OUTPUT MEASUREMENT METRICS IN AN OBJECT-ORIENTED COMPUTER AIDED SOFTWARE ENGINEERING (CASE) ENVIRONMENT: CRITIQUE, EVALUATION AND PROPOSAL

by

Rajiv Banker
Arthur Andersen Professor of Accounting and Information Systems
Carlson School of Business
University of Minnesota

Robert J. Kauffman
Assistant Professor of Information Systems
Stern School of Business
New York University

and

Rachna Kumar
Doctoral Program in Information Systems
Stern School of Business
New York University

October 1990

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

Working Paper Series

STERN IS-90-12

Forthcoming in The Proceedings of the 1991 Hawaii International Conference on System Sciences.

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. We also wish to thank Eric Fisher and Charles Wright for assisting with the data collection. In addition, Dani Zweig provided useful suggestions on the content and presentation of the ideas in this paper. Jon Turner helped us to formulate this research at an early stage with ideas that are central to this paper. Finally, we thank the National Science Foundation for partial funding of the data collection under grant SES-8709044. All errors in this paper are the responsibility of the authors.

Center for Digital Economy Research
Stern School of Business
Working Paper IS-90-12

OUTPUT MEASUREMENT METRICS IN AN OBJECT-ORIENTED COMPUTER AIDED SOFTWARE ENGINEERING (CASE) ENVIRONMENT: CRITIQUE, EVALUATION AND PROPOSAL

Rajiv D. Banker
Carlson School Of Business
University Of Minnesota

Robert J. Kauffman
Stern School Of Business
New York University

Rachna Kumar
Stern School Of Business
New York University

Abstract

Output measurement metrics for the software development process need to be re-examined to determine their performance in the new, radically changed CASE development environment. This paper critiques and empirically evaluates several approaches to the measurement of outputs from the CASE process. The primary metric evaluated is the *function points* method developed by Albrecht. A second metric tested is a *short-form variation* of function points that is easier and quicker to calculate. We also propose a new output metric called *object points* and a related short-form, which are specialized for output measurement in object-oriented CASE environments that include a central object repository. These metrics are proposed as more intuitive and lower cost approaches to measuring the CASE outputs. Our preliminary results show that these metrics have the potential to yield as accurate, if not better, estimates than function points-based measures.

1. INTRODUCTION

The productivity impacts and business value implications of computer aided software engineering (CASE) tools are of increasing concern to researchers as well as practitioners in the software community. However, convincing results in this area have been difficult to obtain. The lack of results can be attributed to a number of difficulties ranging from poor data availability to limitations of current evaluation approaches (KEME89). Thus there is substantial motivation to conduct research on measurement metrics that are conducive to building a cumulative base of valid and reliable estimates for the outputs of CASE development.

A survey conducted by *Software Magazine* reports that only 13% of CASE-using firms out of the 196 surveyed have a productivity measurement program of any kind in place (BOUL89). Such surveys are indicative of an urgent need for measurement approaches which identify and substantiate CASE-related productivity improvements. Appropriate measurement approaches will not only allow comparisons across different software development environments, they will also increase the effectiveness of systems that aim to improve strategic cost management by tracking software development productivity (BANK90b).

However, before we measure, we need to establish robust metrics as measurement units. Existing measurement approaches were developed and validated for third generation language (3GL) software development environments. The CASE environment, however, is radically different in terms of both the structural and functional dimensions of systems development (SENN90). Although these well established methods can be brought to bear on the problem of CASE productivity, they must be scrutinized, and recalibrated to ensure they remain valid in this new CASE development process.

This paper examines the issue of output measurement for CASE-based software development. Our aim is to critique and empirically evaluate several measurement approaches for software development using CASE tools. Our initial emphasis is on the appropriateness of *function points* as a measure of the functionality delivered by CASE-developed systems. The function points methodology was developed by Albrecht to measure the intrinsic size of a system (ALBR79). We also examine a *short-form variation* of function points obtained as an intermediate step in the calculation of function points. We investigate this variation because it is quicker and cheaper to calculate, and as applicable in the CASE development environment as the function points metric.

Another approach, which we call *object points analysis*, represents a new proposal that involves counting repository objects developed in an integrated CASE environment. This approach is proposed as a more intuitive way to measure system functionality when developers use object-oriented CASE development procedures. We will evaluate this third metric which is based directly on the number of

Center for Digital Economy Research
Stern School of Business
Working Paper IS-90-12

objects comprising an application, as well as a fourth metric that weights objects for their relative complexity in terms of the average time it takes to build them.

We present estimation performance results of the four alternative metrics in terms of their ability to predict software development effort. *Estimation performance* refers to the ability of a software output measurement metric to accurately predict the amount of software development labor consumed in a project. This will enable us to assess the extent to which each of the metrics actually measures the size of the software.

1.1. Function Points As An Output Metric

Function points is a metric for the size of the output from the software development process. A function point is defined as the size of one end-user business function (ALBR79). It was originally developed as a means to track productivity in terms of function points delivered per person month of development effort. Subsequent research has investigated the ability of apriori estimates of function points to *predict* the effort required for developing software (LOW90, RUDO84, KEME87).

The function point procedure requires the analyst to identify the occurrence of each unique Input, Output, Logical File, External Interface, and Query types delivered by the software. In this research, we call the sum of all function type occurrences the *RAW-FUNCTION-COUNT*. RAW-FUNCTION-COUNTS are then weighted with numbers that reflect the value of that function type to the user. These are referred to as *WEIGHTED-FUNCTION-COUNTS*. This sum is then adjusted using ratings on fourteen complexity factors that reflect the complexity of the system requirements and the development environment. The adjustment score is called the *TECHNICAL-COMPLEXITY-FACTOR*. Finally, function points are calculated as WEIGHTED-FUNCTION-COUNTS * TECHNICAL-COMPLEXITY-FACTOR. Table 1 gives further details.

A number of factors support the choice of function points as the measurement approach to be evaluated. It is widely accepted as a de facto industry standard (ALBR83, JONE86, SYMO88, LOW90). In use today are many flavors of function point counting, including ESTIMACS (RUBI83), SPQR (JONE86), MARK II (SYMO88), IFPUG (IFPU88) and IBM (IBM89). Rules for counting function points have been rigorously defined, and agreed upon by their more enthusiastic users (DREG89, IFPU88). Function points also have advantages over source-lines-of-code methods of effort estimation because they can be estimated earlier in the development cycle, and are indepenedent of the language and technology used (ALBR79, LOW90). Kemerer (KEME87) reports that Albrecht's function points method led to a smaller average error rate in estimating software applications, when compared to alternate output measurement methods including COCOMO, SLIM (popular source-lines-of-code based models) and ESTIMACS. Finally, Jones states that most CASE customers with measurement plans are basing their metric on function points (BOUL89).

Table 1. The Function Points Procedure

STEP 1: **Identification of RAW-FUNCTION-COUNTS.** Identify each functionality unit and classify into the five user function types: Input Type, Output Type, External Interface Type, Logical File Type, Query Type. This step yields RAW-FUNCTION-COUNTS for the five different function types, and we will refer to the sum of these raw-function-counts as RFC.

STEP 2: **Classification of Simple, Average and Complex Function Types.** The RAW-FUNCTION-COUNTS are further classified into three complexity levels depending on the number of data elements contained in each count, and the number of files referenced. Each subtype is weighted with numbers reflecting the relative value of the function to the user. For example, a Simple Input Type would be weighted by 3, while a Complex Input Type would be weighted by 4. The weighting scheme proposed by Albrecht is:

FUNCTION TYPE	FUNCTION-COMPLEXITY-SCORES		
	Simple	Average	Complex
Inputs	3	4	6
Outputs	4	5	7
Interfaces	5	7	10
Queries	3	4	6
Files	7	10	15

WEIGHTED-FUNCTION-COUNTS for the five different types are then defined as the sum of (RAW-FUNCTION-COUNTS * FUNCTION-COMPLEXITY-SCORES). Hereafter we will refer to this sum as WFC.

STEP 3: **Adjusting WEIGHTED-FUNCTION-COUNTS by TECHNICAL-COMPLEXITY-FACTOR.** The adjustment factor reflects application and environmental complexity, expressed as the degree of influence of 14 "application characteristics" listed below. Each characteristic is rated on a scale of 0 to 5 (COMPLEXITY-FACTOR-VALUE), and then all scores are summed. The TECHNICAL-COMPLEXITY-FACTOR (TCF) = .65 + .01 * $\Sigma_{f \in F}$ COMPLEXITY-FACTOR-VALUE$_f$.

1. Data Communications	8. On-line Update
2. Distributed Functions	9. Complex Processing
3. Performance	10. Re-Usability
4. Heavily-used Config.	11. Installation Ease
5. Transaction Rate	12. Operational Ease
6. On-line Data Entry	13. Multiple Sites
7. End-User Efficiency	14. Facilitate Change

Finally, total FUNCTION-POINTS (FP) are calculated as FP = WFC * TCF.

Center for Digital Economy Research
Stern School of Business
Working Paper IS-90-12

1.2 Data and the CASE Environment Examined

We obtained data on twenty software projects from a large investment bank in New York City. The projects were developed and implemented with an in-house CASE tool over a two-year period. The CASE tool evolved as a multi-million dollar, internally developed software project. Its objective was to increase the responsiveness of the firm's software development operations. It exhibits many of the features of an *Integrated CASE Environment* (**ICE**) (BANK90a). ICE refers to application development using CASE tools that automate the entire life cycle of software development, from the earlier stages of analysis and design to the later stages of code construction and testing. The type of CASE environment used dictates the variety and range of automated software engineering facilities available for programming. This CASE tool provides powerful development support utilities, including entity-relationship modelling, screen and report painters, and 3GL module-integration tools. Its unique features include:

* an *object oriented* approach to applications development. Application programmers use structured, standardized, and rigorously defined objects and modules as building blocks to encode the functionality required for applications.

* a *centralized repository* which stores all modules and objects developed for applications;

* storage of the application model as an abstract *object hierarchy* in the repository. This abstract model defines the functionality of each ICE application.

The structure of this development environment is further detailed later in the paper. Our investigation into the usefulness of a short-form variation of function points, and the development of the object points analysis approach is closely tailored to such CASE environments. The twenty projects were assessed by a team of analysts trained in function points analysis. This resulted in four counts corresponding to the four metrics we discuss in this paper for each of the twenty projects. This data was used for evaluating and comparing the performance of the four metrics.

The remainder of this paper is organized as follows. Section 2 critiques the methodology of function points from the perspective of the changed requirements in a CASE development environment. It also discusses our rationale for testing a short-form variation of function points for CASE-based systems. Section 3 presents a new approach to gauging the outputs of software development : object points analysis using object-oriented CASE environments. Results of an empirical evaluation of the function points metric, its short-form variant, and object points and its short form are presented in Section 4. Section 5 concludes with a discussion of the requirements for metrics which better support the measurement and estimation of productivity in systems developed using CASE.

2. FUNCTION POINTS FROM A CASE PERSPECTIVE: A CRITIQUE

How does the function points procedure stand up to the measurement challenge of the CASE environment? What portions of the procedure present problems in the CASE environment that can be overcome using revised metrics? We argue that each step in the calculation of function points (as presented in Table 1) needs to be reassessed in light of relevant CASE characteristics.

2.1. Step 1 -- Identification of RAW-FUNCTION-COUNTS

First, the classification scheme used in the identification of RAW-FUNCTION-COUNTS (RFC) is not intuitive for CASE-developed software. The components of the function points procedure (Inputs, Outputs, External Interfaces, Queries and Files) do not follow naturally from the building blocks advocated in an object-oriented integrated CASE environment. The CASE methodology used in object-oriented ICE development enforces *modularization* of application code. When modules and objects are the building blocks of CASE applications, identification of the five function types will force the analyst to expend significant effort in stepping within a module or an object to examine the code. Moreover, a sizeable portion of the code may have originated from a powerful feature of CASE: the ability to *generate code*. A programmer or analyst who has not written the actual code and done only the logical design would be forced to deal with the automatically generated code, which may not closely match what the person would have written. Thus, analyzing CASE-generated code would be an onerous, and most likely, an inefficient task. It may result in subjectivity and inconsistency in the classification of RAW-FUNCTION-COUNTS, as well as require a large amount of time and effort.

Second, a straightforward gauge of function types will be prone to double-counting the labor consumed in developing systems with CASE. The feature of a *central repository* in ICE environments presents significant opportunities to reuse code. Reused code

Center for Digital Economy Research
Stern School of Business
Working Paper IS-90-12

adds to the functionality a system delivers without requiring much additional effort. So, when function points are being used for measuring the functionality or size of a CASE-delivered system, the effort estimates should be adjusted to reflect the functionality added by reused code.

Thus, although the five function types represent the intrinsic functionality of CASE-developed systems, it would be useful to have a mechanism that translates functionality into the more natural building blocks of modules and objects in an object-oriented CASE environment. In related research, we have investigated a solution to the problem of mapping between CASE objects and the function types (BANK90c). The proposed mapping forms the basis of automating function points analysis in an object-oriented ICE. This could effectively circumvent the problems of effort, time and inconsistency in manually counting the function points of CASE-delivered systems. However, *estimation* of function points remains unintuitive in such CASE environments.

2.2. Step 2 -- Classification into Complexity Levels

Classification of RAW-FUNCTION-COUNTS into levels of complexity is the second step in function point analysis. It yields WEIGHTED-FUNCTION-COUNTS. The complexity weights that apply to the different complexity levels were determined by Albrecht by trial and error (ALBR79). Symons (SYMO88) concluded that a new set of weights might need to be recalibrated for any new technology, or new development environment. Clearly CASE qualifies as a technology radically different from the traditional 3GL development activities to which Albrecht's weights apply.

It is useful to keep in mind that the rationale for decomposing each function type into simple, average and complex came from a realization that each represented a different level of functionality delivered to the user (ALBR79). For estimation purposes, this is assumed to translate into different amounts of time to code each complexity type. However, in the CASE environment the differential between the time required to code a simple type and a complex type may not be as large as in a 3GL development environment. The ability to do object-oriented development, to reuse code and to generate code, may contribute to an increased uniformity in the levels of effort required for developing different complexity types. The proposition here is that the complexity differentials in CASE function counts may not lead to a significant improvement in estimating the actual development labor consumed. Thus, the complexity classification used in the function points analysis method may not only need recalibration, but in fact, may not be worth (in terms of estimation performance) the extra effort.

For use in CASE environments that exhibit some of the characteristics of ICE, we think it may be worthwhile to consider an aggregate count for each of the five function types, without further classification by complexity level.

Other problems with classification of the function types into three levels of complexity include increased subjectivity and measurement effort. Level of experience in software programming (and by analogy, in CASE tools) affects an analyst's perception regarding the complexity of a function type (LOW90). The time and effort involved in achieving this subclassification through CASE-generated documentation further adds to the cost of counting function points.

2.3. Step 3 -- Adjusting WEIGHTED-FUNCTION-COUNTS by the TECHNICAL-COMPLEXITY-FACTOR

Symons (SYMO88) also advocates a more open-ended approach to the factors affecting external complexity. Availability of CASE utilities such as automatic code generation, graphics generation and screen painting may reduce the development labor required to implement some complexity contributing factors (Table 1). Moreover, in the integrated CASE environment we have been studying, reuse affects development effort far more than any other factor (BANK90a). In short, the list of fourteen factors may not all be relevant to CASE-based system development.

As a result, TECHNICAL-COMPLEXITY-FACTOR (**TCF**) may not explain a significant portion of the variation in labor consumed for developing a CASE-based software application, so that the time and effort spent in calculating TCF would not be of value. (Note from Table 1 that TCF can have a score only in the range of .65 to 1.35, and thus can adjust the final number of function points no more than 35% at the most. In fact, this range proved to be much narrower in the data set we examined).

Thus, it is worthwhile to assess the predictive ability of the TCF factor and its components. At the same time, the effect of the reuse factor needs to be considered in more detail for the measurement procedure to be appropriate for CASE.

2.4. Implications For A *Short Form Variation* Of Function Points

Based on our discussions, we propose a short form variation of function points as a candidate metric appropriate for measuring outputs from object-oriented CASE environments.

To obtain the RAW-FUNCTION-COUNTS metric, Step 1 from the function points analysis procedure is

Center for Digital Economy Research
Stern School of Business
Working Paper IS-90-12

retained but the function counts are not separated into different complexity levels as in Step 2, or adjusted for external complexity as in Step 3. Thus, we use the following definition for this metric:

$$RAW\text{-}FUNCTION\text{-}COUNTS = \sum_{t=1}^{5} FUNCTION\text{-}TYPE\text{-}INSTANCES_t$$

where

FUNCTION-TYPE-INSTANCES = *total number of instances of function type t in an application.*

t = *function types (Input, Output, Query, External Interfaces, and Files).*

We shall later present results to compare RAW-FUNCTION-POINTS and function points in their ability to predict development labor for CASE projects. Results will provide justification for the proposed removal of steps 2 and 3 from the function points procedure as a means of saving calculation time and effort without losing much predictive power.

3. A NEW APPROACH TO OUTPUT MEASUREMENT FOR CASE PRODUCTIVITY

An object-oriented integrated CASE environment presents an interesting opportunity to test these metrics. We have indicated at the outset that object-oriented ICE is not representative of all CASE tools available in the market today. However, object-oriented development is being increasingly practiced in CASE environments, and is widely believed to be the standard analysis and design, and implementation methodology for software development in the 1990s (BOUL89, GOLD90).

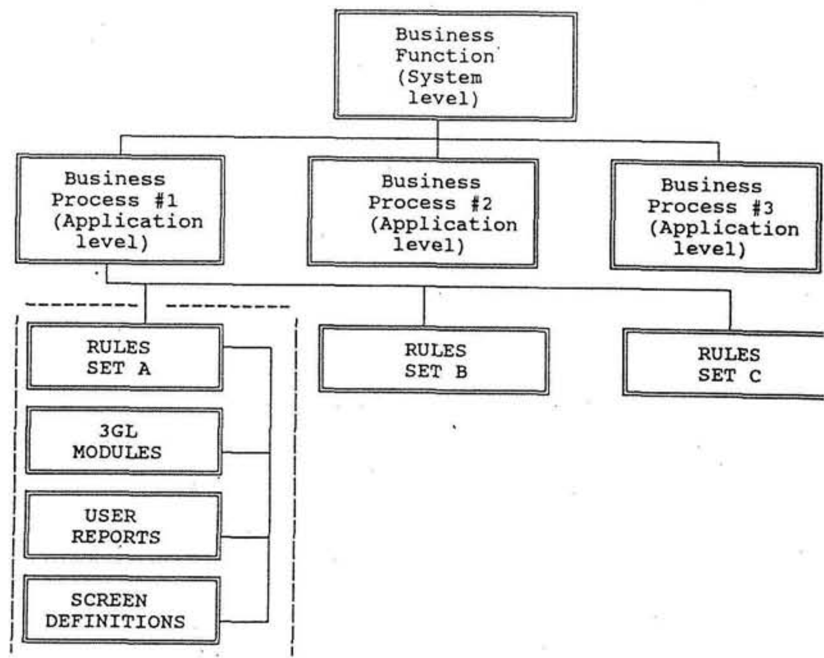
3.1 Object-Oriented Development in a CASE Environment

ICE applications are comprised of *objects* that act as building blocks which the programmer uses to synthesise the functionality required by an application system. Objects provide specific, well-defined functionality in handy, ready-to-use chunks of code. Definitions and code contents of all such objects are stored in the *central repository* which also enforces certain standardization conventions regarding object definition. An object need only be written once, and all subsequent applications that need to deliver similar functionality can make use of the relevant object from the repository. If a system needs to deliver functionality not already embodied in an existing object, a new object may be created according to the standard conventions for its definition. This discipline in the object storage and application version-management features of the central repository streamlines the process of creating software by writing and reusing existing objects.

The central repository stores information about the different kinds of objects used in applications developed with the tool. Examples of object types defined for the CASE tool we studied are: RULE SETS, 3GL MODULES, SCREEN DEFINITIONS and USER REPORTS. Each object type is defined rigorously in order to make the process of software development conducive to object reuse. A RULE SET is a collection of instructions and routines written with the high level language of the CASE tool being used. It is usually called "the program" in 3GL development. As such, RULE SETS have limited flexibility in encoding functionality and generally allow to build the most commonly required functions in the relevant domain. For more complex or uncommon functions, the 3GL MODULE object can be used. A 3GL MODULE is a pre-compiled procedure, originally written using a 3GL. A SCREEN DEFINITION is the logical representation of an on-screen image. A USER REPORT means just the same as it does in development environments other than ICE.

All objects associated with an application are functionally organized into an *object hierarchy*, which is stored in the central repository. An application consists exclusively of these objects and each application can be identified by a high-level BUSINESS PROCESS at the root of the hierarchy. A BUSINESS PROCESS calls other RULE SETS, which in turn use other RULE SETS or 3GL MODULES. These in turn can communicate with a SCREEN DEFINITION, or create a USER REPORT. Figure 1 illustrates this hierarchy of application objects.

Figure 1. Repository Objects in the Integrated CASE Environment



Center for Digital Economy Research
Stern School of Business
Working Paper IS-90-12

The relationships between objects (which RULE uses which 3GL MODULE, which SCREEN invokes which VIEW, etc.) are themselves stored in the central repository. Collectively, the set of object instances and relationships between them make up the model of an application, and this can be used to identify the objects and the object instances that comprise an application. Identification of such objects has two important benefits. First, it follows the natural building process of CASE systems and is therefore intuitive and has the potential to be more accurate and consistent. Second, the application model in the repository can be utilized to facilitate the automation of object identification. This would lead to considerable savings in the effort and cost involved in collecting information about the objects used, and motivate implementation of the revised measurement procedures which we will shortly describe.

3.2. The "Object Analysis" Approach

Do objects represent the *functionality* of an ICE application? We argue that the size and functionality delivered by an ICE application can be derived from the aggregation of the objects used to build it. Will knowing the number of objects that comprise a system provide sufficient information to estimate the labor required to build it?

To explore these questions further, we conducted two sets of interviews with managers and analysts experienced in the use of ICE within the organization. The first set involved conducting Delphi sessions in which a small group of project managers were asked to estimate the time required to build a small application involving a wide variety of functionality requirements. The Delphi sessions were taped for later analysis. Based on the themes that unified the approaches used for reaching group estimates of development labor, we conducted a second set of individual follow-up interviews. Project managers responsible for developing and estimating projects were interviewed and asked more focused questions regarding how they would estimate development labor using ICE objects as the basis of their estimation.

Results of our analysis indicated that project managers employ estimation heuristics to assess the number of different types of objects that need to be developed for a project. Use of heuristics by experts for the estimation of software development costs has been reported previously in other development environments (VICI89). Using these heuristics, a project manager initially estimates the number of RULE SETS, 3GL MODULES, SCREEN DEFINITIONS and USER REPORTS that would comprise the final application software. However, similar to the function types in function points, different objects exhibit different levels of complexity and functionality, and also require different amounts of development labor to construct. The project managers we interviewed classified occurrences of object types into three levels of complexity. Each complexity level within an object type was regarded as requiring a different number of days to develop. Project managers' object-effort estimates are summarized in Table 2 below in terms of the *average time required to build a given object type*.

Table 2. Project Manager Development Effort Heuristics

OBJECT TYPE	PROJECT MANAGER EFFORT HEURISTICS (AVERAGE)
RULE SETS	3 days
3GL MODULES	10 days
SCREEN DEFINITIONS	2 days
USER REPORTS	5 days

We utilized the means of their object-effort responses for the complexity levels because we have not yet explicated the heuristics managers use to classify objects into complexity levels. A deeper investigation into the nature of heuristics for estimation and classification of objects in ICE environments is required in order to specify dimensions of object complexity. We can then generate object-effort tables from a database of actual projects developed using object-oriented ICE environments.

Two new output measures are suggested by our analysis. The first, **OBJECT-COUNTS**, is determined by summing the instances of individual objects of the four types. The second, **OBJECT-POINTS**, is determined by weighting each object type by the development effort associated with it (given in Table 2). These two metrics are defined as follows:

$$OBJECT\text{-}COUNTS = \sum_{t=1}^{4} OBJECT\text{-}INSTANCES_t$$

$$OBJECT\text{-}POINTS = \sum_{t=1}^{4} OBJECT\text{-}EFFORT\text{-}WEIGHT_t * OBJECT\text{-}INSTANCES_t$$

where

$OBJECT\text{-}EFFORT\text{-}WEIGHT_t$ = average development effort associated with object type t, based on project manager heuristics;

OBJECT-INSTANCES = total number of instances of object type t in an ICE application;

t = object type (RULE SETS, 3GL MODULE, SCREEN DEFINITION and USER REPORT).

Center for Digital Economy Research
Stern School of Business
Working Paper IS-90-12

4. EVALUATION OF ALTERNATE METRICS FOR MEASURING CASE OUTPUTS

We next compare the object points analysis approaches with the function points procedure as candidate metrics for measurement of outputs from object-oriented CASE. The OBJECT-COUNTS and OBJECT-POINTS metrics were defined in Section 3.2. *FUNCTION-POINTS* is the original Albrecht version presented earlier in this paper, and its short form variation is *RAW-FUNCTION-COUNTS* defined at the conclusion of Section 2.

4.1. Modelling Output Metric Performance

In order to test the performance of the four metrics for estimation of software development labor, we estimated a set of regression models to predict the *reuse-adjusted development effort* (the dependent variable) in terms of each of the output metrics (the independent variables). The regression results can be used to indicate the extent to which a given output metric is able to explain the variance in development effort, after it has been adjusted to reflect unaccounted effort required for developing reused code. When high levels of reuse are observed, the resulting functionality of a system will not be a very good predictor of the labor required to build it, since reused code does not require an equivalent amount of labor input to construct and implement.

In order that the functionality embodied in the reused code be reflected in the development labor logged against the project, we adjust labor by a factor representing the leverage provided by reused code. Reuse leverage can be measured by the average number of times an object was reused in an application (BANK90a). The average level of reuse requires calculating the ratio of the total objects used in an application to the number of unique objects. This is a leverage metric, which means it *adds* to the labor estimates an amount proportional to the functionality supplied by reused code. This metric for measuring reuse agrees with the reuse measurement approaches advocated by Neighbors (1984) for 3GL environments. Thus, we adjusted development effort, expressed in PERSON-MONTHS, by multiplying it by the average level of reuse as defined below.

$$REUSE = \frac{TOTAL\ OBJECTS\ USED\ BY\ APPLICATION}{UNIQUE\ OBJECTS\ USED\ BY\ APPLICATION}$$

The estimation model we used to compare the various output metrics has the following mathematical form:

$$PERSON\text{-}MONTHS * REUSE = \beta 0 + (\beta 1 * OUTPUT\text{-}METRIC * DUMMY1) + (\beta 2 * OUTPUT\text{-}METRIC * DUMMY2) + \varepsilon$$

where

PERSON-MONTHS	=	*number of person months of development labor consumed in constructing the project;*
REUSE	=	*total number of objects used in an application divided by the unique number of objects used;*
OUTPUT-METRIC	=	*application output, as measured by FUNCTION-POINTS, RAW-FUNCTION-COUNTS OBJECT-COUNTS or OBJECT-POINTS ;*
DUMMY1,(2)	=	*1 if project constructed in Year 1 (2), 0 otherwise;*
$\beta 0$, $\beta 1$, $\beta 2$	=	*regression parameters to be estimated;*
ε	=	*a normally distributed error term.*

A model incorporating the DUMMY1 and DUMMY2 variables enables us to represent information about the relative productivity of the thirteen projects constructed in Year 1, when the CASE tool was being developed, and the seven Year 2 projects developed later. Year 2 projects tended to be much larger development efforts, where the power of the CASE development methodology was more effective and more reuse was observed. As a result, each of the two phases of project development with the CASE tool exhibited different productivity levels. Our study of Year 2 projects indicated an order of magnitude gain in productivity when compared to Year 1 projects (BANK90a). Clearly, developers' use of the tool had begun to mature and the tool delivered more development power by Year 2. The model specified above accounts for this difference in development productivity over the two years.

4.2. Estimation Performance Results

Selected details of the data for the metrics used in the regression are shown in Table 3. Our first step was to examine correlations between the output metrics. Table 4 presents the correlation results. The correlations between
the function point based metrics were all quite high (.98 minimum), while the correlations between the function point metrics and the object metrics were lower (.89 maximum). Since the function points procedure is established and well validated, correlations with this metric are an indication of the convergent validity of the new metric. Low

Center for Digital Economy Research
Stern School of Business
Working Paper IS-90-12

Table 3. Data for Four Software Development Output Metrics

PROJECT DETAILS	FUNCTION POINTS	RAW FUNCTION COUNTS	OBJECT COUNTS	OBJECT POINTS
MEAN	1337.9	242.3	153.9	966.0
MAXIMUM	5911	865	619	3657
MINIMUM	98	27	22	87
STANDARD DEVIATION	1609.0	247.8	168.0	1024.3
MEAN/ STANDARD DEVIATION	1.20	1.02	1.09	1.06

correlation with the function point metric could mean that the new metric is not a good measure of the construct that function points purports to measure (functionality delivered to the user). Or, it could mean that the new metric complements function points by measuring an aspect or dimension of the construct ignored by function points. In our data set, the easier to collect RAW-FUNCTION-COUNTS could thus be as useful a measure of output as FUNCTION-POINTS. The same may not be true for the object analysis based metrics. OBJECT-COUNTS and OBJECT-POINTS may measure a different aspect of the applications' functionality, or they may be measuring an entirely different characteristic of the output.

Table 4. Correlations for Output Metrics

OUTPUT METRICS	CORRELATIONS			
	FP	RFC	OBJECT COUNTS	OBJECT-POINTS
FP	-	-	-	-
RFC	.98	-	-	-
OBJECT-COUNTS	.89	.87	-	-
OBJECT-POINTS	.86	.86	.99	-

Thus, our next step was to examine the quality of the effort estimates produced by the metrics. The regression results for the four estimation models discussed above are presented in Table 5. The table offers information about estimated parameters and the absolute magnitude of the fit of the models.

The RAW-FUNCTION-COUNTS and FUNCTION-POINTS metric were estimated to explain about 54% of the variance in PERSON-MONTHS * REUSE, based on the R^2 value for the estimated model. The similarity between the two metrics' estimation performance is readily explained. Projects in the data set exhibited relatively similar values for TECHNICAL-COMPLEXITY-FACTOR since the implementation environments did not vary much in the applications. However, the results seem to justify the proposition that complexity differentials in ICE-delivered function counts may not lead to significant improvement in estimating development labor.

Table 5. Results For Estimation Performance of Metrics

SOFTWARE OUTPUT MEASUREMENT METRIC	COEFFICIENT ESTIMATES (SIGNIFICANCE LEVELS) β_0	β_1	β_2	REPORTED R-SQUARED
RAW-FUNCTION-COUNTS	464.95 (.08)	6.37 (.001)	1.00 (.15)	.54
FUNCTION-POINTS	485.16 (.06)	1.43 (.001)	0.15 (.16)	.54
OBJECT-COUNTS	90.90 (.76)	19.40 (.001)	2.38 (.03)	.58
OBJECT-POINTS	341.08 (.14)	2.26 (.001)	0.29 (.05)	.65

OBJECT-COUNTS demonstrated a marginally better performance in estimating PERSON-MONTHS * REUSE. R^2 for the estimation model involving OBJECT-COUNTS rose to 58%, a 7.4% increase over FUNCTION-POINTS. The OBJECT-POINTS metric performed even better, with the metric demonstrating the ability to explain 65% of the variance in the output metric, a nearly 20% improvement in R^2 compared to FUNCTION-POINTS and 10% improvement over OBJECT-COUNTS. Once again, regression results indicate the fit of the model, and thereby provide evidence for the *estimation performance* of the metrics. These results, however, are inconclusive about one metric being better than the other as a measure of the intrinsic size and functionality of the software.

The third category of results are derived from an interpretation of the parameter estimates (β_0, β_1 and β_2). The majority of the parameters obtained from these models were positive and significantly different from zero. In fact, the significance of the relationship between labor and output was strongest for the object point metrics. A side result of the modelling approach we have used is that it also provides information on the productivity ratios between Year 2 and Year 1 development using the estimated parameters from the regression. Table 6 presents the productivity ratios, β_1/β_2, for each of the output metrics estimations.

Although the Year 2 to Year 1 productivity ratios exhibit considerable variance, they demonstrate the extent to which productivity increased in the firm's use of CASE over the two years. The low end of the

Center for Digital Economy Research
Stern School of Business
Working Paper IS-90-12

Table 6. Productivity Ratios Based on Estimated Parameters

SOFTWARE OUTPUT MEASUREMENT METRIC	PRODUCTIVITY RATIO -- PHASE 2 VERSUS PHASE 1 (BASED ON β PARAMETERS)
RAW-FUNCTION-COUNTS	6.37/1.00 = 6.37
FUNCTION-POINTS	1.43/0.15 = 9.53
OBJECT-COUNTS	19.40/2.38 = 8.15
OBJECT-POINTS	2.26/0.29 = 7.79

range of productivity ratios is about 6 for RAW-FUNCTION-COUNTS, while on the other hand, using FUNCTION-POINTS as an estimator led to the largest estimated productivity ratio between the phases. One possible interpretation is that RAW-FUNCTION-COUNTS underestimates output because it treats the labor requirements of different complexity levels uniformly. However, as the functionality and complexity embodied in Year 2 projects increased, underestimation of output by RAW-FUNCTION-COUNTS increased more than proportionately. As a result, productivity gains estimated by RAW-FUNCTION-COUNTS was the least. Function points, while accurately capturing the higher functionality of complex CASE-based applications developed in Year 2, tended to overstate the labor required to create them. The mean of the productivity ratios corresponding to the four metrics was 7.96, and this was most closely matched by the object analysis metrics. Thus, each of the models provides clear evidence for the extent of productivity gains observed as use of the CASE tool matured in the firm.

5. CONCLUDING REMARKS

5.1. Contributions

Our investigation into the performance of two function point analysis and two object analysis metrics suggests that there may exist viable alternate approaches for measuring the outputs of the CASE-development process. This study was conducted as an exploratory investigation to provide us with the basis and directions for further developing measurement approaches for object-oriented CASE environments. As such, our findings should be interpreted within the limited validity of the study. Conclusions regarding the performance of alternate metrics from our study were obtained within a single organization, with the 20 projects we studied. At this stage, we have no information about whether the results would also hold true in other integrated CASE development environments, since this research question can only be investigated using data sets that involve multiple organizations. However, we believe that the characteristics of the development environment we studied and utilized in testing the metrics are present in other object-oriented CASE environments. If that is true, then it is a reasonable expectation that results can be generalized to other such CASE environments. We are currently involved in confirming this assertion with larger data sets from multiple sites.

Two alternate measurement approaches exhibited strong potential for further development and validation in object-oriented CASE environements. The RAW-FUNCTION-COUNTS metric proved to be comparable to FUNCTION-POINTS in terms of its estimation performance, and it is readily implemented at much lower cost. We also achieved considerable success in our test of the OBJECT-COUNTS and OBJECT-POINTS metrics as estimators for software development labor. Moreover, these are measures that can be readily automated in an integrated CASE development environment such as the one described in this paper. The results showed that OBJECT-POINTS best fitted our model for estimating software development labor. For the data set we investigated, it actually had a higher R^2 than both FUNCTION-POINTS and RAW-FUNCTION-COUNTS.

Our approach to estimating the productivity gains from the use of CASE in Year 2 versus Year 1 also has important managerial implications for research on CASE productivity. The lessons and insights obtained by studying our data and results can help to build experience in the area of CASE productivity assessment. As a result, we have considerable evidence from our research to suggest that the use and availability of key development facilities made available with the CASE tool clearly affect productivity, as do the wider range of opportunities for reuse and a development environment that is more stable and better understood by developers.

5.2. Future Research

In future research, we intend to further explore object points analysis as an output measurement approach that is tailored to and built into the object oriented CASE development process itself. The first step we will take is to examine another more detailed object points metric in which each object is weighted by the approximate time it takes to construct. Our Delphi sessions and individual project manager interviews suggested that project managers distinguish among the complexity levels of the various objects that they build into ICE applications. Additional work needs to be done to identify the dimensions of the objects that define their complexity levels. We also intend to study a larger set of projects within the same organization and to extend our analyses to the projects of other organization that have implemented object-oriented ICE.

Center for Digital Economy Research
Stern School of Business
Working Paper IS-90-12

Another open question is the automation of object points analysis. Object point analysis reporting tools can be made to analyze the changing contents of the repository as an application is constructed. Since objects were found to be more intuitive to the project managers' mental model of the functionality of software developed with object-oriented CASE, real-time object information will be relevant for the project's operational control. The object information thus made available to the project manager will assist him in proactively managing the software development process and make strategy decisions (BANK90b). We believe that the measurement of object points (weighted for the complexity of objects in the application's object hierarchy) can be automated at low cost, once we have solved the problem of dimensioning object complexity.

When senior managers of software development operations have such tools available, the stage is set for an entirely new approach to managing the software development process -- *software development life cycle productivity management.* To date, the process of tracking software development operations has largely been based on single point estimates of productivity, for example, taken when a project has been completed. But, the data made available by automating the measurement process as a project proceeds through the development life cycle offers many possibilities for rich and insightful analyses that cannot be conducted using traditional performance tracking approaches. Management can increase its effectiveness by proactively fine-tuning software development as it occurs, rather than adjusting it for future development.

REFERENCES

ALBR79 Albrecht, A. J. "Measuring Application Development Productivity " in *Proceedings on the Joint SHARE, GUIDE, and IBM Application Development Symposium*, IBM, October 1979, pp. 83-92.

ALBR83 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.

BANK90a Banker, R. D. and Kauffman, R. J. "An Empirical Assessment of Computer Aided Software Engineering (CASE) Technology: A Study of Productivity, Reuse and Functionality," Working Paper, Center for Research on Information Systems, Stern School of Business, New York University, October 1990.

BANK90b Banker, R. D., Kauffman, R. J. and Kumar, R. "Managing Strategic Costs with Automated Software Metrics" Working Paper, Center for Research on Information Systems, Stern School of Business, New York University, October 1990.

BANK90c Banker, R. D., Fisher, E., Kauffman, R. J., Wright, C., and Zweig, D. "Automating Software Development Productivity Metrics," Working Paper, Center for Research on Information Systems, Stern School of Business, New York University, October 1990.

BOEH84 Boehm, B. W. "Software Engineering Economics", *IEEE Computer*, September 1987, pp. 43-57.

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

COT88 Cot, V., Bourque, P., Oligny, S., and Rivard, N. "Software Metrics: An Overview of Recent Results," *The Journal of Systems and Software*, 8, August 1988, pp. 121-131.

DREG89 Dreger, J. B. *Function Point Analysis*, Prentice Hall, Englewood Cliffs, NJ.

GOLD90 Goldstein, D.G. "Object Oriented Programming", DEC Professional, Vol9, #2, February 1990.

IBM89 *Application Development Productivity Strategy*, World-wide IBM User Group, Application Development Joint Project, June 1989.

IFPU88 *Proceedings of the International Function Points Users Group*, International Function Points Users Group, 1988.

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

JONE88 Jones, T. C. "A New Look At Languages", *ComputerWorld*, November 1988.

KEME87 Kemerer, C. F. "An Empirical Validation of

Center for Digital Economy Research
Stern School of Business
Working Paper IS-90-12

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*, Kona Lua, Hawaii, IEEE, January 1989.

LOW90 Low, G. C., and Jeffrey, D. R. "Function Points in the Estimation and Evaluation of the Software Process," *IEEE Transactions on Software Engineering*, 16:1, January 1990, pp. 64-71.

NEIG84 Neighbors, J.M. "The DRACO Approach to Constructing Software From Reusable Components," *IEEE Transactions on Software Engineering*, SE-10, 5, September 1984, pp. 564-574.

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," *Proceedings of the 22nd Hawaii International Conference on System Sciences*, Kona Lua, Hawaii, January 1989, pp. 957-958.

RUBI83 Rubin, H. A. "Macroestimation of Software Development Parameters: The ESTIMACS System", *IEEE Softfair Conference on Software Development Tools, Techniques and Alternatives*, 1983.

RUDO84 Rudolph, E. E. "Evaluation of a Fourth Generation Language",' *Proceedings of ACS and IFIP Joint Symposium on Information Systems*," April 1984, pp. 148-165.

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.

SYMO88 Symons, C. R. "Function Point Analysis: Difficulties and Improvements," *IEEE Transactions on Software Engineering*, 14:1, January 1988, pp. 2-10.

VICI89 Vicinanza, S., Mukhopadhyay, T., and Prietula, M. J. "Software Effort Estimation: A Study of Expert Performance," Working Paper 89-002, Center for the Management of Technology, Graduate School of Industrial Administration, Carnegie Mellon University.

YELL90 Yellen, R. E. "Systems Analysts' Performance Using CASE Versus Manual Methods," *Proceedings of the 23rd Hawaii International Conference on System Sciences*, Hawaii, January 1990, pp. 497-501.

Center for Digital Economy Research
Stern School of Business
Working Paper IS-90-12