MEASURING THE DEVELOPMENT PERFORMANCE OF INTEGRATED COMPUTER AIDED SOFTWARE ENGINEERING (I-CASE): A SYNTHESIS OF FIELD STUDY RESULTS FROM THE FIRST BOSTON CORPORATION

Rajiv D. Banker

Robert J. Kauffman

Department of Information, Operations, and Management Sciences

Leonard N. Stern School of Business, New York University

44 West 4th Street, New York, NY 10012

MEASURING THE DEVELOPMENT PERFORMANCE OF INTEGRATED COMPUTER AIDED SOFTWARE ENGINEERING (I-CASE): A SYNTHESIS OF FIELD STUDY RESULTS FROM THE FIRST BOSTON CORPORATION

RAJIV D. BANKER
Andersen Chair in Accounting and Information Systems
Carlson School of Management
University of Minnesota
Minneapolis, MN 55455

ROBERT J. KAUFFMAN
Associate Professor of Information Systems
Stern School of Business
New York University
44 West 4th Street
New York, NY 10012

INTRODUCTION

The First Boston Corporation, a large investment bank in New York City, began to build its own integrated computer aided software engineering (**I-CASE**) tool in 1986. This decision was made following a comprehensive survey of the market for CASE tools available at that time. This resulted in a determination that there would be no tools commercially available within the next few years that would:

(1) enable cost-effective expansion of the firm's current applications to support the demand for increased financial market trades processing in a 24-hour a day, global market;

(2) create high functionality, multi-tiered cooperative processing applications that efficiently utilize the power and flexibility of --

* microcomputers and engineering workstations on the trading platform;

Working Paper Series
STERN IS-92-34

* fault-tolerant minicomputers for intraday trades processing and a link to the financial markets;
* mainframe computers for current account and firm securities inventory management, and historical database queries to support trading analytics; and,

(3) further control costs by paring down the overall level of developer expertise that needed to be brought together to create the firm's applications.

Following in-house development of "High Productivity Systems" (**HPS**), an I-CASE tool set that supports the development of reusable software, First Boston's next step was to rebuild and roll out the core applications that formed its investment banking software architecture.

A number of research questions were on our and management's agenda when we began to examine software development using HPS at First Boston. These included:

(1) To what extent did I-CASE support the software development process, leading to improved productivity and higher quality applications?

(2) Did software reuse drive the results?

(3) Are the gains recognizable in small-scale experimental project development?

(4) If so, can they also be replicated in large-scale application development?

This paper provides some insights to these questions by presenting the results of two phases of a multi-year field study that was carried out at the First Boston Corporation. The *first* phase involved three exploratory I-CASE development experiments in which we closely examined development performance. The *second* phase involved data collection to support an empirical study of twenty large-scale software development projects representing the bank's I-CASE-built *New Trades Processing Architecture* (**NTPA**).

We first turn to a more in-depth discussion of the results of the three experimental development projects. Thereafter, we will examine the results of the development in the second phase of the project. We conclude with some ideas on measurement and management approaches to improve the performance of I-CASE development activities.

EVIDENCE FROM SMALL-SCALE EXPERIMENTAL PROJECTS DEVELOPED USING I-CASE

A useful approach to measuring the potential productivity impacts of automated software engineering techniques is to examine how the process of development proceeds in an experimental setting. The reasons for this are threefold:

(1) When a software project is developed as an experiment, the analyst has the opportunity to carefully craft the specifications for the project. This ensures that the developer will focus on developing the kind of system using the tools that management wishes to understand better.

(2) Since the specifications of the product can be controlled and the developer's work can be closely monitored, it is possible to get a more accurate measurement of development productivity for an experimental project than for a real one.

(3) Monitoring the developer also helps the analyst to understand the process behind the product. This enables the analyst to go one step farther: to gain an understanding of what factors drive the level of software development productivity that is subsequently observed.

We applied this approach to estimate the productivity gains that First Boston's HPS delivered for development of three small experimental applications:

(1) a retail industry information system that offers store, district and head office query and data processing capabilities;

(2) an investment banking executive information system; and,

(3) an investment banking trading workstation front-end.

Each was designed to exhibit a high level of user functionality and also require at least two-tier (microcomputer and mainframe) cooperative or client-server processing. Based on surveys of project managers in related work we conducted at First Boston, we learned that developing a system with high functionality and two-tier cooperative processing would require less than twice the effort when compared to development using traditional means, even when project teams were staffed with the most able developers. We were interested to see the extent to which HPS affected development performance, even for a developer with relatively little software engineering experience.

Experiment #1: A Retail Sales Tracking System

Application Description. The experimental development project was a sales tracking system designed for broad use by large firms operating in multiple locations in the retailing industry. The report and inquiry capabilities of the system were meant to serve the needs of two levels of management: senior management at the firm's head office and store managers in the field. The firm's computer architecture was expected to consist of a large mainframe computer at the head office and minicomputers at each of the stores. Management's interest in obtaining on-line, real-time and batch reports based on intra-day and historical sales necessitated cooperative processing, because all data were uploaded to the firm's head office at the end of each business day for long-term storage. The system's high functionality was distinguished by the pull down menus and mouse-driven input screens of the friendly user interface.

Function Point Analysis. We performed a function point analysis to determine the relative size of the application. Function points measure the functionality of an application, as opposed to source lines of code (**SLOC**) (Albrecht and Gaffney, 1983). This metric is increasingly accepted as a meaningful and

reliable measure upon which to base an estimate of development effort (Kemerer, 1990; Symons, 1988). We estimated the size of the application at about 373 function points. Table 1 shows the breakdown by task of the function point total.

Task Description. The functional specifications for the experimental development project were designed in cooperation with First Boston Corporation staff members in advance of engaging the experimental developer. The project consisted of six development tasks. Four of these were primary tasks, which were presented in detail at the beginning of the development period. The final two tasks were enhancements. The enhancements were only discussed with the developer following successful completion of the first four tasks.

EXPERIMENT #1: DEVELOPMENT TASKS	SIZE IN FUNCTION POINTS
Primary Tasks	
Task #1	72
Task #2	80
Task #3	75
Task #4	70
Enhancement Tasks	
Task #5	50
Task #6	26
Overall Project	
Tasks #1-#6	373

TABLE 1. FUNCTION POINTS BY DEVELOPMENT TASK, EXPERIMENT #1 -- RETAILING APPLICATION

Project Manager Perceptions of HPS Development Productivity. There were insufficient time or resources available during the study period to develop the

experimental system in parallel using traditional 3GL tools for comparison purposes. Therefore, we sought to obtain development effort estimates from two knowledgeable sources to support our conclusions about the productivity gains associated with using HPS.

The first estimates were obtained in formal estimation sessions that we moderated involving two teams of First Boston's project managers. The second source was an external consulting firm to whom we gave detailed documentation on the experimental application.

The two formal estimation sessions involved seven project managers overall. They were requested to gauge how long the technical design, construction and testing-implementation phases would take if the application were built:

(1) without HPS and using minimal 3GL development tools;

(2) using HPS to construct a two-tiered cooperative processing application; or,

(3) using HPS to construct a three-tiered cooperative processing application.

Project managers estimated that traditional development of the project would take about ten weeks, even if the system were redefined to incorporate less functionality. Two-tiered HPS development (similar to the experimental system that was later developed), on the other hand, was estimated to require only six weeks total. Increasing the requirements specifications to make the experimental development project a three-tiered system was estimated to take approximately eight weeks.

When project managers were asked to estimate the effort required using traditional methods to provide the minimal functionality of the experimental development project in a single-tiered environment, they reported at least four months would be required. When they considered what would be involved in

duplicating the full functionality provided by HPS development using traditional methods across the micro, mini and mainframe computer environments, that estimate rose to two years of development effort. This estimate parallels what we learned from project managers in another set of structured interviews conducted at First Boston.

For a second independent and unbiased opinion, we provided the functional specifications for the experimental development project to an external consulting firm. They had no knowledge of any other aspects of this project. Their estimates indicated that duplicating the system with minimal functionality in a 3GL development environment would have taken at least two years, while use of commercially available 4GL productivity tools would have required about eight months. These estimates are summarized in Table 2.

Experimental Setting and Subject. HPS Version 2.61 was used for the duration of this experimental development project. During this time, the developer worked in a technically stable development environment. The subject of the experimental application was a First Boston employee with an average knowledge of HPS, based on a little more than six months of experience, and somewhat greater than average programming ability. This person participated in the project on a full-time basis, with the exception of one brief interruption.

Experimental Results. This project was actually completed in six weeks, matching the average of the two estimates provided by First Boston's project managers. Table 3 reports actual productivity levels in function points per person month for each experimental task. The developer observed that HPS Version 2.61 development involving an IBM S/88 minicomputer benefitted the least from HPS; apparently there were few facilities in place at that time to support minicomputer software development. The developer also observed that development time for on-line, real-time screens was greatly reduced due to the implementation of a new screen painting facility.

PROJECT MANAGER PRODUCTIVITY ESTIMATION CATEGORIES	High Functionality, Single-tiered Comparison: HPS to Traditional	High Functionality, Cooperative Processing Comparison: HPS to Traditional
Overall Life Cycle Productivity	30% gain	100% gain
Average of Productivity for Selected Subtasks	70% gain	130% gain
Maintenance/Enhancement Productivity	80% gain	120% gain

TABLE 2. PROJECT MANAGER ESTIMATES OF DEVELOPMENT PRODUCTIVITY GAINS IN TWO DEVELOPMENT SCENARIOS -- RETAILING APPLICATION (EXPERIMENT #1)

Throughout the experiment, we observed no explicit reuse of objects that were constructed in other projects and stored in the repository. However, the developer "templated" a great many new objects, by making slight modifications to objects that she had built. Nevertheless, the productivity results, averaging 149 function points per person month across the six experimental tasks, compared favorably with national estimates of software development productivity in the United States that are presented near the end of this paper (Bouldin, 1989).

We also noted that productivity increased when the developer performed the second of two inter-related tasks. This is indicated by the relatively higher productivity levels observed for the enhancement tasks. We also observed that the developer's productivity declined following the brief, mid-project switch after Task #3 to another job. Finally, we observed that the developer pushed the limits of

HPS' high productivity in completing the final task. We believe that this did not represent normal output, however, because the developer was due to go on vacation at the end of the week the project was completed. Table 3 summarizes these results.

EXPERIMENTAL DEVELOPMENT TASKS	UNADJUSTED ACTUAL PRODUCTIVITY (Function points/ person month)	ADJUSTED ACTUAL PRODUCTIVITY (Function points/ person month)
Primary Tasks		
Task #1	230	138
Task #2	240	144
Task #3	420	252
Task #4	200	120
Enhancement Tasks		
Task #5	360	216
Task #6	775	465
Overall Project		
Tasks #1-#6	248	149

Note: We report both unadjusted and adjusted actual productivity estimates. Adjusting the actual productivity estimates downward by about 40% makes them comparable to development in other First Boston Corporation projects.

The actual development effort we observed commenced at the technical design phase, whereas in most software development shops, strategic planning, business analysis and functional design account for a substantial amount of effort that we have not measured in the experiment.

TABLE 3. PRODUCTIVITY BY DEVELOPMENT TASK -- RETAIL APPLICATION (EXPERIMENT #1)

Clearly, these figures are only estimates; they could not be substantiated at the time because the CASE tool was so new. In addition, the experimental project was small, and one could argue that commercial development of larger systems would be an order of magnitude or two more complex. Still, the results prompted

us to look into HPS-based development performance more deeply, to attempt to understand what factors led to the high level of observed productivity.

Experiment #2: An Executive Information System

Application Description. This experimental application was meant to greatly extend the core functionality of a system that previously had been built using 4GL tools at a large financial institution. The application was intended to offer executives the opportunity to make queries about the content of business relationships with important customers.

Function Point Analysis. This application measured 1509 function points, and was broken into two modules:

(1) a customer reporting module, representing about 1056 function points, or 70% of the application's functionality, derived primarily from external interfaces and input types;

(2) a customer account maintenance module, representing the remaining 30% of the functionality, or 453 function points, derived primarily from input and output types.

The complexity multiplier for the application was 1.03, suggesting that it was of normal complexity, and in fact, the application exhibited a somewhat lower level of functionality than we saw in other systems developed using HPS. Yet, this application was a cooperative processing application, as the experiment was designed to demonstrate three-tiered development productivity. User query databases were located on a mainframe. The front-end graphics were generated by a microcomputer, and employed data that were downloaded from a mainframe and updated in real-time by a fault-tolerant minicomputer.

Task Description. The design specifications of this experimental project were created with the idea of testing the development of an application that incorporated

many features that were believed to be well-supported by various elements of the HPS tool set. Thus, the resulting application included the basic functionality of its 4GL-developed predecessor, but emphasized on-line, real-time functionality.

Estimate of Labor Required. The core elements of the application were estimated by the developers to take about 4 to 5 person months to code using CICS screens and mainframe COBOL. However, we were unable to perform a function point analysis to determine the size of the 4GL-developed system. The developers indicated that the new version of the system that was to be built experimentally could not have been developed without HPS.

Experimental Setting and Subjects. Experimental development was carried out under similar technical conditions as in Experiment #1. HPS Version 2.61 was used and the tool was stable during the time the application was under development. In addition to the design specifications, the primary difference between this experiment and Experiment #1 was that this development was undertaken by a team of seven developers, instead of just one person. Among the members of the experimental project team, only one had more than six months experience in the use of HPS, however, none of the participants was a novice in software development.

Experimental Results. Total observed work effort for the project was 918 hours, or about 5.18 person months, however, work on the project was not continuous for all the developers. Each person spent an average of about 135 hours on the project, with one person spending 10% more and another 10% less. These estimates reflect the fact that the developers were also spending time in demonstrations of the tool, in meetings and in other non-project related activities for 40 hours over the five-week period. This level of effort is consistent with the production of 175 function points per person month for the project overall.

The developers uniformly reported that becoming adept at HPS development did not take very long. The application was developed in a series of

increasingly complex prototypes, with developers alternately playing the role of critical users. The core functionality of the 4GL-developed system was in place within the first two weeks, and developers reported that team members had reused a significant number of objects built by the team members for the project. However, we did not have a measurement approach in place at that time to capture the levels of reuse that were occurring.

Experiment #3: A Trader Workstation Front-end

Application Description. Experiment #3 involved the re-creation and expansion of the functionality of a trader workstation front-end that previously had been built at a large financial institution. The application was re-developed to demonstrate that HPS could support a set of cooperative processing functions that were evenly distributed across the mainframe, minicomputer and microcomputer platforms.

Function Point Analysis. The size of the application was 1389 function points. The functionality was distributed as follows:

(1) 691 function points represented minicomputer functionality; and,

(2) the remainder, 698 function points, ran on the mainframe and microcomputer.

When we examined the function point results more closely, we found that approximately 37% of the functionality was derived from interfaces and 25% was derived from inputs.

This experiment occurred about four months after Experiments #1 and #2, and by that time, we had begun to make progress in understanding that tracking productivity alone would not tell the whole story of development performance with HPS. Thus, for this project we began to measure reuse more directly, in terms of a metric called "reuse leverage". Reuse leverage is defined as follows:

#_HPS_OBJECTS_CALLED / #_UNIQUE_HPS_OBJECTS_BUILT

In addition to the overall level of reuse leverage, we also tracked the greatest observed level of reuse leverage for an object, and individual reuse leverage ratios for 3GL components, and HPS screens and rules.

Experimental Setting and Subjects. HPS Version 2.61 again was used and the tool was stable during the time the application was under development. The team of developers that worked on this experiment had been involved in the development of a 3GL version of the same system at another financial institution.

Experimental Results. Table 4 reports the reuse leverage results for Experiment #3. When examining these results, the reader should keep in mind that all objects (except existing 3GL components) used by the developers were also built by them during the course of their experimental development work.

The reuse leverage results indicated that the developers extensively reused objects that they built themselves. The overall level of reuse leverage of 3.35 times indicates that only about 30% (1/3.35) of the functionality had to be built from scratch, indicating significant potential for a productivity gain to be observed.

Trader workstation software normally requires many calls to well-tested 3GL components that provide specialized functions related to the pricing and trading of financial instruments. In most investment banks such library routines are normally available right off the shelf, so the reuse leverage observed for 3GL components is quite realistic.

REUSE LEVERAGE CATEGORY	REUSE LEVERAGE
Overall Reuse Leverage	3.35 times
Greatest Observed Reuse Leverage for a Specific Object	17.00 times
3GL Component Reuse Leverage	11.10 times
HPS Screen Reuse Leverage	3.43 times
HPS Rule Reuse Leverage	2.72 times

TABLE 4. REUSE LEVERAGE RESULTS FOR TRADER WORKSTATION FRONT-END (EXPERIMENT #3)

The greatest observed level of reuse leverage for a single object was about 17 times, and this object was one that was built by the developers as an HPS object during the project. Such high levels of reuse often occur in financial analytics software, for example, when date or interest rate-related computations must be performed in order to compute the present value of a series of cash flows related to a financial instrument.

More interesting to us was the evidence that two kinds of HPS objects -- "rules" and "screens" -- offer significant reuse opportunities. Rules can be thought of in COBOL as statements in the procedure division. Screens, on the other hand, enable users to interact with the application, input trade-related data and see the results of specific trades. In on-line, real-time applications, these two object types are the most labor-consuming to build. (Batch applications involve the creation of HPS "report" objects, while both batch and on-line applications require developers to build "files" and other less labor-intensive objects.)

A reuse leverage of 2.72 times for rules is consistent with only having to

build about 37% (1/2.72) of the procedure division, if development had occurred using COBOL. Screens tended to be reused even more, 3.43 times, which means that developers only built about 30% (1/3.43) of all application screens from scratch.

Table 5 presents productivity results for Experiment #3, and breaks them out across the minicomputer function points and the combined PC-mainframe function points. The application required 502 person-hours of effort, for an aggregate productivity level of about 272 function points per person month.

DEVELOPMENT ACTIVITY	FUNCTION POINTS	FUNCTION POINTS/ PERSON MONTH
Minicomputer Software Functionality	691	222
PC and Mainframe Software Functionality	698	336
Overall Application	1389	272

Note: The actual productivity estimates were adjusted downward by about 40% to make them comparable to development in other First Boston Corporation projects. The actual development effort we observed commenced at the technical design phase, whereas in most software development shops, strategic planning, business analysis and functional design account for a substantial amount of effort that we have not measured in the experiment.

TABLE 5. PRODUCTIVITY RESULTS FOR TRADER WORKSTATION FRONT-END (EXPERIMENT #3)

The results that were observed in the development of the trader workstation front-end (perhaps to a greater extent than the results observed in the first two experiments), confirmed that software reuse has the power to play a major role in the realization of improved productivity results. Although some of our preliminary questions about the extent of the productivity gains that might be observed in HPS development were answered, many more new questions emerged that would require

additional study. These questions included the following:

(1) Would the order of magnitude of the software development productivity results hold when the project was scaled up from an experiment to the creation of larger, more complex systems?

(2) Would differences in software reuse leverage levels appear in larger projects? In projects that performed predominantly on-line, real-time processing versus batch processing?

(3) How would software development performance change as the use of the I-CASE tool and the tool set itself matured? How rapidly could developers come up to speed to enable large productivity gains to be achieved?

(4) What modifications to standard models in the software engineering economics literature would be needed to capture the impact of reuse on productivity? Does the creation of "reuse leverage" represent a separate "production process"?

EVIDENCE FROM LARGE-SCALE DEVELOPMENT USING I-CASE: FIRST BOSTON'S NEW TRADES PROCESSING ARCHITECTURE (NTPA)

The recent trend in software development in the investment banking industry has been in the direction of applications that deliver a higher level of functionality for the user. Such applications are exemplified by workstation displays that present historical pricing data, graphical analytics and up-to-date prices for financial instruments, in addition to a capability to effect a trade. In this section we will examine the First Boston Corporation's experience with respect to I-CASE-based software development of such applications. The software development performance results that we present emphasize the close relationship between software reuse and the firm's ability to achieve high levels of development productivity.

First Boston's New Trades Processing Architecture: Background

During the latter half of the 1980s, First Boston Corporation's senior IS management believed that to effectively support the firm's investment banking business increasingly sophisticated software applications and growing computer hardware power for high speed securities and money market transactions processing would be needed. This also would require immediate access to large mainframe databases whose contents could be processed in real-time using highly complex financial analysis software. Such applications would require 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 other firms in the industry, First Boston's systems would soon need to operate 24 hours a day across three platforms -- microcomputers, minicomputers and mainframes -- in support of global investment banking and money market trading activities. Much of the power that such software/hardware combinations would deliver was aimed at giving traders a few minutes (or even seconds) worth of time, an advantage that would help them to realize a profit in highly competitive markets. Such *high functionality software* was believed to offer a trader the ability to:

(1) obtain improved information access, through consolidation of multiple digital data feeds of market information on a single trader workstation;

(2) utilize real-time computer-based financial optimization analytics to support trading decisions with respect to existing and newly created financial instruments, and that would take advantage of the consolidated digital feeds; and,

(3) customize a user-friendly, windowing interface to suit a specific need.

In addition, senior management believed that higher functionality software could pay off in other ways. For example, through the delivery of consolidated and unbundled information on customer accounts and trader positions, it might be

possible to improve global and local financial risk management.

The firm's senior management also recognized that it was not possible to bring high functionality systems into production rapidly with traditional development methods. The only way to avoid this "software trap" was to consider automating software development (Feder, 1988). Following a survey of the available technologies then on the market, it was decided that an integrated CASE tool would be built in-house (Clemons, 1991). The result was the commitment of $100 million over the course of the next several years to create a new software development methodology and a new architecture of investment banking software applications. This investment would lay the foundation for *High Productivity Systems* (**HPS**), the firm's I-CASE tool set, and the infrastructure of investment banking applications for the firm that came to be known as the *New Trades Processing Architecture* (**NTPA**).

HPS and the Reusable Software Approach

The approach that the firm implemented emphasized *software reuse*. The technical vision involved rebuilding the firm's information systems architecture in a way that their basic building blocks -- objects and modules -- could be reused repeatedly. The methodology also would help to reduce the bank's reliance on costly language-specialized programmers by making it possible to develop software that could run on any of the three platforms with a single "rules language." This rules language would be defined within the HPS I-CASE tool. Code generators would then process this HPS code so that run-time COBOL, PL/1 and C and other code would be generated for each of the three major development platforms. The automated generation of run-time code was meant to screen developers from the complexity of the development environment. Most developers could focus on development by employing the HPS rules language, instead of traditional 3GLs.

HPS supports reuse because it operates in conjunction with an *object-based centralized repository*. The object types are defined within the rules language and

include programs, rules, output screens, user reports, data fields and 3GL components, among others. The centralized repository is the key enabling technology that supports the firm's reuse methodology. Specifications for the objects used to construct an application are stored in the repository and are widely 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.

At the time we conducted this study, HPS provided application entity relationship diagramming and screen prototyping facilities for enterprise modeling and analysis and design. It also offered code generators for several development languages, as well as tools for debugging code and managing versions of the same application. Table 6 presents an overview of some of the capabilities of HPS in the first two years that it was deployed.

Data Collection

Data were gathered on the development of twenty NTPA applications (some of which were broken in sub-projects), representing substantially all I-CASE development at First Boston during the first two years following the deployment of HPS. Table 7 presents information that will provide the reader with some understanding of the functions these applications provided for the bank's information processing infrastructure.

We obtained data in the following ways:

(1) examination of records on labor charges to projects;

LIFE-CYCLE PHASE	ACTIVITY SUPPORTED	SPECIFIC TOOL SET CAPABILITY
Requirements	Enterprise modeling	Information engineering-based data modeling package
	Information engineering	Diagramming tools to represent: * entity-relationships * business function hierarchies * object-function matrix mapping
System Analysis and Design	Detailed support for enterprise modeling and information engineering	Capabilities of diagramming tools mentioned above apply here also Data dependency diagramming
Construction	Code development for cooperative processing on mainframes, minis and PCs	Languages include: C, COBOL, assembler, PL1 and SQL
	Code generation from HPS "rules language"	Specific generators for: Windows and OS/2; COBOL CICS/MVS batch; IBM S/88 batch and on-line COBOL; IBM 3270 terminal screens; Windows and OS/2 Presentation Manager menus and HELP screens; DB2 databases
Implementation and Testing	Application code debugging	Debugging tool for generated code
	Installation support	Tool capabilities include: * auto version installation control * repository migration control * system rebuild
Production and Maintenance	Miscellaneous	Production version management facilty; software distribution control; debuggers for maintaining code

TABLE 6. THE HPS TOOL SET IN YEARS 1 AND 2 FOLLOWING IMPLEMENTATION

Broker Master	Product Master
Trade Inquiry	Dividend Interest Redemption
Dealers' Clearance	Real-Time Firm Inventory
Producer Master	Affirmation
Trading Account	Mortgage-Backed Securities
Trade Entry	Overnight Firm Inventory
Figuration	Floor/Desk/Breaksheet
Cash Management	Firm Price Management
Customer Account	General Ledger Interface

Note: In some instances, applications were subdivided forming the "projects" that we tracked. This led to the identification of multiple projects for a small number of the applications. In addition, the data set we examined did not actually include all of the applications listed above; some were excluded due to unavailable documentation or labor expense data.

TABLE 7. APPLICATIONS IN THE NEW TRADES PROCESSING ARCHITECTURE -- SOFTWARE FOR THE OPERATING INFRASTRUCTURE OF AN INVESTMENT BANK

(2) function point analysis based on examination of documentation describing NTPA applications;

(3) interviews with project managers and project team members; and,

(4) object analysis based on DB2 queries to the object repository and manual examination of application documentation.

Estimates of labor consumed. We obtained disaggregated and detailed reports on the hours for each developer assigned to an application project. Although this data was relatively complete, the bank did not have a productivity reporting system in place (nor did it track productivity in terms of function points).

As a result, in some cases it was necessary to apply second checks to ensure that we had captured all (or substantially all) of the labor hours expended on a project. In other cases where we believed that the data were too sketchy or likely to be in error, we omitted the project from further consideration in the study.

Function point analysis. To perform function point analyses for NTPA applications, we collected application documentation for as many applications as we could. In some cases, no documentation was yet available. These had been built using HPS prior to the time that application documentation was an automated by-product of system analysis and design procedures.

Function point analyses performed by members of the research team were double-checked for accuracy, and all members of the team were thoroughly trained to reduce the likelihood that the results would be different for different analysts. Project managers offered information about the extent to which the application development environment differed from the norm, making application development more complex.

Interviews with project managers and team members. These interviews were conducted by two members of the research team over the course of two months. The primary purpose of the interviews was to gain assistance with interpreting the labor charges that were made to the projects, how to break those charges out over sub-projects (where they were defined and software developers were broken into smaller teams), and other aspects of a project that might result in different levels of observed productivity. For example, project managers assisted us by specifying the "environmental modifiers" that are applied in function point analysis. In many cases, we learned that I-CASE development tended to reduce environmental complexity for development.

Because the research team was on-site at the bank, the interview process allowed for an initial meeting and then multiple follow-up interviews, when

necessary. In many cases, project managers played a crucial role in helping to ensure that the data we collected were accurate. They also offered advice and guidance that helped us to shape a new modeling perspective that reflects the related activities of reusing software and improving productivity.

Project team members provided useful information to enable us to better understand how the reusable software approach was applied in specific software projects. Through interviews with these developers, we learned about some of the advantages and disadvantages of the approach, and how smaller and larger projects might be affected differently.

The key issue that was discussed had to do with the *incentive compatibility* of software developers to build objects that would be widely reusable by other software developers. In the first two years of software development under HPS, developers "owned" objects that they developed first. Thus they had some measure of responsibility to ensure that the objects performed well in their own and in other developers' applications.

Because guaranteeing the performance of a software object in multiple contexts was difficult for individual developers, an *agency problem* developed which resulted in developers encouraging one another to make slight modifications to existing objects, and then to rename them. This had the effect of shifting ownership from the original developer to the developer who modified the object.

Object analysis. In order to obtain information about software reuse levels in each of the projects, research team members conducted "object analyses" to enable the estimation of project reuse leverage. This proved to be more difficult than we envisioned for two reasons:

(1) It was necessary to ensure that the documented application matched the content of the application that was actually built; and,

(2) the documentation varied in quality, in some cases enabling function point

analysis, but not a detailed count of application objects.

In view of these difficulties, a compromise was necessary. We found this compromise in follow-up interviews with project managers, who informed us that some HPS objects required very little effort to build, while others would be likely to act as the primary cost drivers. This enabled us to focus data collection efforts on the key cost driver objects (rules, screens, files, reports and 3GL components). As it turned out, much of this data was available from the documentation, and was quite accurate.

(More recently, we have been attempting to implement an automated object analysis procedure to confirm the quality of the NTPA project reuse leverage levels that we report in this paper and elsewhere (Banker and Kauffman, 1991). Our attempts to carry out automated object analysis for the NTPA projects have been hampered as the I-CASE tool has evolved. Further analysis requires the migration of prior versions of the applications to the centralized object repository that operates under the current version of HPS.)

Software Reuse Results

Table 8 presents the results obtained for reuse leverage in the twenty NTPA projects. The results contrast software development under HPS in Years 1 and 2 following implementation. They show how reuse leverage differed for on-line, real time versus batch processing application development. The table also shows the distribution of the application projects across these categories.

The observed levels of reuse leverage were lower in Year 1 (1.82 times) than they were in Year 2 (3.95 times). This is a very likely outcome. The lower reuse leverage in Year 1 was probably caused by one of several factors. These include:

CASE TOOL EXPERIENCE CATEGORIES	WEIGHTED AVERAGE REUSE LEVERAGE BY APPLICATION TYPE		
	ON-LINE (# PROJECTS)	BATCH (# PROJECTS)	BOTH (# PROJECTS)
YEAR 1 PROJECTS ONLY	2.95 (5)	1.41 (8)	1.82 (13)
YEAR 2 PROJECTS ONLY	4.11 (6)	3.05 (1)	3.95 (7)

Note: The average reuse leverage results are weighted for project size in terms of the total number of objects in an application.

TABLE 8. REUSE LEVERAGE FOR ON-LINE AND BATCH APPLICATIONS BY CASE TOOL EXPERIENCE CATEGORY

* lack of familiarity on the part of developers with the reusable software approach;
* difficulty in finding the appropriate objects to reuse;
* the practice (discussed earlier and interpreted as a response to the agency problem of object "ownership") of templating and renaming nearly matching software objects to avoid having to debug them; and,
* the small number of objects available in the repository for reuse.

In the Year 1 results, it is also interesting to note that on-line, real-time application development evidenced higher reuse leverage (2.95 times) than batch processing applications (1.41 times). In Year 1, the HPS tool set was biased to support on-line, real-time development to a greater extent than batch processing applications. Although the developers of the HPS I-CASE tools had a year or more lead time to develop its capabilities, the functionality of the tools was still limited. Management decided to focus efforts to create HPS tools that would support on-line, real-time development earlier. Facing substantial risks associated with the large

investment in building an I-CASE tool set, it was important to enable the delivery of applications that would be visible to users early on and whose impact would be felt in the business. In addition, the higher cost of developing more complex on-line, real-time applications made the focus natural.

By Year 2 the HPS tool set increasingly treated on-line, real-time and batch development on equal terms. Year 2 reuse leverage for batch processing application (3.05 times) exceeded the Year 1 level observed for on-line, real-time applications (2.95 times). This improvement can be attributed (in part) to changes in the HPS tool set. For example:

* batch development activities were made more productive through the deployment of a "report painting" facility; this enabled developers to nearly match the productivity that they could obtain for on-line, real-time applications when using a screen painter; and,
* when communication between platforms was required for both batch and on-line applications, highly specialized 3GL components (frequently called "middleware" by the developers we interviewed) had now become available that could be "plugged in".

Developers indicated that they were learning how to use HPS, and in the process, how to reuse more code more often. This perhaps best explains the level of reuse observed for Year 2 on-line, real-time application development (4.11 times). This level of reuse is consistent with building just 24% of an application from scratch, while the remaining 76% results from reused objects.

Large Application Development Productivity

Table 9 presents the function point productivity levels that were observed for the twenty NTPA projects. Similar to our presentation of the reuse leverage results, we include results for Years 1 and 2 to indicate the extent of the learning that was occurring about how to develop software using HPS. We also include

separate productivity figures for on-line, real-time and batch processing applications.

CASE TOOL EXPERIENCE CATEGORIES	PRODUCTIVITY BY APPLICATION TYPE OF PROJECT IN FUNCTION POINTS PER PERSON-MONTH		
	ON-LINE (# PROJECTS)	BATCH (# PROJECTS)	BOTH (# PROJECTS)
YEAR 1 PROJECTS ONLY	32.1 (5)	9.4 (8)	15.6 (13)
YEAR 2 PROJECTS ONLY	135.4 (6)	38.4 (1)	121.6 (7)

Note: The average productivity results are weighted for project size in function points.

TABLE 9. PRODUCTIVITY COMPARISONS FOR ON-LINE AND BATCH APPLICATIONS BY CASE TOOL EXPERIENCE CATEGORY

The productivity results in Year 1 suggest the power associated with software reuse. Productivity for Year 1 on-line, real-time application development was on the order of 32 function points per person month (FP/M), while Year 1 batch processing application development was only 9.4 FP/M. The reuse leverage associated with the on-line projects was 2.95 times (only 34% of the total functionality of the applications had to be built), and batch projects was a more modest 1.41 times (71% of application functionality had to be built from scratch).

By Year 2 productivity for both on-line and batch application development was substantially improved. Year 2 productivity for batch projects (38.4 FP/M) now exceeded Year 1 productivity for on-line, real-time applications. When these results were reviewed with project managers and software developers, most indicated that the increase in reuse leverage for batch development was responsible, and that the

improved capabilities of the I-CASE tool set was a major factor. (Recall that Year 2 reuse leverage of 3.05 times for batch processing application exceeded the Year 1 level of 2.95 times observed for on-line, real-time applications.)

Meanwhile, Year 2 productivity for on-line, real-time projects improved to 135.4 FP/M, four times better than in Year 1. Developers that we interviewed indicated that the primary factors responsible for this result were the availability of a larger pool of reusable repository objects, and the knowledge of how to locate them. In Year 2 developers became more familiar with a facility in HPS that provided key word search for objects. The key words were taken from the object name, still a relatively weak method on which to develop a complete set of candidate objects for reuse, but apparently very useful.

(Since the time that we did this analysis, we have learned much about the process of reusing software in the HPS I-CASE development environment. Banker, Kauffman and Zweig (1992) reported that reuse is often biased towards reuse of "owned" objects or objects created by project team members. Apparently the key word search facility was not the only, and probably not even the primary mechanism that developers used to identify objects that could potentially be reused.)

Comparison of Productivity Results with National Averages

Table 10 summarizes the productivity results obtained in the study and compares them with estimates of national averages of software development productivity made by Capers Jones. The present results compare favorably with the estimated national averages, and suggest the potential for order of magnitude productivity gains that may become possible when I-CASE development tools are used.

PROJECT COMPARISON CATEGORIES	FUNCTION POINTS/ PERSON-MONTH	COMMENTS
Intra-firm Estimates of Year 2 Performance		
BATCH PROCESSING ONLY	38.4	Productivity influenced by lack of 3GL component handling facility in earlier version of CASE tool. Batch report painter and SQL query support added to boost productivity in Year 2.
ON-LINE, REAL-TIME ONLY	135.4	Productivity enhanced by use of rapid, on-line screen painter, and high levels of reuse.
External World Estimates		
MILITARY/ DEFENSE DEPARTMENT	3.0	Large, technically complex development efforts.
TRADITIONAL 3GL	5.0	Averages initial development and subsequent maintenance.
MIS BUSINESS APPLICATIONS	8.0	Averages development activities conducted with and without CASE tools.
MATURE CASE, NO REUSE	15.0	"Mature" defined as a minimum of two years of experience with a relatively stable tool set.
MATURE CASE, WITH REUSE	65.0	A projected target for firms using an I-CASE tool.

Note: The external world figures are found in Bouldin (1989), who attributes them to Capers Jones.

TABLE 10. COMPARISONS BETWEEN INTRA-FIRM AND EXTERNAL WORLD SOFTWARE DEVELOPMENT PRODUCTIVITY

CONCLUSION

This paper provided evidence of the extent to which software reuse and I-CASE tools that operate in conjunction with a centralized repository have the potential to influence software development performance. Clearly, the results of this study can only be generalized to a limited extent. The research examined one I-CASE tool set at one site over two time periods, just following deployment of the tools. Nevertheless we learned much about the process of modeling software development productivity in this kind of development environment and the kinds of new metrics that management will want to track to better understand I-CASE development. In this concluding section, we first offer some preliminary answers to questions that were posed earlier. Finally, we end this paper by offering some thoughts about what implications our work may have for researchers and managers.

Did the order of magnitude of the software development productivity results observed in the experiments hold for larger-scale development? Apparently they did not. Although development productivity was at least one order of magnitude better (135.4 FP/M for I-CASE on-line, real-time application development versus Capers Jones' estimate of 8.0 FP/M for business MIS applications developed using traditional methods) than if 3GL methods had been used, it was evident that the results only held in a limited scenario. Moreover, nowhere did we observe in the NTPA development the 200+ FP/M productivity levels observed in experimental development.

Were the levels of software reuse different in the experimental and large-scale development projects? Here we had just one data point among the experimental projects to make our comparison. The results suggest that they were similar, especially in Year 2. (The comparison is between the overall reuse leverage (3.35 times) observed for Experiment #3, the trader workstation front-end, and the reuse leverages observed for NTPA on-line (4.11 times) and batch processing (3.05 times) applications in Year 2.) Increasing software reuse as project size increases

involves planning, coordination and search costs that were not evident for the experimental projects or for smaller projects. But larger projects may offer more opportunities for reuse, despite their complexity. The relationship between project scale and software reuse observed is an issue that must be addressed in future research.

Did development performance change as the use of the I-CASE tool and the tool set itself matured? There is no doubt from the results that we report and the interpretations offered to us by First Boston Corporation project managers that learning played a very important role in the outcome. Developers were learning to use the new tools as they became available. They were learning to be better at reusing code simultaneously. We observed a very steep learning curve for productivity and reuse leverage between Years 1 and 2 in the use of HPS to develop NTPA. The extent of the potential impact of future learning remains an open issue, however.

What was learned from this study that will assist other researchers in their attempts to model I-CASE development performance? Our research suggests that software development labor is transformed into software outputs (objects, modules or function points in this case) in the presence of a second production process that leads to observed reuse. From what we have seen, reuse leverage is created through a separate production process that involves labor, an existing pool of software objects and significant capital invested in a tool that supports the reusable software approach. Although detailed consideration of the factors that may drive higher levels of software reuse is beyond the scope of this paper, the reader should recognize that such factors must be considered to understand how to manage projects to generate higher levels of software reuse, paving the way for order of magnitude of gains in development productivity.

From a software engineering economics perspective, the well-accepted concept that software outputs are based on a single "software development

production function" may need to be re-evaluated. We have made initial attempts along these lines by estimating two separate production functions using seemingly unrelated regression estimation. For additional details, see Banker and Kauffman (1991).

The implications of this research for managers in I-CASE environments are as follows:

(1) Because software reuse appears to constrain the potential for software development productivity, it makes sense to implement measurement systems that track software reuse, as well as software development performance. Problems with software development productivity may be due to insufficiently high levels of reuse.

(2) If managers believe that it is worthwhile to measure software reuse, they should also recognize the potential difficulties that such measurement may entail. The metric that is discussed in this paper, reuse leverage, is probably new to the reader. There are no widely implemented standards at present, though the IEEE has written a standards document and made it widely available for comment. In addition, measuring reuse leverage manually was very labor and time-consuming. The only real solution is to automate such analysis. (In fact, very little work has been done to date in this area also. One exception is the work of Banker, Kauffman, Wright and Zweig (1992), who proposed a taxonomy of software reuse metrics and suggested an approach to their automation.)

(3) The levels of observed reuse are likely to be influenced by the set of incentive mechanisms that managers devise to overcome the "agency problem" that we described. In the development environment that we studied it is likely that a one-time (if minor) gain in reuse leverage could be obtained by placing objects, once they have been developed and tested, on neutral ground, so that the original developer would no longer be required to guarantee their performance. Other gains could be achieved by implementing incentive mechanisms to increase more directly the observed

levels of reuse.

A natural new owner would be an "object administrator", whose primary roles would involve:

(1) ensuring that a broad base of reusable repository objects is available for other developers to use;

(2) planning for a minimal subset of "reusable objects" to provide the kind of functionality that is needed in many different kinds of projects; and,

(3) proposing incentive mechanisms for senior management review that will assist in the achievement of higher levels of reuse leverage to support improved productivity.

Our call for "object administration" is meant to achieve the same kinds of payoffs in I-CASE development in the 1990s that database administration has delivered since the 1970s.

REFERENCES

[Albrecht and Gaffney, 1983] 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.

[Banker and Kauffman, 1991] Banker, R. D, and Kauffman, R. J. "Reuse and Productivity: An Empirical Study of Integrated Computer Aided Software Engineering (ICASE) Technology at the First Boston Corporation," *MIS Quarterly*, September 1991.

[Banker, Kauffman and Zweig, 1992] Banker, R. D., Kauffman, R. J., and Zweig, D. "Monitoring the 'Software Asset' Using Repository Evaluation: An Empirical Study." Forthcoming in *IEEE Transactions on Software Engineering.*

[Banker, Kauffman, Wright and Zweig, 1992] Banker, R. D., Kauffman, R. J., Wright, C., and Zweig, D. "Automating Reuse and Output Measurement Metrics in an Object-Based Computer Aided Software Engineering Environment." Forthcoming in *IEEE Transactions on Software Engineering.*

[Bouldin, 1989] Bouldin, B. M. "CASE: Measuring Productivity -- What Are You Measuring? Why Are You Measuring It?" *Software Magazine*, 9:10, August

1989.

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

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

[Kemerer, 1990] Kemerer, C. F. "Reliability of Function Points Measurement: A Field Experiment," Working Paper, Sloan School of Management, MIT, December 1990.

[Symons, 1988] Symons, C. R. "Function Point Analysis: Difficulties and Improvements," *IEEE Transactions on Software Engineering*, 14:1, January 1988.

ACKNOWLEDGEMENTS

Special thanks are due Mark Baric, Gene Bedell, Gig Graham, Tom Lewis and Vivek Wadhwa of Seer Technologies. They provided us with access to data on software development projects and managers' time throughout our field study at the First Boston Corporation. We also appreciated the efforts of Eric Fisher, Charles Wright, Vannevar Yu and Rachna Kumar, who assisted in the data collection. An earlier version of this research was presented at the "Software Engineering Economics I Conference," sponsored by the MITRE/Washington Economic Analysis Center, June 1991. Jean Kauffman provided helpful editorial comments to improve the readability of the paper. All errors are the responsibility of the authors.