

**TRACKING THE 'LIFE CYCLE  
TRAJECTORY': METRICS AND MEASURES  
FOR CONTROLLING PRODUCTIVITY  
OF COMPUTER AIDED SOFTWARE  
ENGINEERING (CASE) DEVELOPMENT**

by

**Rajiv D. Banker**

Carlson School of Business  
University of Minnesota  
Minneapolis, Minnesota 55455

**Robert J. Kauffman**

Leonard N. Stern School of Business  
New York University  
New York, New York 10003

and

**Rachna Kumar**

Leonard N. Stern School of Business  
New York University  
New York, New York 10003

**January, 1992**

Center for Research on Information Systems  
Information Systems Department  
Leonard N. Stern School of Business  
New York University

**Working Paper Series**

STERN IS-92-3

Forthcoming in *The Handbook of Software Productivity*, Keyes, J. (ed.), McGraw-Hill.

**TRACKING THE 'LIFE CYCLE TRAJECTORY':  
METRICS AND MEASURES FOR CONTROLLING PRODUCTIVITY  
OF COMPUTER AIDED SOFTWARE ENGINEERING (CASE) DEVELOPMENT**

January 20, 1992

**Rajiv D. Banker**  
Carlson School of Business  
University of Minnesota  
Minneapolis, Minnesota 55455

**Robert J. Kauffman**  
Stern School of Business  
New York University  
New York, New York 10003

**Rachna Kumar**  
Doctoral Program in Information Sy  
Stern School of Business  
New York University  
New York, New York 10003

Forthcoming in *The Handbook of Software Productivity*, Keyes, J. (ed.), McGraw-Hill.

---

---

We wish to acknowledge Mark Baric, Gene Bedell, Tom Lewis and Vivek Wadhwa for the access they provided us to data on software development projects and managers' time throughout our field study of CASE development at the First Boston Corporation and SEER Technologies. Another version of this paper was presented at a conference entitled "Integrating Information Technology and Analysis: How to Deliver Systems Your Clients Will Love," sponsored by the College on the Practice of Management Science of The Institute of Management Science (TIMS/CPMS), and the Operations Research Society of America (ORSA). All errors in this paper are the responsibility of the authors.

---

---



## BIOGRAPHIES OF THE AUTHORS

**RAJIV D. BANKER** holds the Arthur Andersen Chair in Accounting and Information Systems at the Carlson School of Management, University of Minnesota. He received a doctorate from Harvard Business School, specializing in planning and control systems. He currently serves on the editorial boards of six journals and as co-editor of the Journal of Productivity Analysis. He has published over 40 refereed articles. His research interests include strategic cost management, measuring the business value of information technology, assessing software development and maintenance productivity and the economics of information.

**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 his masters degree in international affairs 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. He has published refereed articles in MIS Quarterly, Journal of Management Information Systems, Information and Software Technologies, and elsewhere.

**RACHNA KUMAR** is currently in the Doctoral Program in Information Systems at the Stern School of Business, New York University. She received the Master of Business Administration degree from the Indian Institute of Management, Ahmedabad, in 1983. Her current research interests focus on productivity measurement and cost estimation for computer aided software engineering environments (CASE). Her dissertation work involves a field study of the performance of object-based productivity metrics in the various CASE life cycle phases.



TRACKING THE 'LIFE CYCLE TRAJECTORY':  
METRICS AND MEASURES FOR CONTROLLING PRODUCTIVITY  
OF COMPUTER AIDED SOFTWARE ENGINEERING (CASE) DEVELOPMENT

---

---

ABSTRACT

This paper proposes a new vision for the measurement and management of development productivity related to computer aided software engineering (CASE) technology. We propose that productivity be monitored and controlled in each phase of software development life cycle, a measurement approach we have termed *life cycle trajectory* measurement. Recent advances in CASE technology that make low cost automated measurement possible have made it feasible to collect *life cycle trajectory* measures. We suggest that current approaches for productivity management involve the use of static metrics that are available only at the beginning and end of the project. Yet the depth of the insights needed to make proactive adjustments in the software development process requires monitoring the range of activities across the entire software development life cycle. This can only be accomplished with metrics that can measure performance parameters in each phase of the life cycle. We develop metrics that have the ability to measure and estimate software outputs from each intermediate phase of the development life cycle. These metrics are based on a count of the objects and modules that are used as building blocks for application development in repository object-based CASE environments. The viability of such object-based metrics for *life cycle trajectory* measurement has been empirically tested for the software construction phase using project data generated in Integrated CASE development environments.

---

---



## 1. SOFTWARE PRODUCTIVITY AND THE CASE OPPORTUNITY

Computer aided software engineering (CASE) tools are believed to represent an industrial revolution in the market for software development. They have changed the dynamics of software development from essentially a manual, craft work-like process to a more automated, rigorous and standardized engineering discipline. In this paper, we examine new approaches and opportunities presented by this changed development environment for managing software development performance. We will argue that CASE has the potential to improve control of software development productivity by allowing measurement of software outputs across the entire development life cycle.

### 1.1. The Quest For Improving Software Productivity

The sheer size of corporate investments in software indicates the extent of the hopes that senior managers place in wresting business value from it<sup>(13,7)</sup>. For example, industry specialists estimated that by 1990 the total investment in existing, developed and purchased software was in the neighborhood of 13% of the United States' gross national product, a staggering \$527 billion<sup>(9)</sup>. Other projections reveal an annual increase in software development budgets at the rate of 9% to 12%, exceeding \$150 billion per year by 1990<sup>(9,18)</sup>. However, software development is regarded as a major bottleneck in exploiting the potential of IT<sup>(17)</sup>. Substantial backlogs of software development exist in organizations of all sizes and in many different industries, and they are reported to be increasing at a rapid rate<sup>(42,46)</sup>. One study even reported the existence of "hidden backlogs,"



consisting of user needs that were not formally requested or commissioned; these hidden backlogs were estimated at 535% of known backlogs<sup>(2)</sup>. Reports of software projects months behind schedule and far over budget are also quite common. As a result, senior management perceives that it is critical to find ways to better control the production of corporate software assets.

A common intermediate goal for senior software development managers is to improve the productivity of applications development and the quality of applications execution. The low productivity of software development operations is attributable to a number of factors<sup>(1, 9, 25, 39)</sup>. Table 1 lists the major ones among these.

-----  
 INSERT TABLE 1 ABOUT HERE  
 -----

Improvement of productivity can be achieved by streamlining the life cycle of software creation through the introduction of new development techniques. As a result, in recent years we have witnessed the introduction and adoption of many new software development tools and techniques. These include: structured programming; rapid prototyping and prototyping; fourth generation languages (4GLs); object-oriented and graphical analysis, design and development techniques; and data-oriented methodologies.

The most recent addition to this list is integrated computer aided software engineering (CASE) tools. Input Inc., a California-based research firm, has indicated that about 6% of annual software expenditures by American firms in 1989 were attributable to application development tools in general. In terms of dollars, this puts the total expenditure in the range of

\$6 billion or more, and spending on such off-the-shelf application development tools is conservatively estimated to be growing at a 19% annual rate<sup>(32)</sup>.

### 1.2. The Promise Of CASE Productivity

CASE is often touted as the most promising of all the new tools, and certainly it is the fastest growing segment. Two different surveys have indicated that between 55% to 75% of organizations have adopted CASE tools for various development projects including pilot projects, departmental projects, and corporate-wide applications<sup>(11, 40)</sup>. And, analysts predict that the CASE market will grow at 35% to 45% per year, to something on the order of \$1 billion in the early 1990s<sup>(30)</sup>.

CASE technologies and the methodologies that they promote aim to transform the process of software development. Paralleling the structure of production in other industries such as automobile manufacturing, home construction, and even computer hardware manufacturing, CASE is enabling a move of the software enterprise from an assembly industry to a process industry. This means that each product is no longer custom built, one at a time. Instead, production occurs through the use of pre-fabricated components and reusable templates, plans and procedures<sup>(35)</sup>. CASE advocates and firms investing heavily in CASE argue that software automation and the "modular software" approach is the key to increasing productivity, controlling quality, and introducing predictability into the software development process.

An analysis of the structural and functional dimensions of CASE technology helps to identify the major characteristics of this methodology that contribute towards potential improvements in development productivity. These have very broadly been classified by various authors<sup>(11, 30, 39)</sup> as the standardization of the software development process, and the automation of software

development activities.

*Standardization* of software development is at the heart of the "modular approach" to software creation. It enables reuse of existing software components, which saves the effort in writing, testing, and implementing portions of the software currently being developed<sup>(19, 23)</sup>. Standardization can lead to reduced development time as well as an improved software quality. *Automation* addresses tedious or routine manual tasks such as verification, validation and consistency checking in early development phases, or error checking in code. This not only reduces the labor required for manually performing these tasks, it also ensures that these tasks are satisfactorily and uniformly performed. It also supports an increase in the quality of delivered software.

Thus, standardization and automation can contribute significantly towards development productivity by impacting the efficiency and effectiveness of software creation. *Efficiency* increases productivity directly by increasing software output per unit effort input by software developers when a CASE methodology is used to develop software. *Effectiveness* impacts productivity indirectly by ensuring that CASE-developed software accomplishes the business goals of the organization and therefore the software output is relevant and has maximal value. The efficiency and effectiveness dimensions of CASE development are described in Table 2 below.

-----  
 INSERT TABLE 2 ABOUT HERE  
 -----

A natural question would be the verification of the promised productivity benefits of CASE. Although reports on CASE claim a myriad of benefits ranging from 300% productivity increases to 'zero-maintenance' program code, only a few studies report

rigorously substantiated productivity benefits<sup>(26, 33, 45)</sup>. Most studies report on successful (or confessional) implementations of CASE methods or on surveys compiling usage proportions and profiles of CASE tools<sup>(11, 30, 31)</sup>. Banker and Kauffman<sup>(4)</sup> have presented some of the first empirical results to substantiate large productivity gains from using CASE development techniques, especially the leverage created when a firm implements a software reusability strategy.

This paper examines how management reporting needs to be recast to support the goal of controlling software productivity as much as possible with the tools available in the new environment of CASE. It develops a new vision for the management of the software development life cycle in the presence of integrated CASE technologies via automated software metrics and measures. We will make the case that tracking the *life cycle trajectory* of software projects, made possible by automated analysis of the software development process, will help management to control productivity in a way that was not possible before CASE.

## 2. CONTROLLING CASE DEVELOPMENT PRODUCTIVITY

### 2.1. A New Vocabulary for Tracking Software Development Performance

We propose a framework to measure, control and influence software development performance that builds upon the distinguishing characteristics of CASE environments. We find that existing approaches to the estimation of software development productivity and the measurement of subsequent development performance only provide single point measures -- when a project begins or when it has reached completion. Such static measures for estimation and efficiency analysis do not provide sufficiently detailed or relevant information for proactively managing the software development process. By contrast, *dynamic* measures for software development performance can help management to monitor and

control performance through the entire software development life cycle. We refer to this concept of dynamically measuring and estimating performance in each phase of the development life cycle as "tracking the life cycle trajectory" of a software project. The *life cycle trajectory* approach monitors performance parameters of interest in each life cycle phase and visually depicts the progress of the project along the measured performance dimensions.

Static, single-point software development metrics are snapshots of the results of software development production performance. Dynamic metrics capture the development process on video tape, enabling management to play the action back at will as it occurs, to better understand it, and then to control and improve overall project performance. Boehm<sup>(8)</sup> has equated the problem of accurately estimating development costs for a software project with the problem an author has in estimating the number of pages a book will have when the plot has just been sketched out. Static metrics would only support the comparison of the initial estimate of the length with what the author subsequently writes. But, dynamic metrics are meant to describe the process of producing the book, as the author adjusts the plot, resolves problems in the relationships among the characters, or deals with a crucial mental block which hampers the writing. Figure 1 contrasts the richness of the information provided from dynamic versus static measures.

-----  
INSERT FIGURE 1 ABOUT HERE  
-----

The figure depicts the trajectories of labor consumed by two software projects, A and B. Initially, both are estimated to consume approximately the same level of resources during the life



cycle. Suppose, however, that management's estimates are inaccurate, to an equal extent for both projects. In this situation, we would observe two similar cost estimates and also two similar variances between the estimated and actual costs. Such static metrics might suggest that management take the same kind of action to improve "similar" projects in the future. But note that the labor consumption trajectory suggests that the software development processes occurring in each project were quite different. Let us assume that the area under the phased labor consumption curves and the size of the resulting software are the same for both projects. Project B required relatively more effort during technical analysis and functional design, while Project A consumed more labor during the construction phase.

Similar sketches for the *life cycle trajectory* could be made for other performance measures such as productivity, defects, the development team's expertise profile, and so on. In CASE environments, tracking the *life cycle trajectory* of 'software reuse' is another dimension which offers a diagnostic performance sketch. The point is that utilizing such full trajectory information makes it more likely that managers will ask the right questions. For example: Were the functional design problems experienced due to the qualities of the resulting application or the analysis and design staff? Was the skill mix or experience level of the staff of Project B unsuited to the development requirements of the project? Managers can ask more general questions as well. For example: How much reuse occurs in software development, and what is the extent of its leverage on productivity? Does the skill mix or the experience level of the staff assigned to a project influence the trajectory of its-labor consumption or productivity?

However, such *life cycle trajectory* metrics only become feasible in the CASE environment because the phase activities and phase

boundaries are better defined and more rigidly enforced than in the pre-CASE era. In keeping with the automated character of CASE development, measurement mechanisms can also be built into the CASE toolset enabling management to carry out continuous, low cost monitoring.

## 2.2. Automating Life Cycle Trajectory Measurement

In effect, we are advocating the collection of finer and more "perfect information" in the context of software development cost control, but only to the extent that it is relevant. The collection of more information in a decision setting only can be justified after a careful consideration of the costs and benefits of that information. Traditional software development environments were unable to support the delivery of such information as the life cycle progressed without forcing a project manager to incur unacceptably high costs. But CASE changes this cost-benefit relationship.

**The Benefits Of Measurement:** The value of information describing the software development life cycle to the project manager are a function of the actions that can be taken based on the information, and the consequences that the actions can produce<sup>(14)</sup>. *First*, measures that are collected should be able to resolve decision options. Dynamic life cycle metrics enable actions that influence subsequent software development activities in a manner illustrated in the previous section. *Second*, there is not much value in collecting measures with accurate up-to-the-minute detail if the software operations cannot (or need not) be controlled to that level of fineness. This is likely to be the case in the early phases of development, when order of magnitude estimates of labor may suffice. Figure 2 depicts the high variability and unpredictability of project costs when estimations are made in the earlier phases.

-----  
INSERT FIGURE 2 ABOUT HERE  
-----

Efficient control measures in these phases could be rough, first approximations because they cannot resolve very finely the management actions vis a vis cost control. In the later phases, more accurate, refined measures of the costs and cost drivers will better support decision making for cost control.

**The Costs Of Measurement:** The other issue in committing to trajectory measures is an acceptable cost to implement them. Considerations regarding the decision value of the information affect the nature and design of suitable metrics. Clearly, the cost of measuring should not exceed its decision value, or else it will reduce management's motivation to measure. Johnson and Kaplan<sup>(22)</sup> suggest that the reduction in the costs of information collection and processing no longer justifies highly aggregated, low-detail process information. They comment:

*"... that managers [were] not inclined to compile [disaggregated and] accurate data reflects their judgment on the costs and benefits and feasibility of such information, not a lost sense of what information is relevant to [operational] management decisions" (pp. 144)*

This suggests that managers might have been convinced of the value of measuring across the life cycle, but the cost of such measurement would have deterred them. The cost of collecting data and providing prompt reports for each life cycle phase of software development was too high in the manual programming era to permit the real time trajectory tracking we are now advocating.

But, today's CASE development environments make it possible to automate the measurement and collection of software life cycle trajectory metrics. The reduced cost of automated measures no



longer requires managers to contend with irrelevant, aggregate measures on complex and critical software development processes. The challenge, therefore, is to develop dynamic life cycle performance measures for software development which will be amenable to automation and repeated collection at a minimal cost. Only automated measures and metrics for tracking the *life cycle trajectories* of CASE projects provide ongoing control information such that their decision value outweighs the costs.

In fact, product development in this area is underway for a number of CASE development environments, including Texas Instrument's IEF<sup>(29)</sup>, Andersen Consulting's Foundation<sup>(20)</sup>, and Seer Technologies' High Productivity Systems CASE tools<sup>(4)</sup>. These firms are undertaking the construction of automated metrics facilities at a one time-cost, to defray the cost of repetitive measurements to be made in the future.

### 2.3. Control Framework For Life Cycle Trajectory Measures

Software development productivity is defined as the ratio of the size of software output to the costs required to produce it.

$$PRODUCTIVITY = \frac{SOFTWARE\ SIZE\ OUTPUT}{DEVELOPMENT\ EFFORT\ INPUT}$$

Since the size of software output from the development process is an external specification as defined by the project description, it is not regarded as controllable. Thus most approaches to controlling productivity focus on ways and methods to control the cost of inputs into the development process, that is, development effort in the software context.

Effective cost control systems should deliver three basic capabilities to software development management<sup>(41)</sup>:

- [1] **Measurement** -- The ability to unambiguously and consistently measure costs associated with identifiable units of work.

- [2] Estimation -- The ability to accurately estimate and forecast cost measures.
- [3] Variance Analysis -- The ability to isolate variances between estimated and actual cost measures, enabling corrective measures to be taken to reduce the discovered variances.

We next examine these components more closely, as each relates to our proposal for life cycle trajectory metrics.

Measuring the costs associated with the work of software development should take into account all inputs into the software production process. Costs arise from a number of sources, such as development labor, hardware resources, business transactions, and so on. However, development labor is by far the largest, most significant and most variable cost component<sup>(21)</sup>. Therefore, the measure for the cost of development usually considers only labor inputs and is in terms of the number of person-days or person-months logged on the software project by the development team over the entire life span of the project.

The second requirement, the ability to accurately estimate costs, is required because managers gauge how well an activity is being performed by comparing actuals against estimated performance. Whatever its sophistication, a specific software development performance measurement system cannot be effective in controlling the process unless it incorporates a set of standards which managers can agree upon and use as anchors on which to base their performance expectations. The limited ability of software managers to estimate the time required and costs of development has long been a major shortcoming, and was first brought to the attention of the systems development community by Brooks, in his essay *The Mythical Man Month*<sup>(10)</sup>. Even experts tend to underestimate software project development times, and in spite of this awareness projects continue to be behind schedule and budget. Sometimes irrational political perspectives have been

found to influence the cost estimation process and meaningful managerial actions for improving estimation can be implied<sup>(28)</sup>. Advances in more formal approaches to measuring software size have centered on empirical models that predict development time based on historical relationships between software size and development labor. Models, such as COCOMO, ESTIMACS and SLIM, exemplify these formal approaches<sup>(26)</sup>.

The third requirement, the ability to isolate variances between estimated and actual cost measures is a diagnostic capability which answers an important question: "What is the cause for the difference between estimates and actuals?" Providing a satisfactory answer requires an understanding of cost drivers -- those development attributes that impact and mediate the conversion of development labor into software product. In software development, as in most production processes, the size of the software output is the most important cost driver. But attributes of the development process have also been found to impact development labor<sup>(38, 8)</sup>. These attributes can be classified into program attributes (e.g., reliability requirements), environment attributes (e.g., main memory constraints), personnel attributes (e.g., average experience of project team), and project attributes (e.g., type of development tool used).

$$DEVELOPMENT-EFFORT-INPUT = f(SOFTWARE-SIZE-OUTPUT, OTHER-COST-DRIVERS)$$

In software development, the impact of project development attributes on the labor effort required for delivering the system is not a simple relationship. The impact depends on both the life cycle phase of the software project as well as the value of other attributes<sup>(8, 44)</sup>. Once managers are able to diagnose the causes for the deviation in performance, they should be able to understand what actions are appropriate or necessary to influence the factors causing the deviation. This ability to influence

cost drivers, like isolating the causes of variances, is again dependent on an understanding of the nature and effect of the cost drivers. For example, applications with the project attribute *high reliability* have been found to be adversely affected in terms of development time in the functional design phase, but to a lesser extent than in the coding phase. Similarly, if the personnel attribute for a project is *high experience* for the development team, reliability considerations would not impact development time as much as if the attribute were *low experience*.

So, we see that the cost drivers are phase-dependent and also may exhibit joint effects. This is summarised in the expression below.

$$DEVELOPMENT-EFFORT-INPUT_p = f_p(SOFTWARE-SIZE-OUTPUT_p, OTHER-COST-DRIVERS_p)$$

This considerably complicates the isolation and correction of variances, and meanwhile places a premium on obtaining better and more detailed diagnostic information akin to that advocated in our *life cycle trajectory* measurement proposal.

### 3. LIFE CYCLE TRAJECTORY APPROACHES FOR CASE PRODUCTIVITY

In order to implement a dynamic productivity control system incorporating trajectory measures, we need to identify sound bases for designing metrics which measure DEVELOPMENT-LABOR-INPUT, SOFTWARE-SIZE-OUTPUT and OTHER-CASE-COST-DRIVERS in each development phase.

#### 3.1 Identifying Measures

- DEVELOPMENT-EFFORT-INPUT: These measures for each life cycle phase can be obtained from existing measurement approaches. Existing labor tracking systems generally account for labor hours over the entire life cycle. These labor hours can be summed at

the end of each phase. Linking labor tracking systems to automated software development performance analysis facilities with the proposed trajectory metrics would also help to motivate measurement.

- CASE-COST-DRIVERS: Phase measures for the CASE-COST-DRIVERS require a more substantive change in existing approaches. The prerequisite for establishing measures for cost drivers is the identification of relevant cost drivers: those attributes that significantly affect labor input costs in the different phases. In a CASE development environment, only some factors will impact the software development process enough to make a significant difference in the input labor hours. Thus, the set of relevant software cost drivers identified in prior research needs to be revised, based on what can be learned from new research on CASE development performance. Although more exhaustive, empirical verification is still needed, some preliminary evidence exists to suggest that in CASE environments DEVELOPMENT-TEAM-EXPERIENCE and NEW-OBJECT-PERCENT impact development labor significantly<sup>(5)</sup>. DEVELOPMENT-TEAM-EXPERIENCE can generally be measured with subjective rating methods for each phase.

A bigger challenge is to develop *life cycle trajectory* metrics for NEW-OBJECT-PERCENT and SOFTWARE-SIZE-OUTPUT from each phase. NEW-OBJECT-PERCENT refers to the use of existing software in order to build an application. Reused software adds to the size and functionality of the delivered software product without requiring a proportionate amount of development labor. This justifies its inclusion as an important cost driver for DEVELOPMENT-EFFORT-INPUT. NEW-OBJECT-PERCENT is measured in terms of the proportion of reused code in the total SOFTWARE-SIZE-OUTPUT.

$$NEW-OBJECT-PERCENT = \frac{TOTAL\ SOFTWARE-SIZE-OUTPUT}{UNIQUE\ SOFTWARE-SIZE-OUTPUT}$$

Since NEW-OBJECT-PERCENT is expressed in terms of a proportion of SOFTWARE-SIZE-OUTPUT, both drivers can be measured by the same units of work output. Thus, measures for both SOFTWARE-SIZE-OUTPUT and NEW-OBJECT-PERCENT are dependent on identifying work output measures from the development process. This requires identification of measurable units of work at the end of each of the life cycle phases.

Identifying measurable units of work from phases was not easy until the advent of CASE development tools. In traditional development environments each life cycle phase did not have a unit of delivered work which could be measured with any degree of accuracy. For example, the work done in the business analysis phase was partly represented by diagrams on paper and partly in the analyst's mind. Similarly, a considerable portion of the work completed in the functional design phase went undocumented because of verbal communications between the analyst and the programmer, unwritten contracts, and so on<sup>(15, 37, 43)</sup>.

### **3.2. An Illustration Of Trajectory Metrics: CASE Repository Objects**

CASE technologies make it possible to capture outputs from each life cycle phase. The discipline of CASE development produces well specified, rigorously defined outputs from each life cycle phase. These outputs can form the basis for unambiguous work unit measures.

In keeping with the standardization and reusability aspects of CASE environments, measures for monitoring phase outputs should utilize relevant parameters of the pre-fabricated components that form the basis of the "modular approach." In related work, we explored the possibility of monitoring the use and nature of these pre-fabricated components themselves, which have been



called "objects"<sup>(32)</sup>. The results indicated that because objects act as building blocks to construct the functionality of the software in repository-based CASE environments, they can be used to represent the outputs of development in efficiency metrics.

Objects represent specific, well-defined functions in handy, ready-to-use chunks of code. An object need only be written once, and all subsequent applications that need to deliver the same functionality could merely reuse existing objects. In addition, the definitions and code content of objects in CASE environments are frequently stored in a centralized repository. Examples of objects that are often utilized in repository-based CASE environments are: RULES, SCREEN DEFINITIONS, USER REPORTS, and so on. The complexity of the objects written afresh by a programmer, the level of reuse of existing objects by a programming team, and the total number of objects of all types used to build an application provide a natural avenue along which the design of trajectory metrics can proceed.

In integrated CASE environments (ICASE), i.e., those which automate development in all the life cycle phases), application development is a process of successive refinement of objects as development progresses from the earlier life cycle phases of business analysis and design to the later phases of testing and implementation. For additional details on an integrated CASE environment (ICE) that has some of these features, see (4). The objects created at the business analysis phase are abstract, higher level representations of functionalities required by the application. Each subsequent lower level object of the later phases goes one step further in instantiating the functionality of the previous phases's object, until finally the code is written in the construction phase.

Objects created in earlier phases lay out a road map for subsequent refinement that may occur, or the development of

additional objects in later phases. Thus, a study of the deliverables at the end of each life cycle phase of CASE development would enable the specification of outputs at each stage. Table 3 illustrates this perspective by identifying objects that would be useful to gauge output phase-by-phase. The examples draw on experience we gained in a field study of CASE at the First Boston Corporation and Seer Technologies. The object names are used as illustrations of generic outputs that can be identified from the different life cycle phases.

-----  
INSERT TABLE 3 ABOUT HERE  
-----

The *Business Analysis* phase defines the scope and functions of the system in terms of user requirements. The output of business analysis in CASE environments is a model of the processes and the data involved in the business system. The approach is based on the concepts of the Entity-Relationship (E/R) model developed by Chen. This phase often uses tools such as an entity-relationship diagrammer or a process hierarchy diagrammer, and typically outputs objects such as ENTITIES, PROCESSES and RELATIONSHIPS (between ENTITIES and PROCESSES). These are objects defined according to the E/R model, and their total number and complexity as they exist in the repository at the end of this phase can be used to measure the work output from the business analysis phase.

Similarly, the *Functional Design* phase translates business requirements to the specific needs of the application's users, including features, functions, interfaces, and so on. It uses tools such as a report painter or a window generator, and typically outputs objects such as RULES, WINDOWS, VIEWS, and RELATIONSHIPS (between RULES, WINDOWS, VIEWS, and so on). The *Technical Design* phase further refines the functional specification of objects by including: the data structures; data



flows; and files referenced, input or output. Examples of objects produced in this phase are FIELDS, FILES, RULES details, and so on. *Software Construction* involves adding details to the software for compiling at the source level. Actual code is generated only during the application's run-time. Reusable objects need merely retrieve software construction details from the repository while objects that have to be written from scratch will require much more labor. Thus, the NEW-OBJECT-PERCENT will affect DEVELOPMENT-EFFORT-INPUT very significantly in this phase. (We are currently studying what the relevant object outputs will be for the *Testing/Implementation* and *Maintenance/Enhancement* phases.)

To sum up our argument, repository-based objects can act as distinct and identifiable units of work from each life cycle phase of CASE development. The total number, complexity or size, and origin (reused versus written from scratch) of objects can be used to measure SOFTWARE-SIZE-OUTPUT from each phase. As explained earlier, the NEW-OBJECT-PERCENT cost driver, dependent on the same unit of work as SOFTWARE-SIZE-OUTPUT, can also be distinctly measured USING object-based metrics for each phase. This equips us with productivity metrics to track the *life cycle trajectory* of CASE developed projects.

### **3.3. Implementing Object-Based Trajectory Metrics**

In an exploratory study conducted earlier<sup>(6)</sup> we empirically tested the proposal for object-based trajectory metrics for the construction phase of the CASE life cycle. Data were obtained on software projects developed and produced with a multi-million dollar ICASE tool built by a large investment bank in New York city. The objective of the study was to evaluate the performance of an object-based metric for the 'Software Construction' phase. Performance of the metric was judged on the basis of its ability to:

- (i) measure the cost driver SOFTWARE-SIZE-OUTPUT, and
- (ii) estimate the DEVELOPMENT-LABOR-INPUT for project delivery.

Each project was manually counted for the number of construction phase objects in the final delivered software. The sum of the instances of all object types used for application development in the construction phase was defined as the first object-based metric for this phase. We called this metric OBJECT-COUNTS. A second metric was defined as the effort-weighted sum of the instances of all object types used in the construction phase. The weighting with effort accounts for different amounts of labor required to develop each object type for inclusion in the code for the software project. This metric was called OBJECT-POINTS.

$$OBJECT\ COUNTS_{ConstructionPhase} = \sum_t OBJECT-INSTANCES_t$$

$$OBJECT-POINTS_{ConstructionPhase} = \sum_t OBJECT-EFFORT-WEIGHT_t * OBJECT-INSTANCES_t$$

where

$t$	=	<i>object types used to create application in the Software Construction phase;</i>
$OBJECT-INSTANCES_t$	=	<i>total number of instances of object type <math>t</math> in an application;</i>
$OBJECT-EFFORT-WEIGHT_t$	=	<i>average development effort associated with the construction of object type <math>t</math>;</i>

Table 4 presents correlations between the two object-based metrics defined above and FUNCTION-POINTS. FUNCTION-POINTS is a metric for the size or functionality of the output delivered by a software project to the end user<sup>(3)</sup>. We utilize it as a metric

for SOFTWARE-SIZE-OUTPUT. Its ability to estimate DEVELOPMENT-EFFORT-INPUT has also been investigated and implemented. The FUNCTION-POINT procedure requires counting the occurrence of five function types (Inputs, Outputs, Logical Files, External Interfaces and Queries; for details of the FUNCTION-POINTS procedure, see (15)). These units of software work output refer to the aggregate product delivered; they are not geared towards identifying the output in each phase. Thus, as currently defined, FUNCTION-POINTS cannot be used to implement the *life cycle trajectory* measurement approach to controlling productivity.

In the exploratory study, we viewed obtaining high correlations between FUNCTION-POINTS metric and the object-based metrics being tested as indicators of the validity of the new metrics. Such *convergent validity* with the well-established and well-validated FUNCTION-POINTS metric provides preliminary evidence that the object-based metrics are measures of the same construct that FUNCTION-POINTS purports to measure, i.e., the size and functionality of delivered software as given by SOFTWARE-SIZE-OUTPUT<sub>Construction Phase</sub>. As Table 4 indicates, the OBJECT-COUNT and OBJECT-POINT metrics were highly correlated with FUNCTION-POINTS. OBJECT-COUNTS had a correlation of 0.89 and OBJECT-POINTS had a correlation of 0.86 with FUNCTION-POINTS.

The estimation capability of the object-based metrics was assessed by evaluating their performance in an estimation model to accurately predict the DEVELOPMENT-EFFORT-INPUT that will be consumed for developing a project in the construction phase. Two separate regression models were estimated to predict development effort (the dependent variable) in terms of the output metrics, FUNCTION-POINTS, OBJECT-COUNTS and OBJECT-POINTS (one of these occurred as the independent variable in each model). Results of the regression indicate the extent to which the object-based

metric is able to explain the variance in DEVELOPMENT-EFFORT-INPUT. The results shown in Table 4 reveal an  $R^2$  comparable to the FUNCTION-POINTS metric ( 0.70 and 0.73 for OBJECT-COUNTS and OBJECT-POINTS respectively, compared to 0.75 for FUNCTION-POINTS).

The above research results suggest the viability of the object-based metrics. The two metrics tested, OBJECT-COUNTS and OBJECT-POINTS, performed well as measures for SOFTWARE-SIZE-OUTPUT from the Software Construction phase. They also successfully predicted the total DEVELOPMENT-EFFORT-INPUT required for delivering the completed projects. It remains to test their predictive capability for DEVELOPMENT-EFFORT-INPUT in the Software Construction phase alone. Research is also under way to further define and test object-based metrics for the remaining phases of the CASE life cycle. This is required before the *life cycle trajectory* measurement approach can be more fully implemented.

#### 4. CONCLUSION

In view of the large costs of software, systems for controlling software development should be designed to more closely support the operations and the strategy of the organization. The technology necessary to implement the approach to software development monitoring and control that we advocate is different from what exists in manual software development shops currently. But today, CASE makes implementing our vision of software development tracking increasingly possible.

##### 4.1. Research Contribution

The paper has described a conceptual framework for the development of managerially relevant procedures to enhance software control with *life cycle trajectory metrics*. We also suggested that automating software control is appropriate and

feasible in CASE environments, and that this changes the basic cost-benefit relationship that exists for software project performance tracking. The low cost of measurement made possible through automated analysis and the availability of *repository-based objects* as distinct, identifiable units of development work from each life cycle phase combine to make integrated CASE environments an ideal testbed for research.

Our approach to implementing dynamic control measures forms the first step in a broader attack on CASE project planning and project management methods. Control of software development activities in each phase will support project management activities from the earliest phases of the software life cycle. Tasks such as scheduling, identifying staff requirements, and performing resource planning can be performed on a phase-by-phase basis rather than a project-by-project basis. Moreover, these plans can be revised dynamically as the actual development performance of a phase becomes known. Such an approach will allow more powerful project planning which can more readily adapt to unanticipated changes in performance or parameters.

#### **4.2. Research Agenda**

Our proposals for dynamic trajectory measures open up several new lines of research inquiry for the future.

- [1] Empirical evidence to identify relevant cost drivers for CASE development environments would provide valuable insights into the nature of the cost drivers and the metrics required to track them.
- [2] Research to validate and specify object outputs as measures of work from each of the different phases is needed to provide a rigorous, empirical basis for justifying the implementation of our cost control framework for CASE development.
- [3] Another important extension within our productivity control framework would be to study and compare the estimation accuracy and ease of existing and proposed measurement approaches. Our work on *OBJECT-POINTS* represented an initial

a step in this direction.

We are now involved in investigating the estimation performance of object-based trajectory metrics for phases other than Software Construction. This should result in an integrated cost accounting system for CASE performance tracking which makes use of the features of this development environment. This opens up the opportunity for software production to be integrated with strategy formulation to enable a firm to minimize its strategic software costs.



## REFERENCES

- (1) Alavi, M. High-Productivity Alternatives for Software Development. *Journal of Information Systems Management*, 2(4), Fall 1985, pp. 19-24.
- (2) Alloway, R. M. and Quillard, J. A. User Managers' Systems Needs. *MIS Quarterly*, 7(2), June 1983, 27-41.
- (3) 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.
- (4) Banker, R. D., Kauffman, R. J., Wright, C., and Zweig, D. Automating Output Size and Reusability Metrics in an Object-Based Computer Aided Software Engineering (CASE) Environment. Center for Research in Information Systems Working Paper Series, Stern School of Business, New York University, May 1991.
- (5) Banker, R. D., and Kauffman, R. J. Reuse and Functionality: An Empirical Study of Integrated Computer Aided Software Engineering (ICASE) Technology at the First Boston Corporation. Forthcoming in *MIS Quarterly*, Fall 1991.
- (6) Banker, R. D., Kauffman, R. J., and Kumar, R., An Empirical Test of Object-Based Output Measurement Metrics in a Computer Aided Software Engineering (CASE) Environment. Forthcoming in the *Journal of Management Information Systems*, Winter 1992.
- (7) Benson, R. J. and Parker, M. M. Enterprise Wide Information Management: Strategic Planning For Information Technology - An Introduction for the Business Executive, IBM Los Angeles Scientific Center, G320-2775, January 1986.
- (8) Boehm, B., *Software Engineering Economics*, Prentice Hall, Englewood Cliffs, NJ, 1981.
- (9) Boehm, B. W. and Papaccio, P. Understanding and Controlling Software Costs. *IEEE Transactions on Software Engineering*, 14(10), October 1988, pp. 1462-1477.
- (10) Brooks, F. P., Jr. *The Mythical Man-Month*. Addison Wesley, NY, 1975.
- (11) Burkhard, D. L. and Jenster, P. V. Applications of Computer-Aided Software Engineering Tools: Survey of Current and Prospective Users. *Database*, Fall 1989, pp. 28-37.
- (13) Davis, G. B. Commentary on Information Systems: Productivity Gains from Computer Aided Software Engineering. *Accounting Horizons*, 2(2), June 1988, pp. 90-93.

- (14) Demski, J. S. Information Analysis, Addison-Wesley Publishing, Reading, MA, 1985.
- (15) Dhar, V., Ramesh, B., and Jarke, M. REMAP Project: An Environment for Supporting Requirements Analysis and Maintenance. In Proceedings of Artificial Intelligence and Software Engineering Symposium, AAAI-89, Spring Symposium Series, Stanford, CA, March 1989.
- (16) Dreger, J. B. Function Point Analysis, Prentice Hall, Englewood Cliffs, NJ, 1989.
- (17) Grammas, G. W., and Klein, J. R. Software Productivity as a Strategic Variable. Interfaces 15(3), pp. 116-126, May-June 1985.
- (18) Gurbaxani, V., and Mendelson, H. Software and Hardware in Data Processing Budgets, IEEE Transactions on Software Engineering, SE-13(9), September 1987, pp. 1010-1017.
- (19) Hall, P. A. V. Software Components and Reuse -- Getting More Out of Your Code. Information and Software Technology 29(1), January-February 1987, pp. 38-43.
- (20) Personal communication with Gezinus Hidding, Andersen Consulting, 1990.
- (21) Horowitz, E. and Munson, J. B. An Expansive View of Reusable Software. IEEE Transactions on Software Engineering, SE-10(5), September 1984, pp. 477-487.
- (22) Johnson, H.T. and Kaplan, R.S. Relevance Lost: The Rise and Fall of Management Accounting. Harvard Business School Press, Boston, MA, 1987.
- (23) 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.
- (24) Jones, T. C. Programming Productivity, McGraw-Hill, NY, 1986.
- (25) Kang, K. C., and L. S. Levy. Software Methodology in the Harsh Light of Economics. Information and Software Technology 31(5), June 1989, pp. 239-249.
- (26) Kemerer, C. F. "An Empirical Validation of Software Cost Estimation Models," Communications of the ACM, 30(5), May 1987, pp. 416-429.
- (27) 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, Hawaii, January 1989, pp. 219-227.



- (28) Lederer, A. L., Mirani, R., Neo, B.S., Pollard, C., Prasad, J., and Ramamurthy, K. Information System Cost Estimating: A Management Perspective. MIS Quarterly, Volume 14, Number 2, June 1990, pp. 159-178.
- (29) Mazzucco, F. Automation of Function Counting Techniques, Texas Instruments, 1990.
- (30) McClure, C. The CASE Experience. Byte, April 1989, pp. 235-244.
- (31) McNurlin, B. Building More Flexible Systems. I/S Analyzer, October 1989.
- (32) Moad, J. The Software Revolution. Datamation, February 15, 1990, pp. 22-30.
- (33) 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.
- (34) Nunamaker, J. F. Jr., and Chen, M. Software Productivity: A Framework of Study and an Approach to Reusable Components. In Proceedings of the 22nd Hawaii International Conference System Sciences, Hawaii, January 1989, pp. 959-968.
- (35) Pollack, A. The Move to Modular Software. New York Times, Monday, April 23, 1990, pp. D1-2.
- (36) Ramamoorthy, C. V., Prakash, A., Tsai, W. and Usnda, Y. Software Engineering: Problems and Perspectives. IEEE Computer, 17(10), October 1984, pp. 191-209.
- (37) Sasso, W.C. and McVay, M. The Constraints and Assumptions of Systems Design: A Descriptive Process Model. Center for Research in Information Systems Working Paper #137, Stern School of Business, New York University, September 1990.
- (38) Scacchi, W., and Kintala, C. M. K. Understanding Software Productivity. Technical Report CRI-87-67, Computer Science Department, University of Southern California, Los Angeles, CA, 1987.
- (39) Senn, J. A. and Wynekoop, J. L. Computer Aided Software Engineering (CASE) in Perspective. Working Paper, Information Technology Management Center, College of Business Administration, Georgia State University, 1990.
- (40) Sentry Market Research. CASE Research Report, Westborough, MA, 1990.
- (41) Shah, P. Cost Control and Information Systems, McGraw Hill Book Co., NY, 1981.

- (42) Sprague, R. H. and McNurlin, B. C., Eds. Information Systems Management in Practice, Prentice-Hall, Englewood Cliffs, NJ, 1986.
- (43) Turner, J. A. Understanding Elements of Systems Design. In Critical Issues in Information Systems Research, R. Boland and R. Hirscheim (eds.), John Wiley and Sons, NY, 1986.
- (44) Vicinanza, S., Prietula, M. J. and Mukhopadhyay, T. Case-Based Reasoning in Software Effort Estimation: A Theory, A Model, and A Test, Proceedings of The Eleventh International Conference on Information Systems, Copenhagen, Denmark, December 1990.
- (45) Vipond, S. A. Achieving the Transition to Computer-Aided Software Engineering: A Longitudinal Study of Change and Adaption in Two Software Development Groups. Working Paper, MIS Research Center, Carlson School of Management, University of Minnesota, April 1990.
- (46) Yourdon, E. Whatever Happened To Structured Analysis? Datamation, 32(11), June 1986, pp. 133-138.

Figure 1. Labor Consumption Trajectories for Two Software Development Projects of Similar Size

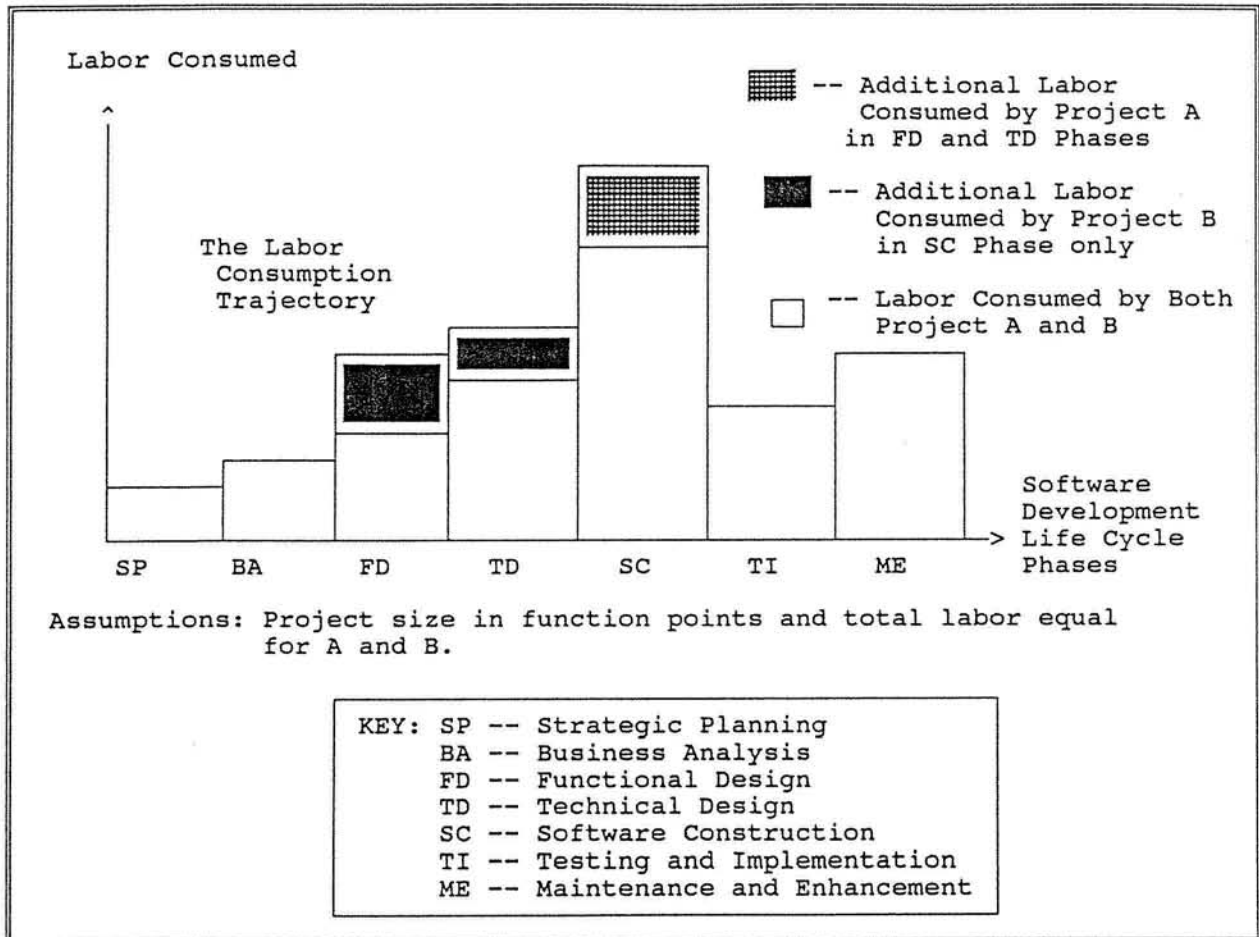
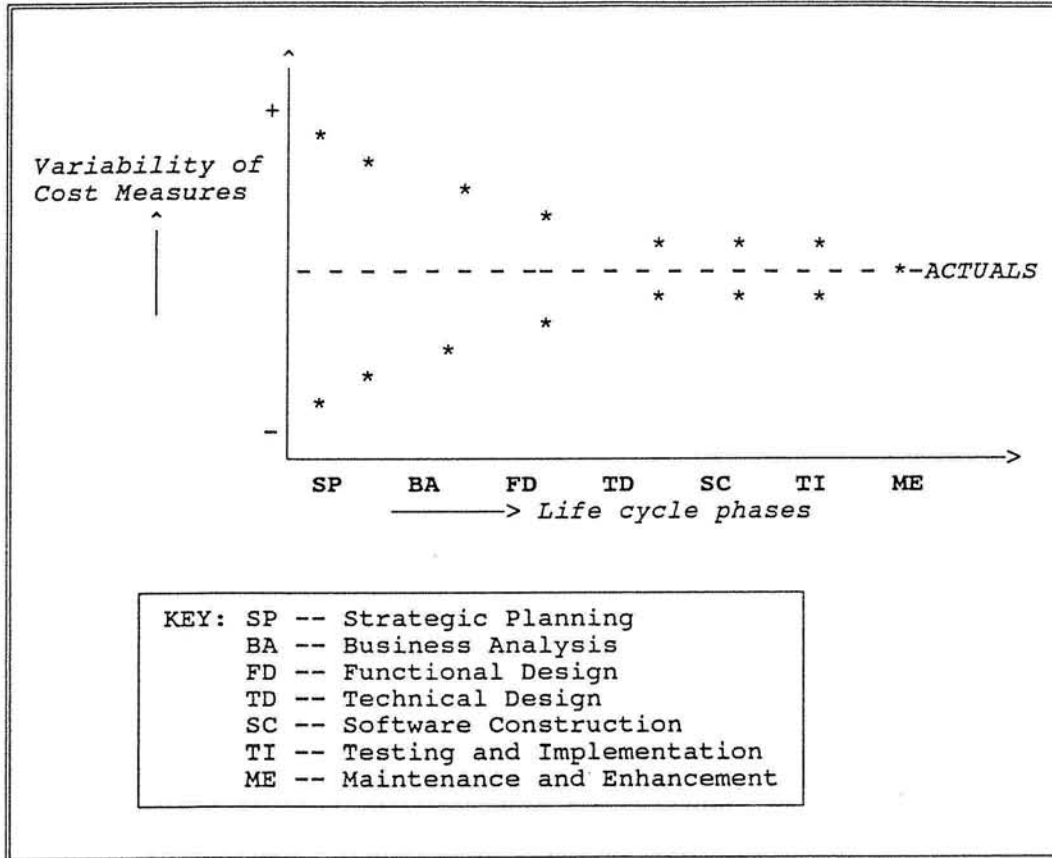


Figure 2. Successive Predictability and Accuracy of Costs



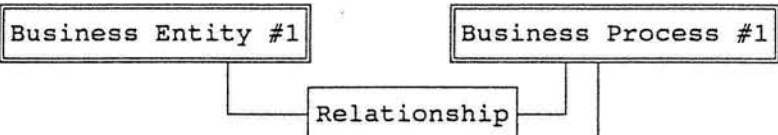
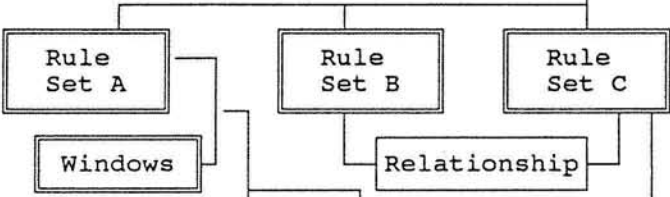
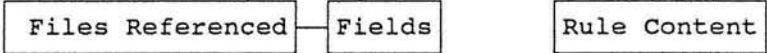
**Table 1: Factors Responsible For Inefficient Software Development**

- Customized application development practices which redevelop from scratch the fundamental procedures and processes that are common across applications or business units in an organization.
- Outdated and error-prone development methodologies that postpone effort to the back end of software development life cycle when the software is coded and implemented; this results in significant additional hidden costs of maintenance.
- Increased complexity, size and scope of the functionality to be incorporated into software for meeting user needs in the competitive environment of a firm's business.
- The labor-intensive nature of software development, which renders software quality and productivity very vulnerable to the skills of the personnel used for development.
- A growth rate in user needs for IT applications that exceeds the growth rate of the supply of experienced and well-trained development staff.

**Table 2. CASE Technology: Cost Impacts of Improved Efficiency and Effectiveness**

MAJOR SOURCES OF CASE BENEFITS	COST IMPACTS	
	EFFICIENCY DIMENSION	EFFECTIVENESS DIMENSION
Productivity Gains	Reuse supports creation of larger amount of software for given level of labor	Products of CASE development create a reusable software infrastructure for the firm, further lowering costs.
Speed of Development	Has potential to help reduce existing backlog of software projects	Allows for flexible, timely response to rapid changes in business goals
Accuracy of Development	Reduces debugging and maintenance costs by lowering error rates	Supports optimizing the functionality of software to meet business/user needs
Methodological Consistency of Development	Provides management with new leverage to manage development labor efficiency across projects	Permits management to make "optimizing" decisions about software labor deployment: software projects need labor with similar toolsets
Traceability of development operations	Enables efficient tracking and coordination of project activities documented on the computer	Enables continuous checking and feedback of project correspondence with initial business specifications
Higher Functionality of Software Product	Brings creation of very complex software within the bounds of routine project development practices	Supports development of visionary projects w/ "blue sky" functionality, and also encourages innovative IT uses
Less onerous technical personnel training requirements	Has potential to combat labor shortages, by reducing the knowledge-intensiveness of software development	Ensures that delivered software is not a function of new programming team's preferences, but dependent on a more fundamental business analysis
Makes the maintenance phase manageable	Maintenance costs are lowered by ensuring that code is highly modularized and well-documented with facilities of the CASE development environment	More careful monitoring of maintenance phase costs can help management to identify the optimal time to stop maintaining and rebuild from scratch to lower overall cost

Table 3. Possible Object Metrics for the CASE Life Cycle

LIFE CYCLE PHASE	POSSIBLE OBJECT METRICS	ILLUSTRATIVE OBJECT HIERARCHY AND COMMENTS
Business Analysis	Entities, Processes, Relationships	<p style="text-align: center;"><u>APPLICATION</u></p>  <pre> graph TD     BE1[Business Entity #1] --- R[Relationship]     BP1[Business Process #1] --- R           </pre>
Functional	Rules, Windows, Views Relationships	 <pre> graph TD     Root[ ] --- RS_A[Rule Set A]     Root --- RS_B[Rule Set B]     Root --- RS_C[Rule Set C]     Root --- W[Windows]     Root --- R[Relationship]     RS_A --- W     RS_B --- R     RS_C --- R           </pre>
Technical Design	Fields, Files, Rules (details)	 <pre> graph TD     FR[Files Referenced] --- F[Fields]     R[Rule Content] --- F           </pre>
Software Construction	Objects built, Objects reused	Above hierarchy must be "navigated" while querying for objects instantiated with code and comprising full functionality. Also must query within and across project hierarchies to identify occurrence of reused objects.
Testing/Implementation	Number of platforms, Number of objects	Currently under investigation.
Maintenance/Enhancement	Number of revisions made to objects	Currently under investigation.

**Table 4: PERFORMANCE OF OBJECT BASED, CONSTRUCTION PHASE METRICS**

METRIC	CORRELATION WITH FUNCTION-POINTS	VALUE OF R-SQUARED
OBJECT-COUNTS	0.89	0.70
OBJECT-POINTS	0.86	0.73
FUNCTION-POINTS	-	0.75





---

---

We wish to acknowledge Mark Baric, Gene Bedell, Tom Lewis and Vivek Wadhwa for the access they provided us to data on software development projects and managers' time throughout our field study of CASE development at the First Boston Corporation and SEER Technologies. Another version of this paper was presented at a conference entitled "Integrating Information Technology and Analysis: How to Deliver Systems Your Clients Will Love," sponsored by the College on the Practice of Management Science of The Institute of Management Science (TIMS/CPMS), and the Operations Research Society of America (ORSA). All errors in this paper are the responsibility of the authors.

---

---