *IEEE Transactions on Software Engineering* (SE-10:5), September 1984, pp. 498–501.

Lenz, M., Schmid, H. A., and Wolfe, P. F. "Software Reuse Through Building Blocks," *IEEE Software* (4:4), July 1987, pp. 34–42.

Loh, M. and Nelson, R. R. "Reaping CASE Harvests," *Datamation*, July 1, 1989, pp. 31–33.

Low, G. C. and Jeffrey, D. R. "Function Points in the Estimation and Evaluation of the Software Process," *IEEE Transactions on Software Engineering* (16:1), January 1, 1990, pp. 64–71.

Mathis, R. F. "The Last 10 Percent," *IEEE Transactions on Software Engineering* (SE-12:6), June 1986, pp. 705–712.

Matsumoto, Y. "Some Experiences in Promoting Reusable Software: Presentation in Higher Abstract Levels," *IEEE Transactions on Software Engineering* (SE-10:5), September 1984, pp. 502–512.

McGuff, F. P. "Cost Cutting Revisited, *Computerworld*, July 17, 1989.

Meyer, B. *Object-Oriented Software Construction*, Prentice-Hall, New York, NY, 1988.

Moad, J. "Cultural Barriers Slow Reusability," *Datamation*, November 15, 1989, pp. 87-92.

Neighbors, J. M. "The DRACO Approach to Constructing Software from Reusable Components," *IEEE Transactions on Software Engineering* (SE-10:5), September 1984, pp. 564–574.

Nezlek, G. S. and Leitheiser, R. L. "Towards Developing a Coherent Research Framework to Measure CASE Effectiveness," *Proceedings of the 24th Hawaii International Conference on Systems Sciences*, January 1991, pp. 438–445.

Norman, R. J. and Nunamaker, J. F., Jr. "CASE Productivity Perceptions of Software Engineering Professionals," *Communications of the ACM* (32:9), September 1989, pp. 1102–1108.

Nunamaker, J. F., Jr. and Chen, M. "Software Productivity: A Framework of Study and an Approach to Reusable Components," *Proceedings of the 22nd Hawaii International Conference on Systems Sciences*, Kailua-Kona, HI, January 1989a, pp. 959–968.

Nunamaker, J. F., Jr. and Chen, M. "Software Productivity: Gaining Competitive Edges in an Information Society," *Proceedings of the 22nd Hawaii International Conference on Systems Sciences*, Kailua-Kona, HI, January 1989b, pp. 957–958.

Parker, J. and Hendley, B. "The Re-usage of Low Level Programming Knowledge in the UNIVERSE Programming Environment," in *Software Engineering Environments*, P. Brereton (ed.), Halstead Press, New York, NY, 1988.

Rombach, H. D. "Software Reuse: A Key to the

Maintenance Problem," *Information and Software Technology* (33:1), January/February 1991.

Rubin, K. "Reuse in Software Engineering: An Object-Oriented Perspective," *Proceedings of COMPCON*, 1990, pp. 340–346.

Scacchi, W. "Understanding Software Productivity: A Comparative Empirical Review," *Proceedings of the 22nd Hawaii International Conference on System Sciences*, Kailua-Kona, HI, January 1989, pp. 969–977.

Schwartz, E. "IBM Sets Sights on Financial Services," *Computer Systems News*, March 19, 1990.

Software Magazine. "CASE Growth Will Skyrocket," March 1988, p. 16.

Sperling, E., Schwartz, E., and Gerber, C. "IBM Makes CASE for Coding," *Computer Systems News*, September 25, 1989.

Sprouls, J. (ed.). *IFPUG Function Point Counting Practices Manual, Release 3.0*, International Function Point Users Group, Westville, OH, 1990.

Standish, T. A. "An Essay on Software Reuse," *IEEE Transactions on Software Engineering* (SE-10:5), September 1984, pp. 494–497.

Symons, C. R. "Function Point Analysis: Difficulties and Improvements," *IEEE Transactions on Software Engineering* (14:1), January 1988, pp. 2–10.

Teory, T. Y., Yang, D., and Fry, J. P. "A Logical Design Methodology for Relational Database Using the Extended Entity-Relationship Model," *ACM Computing Surveys* (18:2), June 1986, pp. 197–222.

Tracz, W. "Software Reuse: Motivations and Inhibitors," *Proceedings of COMPCON 87*, San Francisco, CA, February 1987, pp. 358–363.

Voelckner, J. "Automating Software: Proceed with Caution," *IEEE Spectrum*, July 1988.

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

Wingfield, C. G. "USACSC Experience with SLIM," US Army Institute for Research in Management Information and Computer Science, Technical Report 360-5, 1982.

Wong, W. "Management Overview of Software Reuse," Technical Report PB87-109856, National Bureau of Standards, Gaithersburg, MD, 1987.

Zellner, A. "An Efficient Method for Estimating Seemingly Unrelated Regressions and Tests for Aggregation Bias," *Journal of the American Statistical Association* (57), 1962, pp. 348–368.

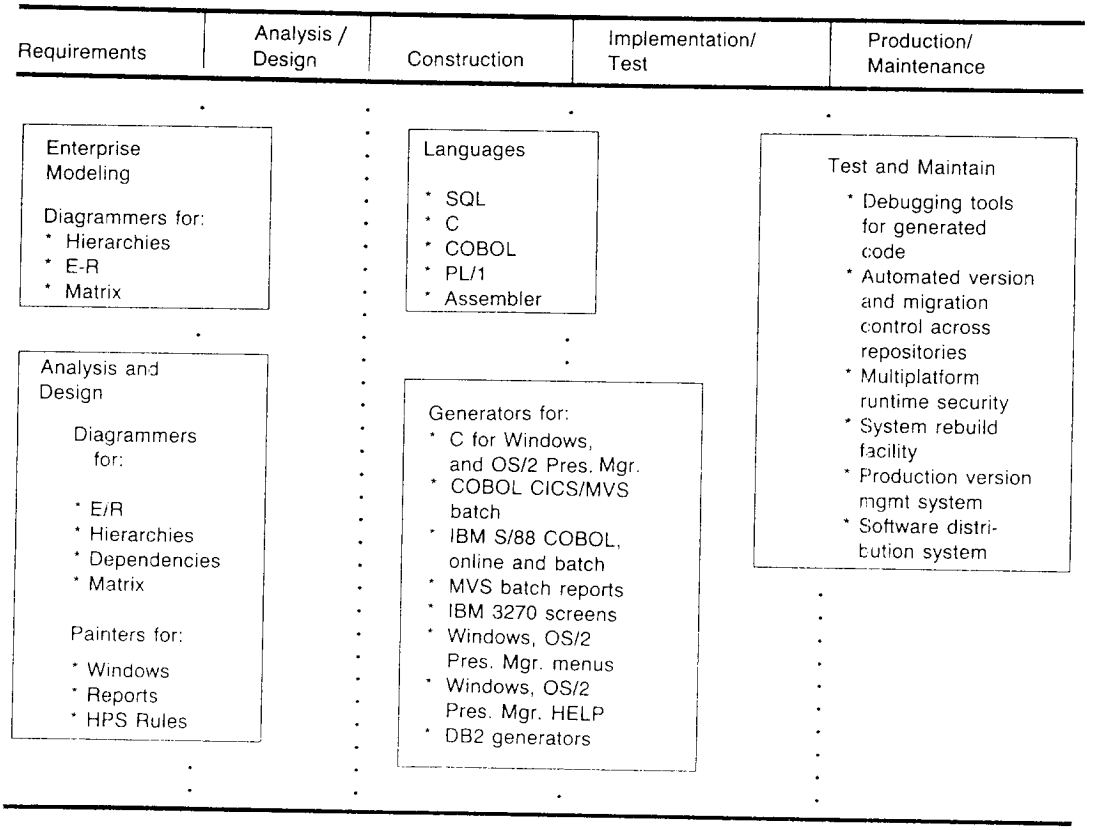
About the Authors

Rajiv D. Banker is the first Arthur Andersen & Co./Duane R. Kullberg Chair in Accounting and Information Systems at the Carlson School of Management, University of Minnesota. He has published numerous articles on information systems, management accounting, operations management and productivity analysis. His current research interests include information systems development and maintenance productivity, computer-aided software engineering performance metrics and evaluation of business value of information technology.

Robert J. Kauffman is an assistant professor at the Stern School of Business at New York University, where he has taught since 1988. He completed a masters degree at Cornell University, and was later employed as an international lending and strategic planning officer at a large money center bank in New York City. He received a doctorate in information systems from the Graduate School of Industrial Administration, Carnegie Mellon University in 1988. His current program of research involves developing new methodologies for measuring the business value of a broad spectrum of information technologies, using techniques from management science and economics.

Appendix 1
The High Productivity Systems (HPS) ICASE Development Tool Set

HPS, the object and repository-based ICASE tool studied in this research, provides support across the systems development life cycle through a series of integrated tools, as shown below:



The HPS application development platform consists of the following features:

- A central DB2-based repository, with PS/2-based local development repositories;
- A technical documentation generator and manager;
- Reuse enabling facilities and management tools;
- Project level security and control;
- Source code library management facilities.

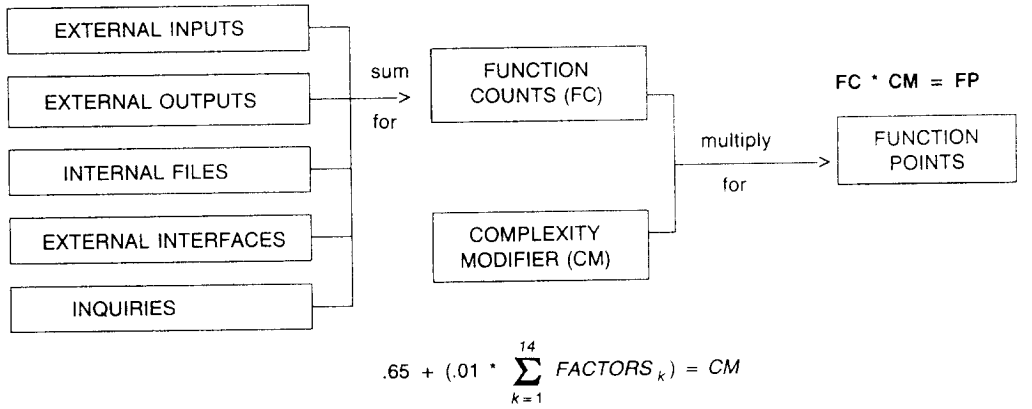
HPS promotes reuse. Developers can search the centralized repository by object type, object name, or keywords that contain an object's description. Once identified, objects can be reused to create functionality in new applications.

Appendix 2
An Overview of the Function Point Analysis Methodology

The function point analysis methodology for the measurement of the size of a software project is based on the identification of functions performed by the software. The methodology was originally developed by Allan Albrecht at IBM. Since then, the methodology has developed, with the help of a national users group, into an operationally well-defined methodology (Dreger, 1989). It continues to gain in popularity, despite the extent of the effort required to analyze a system of moderate size, because of its robustness across different programming environments and its usability as an early life cycle labor estimation tool. (For a description of the International Function Point Users Group Standard, Release 3.0, the interested reader is referred to Sprouls, 1990.)

Function point analysis has two primary components: function counts and complexity measures. *Function counts* (FC) represent a basic measure of the user functionality of a system, independent of the technical features of implementation. The *complexity modifier* (CM) expression provides the final adjustment to the function count obtained to reflect the degree of technical difficulty involved in implementing a system.

$$\sum_{i=1}^{5} \sum_{j=1}^{3} WEIGHT_{ij}\, FUNCTION_{ij} = FC$$

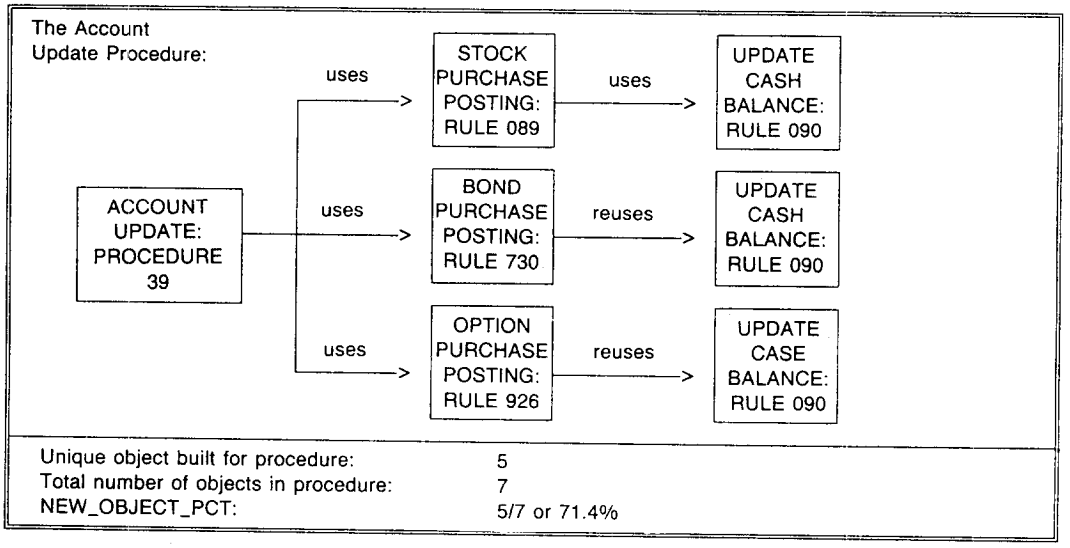


$$.65 + (.01 * \sum_{k=1}^{14} FACTORS_k) = CM$$

Function counts are broken down into five FUNCTION types (i = 1 to 5): external inputs, external outputs, internal files, external interfaces, and inquiries. An application's function count is the sum of the scores an application achieves on each of the external function types. Function counts are determined by applying WEIGHTS (j = 1 to 3; LOW/MEDIUM/HIGH) from a set of tabulated values that represent the number of file types referenced and data elements associated with each occurrence of a FUNCTION type.

The function points method also includes 14 technical FACTORS (k) for implementation that are rated on a scale of "0" to "5." A complexity measure of "0" represents the absence of a factor and thus no adjustment to the original function count, while a "1" means that the technical complexity factor was expected to play an important role in influencing labor consumption. Examples of the factors include batch/online systems, complexity of mathematical logic, required level of reliability, and stability of the development environment. From the above formula, we see that function points are mainly based on function counts. When a system exhibits an average level of development environment complexity, the complexity modifier (CM) will take on the value 1. Less than average complexity reduces functionality, while greater than average complexity increases it.

Appendix 3
An Illustration of the New Object Percentage Metric



Appendix 3 illustrates a software reuse metric called NEW_OBJECT_PCT that measures the extent of the new code that must be developed for an ACCOUNT UPDATE PROCEDURE. In this example, no objects from prior applications are used for development; however, the UPDATE CASE BALANCE RULE 090—built for the first time here—is reused twice. This results in a NEW_CODE_PCT of 71.4 percent. If the CASH BALANCE UPDATE RULE 090 had been available in the repository from a prior software development effort, this would result in a reduction of NEW_OBJECT_PCT to 57.1 percent. Note that we exclude multiple calls to the same object (e.g., a control structure in traditional programming that involves a loop or recursion) in our calculation of reuse.