1988

A MODEL FOR ESTIMATING INFORMATION SYSTEM REQUIREMENTS SIZE: PRELIMINARY FINDINGS

Clive D. Wrigley
McGill University

Albert S. Dexter
University of British Columbia

A MODEL FOR ESTIMATING INFORMATION SYSTEM REQUIREMENTS SIZE: PRELIMINARY FINDINGS

Clive D. Wrigley
Faculty of Management
McGill University

Albert S. Dexter
Faculty of Commerce
University of British Columbia

ABSTRACT

A model has been developed that estimates information system requirements size. The estimating model may be applied relatively early in the systems development life cycle. The model captures system statics with the entity-relationship data model and system dynamics by measuring events at the system boundary. Results on pilot data indicate the model may provide a reliable predictor of system size in terms of lines of code, holding personnel experience and technology constant in a 4GL development environment.

1. INTRODUCTION

In order to successfully plan and develop information systems, it is necessary to obtain an initial estimate of the size of the system being undertaken. While the eventual goal of estimating is to understand and measure complexity and its impact on the amount of effort to deliver software, this paper uses the concept of size as a surrogate for system complexity. Most estimating models lack a theoretical basis, are complex, somewhat intractable, and cannot be used with accuracy until late in the systems development process, i.e. after considerable resources have already been consumed. It is the purpose of this paper to:

a) develop an estimating approach that is theoretically based;

b) create a relatively parsimonious model;

c) provide an estimating model that may be applied early in the development life cycle; and

d) initially validate the model.

The objective is to establish a theoretical and empirical link between the entities, relationships, and events that occur in the real world and the human effort required to analyze, design, and implement the information system that models them. This paper attempts to establish a linkage between requirements size and the amount of FOCUS code needed to implement these requirements.

The paper proceeds as follows: The relevant literature is briefly addressed describing previous estimating techniques and when in the development life cycle enough estimating information is available for their use. The role of our approach is then compared to previous estimating techniques. Next a theoretical estimating approach is presented and operationalized in a parsimonious model for estimation. The basic input parameters to the model are developed during requirements analysis using Entity-Relationship (ER) diagrams and Data Flow Diagrams (DFD). This operationalization allows the tracing of measures of information requirements size into eventual lines of FOCUS code. Finally our preliminary validation efforts with 75 FOCUS programs, the limitations of these results and current research directions are discussed.

2. ESTIMATING MODELS

A number of estimating models have appeared in the literature over the past two decades. Several authors have suggested a taxonomy of these estimating approaches (Wolverton 1974; Basili 1980; Benbasat and Vessey 1980; Kitchenham and Taylor 1984; Conte, Dunsmore and Shen 1986).

Reviews and critiques of these approaches appear in Boehm (1981), Mohanty (1981), Golden, Mueller and Anselm (1981), Kitchenham and Taylor (1984), and Wrigley and Dexter (1987). The issue which emerges from existing estimating models is that an accurate, early estimate of system size is crucial in order for the estimate to be useful in predicting actual effort to build the given system. At present, the two sizing approaches are the lines of code (LOC) approach and the function point (FP) approach. Unfortunately, neither of these approaches utilize information about the system being developed which is captured with structured analysis methods.

Several models exist in which a size estimate in LOC is the prime input into the model: Meta model (Bailey and Basili 1980), COCOMO (Boehm 1981), SLIM (Putnam 1978; Putnam and Fitzsimmons 1979; and Wolverton 1974). The rationality of using LOC, or any other output metric of a system development effort, as an input to an estimating model, is questionable. The problems with using code size as an input into an effort equation are:

1. The size of a software system is the result of many contingencies. It is the by-product or cumulation of all factors in the development process.

2. Size, measured in LOC, is that which results after requirements have been met. It should not be considered as a target.

3. Size estimates at the requirements phase are quite subjective. Accurate estimates may not be available until after the detail design is complete.

One example of how difficult the sizing problem is comes from a report by Conte, Dunsmore and Shen (1986, pg. 214) on a study by Yourdon. Several experienced software managers were asked to estimate the size of 16 completed software projects given only the complete design specifications for each project. An R^2 of .07 between actual size and estimated size is reported. An interesting observation from this study is that the expert analysts consistently underestimated the actual product sizes. The significance of these results is that, even with the design specification in hand, the ability to subjectively size a project in terms of LOC is elusive.

One conclusion that may be drawn is that SLIM, COCOMO and the others are not sizing models but are better suited to estimating resource consumption and scheduling once a size estimate is available. Notwithstanding their contribution to our understanding of the issues, significant sizing problems remain.

Additionally, a number of models use Function Points as prime inputs: Albrecht (1979), Albrecht and Gaffney (1983), Rubin (1983, 1985), Jones (1986), Symons (1988). This more recent approach considers larger units of software than LOC, such as screens, reports, inquiries, files and interfaces as inputs to an estimating model. Our ability to estimate these larger units of software is better than estimating LOC, as the information necessary to measure them is available during the design phase of development. However, ideally, what is sought are properties of an information system which are measurable during analysis and which are found to causally affect the amount of effort and code required to build the system. This paper addresses the problem of obtaining size estimates based on system requirements, i.e., estimates based on inputs to the development process.

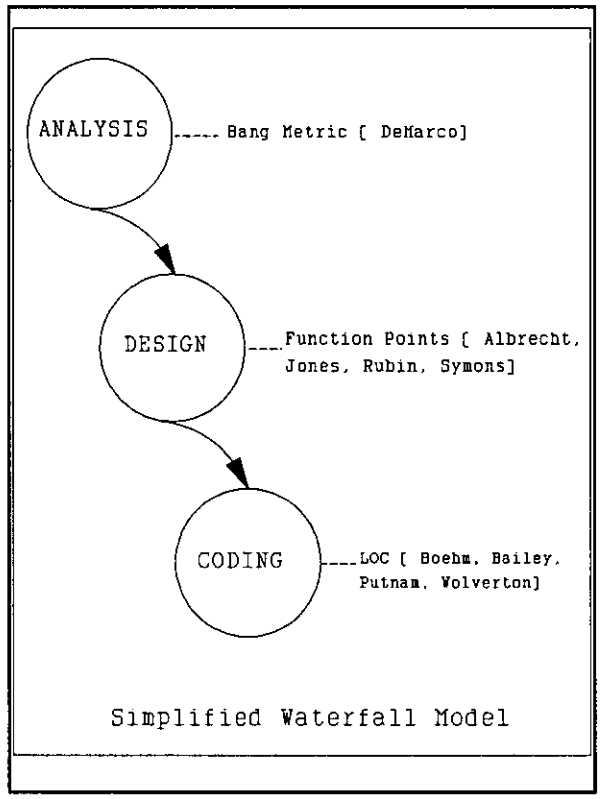

Figure 1

The various sizing[1] approaches may be placed on the Waterfall model of system development. It can be seen from this placement when information is available to make an estimate. As can be seen from Figure 1, there is a paucity of sizing strategies at the analysis phase. The notable exception is De Marco (1982). He has developed a "Bang" metric to estimate system effort. De Marco's "p-counts" (system primitives) include 12 different ways of counting system properties which are indicators of system complexity. However, in his own words:

> You might reason, as I originally did, that all work in a project is work spent implementing one of the things counted by the various p-counts. This theory implies that you ought to base your function metric on all of the p-counts, with each one weighted by its unique factor. I have never had much success with this approach; it is statistically intractable and *some of the counts overlap and measure redundantly*. A simpler and more productive way to characterize Bang is to choose one of the counts as a principal indicator. [DeMarco 1982, pp. 83, emphasis added.]

This paper attempts to find the simpler, more parsimonious method that De Marco suggested. As a step towards identifying which principle indicators to use, De Marco differentiated systems on two axes: scientific to business processing and function strong to data strong. Much of business data processing can be characterized as data strong. This paper takes the position that the requirements size of these business applications can be captured in terms of data measurements. The next section of this paper develops the general theory for this approach.

3. THEORETICAL DEVELOPMENT

We now develop a theoretical model of system requirements size. To achieve this, several major concepts are developed beginning with a discussion of the basic inputs to the development process. We then discuss the concepts of isomorphic transformational properties from requirements through to implementation, methods of requirements modelling, and an overview of requirements sizing. Next, concepts of processing complexity and how they relate to requirements are introduced. Finally, these concepts are synthesized into a formal statement of requirements size.

The simplest statement of the estimating problem is to determine the amount of effort[2] that is required to produce the working system. Effort is then the independent variable that causes a system to be produced. However, what we are after is an estimating model which considers effort as the dependent variable. The central question is: What does the amount of effort depend on? A model to help structure this question has been synthesized from the literature and appears in Figure 2. The justification for the model is described in detail by Wrigley and Dexter (1987).

Figure 2 may be interpreted as follows: An increase in system requirements size increases effort, while increases in personnel experience, and methods and tools mitigate effort.

Since the constructs in Figure 2 are temporally antecedent to the development process, this model suggests that these constructs causally affect system development effort. While the model in Figure 2 is the general statement of the problem, it is necessary to simplify it even further to establish a sizing metric based on requirements. If Personnel and Methods are held constant in this model, then this is equivalent to the simplified model of Figure 3.

We rationalize the estimating approach by arguing that any working system is the result of a human thought process which consists of transformations of requirements, through design specification and program coding to the working system. This transformation process is shown in Figure 4.

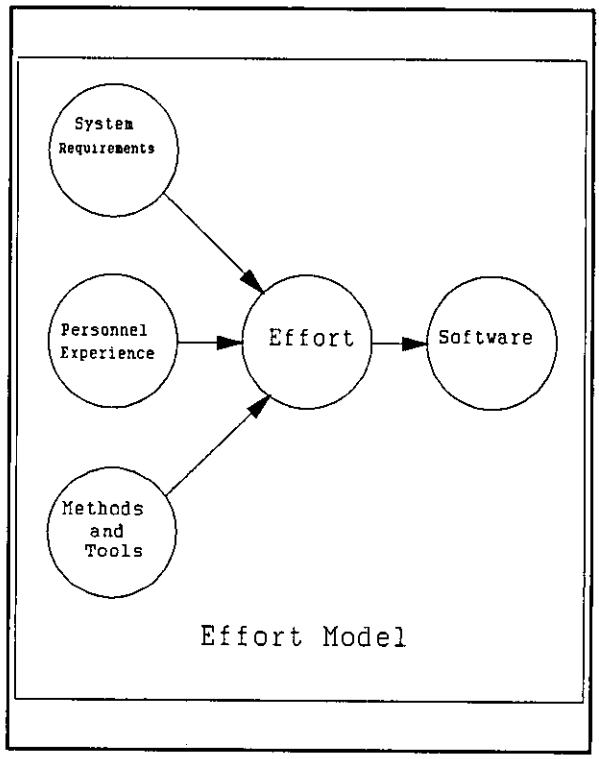

Figure 2

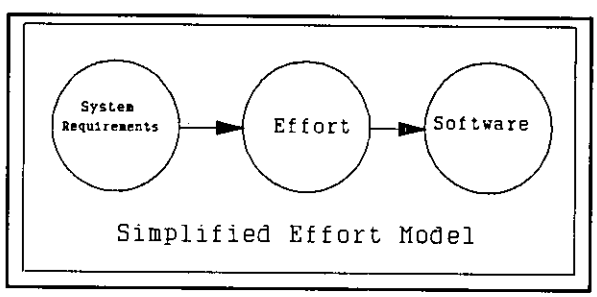

Figure 3

Effort is now the only intervening variable between requirements and code. This allows us to measure both requirements and code and determine which properties of requirements *cause* code to be produced. Hence measures of requirements are the independent variables while code produced is the dependent variable.

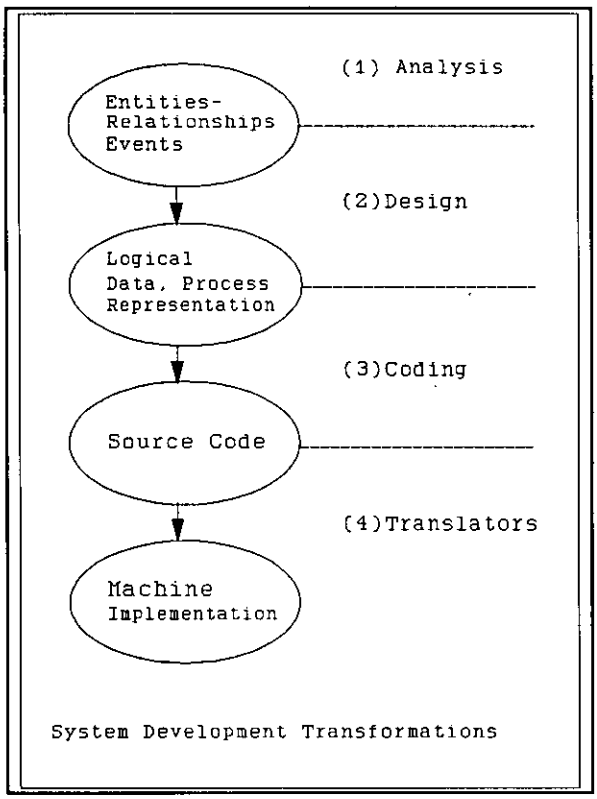

Figure 4

For estimating purposes, what is important, and is our central premise, is that there exist properties of a system's requirements that remain invariant over the transformations necessary to bring about the working system.[3] Moreover these properties are measurable. If the working system accurately reflects its requirements, then the transformations have maintained the basic isomorphism. The property of maintaining the structural form of a system requirement through to the working system is referred to as an isomorphic transformation.[4]

We now discuss estimation so as to link the concept of isomorphism to an estimating approach. Currently, two separate but parallel schools of thought exist with respect to system analysis: the Data Structure or Data Model approach and the Data Flow approach. While both are extremely useful for purposes of analysis and design, this paper begins to integrate both approaches for the purpose of estimating. The central reason for this is that both a system's statics (data structure) and dynamics (events which generate data flow) must be captured if a measure of requirements size is desired. It is the combination of statics and dynamics which drive the transformations (processes) into a working information system.

The most widely accepted data model is the Entity-Relationship (E-R) model (Chen 1976), also called the Entity Relationship Attribute model (E-R-A). We use the E-R model to capture system statics and by adding Data Flow Diagrams we include events (system dynamics) in our approach to modelling requirements size. An information system is a representation of the entities and their relationships that exist in the real world. The state of these entities and relationships must be maintained if the IS is to preserve its "faithful representation"[5] of the statics and dynamics of the real world. The variety of events that occur, and their effect on the entities and relationships, add to the size of the system. The more entities, relationships and events a system incorporates, the more complex its information requirements become. As the size of these requirements increases, the size of the software, if isomorphic to the requirements, will also increase. More specifically, statements in the requirements definition about events, entities and relationships that are to be modelled in the information system eventually will be identifiable in the machine implementable code.

Our position, that the basic structure of system requirements remain isomorphic from analysis through transformation into design and code is central to numerous methodologies that use top down decomposition and step wise refinement development strategies. The data structure school authors have suggested that a well structured function should match the logical data structure on which the function operates. The extension of the above reasoning is that processing performed by a program monotonically increases with the complexity of the data at the program interface. If this proposition is true, then by measuring the number of the inputs and outputs to a system as a whole (based on requirements) we can approximate the size of the processing task. The theory, which explains the above practitioner's observations and a major premise of this paper, can be found in Ashby's Theory of Requisite Variety (Ashby 1956). Simply stated, a system, to remain ecologically viable, must have sufficient internal variety to be able to respond to various changes in the environment, i.e., a system must be at least as complex as its immediate environment. Therefore, the canonical form of a requirements definition, should provide a measure of the overall complexity of the information system to be implemented in software. Our premise is that the logical structure of the data requirements, if measured correctly, can be used as a measure of system size.

Halstead (1977) made the theoretical connection between complexity and effort. He claimed that programmers undertake some search process through the operand and operator space of a language to transform a program specification into code. The number of primitive mental operations required to locate the right operand-operator sequences, divided by the number of primitive mental operations per second (Stroud 1954), gives the total time required to program a given specification. If this theory is generalizable to the analysis and design transforma-

tions, then by counting the things that are likely to induce mental load on both users and developers we should obtain a high correlation with actual effort to build the software. Moreover, this relation will be causal. There is need then to create a usable metric for counting the size of system requirements.

The problem we face in estimating IS development effort is that there is no standardized metric of requirements size which can be used early in the IS planning process. We are now developing such a metric in order to create an estimate for the eventual code and the effort to produce a system.

Conceptually an organization can be described in terms of its statics and its dynamics. If a comprehensive E-R diagram were to be constructed for a firm, this diagram would represent the firms data map or static view. An organization's response to environmental changes on the other hand, can be considered as the firm's dynamics. These environmental changes, are defined as events at the organization or system boundary. These dynamics can be modelled with the use of DFDs.

For each development project, we can view the process as implementing a small segment of the organization's data map and environment linkages. Within a segment, the entities, relationships and events will then give us an early indication of the size of the system's requirements. We now provide a formalization of the above.

4. A FORMAL STATEMENT OF REQUIREMENTS SIZE

The ideal situation is where we have complete information about the events that exist at the system boundary, the entity sets, the known relationships among the entity sets and entity occurrences, and the controls or laws which specify allowable combinations of one or more events. It is then possible to describe both the statics and dynamics of a system with the use of set theory and matrix notation. The discussion below represents an initial attempt at rigorously specifying these concepts.

We can define the event space of a system by a vector **E** which contains an element for each possible input and output event at the system boundary:

$$\mathbf{E} = [e_1,...,e_n]$$

The event vector, **E** is comprised of two sub-vectors, $\mathbf{E^I}$ and $\mathbf{E^O}$ which correspond to the input events and output events respectively. Further, the e_i's, i=1...n are defined as vectors themselves which contain information about the specific event that has taken or is taking place. The elements in the e_i's are attribute values of the entity occurrences involved in the event.

A second vector **S** is defined as the system state vector. The elements of **S** are defined as vectors (sets) whose elements contain the values of the entities and relationships that exist in the system's memory.

The two vectors $\mathbf{E^I}$ and **S** are the inputs into the matrix **T** which transforms $\mathbf{E^I}$ and **S** into the new system state vector, **S'**, and possibly creates new output events at the system boundary $\mathbf{E^O}$. The transformation process **T** is what we usually refer to as code. This model of an information system is shown in Figure 5.

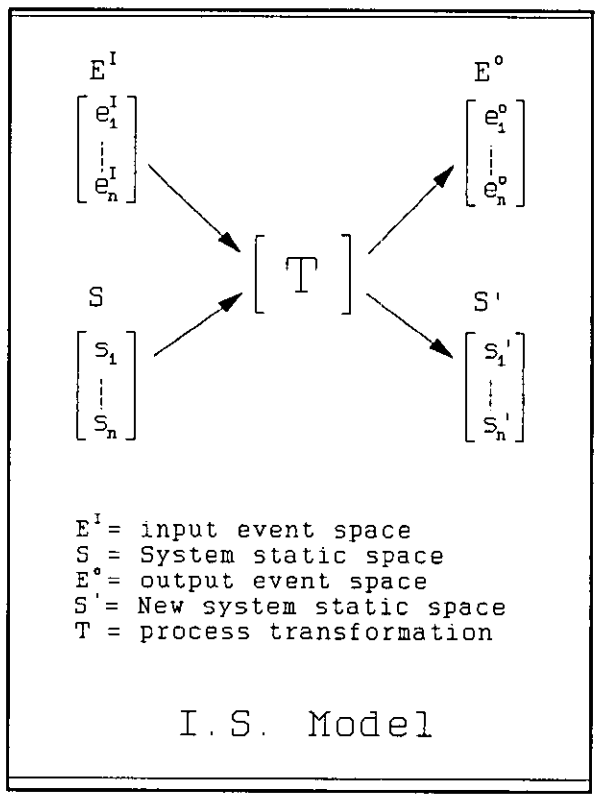

Figure 5

If the assumtion regarding isomorphic transformations holds, we would expect that the complexity of the transformation **T** reflects the complexity of **E** and **S**. The task now becomes one of measuring the size of both **E** and **S** in order to predict the size of **T**.

From Figure 5 we can see that the first approximation of requirements complexity can be obtained by simply counting the number of events that a system either must respond to or generates and the size of each event in terms of the number of attributes or data elements in each event. This is a measure of **E**. Additionally, **S** can be counted in the same way. A count of the number of entities and relationships weighted by the size of each entity or relationship will provide a measure of **S**. The

combination of these two requirements measures provide a first estimate of the size of T necessary to operate a system. While these measures may be incomplete with respect to the eventual complexity of T, they are available early in the development process. It must be kept in mind that estimating is a task carried out in the presence of incomplete information.

When more information about the E's and S's is introduced it is possible to refine our initial estimate of T. Here we can see the benefit of establishing early macro measures of system requirements which can be tracked and refined as more information becomes available.[6]

In the case of simple input events, there should be a one to one correspondence between the state change in the system's environment and a single variable update, i.e., there is a direct transformation of the input variable into storage. The trivial case is where a change in the environment is not recorded, i.e., the system has no defined response for the event. In the case of more complex events, changes to more than one set may be required. This would be the case where the event is comprised of more than one data element.

A first order approximation to the complexity of an individual input event can now be defined as the number of set changes in S that must be made given the event e_i.[7] Additionally, a complex event may dictate that updates to one set are conditionally dependent on the state of one or more other sets. Hence, a second order upper limit to the complexity of an event e_i is the number of possible dependencies across sets for each set change.

Given the above event-system state interaction information, it is anticipated that an extremely accurate estimate of T could be generated. Unfortunately the previous two measures of complexity would not be available until well into the design phase. What we produce at the analysis phase are approximate measures of the e_i's and the s_i's. These individual estimates of e_i and s_i are aggregated to form E^I, E^O, and S. Essentially, these estimates represent data flows to and from the system processes and can be used to estimate T.

We began with the proposition that data requirements complexity is the driving force behind the effort to build commercial software. It is claimed that requirements complexity could be estimated by measuring a system's event space and its internal state space. It is the size of these vectors that drive the transformation process. The conclusion we now draw from this section is that software processes are driven by the data. Without data there are no processes. While it is true that events reflect processes occurring in a system's environment, these events must be represented as data to a system. The significance of this from a research perspective is that the data can be viewed as the independent variable and the software processes as the dependent variable. Therefore, measurements of the data should predict the size of the process.

We have presented an approach to estimate system requirements based on measures of system statics and dynamics. This has been achieved by linking theory of requirements modelling with theory of system development transformations. We stated that measurements of requirements are available earlier in the development process than other measures such as function points and lines of code. We now discuss the empirical investigation of this model.

5. EMPIRICAL ANALYSIS

We model empirically the following general equation for estimating process size:

T = fn(E, S) where **E** represents event measurement and **S** represents static system measurements. For this initial investigation, events (dynamics) E, at the program level, are measured by:

1. E^I
   a. Input events = number of screens
   b. Input event size = number of screen variables
2. E^O
   a. Output events = number of reports
   b. Output event size = number of report variables

The System Static Measurement, S, is measured by:

1. Entities = number of files accessed
2. Entity size = number of fields available
3. Relationships = number of projections and joins

Readers will observe that there is similarity between these measures, De Marco's Bang Metrics, and Albrecht's Function Point measures. There are two related reasons for this. First, if the theory of isomorphic transformations holds, then it is expected that macro measures of system requirements will show up as function points (and eventually as lines of code). Second, since analysis of the pilot data is done at the program level, it is more appropriate to measure program dynamics and statics with function point-like counts. The rationale behind this is that a system is decomposed to the program level during the design phase. As we have argued that the function point approach is a reasonable size estimating strategy at the design phase, it is appropriate to use function point-like measures at the program level. It is anticipated that early measures of requirements size are predictors of both function point size and lines of code size. Table 1 contains the mapping between requirement measures and function points.

Table 1.

Entities	→	Logical internal files, external files
Relationships	→	Logical internal files
Events	→	Input transaction types, external outputs, inquries

Table 1 also shows how the entities, relationships, and events approach is more general than the Function Point approach to estimating.

Process Measurement, T, is measured as lines of FOCUS code.[8]

T = number of FOCUS lines of code (LOC)

6. RESEARCH SETTING

The research setting can be described as typical of modern commercial development shops. The DP services group employs nearly 80 people in a variety of jobs including clerical, machine operators, information centre staff, programmers, senior analysts and project leaders. The information systems in use consist of those developed in-house as well as modified packages. Both 3rd GL (COBOL, PL/1) and 4th GL (FOCUS) languages are used. The staff turnover is comparatively low, i.e., most professional personnel were on staff throughout the time period of the systems analyzed. This helped greatly as the person who built the system was available to answer any questions that arose during the course of the investigation. The hardware development environment was constant, e.g., the same operating system, screen editors, etc., over the development period of the systems analyzed.

The total data available for this study comprises approximately 28 application systems written in FOCUS. All systems were developed by the Small Projects Group in the company between 1984 and 1987. The systems analyzed represent all of the systems developed by the company in FOCUS. Data on each system include:

1. Business function/application type.
2. Programmer(s) and skill level.
3. Hours to build: from analysis to implementation plus maintenance.
4. Project elapsed time.
5. Total lines of FOCUS code (LOC).
6. Metrics of data structures and events.
7. Number of programs broken down into four classes:
   a. Online and internal file update
   b. Menu selection
   c. Report generation
   d. Report batch drivers

The 28 systems contain over 800 FOCUS programs and over 100,000 FOCUS LOC.

For the pilot study, two of the 28 systems were analyzed. These two systems are comprised of 75 programs representing about 11,000 LOC. One system is 2700 LOC while the other is 8300 LOC. All programs were designed and written by the same senior systems analyst. The two systems are functionally similar in that they are basically menu driven by users. The users typically enter transactions from a terminal and may select a number of reports by specifying report parameters. Means tests were performed to determine if length of program by program class was system dependent. These tests were not rejected allowing pooling of the programs from each system.

7. PILOT RESULTS

All statistical analysis was performed using the statistical package MIDAS. Of the 75 programs, four distinct program classifications emerged: updates reflecting input events (n = 19); reports reflecting output events (n = 41); menus controlling access to these processes (n = 12), and batch report drivers (3). The program demographics are shown in Table 2.

Table 2. Program Demographics LOC

Frequency	Classification	Min Size	Max Size	Mean	Std Dev	% of Total Code
19	Updates	15	740	228.3	223.3	38.3
41	Reports	10	440	146.3	95.1	53.0
12	Menus	30	87	67.5	18.1	7.2
3	Batch Drivers	31	88	55.7	29.3	1.5
75	Total	10	740	150.9	141.9	100.0

Since the sample size for batch drivers was so small, no further analysis was done on this program class. For the remaining programs, a means/variance test was run to determine if the program classifications differed signifi-

cantly from each other. Pair wise comparisons are shown in Table 3.

Table 3.

	MEAN			VARIANCE	
Updates >	Reports	(p=.05)	Updates >	Reports	(p=.00)
Reports >	Menus	(p=.01)	Reports >	Menus	(p=.00)
Updates >	Menus	(p=.02)	Updates >	Menus	(p=.00)

7.1 Regression Analysis by Program Class

There is evidence from the descriptive measures above that programs in each class differ significantly from each other in terms of their functions. Therefore, regressions were performed on each class separately. The regression model depicts the information on requirements that may be measured early in the development life cycle. For the pilot study, the general regression model to predict process size at the program level was:

T = fn (number of screens, number of screen variables, number of reports, number of report variables, number of files accessed, number of fields in files, number of projections and joins used)

The summary results for update and menu programs are shown in Table 4.

Table 4.

	Update Programs T (p)	Menu Programs T (p)
Constant Term	0.05 (n.s.)	-4.0 (.003)
Screen Variables	7.68 (.000)	7.8 (.000)
Master Files Accesssed	3.47 (.003)	N/A
R^2	.86	.86
F(p	48.7 (.000)	61.0 (.000)
S.E.E.	89.0	7.1
n	19	12

Table 4 may be interpreted as follows. Early in the development life cycle an estimate of the amount of data flowing to and from the user provides the measure of the event size. In addition the number of master files accessed during system update significantly contributes to code. Combined, these two variables explain 86 percent of the variance in source code size. MENUS can be predicted easily by simply counting the number of variables to and from the user; here the R^2 is 86 percent. In this sample the large negative intercept is not meaningful, as there were no observations near zero.

Predicting reports, however, is more problematical with an R^2 of 35 percent; however, the amount of user control over report parameterization and the amount of data in terms of fields accessible contribute to the amount of code. The screen variables simply indicate whether the user could customize the report or not. Surprisingly, the output data flow in terms of report variables was not significant in contributing to the variance in code length. Obviously, as the development process unfolds, more becomes known about the intricacies of the project. After data-base design, the number of projections and joins needed to produce reports are better understood. An independent variable defined as a measure of complexity of the systems' static space, i.e., the number of projections and joins, was added. The results of this regression for reports is shown in Table 5.

Table 5.

	Report Programs: Preliminary T (p)	Report Programs: Detailed T (p)
Constant	.45 (.65)	.47 (.64)
Screens	2.20 (.03)	3.30 (.002)
Fields in Master	2.10 (.04)	1.20 (.24)
Report Variables	.89 (.38)	.55 (.60)
Projections and Joins	Not Used	7.40 (.000)
R^2	.36	.74
F (p)	6.78 (.001)	26.17 (.000)
S.E.E.	79.40	50.7
n	41	41

7.2 A Parsimonious Model

While the previous regression results appear promising, there appear problems with multi-collinearity among the independent variables. Specifically, the independent variables show a multicollinearity between screens and screen variables and between master files accessed and the number of fields. The significant correlation coefficients for for each class of programs is shown in Table 6.

To remove the multicollinearity, the first principle component was extracted from screens and screen variables, and from masters and fields respectively. The reader will note that the intuitive interpretation of these principal

components corresponds to the theoretical notion of the dynamic events E and system statics S. The reduced model for each program class is shown in Table 7.

Table 6.

	Updates	Menus	Reports
Screens/ Screen Variables	.87	.78	.90
Masters/Fields	.76	N/A	.59

Table 7.

Variable	Updates T (p)	Menus T (p)	Reports T (p)
Constant	.14(n.s.)	-4.14 (.002)	.93 (n.s.)
Event Factor	7.83(.000)	8.12 (.000)	1.82 (.077)
Static Factor	2.75 (.01)	N/A	41.88 (.068)
Projections	N/A	N/A	6.98 (.000)
R^2	.83	.87	.69
F(p)	39.6 (.000)	66.0 (.000)	27.9 (.000)
S.E.E.	97.1	6.8	54.8
n	19	12	41

8. DISCUSSION

The results of the pilot study show a significant causal relation between the measures defined and the amount of code needed to implement the systems. However, these results are not too surprising as measures of requirements described in this pilot data are the result of "reverse engineered code." The source code from the company was obtained from its production library which contains the current versions of all its working systems. The advantages of reversing the code over using existing analysis and design documents lies in the questionable accuracy of these latter documents. In the field, it is common for documentation to lag the software and therefore requirements changes may not be reflected in the current requirements definition. From a theoretical perspective, the reverse engineered code has under-gone reverse transformations. Hence, it is assured that the analysis and design measures are "true" representations of the final working system, subject only to the interpretations of the researcher. The reversing operations, however, are objective and reproducible.

While these empirical results appear promising, we note the following limitations. First, the analyses have been performed with only two small business application systems; thus no generalizability is being suggested. Second, the reverse engineering operations imply that the requirements were accurate, therefore the methodology has not allowed for volatility in the requirements specifications. Third, the restriction of this estimation approach is to be used with data strong applications only; it does not purport to predict scientific or other programs that are function strong. Finally, we have used simple measures of requirements, design, and code size as surrogates for system complexity at the various phases of system development. The concept of system development complexity is multi-dimensional and is affected by software development tools, personnel experience, team development, class of system, and implementation language, at a minimum. In this research all of the above have been held constant, allowing only for variations in the size of the software analyzed.

9. IMPLICATIONS FOR RESEARCH AND PRACTICE

The paper begins to address the problem of how to estimate the requirements that software must support. We have proposed a method that uses current systems analysis and design techniques to measure a system's requirements.

Current research is underway to develop an automated FOCUS code analyzer to eliminate the chance of researcher error in the reverse engineering process. The automated analyzer will first be used to expand the current data set from two systems to 28. Subsequent research will use the analyzer to calibrate other FOCUS development environments, leading to standards of measurement for multiple organizations and programmers. With sufficient data collected on a wide variety of systems and environments, it will be feasible to generalize the estimation parameters to include both function and data strong systems, various levels of tool use, team development, and other implementation languages. This research will contribute to the growing body of knowledge on the measurement and evaluation of the system development process.

10. REFERENCES

Albrecht, A. J. "Measuring Application Development Productivity." *Proceedings of the IBM Applications Development Symposium*, GUIDE/SHARE, October, 1979, pp. 83-92.

Albrecht, A. J., and Gaffney, J., Jr. "Software Function, Source Lines of Code, and Development Effort Prediction: A Software Science Validation." *IEEE Transactions*

on Software Engineering, November, 1983, SE-9(6), pp. 639-648.

Ashby, R. W. *Introduction to Cybernetics*. New York: John Wiley & Sons Inc., 1956.

Bailey, J. W., and Basili, V. R. "A Meta-Model for Software Development Resource Expenditures." *Proceedings of the Fifth International Conference on Software Engineering*. IEEE/ACM/NBS, March 1981, pp.107-116.

Basili, V. R. *Models and Metrics for Software Management and Engineering: Tutorial*. IEEE Computer Society Press, 1980.

Benbasat, I., and Vessey, I. "Programmer and Analyst Time/Cost Estimation." *MIS Quarterly*, June 1980, pp. 31-42.

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

Bunge, M. *Treatise on Basic Philosophy: Ontology II: A World of Systems*. Boston: Rediel, 1977.

Chen, P. "The Entity Relationship Model: Towards a Unified View of Data." *ACM Transactions on Database Systems*, Vol. 1, No. 1, June, 1976, pp. 9-36.

Conte, S. D.; Dunsmore, H. E.; and Shen, V. Y. *Software Engineering Metrics and Models*. Menlo Park, CA: The Benjamin/Cummings Publishing Company, Inc., 1986.

DeMarco, T. *Structured Analysis and System Specification*. New York: Yourdon Press, 1979.

DeMarco, T. *Controlling Software Projects*. Englewood Cliffs, NJ: Prentice-Hall, Inc., 1982.

Gane, C., and Sarson, T. *Structured Systems Analysis: Tools and Techniques*. Englewood Cliffs, NJ: Prentice-Hall, Inc., 1979.

Golden, J. R.; Mueller, J. R.; and Anselm, B. "Software Cost Estimating: Craft or Witchcraft." *DATABASE*, Spring 1981, pp. 12-14.

Halstead, M. H. *Elements of Software Science*. New York: Elsevier, North-Holland, 1977.

Jackson, M. *Principles of Program Design*. London: Academic Press, 1975.

Jones, C. *Programming Productivity*. New York: McGraw-Hill, Inc., 1986.

Kitchenham, B. A., and Taylor, N. R. "Software Cost Models." *ICL Technical Journal*, May 1984, pp. 73-102.

Mohanty, S. N. "Software Cost Estimation: Present and Future." *Software-Practice and Experience*, Vol. 11, 1981, pp. 103-121.

Putnam, L. H. "A General Empirical Solution to the Macro Software Sizing and Estimating Problem." *IEEE Transactions on Software Engineering*, July 1978, Vol. SE-4, No. 4, pp. 345-361.

Putnam, L. H., and Fitzsimmons, A. "Estimating Software Costs." *Datamation*, September, October, November 1979. Vol. 25, No. 10, 11, 12.

Rubin, H. A. "Macro-Estimation of Software Development Parameters: The ESTIMACS System." *SOFTAIR - Software Developement: Tools, Techniques, and Alternatives*, IEEE, July 1983, pp. 109-118.

Rubin, H. A. "The Art and Science of Software Estimation: Fifth Generation Estimators." *Proceedings of the Seventh Annual ISPA Conference*, Vol. 5, June 1985.

Stroud, J. M. "The Fine Structure of Psycholocial Time." In Henry Quastler (ed.), *Information Theory in Psychology*, Glenco, IL: The Free Press, 1954.

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

von Bertalanffy, L. *General Systems Theory*. New York: George Braziller, 1968.

Wand, Y., and Weber, R. "Formalization of Information Systems Design." University of British Columbia Working Paper, May 1987.

Warnier, J. *Logical Construction of Programs*. New York: Van Nostrand Reinhold, 1974.

Wolverton, R. W. "The Cost of Developing Large-Scale Software." *IEEE Transactions on Computers*. June 1974. Vol. c-23, No. 6, pp. 615-636.

Wrigley, C. D., and Dexter, A. S. "Software Development Estimation Models: A Review and Critique." *Proceedings of the Administrative Sciences Association of Canada: MIS Division*, Toronto, Canada, June 1987, pp. 125-138.

11. ENDNOTES

1. For a complete review of software metrics see Conte, Dunsmore and Shen (1986).

2. Effort is used in place of labour as effort is considered "professional labour."

3. Initial attempts to formalize this concept appear in Wand and Weber (1987).

4. For a more general articulation of this concept see Ashby (1956).

5. The system theoretic approach as articulated by Ashby (1956), von Bertalanffy (1968), and Bunge (1977), among others, and the more recent IS model of events, states and laws being developed by Wand and Weber (1987), turn out to be a useful vehicle for describing requirements complexity.

6. The importance of this from a project management perspective is that the series of decisions to proceed or not to proceed with a project must have consistent units for comparison.

7. An issue which may be raised at this point is that a "good" design would minimize the number of set changes, possibly to one. The position of this paper is that even with the "best" design, complex events will affect more than one set.

8. Clearly not all lines of code are equivalent. Ideally, with an automated code counter, one could count at the operator-operand level such as Halstead (1977). We used lines of FOCUS code based on statistical stability over a large number of LOC.