MANAGING DEVELOPMENT PRODUCTIVITY OF THE COMPUTER AIDED SOFTWARE ENGINEERING (CASE) PROCESS WITH DYNAMIC LIFE CYCLE TRAJECTORY METRICS

by

Rajiv D. Banker

Robert J. Kauffman

Rachna Kumar

Center for Digital Economy Research
Stern School of Business
Working Paper IS-90-23

MANAGING DEVELOPMENT PRODUCTIVITY OF THE COMPUTER AIDED SOFTWARE ENGINEERING (CASE) PROCESS WITH DYNAMIC LIFE CYCLE TRAJECTORY METRICS

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

December, 1990

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

Working Paper Series

STERN IS-90-23

Center for Digital Economy Research
Stern School of Business
Working Paper IS-90-23

MANAGING DEVELOPMENT PRODUCTIVITY

OF THE COMPUTER AIDED SOFTWARE ENGINEERING (CASE) PROCESS

WITH DYNAMIC LIFE CYCLE TRAJECTORY METRICS

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 they be monitored and controlled via the application of *dynamic* software development "*life cycle trajectory metrics*." This view develops out of management accounting approaches for process control and recent advances in CASE technology that make automated measurement possible. We suggest that current approaches involve the use of "*static metrics*" for estimation and evaluation, with the result that the depth of the insights they can provide to management is necessarily limited. They only provide "point estimates" of output or productivity at the beginning and end of the project. Yet to manage software development proactively for improved efficiency and effectiveness, management needs to track the range of activities and effort across the entire software development life cycle. This can only be accomplished when timely and relevant information is obtained about the software size output, as well as costs, via "*dynamic metrics*," which provide a richer phase-by-phase view.

We 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. We also thank Jon Turner, Vasant Dhar and the participants of the Technology and Strategy Roundtable at the Wharton School of Business, University of Pennsylvania (November 16, 1990). All errors in this paper are the responsibility of the authors.

Center for Digital Economy Research
Stern School of Business
Working Paper IS-90-23

1. INTRODUCTION

The introduction of computer aided software engineering (**CASE**) tools in software development has radically changed the dynamics of software creation. In fact, CASE tools are believed to represent an industrial revolution in the market for software products. In light of these changes, it is worthwhile to re-examine the methods and approaches for managing software development performance. In this paper, we will argue that CASE offers new opportunities to improve software development control by matching software product to software costs across the development life cycle.

1.1. The Crisis in Software Costs

Cost-effective software development is strategically important for firms seeking to achieve competitiveness through the use of information technology (**IT**) (BENS86, DAVI88, JONE86). The sheer size of the investments in software indicates the depth of the commitments made to IT. For example, industry specialists estimate that by 1990 the total investment in existing, developed and purchased software will be in the neighborhood of 13% of the United States' gross national product, a staggering $527 billion (RAMA84). Other projections reveal an annual increase in software development budgets at the rate of 9% to 12%, exceeding $150 billion per year by 1990 (BOEH88, GURB87). The extent of the hopes that senior managers place in wresting business value from their software investments parallels the magnitude of the dollars spent.

Despite their expenditures, senior managers still regard software development as the major bottleneck in exploiting the potential of IT (GRAM85, BOUL89). 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 (SPRA86, YOUR86). 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 (ALLO83).

Center for Digital Economy Research
Stern School of Business
Working Paper IS-90-23

Reports of software projects months behind schedule and far over budget are also quite common, and, in fact, up to 15% of ongoing software projects are thought to be abandoned due to gross underestimation of required resources (JONE86, MART83). If senior management finds no way to better manage the production of software, their commitment to IT could end up becoming a liability, rather than an asset, for the firm.

This software crisis is attributable to multiple factors (ALAV85, BOEH88, KANG89, SENN90). The most often cited ones include:

* 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.

With the increasing emphasis placed on the role information systems play in obtaining the strategic goals of an organization (CASH88, IVES84, PORT87), the management and control of the software development process represents an increasingly difficult problem that must be solved. A common intermediate goal for senior software development managers is to improve the productivity and quality of software operations. They aim to achieve this by streamlining the life cycle of software creation through the introduction of new development techniques. As a

Center for Digital Economy Research
Stern School of Business
Working Paper IS-90-23

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; 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**), a technology that provides new options for managing and controlling the productivity and costs of software development. Input Inc., a California-based research firm, figures 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 (MOAD90).

1.2. CASE -- An Industrial Revolution in Software Development

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 (BURK89, SENT90). 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 (MCCL89).

CASE technologies and the methodologies that they promote aim to transform the process of software development. Up to the present, software development has essentially been a manual, craft work-like process, but CASE is at the heart of an industrial revolution in the making. It is rapidly transforming the creation of software into a more automated, rigorous and standardized engineering discipline. Paralleling the structure of production in other industries such as automobile manufacturing, home-construction, and even computer hardware

Center for Digital Economy Research
Stern School of Business
Working Paper IS-90-23

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 from reusable templates, plans and procedures (POLL90). This "modular software" approach offers considerable promise to alleviate the major problems causing the software crisis cited above. CASE advocates and firms investing heavily in CASE argue that software automation is the key to increasing productivity, controlling quality, and introducing predictability into the software development process. Thus, CASE is increasingly classified as a "strategic technology," especially among those firms which have moved to implement it early to control longer term software development costs.

Reports on CASE claim a myriad of benefits ranging from 300% productivity increases to 'zero-maintenance' program code. But only a few of these benefits have been rigorously substantiated (KEME89, NUNA89). Studies describing successful implementation of CASE methods and surveys reporting on usage proportions and profiles of CASE tools abound (BURK89, MCCL89, MCNU89).

Norman and Nunamaker (NORM89) investigated the functional and behavioral aspects of CASE technology that contribute most favorably towards increasing the productivity of software engineers. They found that the standardization aspects of CASE technology, enforcing adherence to a disciplined, rigorous and higher quality software development methodology, were perceived to provide the most productivity gains. A different approach to investigating the impacts of CASE techniques was taken by Vipond (VIPO90) in a longitudinal study to identify the behavioral implications of introducing CASE methods into software groups. The study indicated that impacts of CASE on job attitudes and communicative behaviors of software developers can be complex and profound; improvements in the software development process will ultimately need to take into account the behavioral aspects of CASE, as well as carefully manage and control the technical

Center for Digital Economy Research
Stern School of Business
Working Paper IS-90-23

aspects.

While the actual impacts of CASE are yet to be exhaustively validated, the major sources of benefits from CASE can be identified. Banker and Kauffman (BANK91B) present some of the first empirical results to substantiate large productivity gains from using CASE development techniques, especially the leverage created by reusable code. An analysis of the structural and functional dimensions of CASE technology helps to identify the major characteristics of this methodology that contribute towards potential benefits from CASE. These have very broadly been classified by various authors (see, for example, BURK89, MCCL89 and SENN90) 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 (HALL87, JONE84). Standardization could thus lead to reduction in development time as well as an improvement in the quality of software developed. 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 leads to increases in the quality of delivered software.

Thus, standardization and automation can have significant impacts on the efficiency and effectiveness of software development, and thus strategic costs. *Efficiency* refers to how productive software developers are when a CASE methodology is used to develop software. *Effectiveness* relates to how well CASE-developed software accomplishes the business goals of the organization.

The major benefit and cost implications that result from the standardization and automation of software development are

Center for Digital Economy Research
Stern School of Business
Working Paper IS-90-23

described in Table 1 below. What remains is to re-think how management reporting needs to be recast to support the goal of reducing software costs as much as possible with the tools available in this new environment.

INSERT TABLE 1 ABOUT HERE

The remainder of this paper develops a new vision for the management of the software development life cycle in the presence of integrated CASE technologies via automated software metrics. We will make the case that *dynamic life cycle trajectory metrics* made possible by automated development of software projects will help management to realize the benefits of *"software process control"* in a way that was not possible before.

2. A PROPOSAL FOR CASE DEVELOPMENT PROCESS CONTROL

2.1. A New Vocabulary for Software Development Performance Tracking

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 costs and the measurement of subsequent development performance only provide single point measures -- when a project begins or when it has reached completion. Such *"static software development performance metrics"* for cost estimation and efficiency analysis do not provide sufficiently detailed or relevant information for proactively managing the software development process. By contrast, *"dynamic software development performance metrics"* can help management to monitor and control development performance throughout the software development life cycle.

Boehm (BOEH81) has equated the problem of accurately

Center for Digital Economy Research
Stern School of Business
Working Paper IS-90-23

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.

In a similar vein, static 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. 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 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

Center for Digital Economy Research
Stern School of Business
Working Paper IS-90-23

effort during technical analysis and functional design, while Project A consumed more labor during the construction phase.

A similar sketch could be made for productivity in function points, month-by-month, as the construction of a software application proceeds. 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 experience of the analysis and design staff? Was the skill mix or experience of the analysis and design staff of Project B unsuited to the development requirements of the project?

Managers can ask more general questions as well. For example: How much code 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?

Our approach to monitoring software development can be implemented with *dynamic trajectory metrics* which measure performance parameters in each life cycle phase of software development. However, such metrics only become feasible in the CASE environment because the phase activities and phase boundaries are better defined and more rigidly enforced. 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. Process Control Systems and Software Development

The intellectual backdrop of our proposal is found in recent developments in the field of management accounting. Today, it is increasingly recognized that two different types of control systems are needed to facilitate effective management: *product costing* and *process control* (KAPL88). The normal approach to software development productivity management compiles the total costs for producing software, and accounts it against the

Center for Digital Economy Research
Stern School of Business
Working Paper IS-90-23

aggregate software delivered, as described in the equation below:

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

This is akin to *product costing* systems. Product costing is advocated in the accounting discipline in such contexts as pricing and valuing products. It is useful, for example, to provide information to support project bidding, but product costing is not capable of providing information that enables *dynamic* performance evaluation as a project proceeds. The problem with obtaining dynamic productivity measures arises because existing output measurement approaches are not geared to gauging software size at intermediate points of the software development life cycle. Examples of such "end-point" output size estimation and measurement approaches include source-lines-of-code-based models like COCOMO (LOW90) and SLIM (KEME87), and function points (LOW90).

By contrast, *process control systems* are responsible for facilitating operational functions (COOP88). Operational control allows management not just to *value* the total cost of the delivered software (as in product costing), but also to *control* the costs as software development occurs over the project life cycle. Dynamic measurement can be performed to diagnose factors driving the costs of operations as the development proceeds. Information on the nature and impact of cost drivers can be used to make tradeoff and compromise decisions, and adjustments in the process based on sensitivity analyses.

Both the software costing approach and the software process control approach to measuring software activities are relevant in the management of software development. However, the ability to control and influence process costs is critical in meeting the challenge of building strategically beneficial software assets. Thus, the primary productivity control framework in terms of the frequency and approach to measuring a software project should be based on the measurement principles that support optimal process

Center for Digital Economy Research
Stern School of Business
Working Paper IS-90-23

control.

Management accounting distinguishes among four requirements for software product costing and process control systems (COOP88, JOHN87). Table 2 summarizes these requirements.

INSERT TABLE 2 ABOUT HERE

Nature of Costs: For effective control of the software development process, development costs should be considered variable with respect to all relevant cost drivers. Software product costing systems don't adequately diagnose the causes of cost variances; they only use labor cost figures captured when development has been completed. So, an approach that incorporates a more effective treatment of cost drivers is needed to reflect their nature and impacts on project costs.

Management Scope: A time-tested principle of management is that managers should only be accountable for those activities that they can influence directly. Individual project managers are held accountable for their project's development performance. But, they only can influence the costs of their projects by reacting to process control measures that permit corrective actions to be taken as development proceeds. The information provided by static product costing approaches can best be used by departmental or senior managers in comparing performance across projects being developed at that same time or historically over time. Thus, if controlling or influencing the internal operations of a project is the major concern, project managers should be supported by process control systems that cover their responsibilities across the entire life cycle.

Time Horizon: Another important characteristic of process management systems is their ability to explain variances in *short term* software development costs. The key to achieving this is the definition of the "short-term" time horizon in the context of software development. This is the time period during which we expect constructive process control opportunities to occur

Center for Digital Economy Research
Stern School of Business
Working Paper IS-90-23

(BRUN87). Control opportunities are traditionally known to coincide with the occurrence of a measurable unit of work. In other words, to be useful, the frequency of process control information should match the cycle of the software production. Our premise is that productive decisions only can be made at the natural breakpoints that occur during the production -- especially as phases end -- and so measurement procedures should deliver information that is relevant to decision making at these points.

Reporting Frequency: The design of existing static software development productivity measures was justified in manual development environments since a traditionally-developed software project was only concretely and unambiguously measurable upon completion. But, project completion is not the only time that decisions can be made which affect the software development process, and this is especially true for CASE development. For example, a manager may wish to determine whether schedule overruns are being caused by inefficient design, error-prone coding, or unexpected implementation difficulties. Thus, there is a need for more frequent reporting to support the shorter time horizon of measurable software development in each life cycle phase.

Since software development projects have become so much larger and more complex, the completion point should no longer be viewed as the only concrete decision time. As the software development life cycle proceeds, each phase becomes a distinct sub-process of the overall production of software. Upon closer inspection, each phase would seem to have different outputs, different conversion efficiencies, and, thus, different parameters for management action. And, each phase often has qualitatively different inputs as the composition of the development team assigned to the project changes over time to match the difficulties presented by development in each life cycle phase.

Center for Digital Economy Research
Stern School of Business
Working Paper IS-90-23

2.3. Automating Dynamic Trajectory Metrics for Process Control: Benefits and Costs

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 deliver perfect information as the life cycle progressed without forcing a project manager to incur unacceptably high costs. But CASE changes this cost-benefit relationship.

Benefits of Measurement: The benefits of information that describe 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. *First*, measures that are collected should be able to resolve decision options. Dynamic life cycle metrics enable actions that influence subsequent software development activities. *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.

Thus, it is reasonable to expect that the value of very accurate and detailed information to a project manager in the earlier life cycle phases is probably less than its value in the later phases. 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,

Center for Digital Economy Research
Stern School of Business
Working Paper IS-90-23

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.

Costs of Measurement: The other issue in committing to dynamic 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, else it will reduce management's motivation to measure. Johnson and Kaplan 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"* (JOHN87, 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 process control we are now advocating.

Automated Measurement: But, today's CASE development environments make it possible to automate dynamic software development life cycle 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 in developing dynamic cost measurement procedures for software development is to reduce the costs of measurement itself. Automation of measurement metrics in the CASE environment can provide ongoing control information such

Center for Digital Economy Research
Stern School of Business
Working Paper IS-90-23

that the decision value outweighs the costs.[1]

3. CONTROLLING CASE DEVELOPMENT COSTS WITH DYNAMIC TRAJECTORY METRICS

3.1. Requirements For Dynamic Control of CASE Development Costs

Effective software cost control systems should deliver three basic capabilities to management (SHAH81):

[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 in subsequent stages of the production process.

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

Measuring the costs associated with the work of software development should take account of 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 (HORO84). 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

---

[1]In fact, product development in this area is underway for a number of CASE development environments, including Texas Instrument's IEF (MAZZ90), Andersen Consulting's Foundation (HIDD90), and Seer Technologies' High Productivity Systems CASE tools (BANK90). 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.

Center for Digital Economy Research
Stern School of Business
Working Paper IS-90-23

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* (BROO75).

In fact, even experts tend to underestimate software project development times, and in spite of this awareness projects continue to be behind schedule and budget. Moreover, sometimes irrational political perspectives influence the cost estimation process, and have important ramifications for taking meaningful managerial actions to improve estimation (LEDE90). Advances in more formal approaches to measuring software size have tested empirical models that predict development time based on historical relationships between software size and development labor. (These include models such as COCOMO, ESTIMACS and SLIM, as discussed in KEME87.)

The third requirement, the ability to isolate variances between estimated and actual cost measures is a diagnosis capability which provides answers to 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 input (SCAC87, BOEH81). These attributes can be classified into program attributes (e.g., reliability requirements), environment attributes (e.g., main memory constraints), personnel attributes (e.g., average

Center for Digital Economy Research
Stern School of Business
Working Paper IS-90-23

experience of project team), and project attributes (e.g., type of development tool used).

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 (BOEH81, VICI90). 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 considerably complicates the isolation and correction of variances, and meanwhile places a premium on obtaining better information throughout the life cycle.

3.2. CASE Repository Objects: A Basis for Dynamic Trajectory Metrics

In order to implement a dynamic software process control system incorporating trajectory metrics, we need to identify a sound basis for designing the specific metrics which measure cost efficiency parameters at relevant intermediate points in the development life cycle. We have established that these relevant intermediate points are the endpoints of the life cycle phases. We have also stressed that diagnostic ability in controlling costs can be achieved only by regarding costs as variable with respect to all cost drivers. This suggests the need for the

Center for Digital Economy Research
Stern School of Business
Working Paper IS-90-23

following functional relationship to be tested:

$$DEVELOPMENT\text{-}LABOR\text{-}INPUT_p = f(COST\text{-}DRIVERS\text{-}FOR\text{-}CASE)_p$$

where p indicates the phase of the life cycle in which measurement occurs. Thus, trajectory metrics should be based on measures of DEVELOPMENT-LABOR-INPUT and COST-DRIVERS-FOR-CASE for each development phase.

DEVELOPMENT-LABOR-INPUT 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.

Phase measures for the COST-DRIVERS require a more radical 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, SOFTWARE-PRODUCT-OUTPUT and REUSE-LEVEL impact development labor significantly (BANK91B, KARI90). DEVELOPMENT-TEAM-EXPERIENCE can generally be measured with subjective rating methods for each phase. A bigger challenge is to develop trajectory metrics for the SOFTWARE-PRODUCT-OUTPUT from each phase.

REUSE-LEVEL refers to the use of existing code in order to program an application. Reused code thus adds to the size and functionality of the delivered software product without requiring

Center for Digital Economy Research
Stern School of Business
Working Paper IS-90-23

a proportionate amount of development labor. This justifies its inclusion as an important cost driver for DEVELOPMENT-LABOR-INPUT. REUSE-LEVEL is measured in terms of the proportion of reused code in the total SOFTWARE-PRODUCT-OUTPUT. The proportion of reused code in the final software product is measured in terms of the same units of work output that are used for SOFTWARE-PRODUCT-OUTPUT. Thus, measures for both SOFTWARE-PRODUCT-OUTPUT and REUSE-LEVEL 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 (DHAR89, SASS88, TURN86).

However, 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 we call *"objects"* (BANK91A). The results indicated that because objects act as building blocks to construct the functionality of

Center for Digital Economy Research
Stern School of Business
Working Paper IS-90-23

the software, 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 business CASE environments are: RULES, SCREEN DEFINITIONS, USER REPORTS, and so on.[2] 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.

3.3. Trajectory Approaches for the CASE Life Cycle: Some Proposals

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. In integrated CASE environments (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. 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

---

[2]For additional details on an integrated CASE environment (ICE) that has some of these features, see BANK90A.

Center for Digital Economy Research
Stern School of Business
Working Paper IS-90-23

additional objects in later phases. 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. This stage often uses tools such as an Entity-Relationship Diagrammer or a Process Hierarchy Diagrammer, and typically outputs objects such as ENTITIES, PROCESSES, RELATIONSHIPS (between ENTITIES and PROCESSES). These are generic objects, 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 technical phase are FIELDS, FILES, RULE details, and so on. *Software Construction* involves generation of all code at the source level. Reusable objects need merely retrieve code from the repository while objects that have to be written from scratch will require much more labor. Thus, the

Center for Digital Economy Research
Stern School of Business
Working Paper IS-90-23

REUSE-LEVEL will affect DEVELOPMENT-LABOR-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 the 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-PRODUCT-OUTPUT from each phase. Since the reuse cost driver is also dependent on the object unit of work, REUSE-LEVEL can also be distinctly identified for each phase. This equips us with the capability to perform the dynamic software process control necessary for reaping significant cost savings from CASE development methodologies.

4. CONCLUSION

In view of the large costs of software, cost control systems for 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 systems that we advocate is radically different from what exists in most 3GL development shops today. But today, CASE makes implementing our vision of software development tracking increasingly possible.

4.1. Research Contribution

The paper has described the conceptual framework for the development of managerially relevant procedures to enhance software process control with *dynamic software development performance trajectory metrics*. We also have suggested that automating software process 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

Center for Digital Economy Research
Stern School of Business
Working Paper IS-90-23

each life cycle phase combine to make integrated CASE environments ideal testbeds for research on trajectory metrics.

Our approach to implementing dynamic cost 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 scheduling, identifying staff requirements and performing resource planning can be performed on a phase-by-phase basis, rather than on a project-by-project basis. Moreover, these plans can be revised dynamically as the actual performance occurring in a phase becomes known. Such an approach will allow more powerful project planning which can more readily adapt to unanticipated changes in performance or project parameters.

A great deal of potential exists to increase management effectiveness by using phase-based performance measures. Their use opens up the possibility of conducting new, rich and insightful analyses that cannot be conducted using today's aggregate and static performance measures. As management's database of phase-by-phase performance results grows, project managers can utilize the historical experience to fine tune their management actions in the short term. They can identify the most productive practices, perform sensitivity analyses of environmental and functional features on project parameters, and make trade-off to optimize productivity, hit a target cost, reach a given level of software quality, or match a tough deadline.

We envision a dynamic software project control environment which integrates:

* more accurate project planning based on better phase estimates;

* more proactive project management whose decisions are based on sensitive phase measures;

* more flexible plan revision based on diagnosing differences between phase plans and actuals.

Such an integrated software project control environment

Center for Digital Economy Research
Stern School of Business
Working Paper IS-90-23

places the control back in the hands of the project manager.

4.2. Research Agenda

The paper has described the conceptual framework for the development of managerially relevant procedures to enhance software process control with *software development performance trajectory metrics*. We also have suggested that automating software process control is appropriate and feasible in CASE environments. The low cost of measurement and the availability of *objects* as distinct, identifiable units of development work from each life cycle phase combine to make the CASE tool we are studying at Seer Technologies and First Boston Corporation an ideal testbed for research on trajectory metrics.

Our proposal for trajectory measures opens 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 the different phases is also needed to provide a rigorous, empirical basis for justifying the implementation of our cost control framework CASE.

[3] Another important extension within our cost control framework would be to study and compare the estimation accuracy and ease of different measurement approaches. Our work on "*object points*" is a step in this direction (BANK91A).

We are now involved in investigating the estimation performance of dynamic, object-based trajectory metrics. This should further our goal of developing an integrated CASE process control system which makes use of the features of the CASE development environment. Software production can then be matched more closely with strategy formulation to enable a firm to minimize its strategic software costs.

Center for Digital Economy Research
Stern School of Business
Working Paper IS-90-23

REFERENCES

ALAV85 Alavi, M. High-Productivity Alternatives for Software Development. *Journal of Information Systems Management*, 2(4), Fall 1985, pp. 19-24.

ALLO83 Alloway, R. M. and Quillard, J. A. User Managers' Systems Needs. *MIS Quarterly,* 7(2), June 1983, 27-41.

BANK90 Banker, R. D., Fisher, E., Kauffman, R. J., Wright, C., and Zweig, D. Automating Software Development Performance Metrics. Working Paper, Stern School of Business, New York University, September 1990.

BANK91A Banker, R. D., Kauffman, R. J., and Kumar, R., Output Metrics for Object-Oriented, Integrated Computer Aided Software Engineering (CASE): Critique, Evaluation and Proposal. Forthcoming in the *Proceedings of the Twenty-Fourth Hawaii International Conference on System Sciences*, Hawaii, January 1991, pp. 327-339.

BANK91B Banker, R. D., and Kauffman, R. J. An Empirical Study of Computer Aided Software Engineering (CASE) Technology: Productivity, Reuse and Functionality. Forthcoming in *MIS Quarterly*, 1991.

BENS86 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.

BOEH81 Boehm, B., *Software Engineering Economics*, Prentice Hall, Englewood Cliffs, NJ, 1981.

BOEH88 Boehm, B. W. and Papaccio, P. Understanding and Controlling Software Costs. *IEEE Transactions on Software Engineering*, 14(10), October 1988, pp. 1462-1477.

BOUL89 Bouldin, Barbara M. CASE: Measuring Productivity -- What Are You Measuring? Why Are You Measuring It? *Software Magazine* 9(10), August 1989, pp. 30-39.

BROO75 Brooks, F. P., Jr. *The Mythical Man-Month*. Addison Wesley, NY, 1975.

BRUN87 Bruns, W. and Kaplan, R. S. eds., *Field Studies in Management Accounting and Control*, Harvard Business School Press, Boston, MA, 1987.

Center for Digital Economy Research
Stern School of Business
Working Paper IS-90-23

BURK89 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.

CASH88 Cash, J., McFarlan, F., McKenney, J. and Vitale, M. *Corporate Information Systems Management: Text and Cases,* Irwin, Homewood, IL, 1988.

COOP88 Cooper, R. and Kaplan, R.S. Measure Costs Right: Make the Right Decisions. *Harvard Business Review*, September-October, 1988, pp 96-103.

DAVI88 Davis, G. B. Commentary on Information Systems: Productivity Gains from Computer Aided Software Engineering. *Accounting Horizons*, 2(2), June 1988, pp. 90-93.

DHAR89 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.

GRAM85 Grammas, G. W., and Klein, J. R. Software Productivity as a Strategic Variable. *Interfaces* 15(3), pp. 116-126, May-June 1985.

GURB87 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.

HALL87 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.

HIDD90 Personal communication with Gezinus Hidding, Andersen Consulting, 1990.

HORO84 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.

IVES84 Ives, B., and Learmonth, G. The Information System as a Competitive Weapon. *Communications of the ACM,* 27(12), December 1984, pp. 1193-2101.

JOHN87 Johnson, H. T. and Kaplan, R. S. *Relevance Lost: The Rise and Fall of Management Accounting*. Harvard Business School Press, Boston, MA, 1987.

Center for Digital Economy Research
Stern School of Business
Working Paper IS-90-23

JONE84 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.

JONE86 Jones, T. C. *Programming Productivity*, McGraw-Hill, NY, 1986.

KANG89 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.

KAPL88 Kaplan, R. S. One Cost System Isn't Enough. *Harvard Business Review*, January-February, 1988, pp. 61-66.

KARI90 Karimi, J. An Asset-Based Systems Development Approach to Software Reusability. *MIS Quarterly*, 14(2), June 1990, pp. 179-200.

KEME87 Kemerer, C. F. "An Empirical Validation of Software Cost Estimation Models," *Communications of the ACM*, 30(5), May 1987, pp. 416-429.

KEME89 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.

LEDE90 Lederer, A. L., Mirani, R., Neo, B. S., Pollard, C., Prasad, J., and Ramamurthy, K. Information System Cost Estimating: A Management Perspective *MIS Quarterly*, 14(2), June 1990, pp. 159-178.

LOW90 Low, G. C., and Jeffrey, D. R. Function Points in the Estimation and Evaluation of the Software Process. *IEEE Transactions on Software Engineering* 16, January 1990, pp. 64-71.

MART83 Martin, E. W. Strategy for a DoD Software Initiative. *IEEE Computer*, 16(3), March 1983, pp. 52-59.

MAZZ90 Mazzucco, F. Automation of Function Counting Techniques, Texas Instruments, 1990.

MCCL89 McClure, C. The CASE Experience. *Byte*, April 1989, pp. 235-244.

MCNU89 McNurlin, B. Building More Flexible Systems. *I/S Analyzer*, October 1989.

MOAD90 Moad, J. The Software Revolution. *Datamation*, February 15, 1990, pp. 22-30.

Center for Digital Economy Research
Stern School of Business
Working Paper IS-90-23

NORM89 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.

NUNA89 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.

POLL90 Pollack, A. The Move to Modular Software. *New York Times*, Monday, April 23, 1990, pp. D1-2.

PORT87 Porter, M. From Competitive Advantage to Corporate Strategy. *Harvard Business Review*, May-June 1987, pp. 43-59.

RAMA84 Ramamoorthy, C. V., Prakash, A., Tsai, W. and Usnda, Y. Software Engineering: Problems and Perspectives. *IEEE Computer*, 17(10), October 1984, pp. 191-209.

SASS88 Sasso, W.C. and McVay, M. The Constraints and Assumptions of Systems Design: A Descriptive Process Model. Working Paper, CRIS #137, Stern School of Business, New York University, September 1988.

SCAC87 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.

SENN90 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.

SENT90 Sentry Market Research. CASE Research Report, Westborough, MA, 1990.

SHAH81 Shah, P. *Cost Control and Information Systems*, McGraw Hill Book Co., NY, 1981.

SPRA86 Sprague, R. H. and McNurlin, B. C. (eds.) *Information Systems Management in Practice*, Prentice-Hall, Englewood Cliffs, NJ, 1986.

TURN86 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.

Center for Digital Economy Research
Stern School of Business
Working Paper IS-90-23

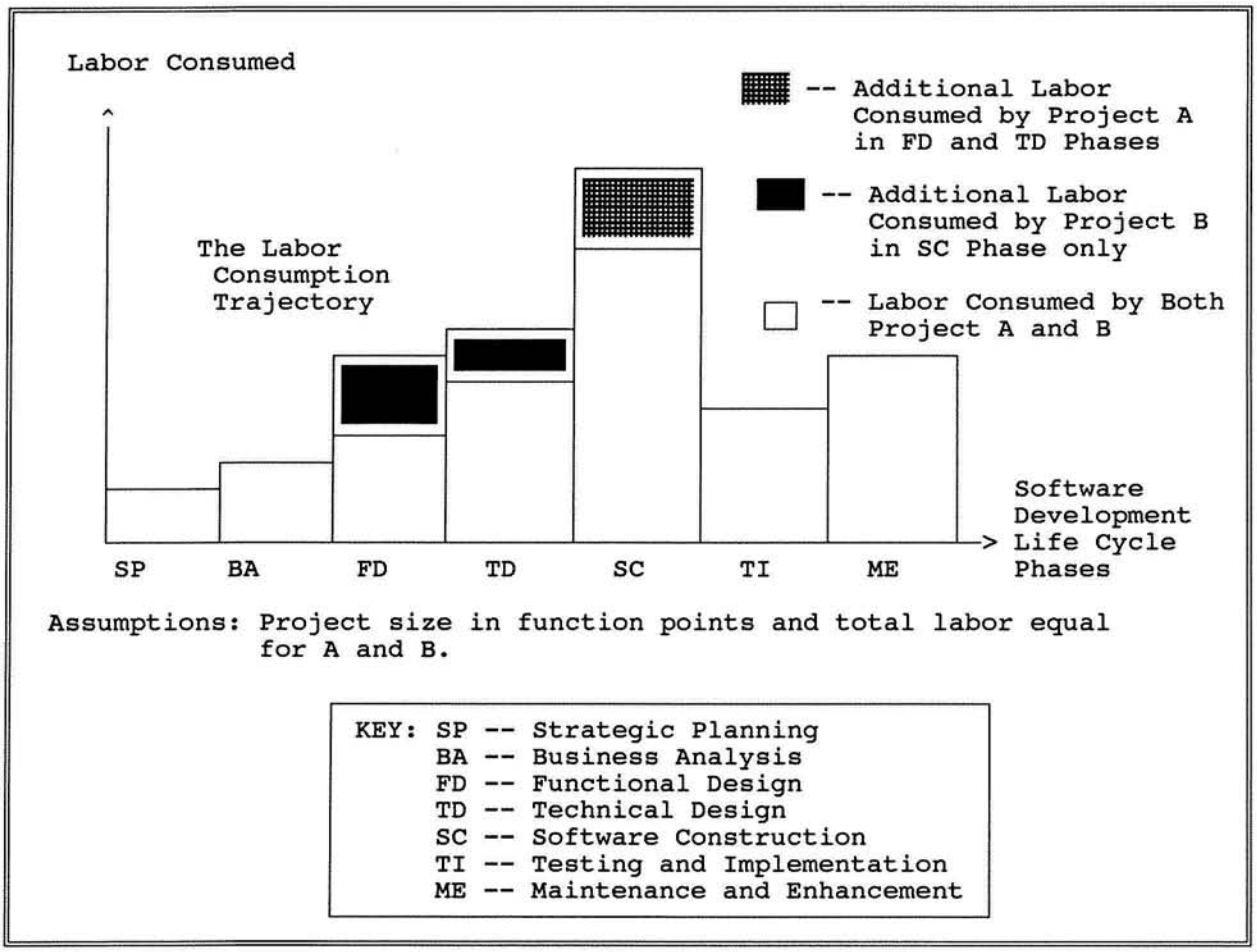
VICI90 Vicinanza, S., Prietula, M. J. and Mukhopadhyay, T. Case-Based Reasoning in Software Effort Estimation: A Theory, A Model, and A Test, forthcoming in *Proceedings of The Eleventh International Conference on Information Systems*, Copenhagen, Denmark, December 1990.

VIPO90 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.

YOUR86 Yourdon, E. Whatever Happened To Structured Analysis? *Datamation*, 32(11), June 1986, pp. 133-138.

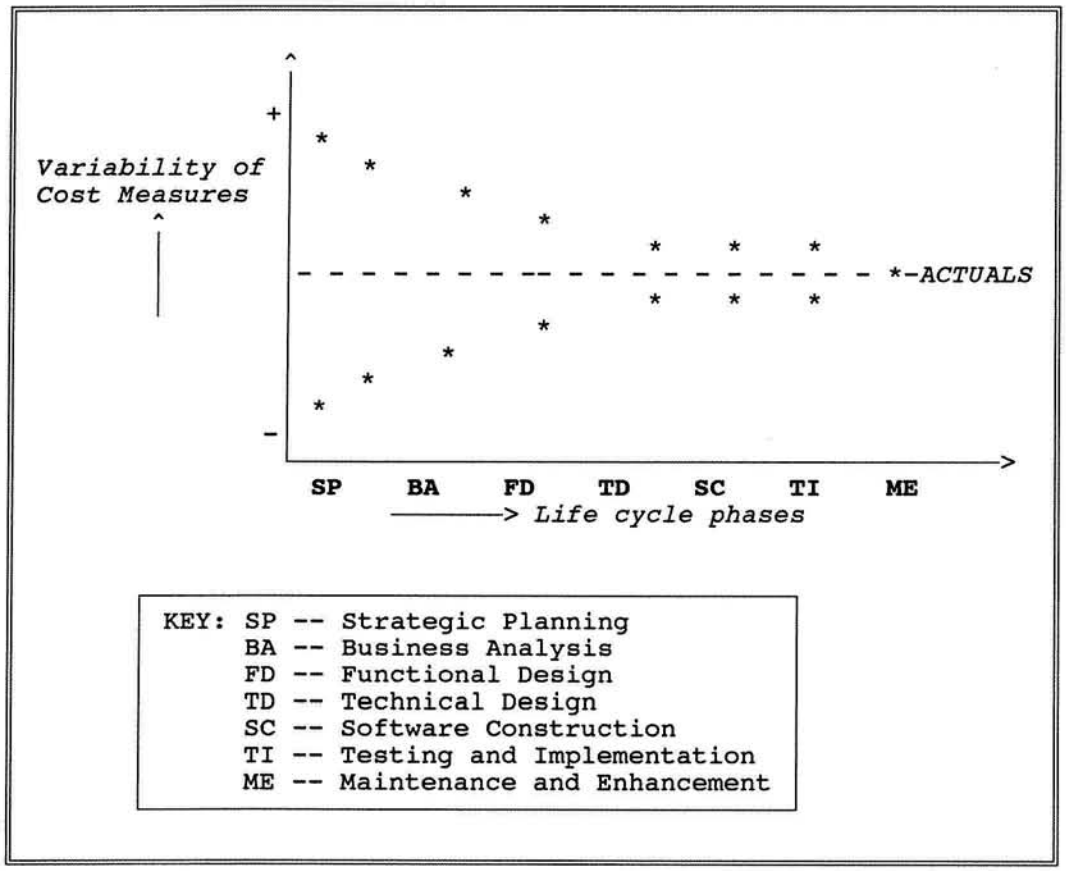
Center for Digital Economy Research
Stern School of Business
Working Paper IS-90-23

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



Center for Digital Economy Research
Stern School of Business
Working Paper IS-90-23

Figure 2. Successive Predictability and Accuracy of Costs



Center for Digital Economy Research
Stern School of Business
Working Paper IS-90-23

Table 1. 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 training requirements for personnel	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

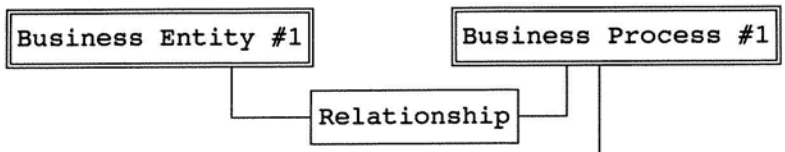
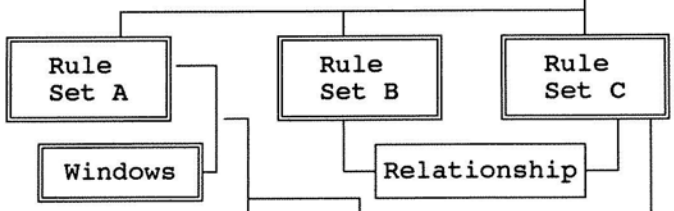
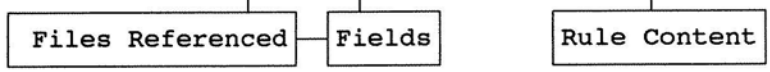
Center for Digital Economy Research
Stern School of Business
Working Paper IS-90-23

Table 2. Determinants of Product Costing and Process Control Systems for Software Development

MANAGEMENT ACCOUNTING SYSTEMS	ATTRIBUTES OF ACCOUNTING SYSTEM FOR SOFTWARE DEVELOPMENT			
	NATURE OF COSTS	*MANAGEMENT SCOPE*	*TIME HORIZON*	*REPORTING FREQUENCY*
PROCESS CONTROL	Variable; incorporates examination of all key cost drivers	Effected by project manager and limited to single project	Short-term; repetitive; and future-oriented	By unit of measurable software development work
PRODUCT COSTING	Variable; focuses only on labor consumed in terms of dif-sizes of software development outputs	Effected by senior manage-ment as gauge of comparative development performance of multiple project teams	Long-term; non-repeti-tive since linked to completion of specific projects; historical perspective	At completion of the entire project(s)
Adapted for software development from Kaplan (KAPL88).				

Center for Digital Economy Research
Stern School of Business
Working Paper IS-90-23

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, Relation-ships	
Func-tional	Rules, Windows, Views Relation-ships	
Tech-nichal Design	Fields, Files, Rules (details)	
Software Construc-tion	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/ Implemen-tation	Number of platforms, Number of objects	Currently under investigation.
Mainte-nance/ Enhance-ment	Number of revisions made to objects	Currently under investigation.

Center for Digital Economy Research
Stern School of Business
Working Paper IS-90-23

Center for Digital Economy Research
Stern School of Business
Working Paper IS-90-23