A MODEL AND METHOD FOR MEASURING INFORMATION SYSTEM SIZE

by

CLIVE DONALD WRIGLEY

B.A. (Honors), Simon Fraser University, 1980

A THESIS SUBMITTED IN PARTIAL FULFILMENT OF
THE REQUIREMENTS FOR THE DEGREE OF
DOCTOR OF PHILOSOPHY

in

THE FACULTY OF GRADUATE STUDIES
(Faculty of Commerce and Business Administration)

We accept this thesis as conforming
to the required standard

THE UNIVERSITY OF BRITISH COLUMBIA

December 1988

© Clive Donald Wrigley, 1988
In presenting this thesis in partial fulfilment of the requirements for an advanced degree at the University of British Columbia, I agree that the Library shall make it freely available for reference and study. I further agree that permission for extensive copying of this thesis for scholarly purposes may be granted by the head of my department or by his or her representatives. It is understood that copying or publication of this thesis for financial gain shall not be allowed without my written permission.

Department of COMMERCe

The University of British Columbia
Vancouver, Canada

Date DEC. 28, 1988
ABSTRACT

This thesis develops a measurement model and method that allows information system professionals to establish measures of information system size that are accurate and may be established early in the system development life cycle. It reports the results of an empirical investigation into the aspects of requirements and design metrics which lead to the production of source code. The theoretical foundations used to investigate this topic originate in systems theory, models of information systems and organizational theories of structure and complexity. Models of system development are reviewed as they play a key role in our notions of activities to perform during development. By drawing on existing estimating models from software engineering and other sources of practitioner literature, a model to predict development effort was synthesized. Three distinct constructs emerged: 1) system requirements, which drive effort; 2) personnel experience, which can mitigate effort; and 3) technology, which can also mitigate effort. System requirements was chosen to further define and operationalize, since they are the principal source of development complexity and hence system size. An Entity-Relationship and Event approach was taken to establish an early measure of System Requirements size. A theoretical framework of data and process complexity was developed which may be used to initially size a development project based on the information available at the requirements specification and design phases. It is argued that this new approach is more general than existing sizing techniques; namely, Function Points and Lines of Code. This new sizing approach is tested against 26 simple transaction based processing systems developed in FOCUS. An automated Code Analyser was developed to reverse engineer these systems back to their design measures. The personnel and technology variables were held constant for this initial test. For the thesis, only retrospective measurement occurred but it is expected that longitudinal measurement will eventually be possible. Two primary research contributions are seen emanating from this study. First is the development and application of theory to the problem of information system sizing. Second is a method for data collection and analysis which will help the software development industry move towards the goal of system development measurement and evaluation. This will improve planning for, and management of, information systems development.
# TABLE OF CONTENTS

List of Tables ................................................................................................. vi
List of Figures ................................................................................................. vii
Abstract ........................................................................................................... ii
Acknowledgements ........................................................................................ viii

Chapter 1. Introduction .................................................................................. 1
  1.1. Thesis Statement .................................................................................. 1
  1.2. Motivation: The Estimating Problem And Its Significance .................. 1
      1.2.0.1. Pilot Survey .............................................................................. 2
  1.3. Goals of the estimating process ............................................................ 3
  1.4. The measurement issues ...................................................................... 5
  1.5. Research Methods used in this thesis ................................................... 6
  1.6. Contribution to Knowledge and Profession .......................................... 7
  1.7. Thesis Structure .................................................................................. 8

Chapter 2. Relevant Literature Review ........................................................ 10
  2.1. Model Classification .......................................................................... 10
  2.2. Historical Development ..................................................................... 12
  2.3. Empirical Validations ....................................................................... 18
  2.4. Comparison of Factors and Model Sensitivity .................................... 20
      2.4.1. Discussion: Effort Estimation and Productivity ......................... 23
  2.5. System Sizing .................................................................................... 26
  2.6. Summary: The missing Link ............................................................... 31

Chapter 3. Theoretical Framework ............................................................... 33
  3.1. Framing the Issues ............................................................................. 33
  3.2. Key Information System Concepts ...................................................... 36
      3.2.1. System Development Transformations ....................................... 36
      3.2.2. Requirements Modelling .......................................................... 38
      3.2.3. A Dynamic Information System Model ..................................... 40
      3.2.4. Requirements Transformation ............................................... 42
      3.2.5. Requirements Complexity ....................................................... 45
      3.2.6. Requirements Sizing ................................................................. 46
      3.2.7. Square footage ......................................................................... 46
  3.3. Processing Complexity ....................................................................... 47
      3.3.1. Processing complexity based on E-R-E concepts ....................... 48
      3.3.2. Input Events ............................................................................. 49
      3.3.3. Output Events ........................................................................... 49
      3.3.4. Functionality ............................................................................ 51
  3.4. E-R-E and Function Points ............................................................... 52
  3.5. Formalizing Requirements Size ......................................................... 53
      3.5.1. Events, States and Transforms .................................................. 53
  3.6. Summary ............................................................................................ 58

Chapter 4. Empirical Research Design ........................................................ 60
<table>
<thead>
<tr>
<th>Section</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>6.1.4.2. Test For Learning Effects</td>
<td>129</td>
</tr>
<tr>
<td>6.2. Unit of Analysis: System Level</td>
<td>130</td>
</tr>
<tr>
<td>6.2.1. Demographics</td>
<td>131</td>
</tr>
<tr>
<td>6.2.2. Metric Linkage at System Level</td>
<td>132</td>
</tr>
<tr>
<td>6.2.3. Resource Consumption</td>
<td>133</td>
</tr>
<tr>
<td>6.3. Unit of Analysis: Firm Level</td>
<td>136</td>
</tr>
<tr>
<td>6.4. Summary and Discussion</td>
<td>138</td>
</tr>
<tr>
<td>6.4.1. Generalizability</td>
<td>139</td>
</tr>
<tr>
<td>6.4.1.1. System Size</td>
<td>139</td>
</tr>
<tr>
<td>6.4.1.2. Model Validation</td>
<td>140</td>
</tr>
<tr>
<td>6.4.1.3. Reverse Engineering</td>
<td>141</td>
</tr>
<tr>
<td>6.4.1.4. Forward Engineering</td>
<td>142</td>
</tr>
<tr>
<td>6.4.1.5. Hours Data</td>
<td>142</td>
</tr>
<tr>
<td>6.4.1.6. Language</td>
<td>143</td>
</tr>
<tr>
<td>6.4.2. Maintenance</td>
<td>147</td>
</tr>
<tr>
<td>6.4.3. System Decomposition</td>
<td>148</td>
</tr>
<tr>
<td>Chapter 7. Summary and Research Directions</td>
<td>150</td>
</tr>
<tr>
<td>7.1. Thesis Summary</td>
<td>150</td>
</tr>
<tr>
<td>7.1.1. Empirical Limitations</td>
<td>151</td>
</tr>
<tr>
<td>7.2. Direct Extensions</td>
<td>152</td>
</tr>
<tr>
<td>7.3. Expert Behaviour</td>
<td>154</td>
</tr>
<tr>
<td>7.4. Modelling</td>
<td>155</td>
</tr>
<tr>
<td>7.5. Management</td>
<td>156</td>
</tr>
<tr>
<td>7.6. Closing Remarks</td>
<td>157</td>
</tr>
<tr>
<td>References</td>
<td>158</td>
</tr>
<tr>
<td>Appendix A: Operand Coding Scheme</td>
<td>164</td>
</tr>
<tr>
<td>Appendix B: Pilot Validation Results</td>
<td>166</td>
</tr>
<tr>
<td>Appendix C: Systems Database Detailed Structure</td>
<td>177</td>
</tr>
<tr>
<td>Appendix D: Code Analyser Source Code</td>
<td>180</td>
</tr>
</tbody>
</table>
List of Tables

Table 2.1: Comparison of Estimating Approaches ......................................................... 11
Table 2.2: Effects on Productivity .................................................................................... 21
Table 5.1: Pilot Regression - Explained Variance in Code Size ........................................ 105
Table 6.1: Program class differences .............................................................................. 115
Table 6.2.1: Regression model - Update class ................................................................. 118
Table 6.2.2: Regression model - Output class ................................................................. 119
Table 6.2.1: Regression model - Control class ............................................................... 120
Table 6.3: Reduced Model .............................................................................................. 123
Table 6.4: Programmer by Program Class ...................................................................... 124
Table 6.5.1: Programmer Analysis - Updates .................................................................. 126
Table 6.5.2: Programmer Analysis - Outputs .................................................................. 127
Table 6.5.3: Programmer Analysis - Control .................................................................. 128
Table 6.6: System Size Measures .................................................................................... 132
Table 6.7: Design Predictors of System Size .................................................................. 133
List of Figures

Figure 2.1: Simplified Water Fall Model .......................................................... 29
Figure 2.2: De Marco’s Project Classification ................................................... 31
Figure 3.1: Requirements, Personnel and Tools Model .................................... 35
Figure 3.2: System Development Transformations ........................................... 38
Figure 3.3: Information System Model ............................................................. 55
Figure 4.1: Research Model .............................................................................. 63
Figure 4.2: Detailed Research Model ............................................................... 66
Figure 4.3: Metric Linkages ............................................................................. 69
Figure 4.4: System Development - Measurement and Evaluation .................. 72
Figure 4.5: Asset Disposal System: E-R-E Structure ..................................... 78
Figure 4.6: Asset Disposal System Processes .................................................. 80
Figure 4.7: Top Level Data Flow Diagram ..................................................... 81
Figure 4.8: ADS Main Menu Processes ........................................................... 83
Figure 4.9: Sub-Menu 1020: Report Processes ................................................ 85
Figure 4.10: Batch Report Run 4080 ............................................................... 86
Figure 4.11: Asset Status Report 4160 ............................................................. 87
Figure 4.12: Outputs from Code Analyzer ...................................................... 92
Figure 4.13: Systems Database ....................................................................... 97
Figure 6.1: Program size distribution .............................................................. 114
Figure 6.2: Hours vs. Lines of Code ............................................................... 135
Figure 6.3: Program size distribution - Site 2 ................................................ 138
Figure 6.4: Calculated vs. Actual length ........................................................ 146
ACKNOWLEDGEMENTS

It is impossible of course to identify all the contributors towards a thesis. Our thoughts are mosaics of what we have read, heard and discussed. Many ideas come from off-hand comments in seminars and conversation. It is amusing to remember oneself bask in the delight of what appeared to be a precocious new thought only to find several days later that another author had articulated the concept much better five years earlier.

The role of the thesis advisor in this process is incalculable, so first and foremost I would like to acknowledge and thank Albert Dexter whose continued patience, support and humour smoothed out the rough spots while continually challenging me not to be satisfied with the ordinary. Although, at times this can be a curse without the self discipline to constrain ones ambitions. Al, you will be a friend and colleague for many long years to come.

To the other members of my committee, Craig Pinder and Son Vuong, for your insight and guidance through this lucubration and to the faculty and graduate students in the MIS division, I thank you for your candour. Critical appraisal is the route to knowledge.

To the managers who provided access to their data and to those working in the data sites, who unwittingly became involved in this project yet gave their fullest co-operation, I extend my gratitude. Hopefully the output from this research is not too small a payment.

Finally, to my family, Don, Dorothy, Lynda, Joyce and Dugal, whose unending patience and moral support, if not understanding, gave me the motivation to keep going. And of course to Shandy - who really knows how to live.
A MODEL AND METHOD FOR
MEASURING INFORMATION SYSTEM SIZE
CHAPTER 1. INTRODUCTION

1.1. THESIS STATEMENT

The objective of this dissertation is to establish a theoretical and empirical link between:

1. The entities, events, and their inter-relationships that occur in the real world, and
2. The size of the implemented information system which represents those real-world entities, relationships and events.

1.2. MOTIVATION: THE ESTIMATING PROBLEM AND ITS SIGNIFICANCE

There are essentially two distinct issues with which the Information Systems Manager is concerned. The first is the decision of whether or not to proceed with a software development project. Here the accuracy of the total life cycle development cost estimate is critical as it is the prime input into the cost/benefit decision models which we use to justify our investments. As Boehm [1981] writes:

"There is no good way to perform a software cost benefit analysis, break even analysis, or make or buy decision without some reasonably accurate method of estimating software costs, and their sensitivity to various product, project, and environmental factors." (pg. 30)

The second issue is the management of the development process. Total effort to be expended must be distributed over the project duration. The two aspects of this later issue are scheduling of resources and project control. Estimates form a baseline against which the management process can be carried out. The usefulness of project
management tools, such as PERT and Gantt charts, depend mainly on the accuracy of the original estimate. A good estimate will identify areas of project development that may cause problems or require a greater degree of managerial control. Without this baseline estimate, projects are out of control from their inception.

Two measures of system success are whether a project is delivered on time and within budget. Obviously the accuracy of the original estimate may determine if a system is considered successful given these criteria. As Pressman [1987] explains:

"In the early days of computing, software costs represented a small percentage of the overall cost of a computer based system. A sizable error in estimates of software costs had relatively little impact. Today, software is the most expensive element in many computer based systems. A large cost estimation error can make the difference between profit and loss or between system success and system abandonment." (pg.98)

It is the purpose of this thesis:

1. To argue that system sizing is a major area of difficulty in estimating resource consumption,
2. To develop a theoretical approach to system sizing,
3. To demonstrate that the sizing model and method can be used to explain system size at the various phases of information system development, and
4. To demonstrate that units of system size can be related to units of resource consumption.

1.2.0.1 Pilot Survey

To establish if, in fact, estimating software costs is of real concern to practicing IS professionals, a pilot survey was conducted in the Vancouver Area among data processing managers. These managers are members of the FOCUS users group and represent most of the larger DP shops in Vancouver. Fifty questionnaires were distributed which asked basic questions about the use of estimating methods and the
percentage of software development projects delivered on time and within budget. Of the 30% who responded, it was discovered that very little use was made of formal estimating methods. Most managers relied on past experience and gut feel. Further, their self-reported data on estimate accuracy revealed that over 50% of all projects were completed either over time or budget schedule. While we could speculate that the large percentage of projects being over budget was due to not using formal estimating techniques, it may also be due to the lack of usefulness of these techniques. The basic findings of the survey were presented at a FUSE (Focus Users group) meeting in June 1987. A great deal of interest was expressed by these practitioners to the extent that several firms agreed to provide access to data on their own systems development projects.

1.3. GOALS OF THE ESTIMATING PROCESS

In order to construct an estimate, resources must be expended to gather information on which the estimate is based. This introduces two questions: 1) What is the appropriate information to gather? and, 2) What quantity of resources should be expended to gather this information in order to make an estimate? Once these two questions have been addressed an estimate is constructed. Finally, a decision must be made as to the quality of the estimate. This last point is best expressed by Aristotle (330 B.C.E.):

"...it is the mark of an instructed mind to rest satisfied with the degree of precision which the nature of the subject admits, and not to seek exactness when only an approximation of the truth is possible..."

1In Pressman [1987] pg.82
Within the broader scope of IS investment decisions and project management many issues enter into the picture. More specifically, an IS development estimate should attempt to address the following basic questions [Rubin 1983]:

1. What quantity of human resources are required?
   a. What kind of skills are needed?
   b. How many people are required?
   c. What are the constraints?

2. How much will it cost?
3. How long will it take?
4. What are the risks?
5. What will the effect be on the existing portfolio?
6. What are the trade-offs?

While all of the above are clearly important, this research will focus on a more critical issue on which all of the above questions are based. The central issue revolves around the concept of system size. Early in the System Development Life Cycle (SDLC) what properties of a system are available for counting and what units should be used? Can these measurable properties of system requirements be directly linked to implementable code. If this is the case, then by establishing the linkages between requirements and code, it is possible to construct estimates of code size based upon the requirement measures. In general, once this central question can be accurately assessed, dollar cost figures, schedule, skills, and number of people required can be derived reasonably well given existing project management tools.

The estimating process can be considered as a semi-structured problem which requires information in order to predict the value of a number of variables. We can safely ignore the pathological case where an estimate is requested in the absence of any information. The estimating process is ongoing, since, as more information is gathered...
refinements on the initial estimate are possible. After all information has been gathered, i.e. at project completion, the difference between the estimate and the actual should be zero. The principal goal is to produce an accurate estimate of resource consumption as early as possible with the minimum amount of information.

1.4. THE MEASUREMENT ISSUES

In order to construct an estimate of any kind, a measurement must be made. The first measurement issue is to determine which variables in the development process are causally connected. The second, is to actually measure those variables. For example, in order to estimate system development dollar costs you must first be able to determine the functional relationship between resource consumption and dollars. Clearly, the largest determinant of costs in software development is the consumption of the labour resource, but estimating labour requirements is the central problem. We first have to be able to measure some properties of the software project in order to estimate labour effort. One common practice is to estimate overall project size by first estimating Lines of Code (LOC) and then estimate labour effort as a function of this size metric. As we shall see there is a serious conceptual flaw with the LOC estimating approach. In brief, LOC is correlated with effort but does not causally determine effort. Effort is expended to produce LOC. It is effort that causes the production of LOC, not the other way around. A second approach is to size a project using Function Points (FP). While this basic approach is sound it is limited in terms of when accurate information is available to make an estimate.

The timing of the measure is critical. If a size estimate of LOC is made after the
detail design is complete then we would feel more comfortable with its accuracy than if LOC were estimated during analysis. LOC may be an appropriate basis for estimating effort but only at a certain point in time, usually late in the development process. Likewise the accuracy of a FP measure will be higher if it is made when sufficient information is available.

1.5. **RESEARCH METHODS USED IN THIS THESIS**

1. Existing estimating methods were investigated and found to have deficiencies.

2. A causal model was synthesized from the literature based on existing models and the reported effects of various factors on the development process. The model contains three major variables which are claimed to causally affect effort: System requirements, Personnel experience, and Technology.

3. A pilot survey was conducted to establish the practical need for research in this area and to solicit access to live data in order to test the model.

4. This thesis starts from the theoretical position that the underlying entities and events which the information system models is the root source of effort to build the system. Theoretical development from this position draws from systems theory concepts, theory of the system development process, and requirements modelling using both the data flow and the Entity Relationship data model approaches.

5. Data on completed systems was collected and used to calibrate the proposed requirements and design metrics. Technology and experience were held constant in the field setting.

2 Here accuracy is defined as (Actual LOC - Estimated LOC)/ Actual LOC. This is generalizable to other accuracy measures in the general form (Actual measure - Estimated measure)/ Actual measure
6. The requirements and design size metrics were empirically validated by predicting code size based on information present in the source code of completed systems using regression analysis.

1.6. CONTRIBUTION TO KNOWLEDGE AND PROFESSION

1. Current techniques for sizing system development projects are based on practice not theory. It is anticipated that by applying a theory of requirements size to estimating effort required to build an information system a theoretically based metric can be developed.

2. The focus of existing estimating techniques can be considered to be on the software processes and not on the data requirements. This research will attempt to show, in part, that data structures are more general than processes for the purposes of estimating.

3. An earlier, more parsimonious, more reliable metric of requirements and design size is expected. It is anticipated that the E-R data structure and event approach to estimating (E-R-E) encompasses the two most common sizing techniques, namely, Function Points and Lines of Code. An E-R-E diagram is used to capture the essential objects and relationships that exist in the real world and their dynamics. By measuring more permanent, more basic objects in a business system, we should be able to predict not only effort but also Function Points and Lines of Code. If this can be achieved, then the E-R-E approach could be considered a more generalized form of the other two approaches.

4. The design and development of a prototype source code analyser provides the capability to reverse-engineer installed systems. This provides management with a

---

It may also be the case that a data structure approach is a more general approach to analysis and design. However, this issue will not be addressed in this thesis.
tool to fully understand their installed software base by providing reliable software metrics.

5. This study is the first to investigate estimating in system development environments using a 4th generation language. If generalizable, this will open the door to estimating in modern development environments.

1.7. **THESIS STRUCTURE**

Chapter 2 of this thesis reviews the relevant literature on estimating and software metrics. Previous estimating models are also reviewed and assessed. The factors found to affect development effort are organized into a table where a basic structure emerges. Three major influences are found which lay the groundwork for the theoretical development in Chapter 3.

Chapter 3 develops a theory of requirements size and proposes a method of measurement based on entities, relationships and events.

Chapter 4 moves from the theory to the empirical domain. It presents a research model containing both the theoretical constructs developed in chapter 3 and the empirical measures proposed to substantiate the theory. An example system is used to articulate the measures suggested. A field study research design is described where the theory may be tested. The design and construction of an automated Code Analyser is presented to address the issue of measurement reliability.

Chapter 5 describes the empirical setting and the data collected in order to test the
estimation model. Data reliability and validity issues are addressed by comparing manual counting methods with the automated tool.

Chapter 6 discusses the results of the data analysis and identifies the limitations of the research.

Chapter 7 contains a discussion on the ramifications of this research both in terms of extending the academic investigation and in terms of its potential contribution to industry.
CHAPTER 2. RELEVANT LITERATURE REVIEW

2.1. MODEL CLASSIFICATION

Several authors have suggested a taxonomy of estimating approaches: [Wolverton, 1974; Basili, 1980; Benbasat and Vessey, 1980; Kitchenham and Taylor, 1984; Conte, Dunsmore and Shen, 1986]

1. **Expert judgement:** Estimates are arrived at from "gut feel" and perhaps some experience with other projects sharing a common attribute. Subjective evaluations and comparisons are used heuristically. This technique is used when completely new projects are considered. It marks the most common approach used in the early years of computing. Also, a pilot survey found this approach still common practice.

2. **Mathematical or Algorithmic models:** The next stage of development appears to be the construction of formulas based on attributes of a number of completed projects. Curve fitting and factor analysis are applied to a dataset in the hopes of teasing out some underlying effects. The resulting equation(s) has a number of parameters which presumably correlate highly with system development effort. Distribution of this effort over the schedule is derived from assumptions about the shape of the life cycle curve.

3. **Bottom up:** This approach is widely employed when a structured systems development methodology is available. The project is broken down into identifiable tasks, each task is separately estimated, then summed up over the entire project. While this is expected to reduce error variance it requires far more effort than is feasible for initial job sizing.

4. **Top down:** Global properties of the software product, technology, environment and personnel are used to create an overall estimate. Effort is then distributed
over the project according to the assumed shape of the life cycle curve. It is this technique which offers the most promise for generating early accurate estimates.

Kitchenham and Taylor [1984] point out that all of the above approaches share a common attribute: they all require historical data of completed projects, and even the so-called theoretical mathematical models are constructed and calibrated from real data. It would appear from their analysis that there is an absence of pure theory in this area. A brief comparison of the strengths and weaknesses of these approaches appears in Table 2.1.

Table 2.1

<table>
<thead>
<tr>
<th>Approach</th>
<th>Strengths</th>
<th>Weaknesses</th>
</tr>
</thead>
<tbody>
<tr>
<td>Expert judgement</td>
<td>• Assessment of reasonableness, interactions, exceptional circumstances</td>
<td>• No better than experience, imperfect recall</td>
</tr>
<tr>
<td>Algorithmic</td>
<td>• Objective, repeatable, analyzable formula, efficient, good for sensitivity analysis</td>
<td>• Subjective inputs, calibrated to past not future</td>
</tr>
<tr>
<td>Bottom-up</td>
<td>• High level of detail, more stable</td>
<td>• May overlook some costs, requires more effort</td>
</tr>
<tr>
<td>Top Down</td>
<td>• System level focus</td>
<td>• Less detailed basis, less stable</td>
</tr>
</tbody>
</table>

4 Adapted from Boehm [1981], pg. 342
2.2. HISTORICAL DEVELOPMENT

The first published work on software cost estimation came from Nelson [1966]. He constructed a regression equation from 169 software projects studied by System Development Corporation (SDC). This regression model contains 14 parameters which are rated by the estimator. The dependent variable is measured in man-months (MM). Against its own data base, the model produces a mean estimate of 40 MM with a standard error of 62 MM. With a 1.5 co-efficient of variation, clearly something may be missing. The principal conclusion from the study was that there were too many non-linear relationships for a linear regression to be meaningful.

During the early seventies, TRW was involved with the development of numerous real time military projects. Out of this project base, Wolverton [1974], developed a graph relating cost per instruction to difficulty of project, for 6 different project categories. Difficulty is on a percentile scale, i.e. relative degree of difficulty compared to other projects. Cost per instruction relates to a line of assembler code. His model is basically algorithmic taking the three input parameters, lines of code, level of difficulty, the historical cost base and produces a total project cost estimate. His model forecasts that difficulty can change cost per instruction by 100%.

In 1975 Brooks wrote his now famous book "The Mythical Man Month". While not having a model per se, Brooks motivated the central idea that personnel and time are not interchangeable. It is this non-linear relationship which had confounded the

management of projects for years. "Adding people to a late project makes it later."

A major study to investigate programming productivity was conducted by Walston and Felix [1977] at IBM's Federal Systems Division (FSD). They investigated 60 projects, written in 28 different languages, ranging in size from 4000 to 467,000 lines of code. Productivity was found to range from 27 to 1000 delivered source lines per man-month, nearly two orders of magnitude! From this data set 68 project attributes were collected and compared to lines of code per man-month, 29 of these were found to be significant. The basic model calculates a productivity index based upon the 29 significant "cost drivers":

\[ I = \sum_{i=1}^{29} W_i X_i \]

where:
- \( W_i = 0.5 \log_{10}(PC)_i \)
- \( X_i \) = project attribute

and \((PC)_i\) is the productivity change between a low and high rating for the \(i\)th cost driver attribute.

Putnam's SLIM model [Putnam, 1978; Putnam and Fitzsimmons, 1979] uses the Rayleigh distribution to relate the three critical variables: source lines of code, total man-years, and project duration. The software equation is:

\[ S = CK^{33}t^{1.33} \]

where:
- \( S \) = delivered source lines
- \( K \) = life-cycle effort in man-years
- \( t \) = development time in years
- \( C \) = a technology constant

\[ ^{6} \text{Brooks [1975], pg.25} \]
The user of the model typically inputs an initial size estimate, then manipulates the shape of the curve to arrive at the final effort estimate. The shape is modified by two parameters, the productivity factor $C$, and its initial slope, representing the constraint on personnel buildup. These parameters can be calibrated either by accepting input data from completed projects or answering a series of 22 questions. In 1973, $C$ was calibrated at 4900. By 1978 this figure had reached 10,000, indicating either productivity gains of 100% over 5 years or lack of stability in the parameters.

Later, Alan Albrecht of IBM changed the preoccupation with lines of code as the principal sizing metric by introducing function points [Albrecht, 1979]. Instead of estimating lines of code, a count is made of identifiable system functions: external inputs, external outputs, logical internal file, external interface file, external inquiry. Each function has an associated number of points:

- External Input = 4
- External Output = 5
- Logical Internal file = 10
- External Interface file = 7
- External Inquiry = 4

The software product is initially sized by multiplying the number of identifiable software functions in each category by its associated function points. This function count is modified by a number of complexity adjustment factors to account for the different kinds of system requirements and development environments. In the original model the adjustment could modify the initial function count by plus or minus 35%; however, the effects of complexity have since been revised upwards to plus or minus 300%. This model is the first example of a top down approach to estimation.

In 1981 Barry Boehm of TRW published his COstructive COst MOdel (COCOMO). This
model is based on the analysis of 63 software projects during his tenure as Director of Software Research and Technology. The basic model relates "thousands of delivered source instructions" (KDSI) to "effort" (MM):

$$\text{MM} = 2.4(K\text{DSI})^{1.05}$$

The development schedule in months (TDEV) is estimated next with:

$$\text{TDEV} = 2.5(\text{MM})^{0.38}$$

The distribution of effort by phase within this time period is taken from the normative prescriptions of the traditional life cycle curve.

The detailed COCOMO considers 15 "cost drivers" which have been found to affect productivity. These cost drivers are applied to the different phases of development to adjust the original effort estimate upward or downward. The basic COCOMO can be considered an algorithmic model while the consideration of macro effects in the detailed version makes it a hybrid somewhere between top-down and algorithmic. When applied to its own database of projects, COCOMO predicts development costs within 20% of actuals 70% of the time. Boehm claims this translates to a standard deviation of the residuals of roughly 20% of the actuals.7

The Bailey-Basili meta model was published in 1981 and is similar to Boehm's COCOMO model. Developed from 18 projects in the NASA-Goddard Software Engineering Laboratory it uses prior effort and size data to scale three parameters:

$$\text{Effort} = a(\text{Size})^b + c$$

Two other project attributes, total methodology (METH) and cumulative complexity (CMPLEX) are then used to determine effort multipliers and error ratio (ER) in the

7Boehm [1981], pg. 521
Effort = ER \[a(\text{Size})^b + c\]
ER = d(METH) + e(CMPLX) + f

When applied to its own database the resulting equations were:

Effort = ER[0.73 + (Size)^1.16 + 3.5]
ER = -0.036(METH) + 0.009(CMPLX) + 0.80

with a standard error of estimate of 1.15.

The ESTIMACS model was developed and published in 1983 by Howard Rubin of Hunter College [Rubin, 1983]. As it is a proprietary product, no detailed information is currently available. The model engages the user in a series of 25 questions. It uses a modified function point approach for a size estimate which is then adjusted by assumptions about project complexity. This adjustment can change the function point estimate by a factor of 2. In addition, ESTIMACS produces staffing profiles, a development schedule and a project risk estimate.

International Telephone and Telegraph formed a study group to investigate the factors affecting productivity. The results were published by Vosburgh [1984]. The group analyzed 44 projects, written in several different languages, ranging in size from 5000 to 500,000 lines of code. The projects were a mix of switching systems, defense applications, and process control. Beginning with a list of 100 factors, the researchers found 5 to be significant. Regressed against the dependent variable effort, these 5 explained 65% of the variance:

1. Complexity and resource constraints 16%
2. Client Interface and experience 12%
3. Modern Programming practice 24%
The most recent estimating product, SPQR/20 [Jones, 1986], was also derived from Albrecht's function approach. Developed by Capers Jones of Software Productivity Research Inc., it is an interactive package which provides similar outputs to ESTIMACS. The user is prompted by a series of product and project related questions. Then the project size estimate is input by the user in terms of function point categories.

In addition to the above models, a number of bottom-up estimating approaches have been developed from proprietary system development methodologies. One example of this class is ESTIMATE/1 developed and used by Arthur Andersen and Co. The basic structure of the model is as follows:

\[
\begin{align*}
F_f &= \text{Estimating factor, } f = 1..n \\
K_k &= \text{Constant to adjust units to common base, } k = 1..n \\
T_t &= \text{Task in work breakdown structure, } t = 1..m \\
S_s &= \text{Segment: includes a group of tasks, } s = 1..p \\
C_s &= \text{Estimate by analyst of overall complexity for segment } s \\
UT_t &= \text{# of units in task } t \\
UT_t &= fn(F/K_f...F/K_j) \\
HT_t &= \text{Hours estimated to complete task } t \\
HT_t &= UT_t * fn(C_s, T_t)^8 
\end{align*}
\]

A base estimate is calculated by summing up over all tasks. This base is then adjusted for several macro complexity factors. The complexity multipliers at this macro level plus the multipliers at the task level can effect total project costs by several hundred percent. The distribution of effort is explicit in the work breakdown structure. Cost figures for each category of personnel, e.g. senior analyst down to clerks, are

\[^8 \text{fnC is derived empirically and is considered proprietary information.}\]
multiplied by the number of hours found in the task estimates. Each person on the project team also has a productivity multiplier which is used to adjust the total cost figure up or down.

2.3. **EMPIRICAL VALIDATIONS**

There have been few attempts to externally validate the various models against other projects to date. There are several possible reasons for this. First, many consider their project database as proprietary information. The second, and far more likely is that the data is simply not collected. Once a system is up and running there is little motivation to conduct a post mortem. It is unlikely that project management systems are geared to track critical data points. Third, formal estimation techniques are not widely used. Expert judgement may still be the dominant approach.

Brooks [1981] reviewed the Walston and Felix database and noted that 4 of the 29 variables related to structured programming. These variables were grouped as one and then the projects were re-analyzed. He found that productivity gains of 35% were achieved for small projects reporting the use of structured programming, while gains of 200% were found for larger projects using similar techniques. Further, he found that in projects using unstructured techniques low productivity was highly correlated with high customer interface complexity, high application complexity, timing and storage constraints, and low personnel experience.

Albrecht and Gaffney [1983] collected data on a new set of projects in an attempt to established a correlation between function points and source lines of code. This is
important because a major problem in estimating is in obtaining a reasonable feel for the software size. His sample consisted of 24 projects developed by IBM DP Services. Relations were found between language type, size, and work effort. It appears from this article that function points map well to eventual lines of code.\(^9\) In a separate validation study, Kemerer [1987] found a .65 correlation between function points and lines of code for 15 Cobol projects.

Behrens [1983] collected data from 11 projects completed in 1980 and 14 completed in 1981. The projects were sized using Albrecht's function point technique. Cost data was made available from an automated project management system. He found an exponential relation between function points and work effort. Language used and development environment were also found to affect work effort. In contradiction to many studies, years of experience was not found to be a significant factor.

Kitchenham et.al. [1984] evaluated both SLIM and COCOMO in a joint research project between ICL and British Telecom. There objective was to find an estimating model to use in future development projects. Their dataset consisted of 20 completed projects dealing with advanced telephone switching. Many of these systems were of a real-time nature and developed at many different sites. Plots of actuals vs. estimates show little systematic relation. They pointed out many difficulties in using and calibrating the models to their particular environments. Their conclusions were to reject the use of either model and to establish their own historical database.

Kemerer [1987] conducted a post hoc evaluation of 15 projects using 4 of the

\(^9\) Jones [1986], pg. 77, gives examples of the size mapping between 25 languages and function points.
Relevant Literature Review / 20

previous models: SLIM, COCOMO, ESTIMACS, and FUNCTION POINTS. Data were collected from completed projects within the same organization. Nearly all projects were developed in COBOL and ranged in size from 39 thousand source lines of code (KSLOC) to 1107 KSLOC, with the mean around 190 KSLOC. Each of the models were used to estimate total effort which were then compared to actual effort. SLIM had an average percent error of 772%, i.e. SLIM consistently estimated more effort than was actually used. Similarly, COCOMO had an average error of 614%. Albrecht’s function point technique did much better with the average error at 102%. Finally, Rubin’s ESTIMACS package produced errors of 167%.

2.4. COMPARISON OF FACTORS AND MODEL SENSITIVITY

Five of the preceding models have been selected for further analysis. The criteria for inclusion was made informally based primarily upon the publication of empirical results but also from the frequency of references by other authors.

Table 2.2, from Wrigley and Dexter [1987], contains a collection of 74 factors which have been found to affect development effort. Several factor classifications appear in the literature. Boehm [1981] organizes 29 factors into 5 major groupings: size, program, computer, personnel, and project attributes. Factors are equated to attributes in his analysis. Kemerer [1986] builds on this basic taxonomy but adds a sixth group, user factors, and renames program to product, and computer to environment.

Although Boehm’s factor classification has been generally maintained, a higher level of abstraction has been added to serve as the basis of a conceptual model presented in
### System Requirements:

<table>
<thead>
<tr>
<th>Requirement</th>
<th>IBM</th>
<th>COC</th>
<th>PSD</th>
<th>OMO</th>
<th>SLIM</th>
<th>MACS</th>
<th>SPOR</th>
</tr>
</thead>
<tbody>
<tr>
<td>Product Size</td>
<td></td>
<td></td>
<td>1000</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Data Base Size</td>
<td>73</td>
<td></td>
<td>44</td>
<td>*</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td># of Items in Database/1000 LOC</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td># of Logical Files</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td># of Logical Business Inputs</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td># of Logical Business Outputs</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td># of On-line Inquiries</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td># of Business Functions To Implement</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>% Functions User Already Performs</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>% Functions Already Automated</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Source Code Re-usability</td>
<td>1000</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Complexity</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Customer Interface Complexity</td>
<td>300</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Complexity of Code</td>
<td>70</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Complexity of Application Processing</td>
<td>100</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Complexity of Program Flow</td>
<td>30</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>% Code Non-Mathematical or I/O</td>
<td>42</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Software Complexity</td>
<td>135</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Complexity of Processing Logic</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Logical Complexity</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Structural Complexity</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Data Complexity</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Time</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Batch, On-line, Real Time</td>
<td>77</td>
<td>95</td>
<td></td>
<td>*</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Execution Time Constraints</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Performance Explicit Design Criteria</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Response Time Constraints</td>
<td>37</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>% Code for Real Time or Interactive</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Documentation</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>User Doc. (Excellent..Informal)</td>
<td>64</td>
<td></td>
<td>30</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td># Pages Documentation/1000 LOC</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Reliability</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Required Reliability</td>
<td>180</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Back-up System Required</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Automated Error Detection Needed</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Special Disaster Recovery Needed</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Critical Business System</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Project Class (Personnel..Military)</td>
<td>140</td>
<td></td>
<td>15</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Project Type (Batch..AI)</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>User Participation</td>
<td>52</td>
<td></td>
<td>50</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Requirements Volatility</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

### Personnel Experience:

<table>
<thead>
<tr>
<th>Requirement</th>
<th>IBM</th>
<th>COC</th>
<th>PSD</th>
<th>OMO</th>
<th>SLIM</th>
<th>MACS</th>
<th>SPOR</th>
</tr>
</thead>
<tbody>
<tr>
<td>Overall Personnel Experience</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>210</td>
<td>144</td>
<td>200</td>
</tr>
<tr>
<td>Analysts Capability</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Programmer Capability</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Language Experience</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Experience with Operational Hardware</td>
<td>43</td>
<td>64</td>
<td>116</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Application Experience</td>
<td>180</td>
<td>47</td>
<td>50</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Project Novelty (Conversion..New)</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Comparable System in Operation</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>User Knowledge of Data Processing</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Customer Experience</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

### Project Development:

<table>
<thead>
<tr>
<th>Requirement</th>
<th>IBM</th>
<th>COC</th>
<th>PSD</th>
<th>OMO</th>
<th>SLIM</th>
<th>MACS</th>
<th>SPOR</th>
</tr>
</thead>
<tbody>
<tr>
<td>Methods and Tools</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Use of Structured Programming</td>
<td>78</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Use of Design and Code Inspections</td>
<td>54</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Use of Top Down Development</td>
<td>64</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Use of Chief Programmer Team</td>
<td>85</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Use of Modern Programming Practices</td>
<td>130</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Use of Software Tools</td>
<td>107</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Use of Automated Design Tools</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Language Used</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Environment</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Distributed, On-line, or Batch</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Computer turnaround Time</td>
<td>44</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Computer Access Open</td>
<td>58</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Computer Access Closed</td>
<td>78</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Classified Security Environment</td>
<td>85</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Office Facilities (Separate..Crammed)</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td># Sites in Other Cities</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td># People Relocated for Project</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Travel Time to Involved Sites</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Process</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Development Schedule Constraint</td>
<td>20</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Constraints on Program Design</td>
<td>64</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Main Storage Constraints</td>
<td>85</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Virtual Machine Volatility</td>
<td>75</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Hardware Concurrently Developed</td>
<td>68</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>% Programmers in Design &amp; Develop</td>
<td>155</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>(Mean Staff Size)/Duration</td>
<td>76</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>User/Management Agreement</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td># of People in Organization Involved</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td># of Organizational Units in Project</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td># Users on Development Team</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Table 2.2 Effects on Productivity (%)
Chapter 3. The three top level factor groupings are system requirements, personnel experience and development technology. Analysis of Table 2.2 shows that most of the 74 factors map reasonably well into these three major groupings.

Another aspect of Table 2.2 is worth noting. Complexity has been split off as a separate class. One reason for this is the general lack of clarity as to what the definition of complexity is and the resulting lack of rigour in measuring its effects. But more importantly, complexity has been found to influence much of the development effort. For example, in 1979, Albrecht allowed complexity factors to impact the estimate by plus or minus 35%. This has since been revised upward to 300% potential effect [Jones 1986].

Since authors publish their results in different ways it was necessary to convert the various findings to a common base for comparison. The figures in the columns under each model represent the potential impact on productivity measured in percent. What this means is that, ignoring all other factors and possible correlations, the productivity of projects having a high rating on the factor was higher by X%, when compared to projects having a low rating on the factor (at some level of significance). Where not otherwise obvious, the usual scale for each factor was low, medium and high. For example, in the IBM-FSD database, projects having a high level of personnel experience reported productivity gains of 210% compared to projects having low personnel experience. Clearly for some factors the reverse effect is expected. For example, in the same dataset, productivity gains of 300% were found for projects with low customer interface complexity. The 's are used to show that the author includes this factor in
his model but the weight of its impact is not presently available\textsuperscript{10}. More than one entry per line indicates that this factor is referred to exactly the same way in each of the indicated models. It is obvious that there is much duplication in Table 2.2. One purpose of presenting the data in this manner is to show the diversity and ambiguity surrounding these models.

2.4.1. Discussion: Effort Estimation and Productivity

The concepts of software effort estimation and programmer productivity are inextricably linked. Explicit in all models of cost estimation are notions of factors which either increase or decrease software development effort. Programmer productivity gains are achieved by enhancing those factors which are believed to decrease effort and mitigating those that are believed to increase effort. For the most part these factors have been empirically derived from collections of projects. A question of external validity can be raised when models are constructed from historical data. The project mix may significantly influence the model parameters. For example, of the 63 projects in Boehm’s COCOMO database only 7 are classified as business applications while 24 were developed in FORTRAN and 20 in Assembler. Generalizations from this sample to other areas may be risky.

The better empirical studies report the effects of various factors on productivity. Productivity, however, is a deceptively amorphous concept. In fields other than software

\textsuperscript{10} In an independent test of SPQR/20 the authors found each of the complexity factors to impact development costs by up to 80\%, and by nearly one order of magnitude when combined. However, this may be due to the specific attributes of the software system used as a test case.
engineering there is usually some reasonable metric for the output or yield from a process. Likewise, inputs can be measured. Economic productivity can be easily computed as yield/input. The problem we face in software development is that both the inputs and the outputs have not yet had a stable metric for comparison. Nevertheless, if we are to progress in this area, some measures are required. Currently the common usage is to measure inputs in terms of man-months (MM) and outputs in lines of code (LOC). Productivity is then defined as LOC/MM.

One difficulty in interpreting the empirical studies is that factors affecting LOC and those affecting MM are not explicitly separated. Another anomaly in using LOC/MM as a measure of productivity is that the majority of development effort is not coding [McKeen 1981, 1983]. If we are simply interested in the coding phase then LOC/MM may be reasonable, but many other types of people are involved in project development such as users, analysts, designers etc. Assigning productivity indices for these people is more problematical.

The ability to construct an accurate estimate of effort critically depends upon implicit assumptions about productivity. Even assuming the size estimate is accurate, productivity, and hence effort is influenced by many of the factors identified in Table 2. Currently, there are insufficient studies to accurately assess the effects of these factors on development effort.

As a side issue, activities included in the definition of effort can affect estimates. Brooks [1975] points out that while coding effort may be the easiest to estimate it

\[^{11}\text{Non-gender specific}\]
may only comprise 20% of the total system effort. Management effort, clerical support, maintenance, training etc. are significant activities to include. Comparisons among published studies, therefore, are confounded through differences in definitions.

Perhaps the most important idea to come out of Brooks' and Putnam's work is that effort and schedule are strongly linked. Total effort can rise exponentially as project schedule is shortened to less than some optimum time. The reasons for this are cited as coming from the communication load among project personnel and diminished productivity as stress levels go beyond some optimum point. The central idea is that projects must go through a natural gestation period. It is claimed that minimum effort, and hence minimum cost, may be achieved with the appropriate schedule. The theoretical reason for this has received little, if any, attention in the literature.

There is the implicit assumption in all estimating models to date that the factors which affect development effort are orthogonal. This may be due to insufficient data points to check for interdependencies. But, if we are to build more parsimonious estimating models, then redundant factors must be identified and removed to avoid double counting and confounded results.

As can be seen from the preceding discussion, many theoretical and empirical limitations exist with the various estimating models. A further review and critique of these approaches appears in Boehm [1981], Mohanty [1981], and Wrigley & Dexter [1987]. We turn now to an area of the literature which forms key components in later theoretical development.
2.5. SYSTEM SIZING

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. (see e.g. Gane & Sarson [1979], Warnier [1976], Jackson [1975], De Marco [1978,1982]).

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, 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 as follows:

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.

The fundamental assumption in using any of the code oriented sizing models is that a priori, a reasonable estimate of system size in LOC is available. This requires a
subjective estimate of project size to be initially input to the model. Statements such as:

"An initial study has determined that the size of the program will be roughly 32,000 delivered source instructions..." (Boehm [1981], pg. 63).

"It is usually reasonable to estimate a range of possible sizes for the system..." (Putnam [1979], pg. 356).

"Given that S (LOC) can be estimated..." (Kitchenham and Taylor [1984], pg. 82).

are not explicitly stated as the critical assumption. In the Golden et al. [1984] review of Putnam's model, it is conceded in a closing paragraph:

"One difficulty and unresolved problem with the use of this or any similar model is system sizing, that is, estimating lines of code. It follows that a good requirements and system specification is needed." (pg. 14).

An example of how difficult this sizing problem is comes from a report by Conte et al. [1986] 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.

The conclusion from the above discussion 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 & Caffney [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 thesis addresses the problem of obtaining size estimates based on system requirements, i.e. estimates based on inputs to the development process.

The various sizing approaches may be placed on a simplified version of the Waterfall model [Boehm 1981] of system development. It can be seen from this placement when information is available to make an estimate. As can be seen from Figure 2.1 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
Figure 2.1: Simplified Waterfall Model
and more productive way to characterize Bang is to choose one of the counts as a principal indicator..." (pg. 83, bold added).

Jones [1986], the leading proponent for the use of function points, reflects on this unresolved issue:

"As of 1985, software engineering is slowly emerging from the dark fog of lines of code measurements to explore new methods and new concepts of measurement" ... "...new software measures are starting to appear that come to grips with functions, with structural complexity, and with data complexity. Although these methods were developed independently by different researchers, new synergistic hybrid measures are starting to be explored, with significant potential values". (pp. 81-82)

One purpose of this thesis is to find the simpler, more parsimonious method that Jones and 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 (Figure 2.2). Much of business data processing can be characterized as data strong. This thesis takes the position that requirements complexity of these business applications can be captured in terms of data measurements. The development of Requirements metrics is critical if estimating is to advance on a scientific footing.

2.6. SUMMARY: THE MISSING LINK

While our assumptions about the distribution of effort over a system's life and the effect of development methods and tools on productivity are important, they are secondary to the central measurement question. The critical shortfall of existing software metrics is that they are measures of the product of development. As such, their
Figure 2.2: De Marco's Project Classification
usefulness as estimating metrics are limited. If metrics of the product of development are used, semantic information about the requirements is lost as these product metrics are just shadows of the underlying reality. The essential problem is this: If what we really want to do is estimate effort, then we must measure those things that cause effort. By using a metric of the end product as a basis for determining effort, we immediately introduce error in our estimate of effort. This is because our measure of the end product is itself an estimate complete with its own error margin. Therefore, to minimize this error we must estimate effort not on another estimate, but on an actual measure at the time the estimate of effort is made. An example will clarify this concept: If an estimate of effort is made based on an estimate of Lines of Code, then there are two sources of error between an estimate of effort and eventual actual effort. The first is the error margin between the estimate of LOC and the actual LOC, i.e. the amount by which the subjective estimate of LOC is different than the eventual actual LOC. The second source of error is introduced when LOC is an imperfect predictor of effort, i.e. when LOC is not perfectly correlated with effort.

To remove this inherent source of error in estimating, it is necessary to move from techniques and rules of thumb developed in practice to a more substantial view of the basic sources of effort in system development. This will occur by constructing estimates based on the real world entities, relationships and events that an information system models. The next chapter develops the general theory for this approach.
CHAPTER 3. THEORETICAL FRAMEWORK

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

3.1. FRAMING THE ISSUES

Building information systems is a production process. The prime inputs into any production process are labour (effort), technology and capital. The output from the system development process is a working information system. The resulting system can be measured in many ways, the easiest of which is to simply count the amount of software produced using some meaningful metric such as lines of code. Alternatively, user satisfaction or system reliability are increasingly common ways of evaluating systems. Although measures of the end product are clearly important, it is not the focus of this thesis. Instead, we will turn our attention to the inputs of the system development process.

The estimating problem is, inter alia, to determine the amount of effort\textsuperscript{1,2} that is

\textsuperscript{1,2}Effort is used in place of labour as effort is considered to be "professional labour".

33
required to produce the working system. While the production model is useful in understanding the building process, its usefulness in estimating is limited because the effort input is the independent variable. What we are after is a 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 3.1. The justification for this model comes from the analysis of Table 2.2 in Chapter 2 and from Wrigley & Dexter (1987):

The numbers in the columns of Table 2.2 represent the effects of each factor on system development productivity. What is important to note, however, is the direction of the effects. For all of the factors under "System Requirements", an increase in the factor is associated with a decrease in productivity and hence an increase in effort. For all of the factors under "Personnel Experience", an increase in the factor is associated with an increase in productivity and hence a decrease in effort. For all the factors under "Development Technology", an increase in the factor is associated with an increase in productivity and hence a decrease in effort. Moreover, since these three constructs are temporally antecedent to the development process this model suggests that these constructs causally affect system development effort. The face validity of this model motivates three assumptions which this thesis accepts as unproven premises, namely:

1. Holding personnel and technology constant, as the size of system requirements increase, required effort will also increase.
2. For a given set of requirements and level of technology, as personnel skill and experience (of both users and developers) increases, required effort decreases.
3. For a given set of requirements and level of personnel experience, as the use
Figure 3.1: Requirements, Personnel and Tools Model
of development methods and tools advances, required effort will decrease.

It may be argued that some interaction effects will exist among the independent variables. For example, the more difficult or larger systems may be developed by senior personnel. While this is most likely the case, it is presumed that any interaction effects are small in comparison to the main effects.

The construct which is hypothesized as the principal independent source of effort is system requirements. Clearly this construct is quite complex requiring further definition if it is to be useful. Therefore the remainder of this Chapter will focus on its operationalization and measurement. For estimating purposes, how can we measure system requirements? Before we start to answer this question it is first necessary to understand the process of building information systems and exactly what it is we are building. I will start by describing a few key concepts.

3.2. *KEY INFORMATION SYSTEM CONCEPTS*

3.2.1. System Development Transformations

It is generally accepted that building any machine artifact consists of a series of transformations which start from some conceptual reality in human thought, through various phases, into a working system. The demarcations between phases are not always clear but nevertheless must reflect an identifiable stable point in the process. For our purposes the stable points in the transformations from conceptual reality to working computer systems are defined as 1) requirements definition, 2) logical representation,
Theoretical Framework / 37

and 3) machine implementable code. The corresponding transformations are analysis, design and coding. The final transformation, from machine implementable code to the working system, is done by machines via compilers, linkers, translators, etc. This simplified development model is portrayed in Figure 3.2 (see also Wand and Weber [1987, 1988]).

For estimating purposes, what is important, and is a central premise of this thesis, is that there exist properties of a system's requirements that remain invariant over the transformations necessary to bring about the working system. Moreover these properties are measurable. If the working system accurately reflects its requirements, then the transformations have maintained a 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\(^1\). This premise of isomorphic transformation is a cornerstone in the theoretical framework of this chapter and will be expanded in a later section under Requirements Transformations.

Before doing so, the next two sections introduce requirements modelling concepts and then define an I.S. model.

3.2.2. Requirements Modelling

As described in Chapter 2, two separate but parallel schools of thought exist with respect to system analysis: 1) the Data Structure or Data Model approach and 2) the Data Flow approach. While both approaches are extremely useful for purposes of

\(^1\) For a more general articulation of this concept see Ashby [1956]
Figure 3.2: System Development Transformations
analysis and design, this thesis takes the position that for estimating purposes an integration of both is required. 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) in 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). Various extensions and interpretations of this model appear in the literature. For the purposes of this thesis the terminology chosen is stated by Atzeni et al. [1983]:

Three different classes of objects exist in the model: entities, relationships and attributes. Each object is represented and identified by an object name (entity name, relationship name, attribute name); moreover it may have associated both a set of synonyms and an explicative text in natural language.

*Entities* represent those classes of objects in the real world involved in the application. An elementary object within a class will be referred to as an occurrence (or instance) of an entity.

*Relationships* represent classes of logical associations between entities. An element of one of these classes will be referred to as an occurrence of a relationship. We use the term "entity" ("relationship") as a synonym of the term "entity set" ("relationship set"). The model can describe the cardinality (type) of each relationship, i.e. it can distinguish between 1:1, 1:n, m:n (binary) relationships. ...

*Attributes* represent properties of entities or relationships; an attribute is a mathematical function; the domain is the set of occurrences of an entity or a relationship and the codomain is a set of values. The set of attributes of an entity that uniquely identifies its occurrences is the key
The data model approach, however, is not without its limitations. The first, and less critical shortfall, lies in the modelling of entity-relationship connectivity with the use of mapping ratios. Many combinations of set mappings exist in reality other than the three mentioned above. Further, there is the implicit assumption that entity sets are connected to other entity sets via a single attribute. For estimating requirements size the simplistic mappings of 1:1, 1:n, n:m lose much information as to the real underlying entity linkages. This observation will become apparent when we try to predict processing complexity. The second, and far more serious situation, lies in the modelling of environmental changes, i.e. events. While a data model may capture the essential static complexity of a system, it fails to capture the dynamic aspects of a system, i.e. how the system responds to real world events. Since software processes are the dynamics of a system, the E-R model has a fundamental weakness in its pure form. It is, however, possible to describe events in terms of the attributes of the entities which are involved in the event. This concept will be discussed in greater detail shortly. When the E-R model extends to include events it becomes the Entity-Relationship-Event (E-R-E) approach to capturing both the statics and dynamics of requirements. With these terms defined, we can now describe what an information system is.

3.2.3. A Dynamic Information System Model

An information system is a representation of the attributes of entities and the inter-relationships of entities that exist in the real world. The state of these attributes
and object relations must be maintained if the IS is to preserve its "faithful representation" of the statics and dynamics of the real world. The more attributes an object has and the more these attributes can vary the more complex it is. When the attributes of an object change, this change can be considered as an event to the information system. These attributes, or, to keep later terminology consistent, data elements, must be originally specified to the system and then kept current as events occur in the environment.

Entities also have relationships with other entities. The knowledge of these relationships must also be specified to the information system in some way. When the relationship among objects change in the real world, this can also be considered an event to the information system. These events must also be recorded in the information system.

A data element is essentially a representation of some real world state, or the cumulation of events in the real world. Hence, a data element is a state or event tracker. Events and entities in the real world must be reflected in one or more data elements. "Code" is required to maintain these variables to ensure they parallel their counterparts in the real world. Code is needed to locate, retrieve, change, delete or add to instances or sets of instances of these data elements. The dependency relationships among variables introduce the need for additional logic. Retrieval requirements, or views of the data also introduce the need for processing logic. If this description of an information system is acceptable then it is important to incorporate the concept of transformations developed earlier to understand how the size of

\[\text{4}^\text{The system theoretic approach as articulated by Ashby [1956], von Bertalanffy [1968], Bunge [1977], among others and the more recent IS model of events, states and laws being developed by Wand and Weber [1987], turns out to be a useful vehicle for describing requirements size.}\]
requirements end up as complexity in processing.

3.2.4. Requirements Transformation

Restated, a central premise of this thesis is that the complexity of the software remains isomorphic to the complexity of its requirements. Specifically, statements in the requirements definition about events, entities and relationships that are to be modelled in the information system will be identifiable somehow in the machine implementable code. As an example, consider the English phrase, "customer buys part" (C-B-P), that would likely exist in the requirements statement of an online inventory system. It is basically a description of an event, that is of interest to the system. We parse the sentence into subject and predicate, yielding two nouns, customer and part, and one verb, buys. Interestingly enough this phrase can also be described in terms of entities, relationships and events. The two entities, customer and part have a relationship formed between them by the buying action. The entire sequence, C-B-P, is considered as the event. Here it can be seen how the E-R model could been extended to capture system dynamics by including events. In this light a working definition of an event is:

An event to the system is that which creates or destroys a relationship between two or more existing entity occurrences. An event also may create, destroy, or change one or more attributes of an entity occurrence.

For example, in the above system we may wish to add a customer entity occurrence. In addition, if we assume that the system is well designed, customers should be represented and identifiable as unique entity occurrences within the system. Likewise, parts will be represented as entity occurrences. If the isomorphism is to be preserved from the requirements definition to the machine implementation, then somewhere in the
software we would expect to find the C-B-P event fully articulated in the form of variables and processing. From a machine perspective (assuming current processor architecture) transformations can be expressed in the form: operand-operator-operand. It is conceivable that our C-B-P event could eventually be represented in this manner. The complete transformation from the original English phrase to machine form is shown in Table 3.1, where the items in the three righthand columns form the inputs into the system development phases shown in the lefthand column.

<table>
<thead>
<tr>
<th>Event:</th>
<th>Customer</th>
<th>Buys</th>
<th>Part</th>
</tr>
</thead>
<tbody>
<tr>
<td>Requirements</td>
<td>Noun</td>
<td>Verb</td>
<td>Noun</td>
</tr>
<tr>
<td>Design</td>
<td>Entity</td>
<td>Relationship</td>
<td>Entity</td>
</tr>
<tr>
<td>Code</td>
<td>Data</td>
<td>Process</td>
<td>Data</td>
</tr>
<tr>
<td>Implementation</td>
<td>Operand</td>
<td>Operator</td>
<td>Operand</td>
</tr>
</tbody>
</table>

If requirements specification contained only simple noun-verb phrases the life of system developers would be easy. Unfortunately, the real world places restrictions on whom and how many parts can be bought at any one point. This introduces the concept of control which must accompany the C-B-P description. The controls placed on events usually take the conjunctive "if" form in requirements. The "if" has a similar effect as a verb in that a relationship is defined between entities. This is equivalent to making the state of one variable dependent on another. Complex controls will result in complex logic, eg. validity checks, in software. The disturbance to the system, namely the C-B-P event, will require a series of transformations by the software to ensure that all checks and balances are carried out, i.e. to return the system to an equilibrium
state if possible.

Other examples of language forms which we would expect to have isomorphic transformational properties are:

1. **Has**: implies ownership or a relationship between two entity sets.
2. **Is a**: An entity occurrence is a member of an entity set.
3. **Adjectives**: attributes of entities
4. **Adverbs**: attributes of relationships

While it is beyond the scope of this thesis to even attempt to decompose the English language into its functional primitives, it is a central premise that, if done so, the structure provided would serve well as a requirements analysis tool.

The basic structure of these kinds of phrases should remain isomorphic through the several transformations necessary to go from requirements to implementation. Support for this basic position can be found in Halstead [1977] who found significant regularity in the structure of English prose and machine language. Therefore, the canonical form of a requirements definition, i.e. stripped of redundant prose, should provide a measure of the overall size of the information system to be implemented in software. For now it is sufficient to accept that the logical structure of the requirements, if measured correctly, can be used as an early measure of system size.
3.2.5. Requirements Complexity

Requirements complexity can be considered along a number of dimensions. With respect to an information system, complexity may be measured as 1) the variety in the classes of things that must be dealt with, 2) the level of interaction among its parts, 3) the uncertainty of event occurrences, and 4) the timing of system responses [Kottemann and Konsynski 1984]. While this list is not meant to be exhaustive, it provides a basis for understanding the nature of complexity.

In many senses complexity can be defined as anything which causes confusion in human beings or induces mental load. If a complex phenomenon is well understood, it may no longer seem confusing to those who understand it but will undoubtedly induce cognitive strain on those who don't. For our purposes though, a complex phenomenon requires effort to understand it, define it, and give it structure.

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 transformations 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 relationship will be causal.
3.2.6. Requirements Sizing

In the Construction industry estimating practices are well developed. In the case of buildings, the metric most commonly used to estimate cost is square footage. A reasonable estimate for the cost of building a house can be obtained simply by multiplying the desired size of the house, in square feet, by some quality (luxury) index per square foot. In 1988 the standard residential house can be built for around $60 per square foot.

The problem we face in IS development is that there is no similar standard metric, such as square feet, which can be used early in the IS planning process. If we consider that an information system somehow maps onto the organization which it supports, then the area of the organization which is to be computerized could be used as a basis for estimating system development costs.

3.2.7. Square footage

Assume that a comprehensive E-R-E diagram can be constructed for a business firm. This diagram would represent the entire organization's knowledge or data base. It could be considered as the organization's data map. The area on this map is referred to as "square footage". Further, it would also capture the organization's linkages to its environment. These linkages, or relationships with its environment can be used to capture the events of interest to the firm.

Recognize that the E-R-E diagram is used as a vehicle, albeit imperfectly, to represent knowledge. It is hoped that the approach suggested in this thesis will be extendable to any knowledge representation scheme.
In a well designed business there should be concentrations or clusters of entities and relationships with high connectivity. These clusters should correspond to our notion of a business function or department. There should also be a simple minimal linkage between these clusters. Minimal linkage would be where a single entity in one cluster is related to a single entity in another. A simple linkage would be where the mapping ratio between the two is simple, e.g. (1,1) or (1,*). The above description is similar to our intuitive notions of loose coupling and high internal cohesion.

Hence we should be able, hypothetically at least, to draw a circle around a cluster of entities on a business' data map and have those entities represent some logical business unit. We can then think of a system development project as computerizing or implementing a small portion of the business' data map. The entities, relationships and events within the implementation circle give us an early indication of the "size" of the requirements.

3.3. PROCESSING COMPLEXITY

Initially, we will deal with the processing complexity of a relatively simple transaction processing information system. These types of systems are the best understood by all and have the best empirical data available for model validation. The objective of the discussion below is to show how one could start to measure the size of a system's requirements. The principal assumption is that the source of all processing requirements come from events that occur in the system environment and the complexity of the data structures to which those events are related.
3.3.1. Processing complexity based on E-R-E concepts

Jackson [1975], Warnier [1976], De Marco [1982] and others, have suggested that a well structured function should match the logical data structure on which the function operates. De Marco [1982] explains:

"The reasoning behind Warnier's observation is this: All decisions in the code are based on the data processed by the code; the structure of that data - that is, the associations among its component pieces gives a strong hint as to how the code could be written, eg. if there is a repeated substructure in the arriving data there should be a loop in the code to deal with it. If there is an option in the data, i.e. a field that may or not be present, then there will have to be an IF-ELSE sequence to deal with the two situations." (pg. 107)

The extension of the above reasoning is that processing performed by a program is proportional to the complexity of the data at the program interface. If this proposition is true then by measuring the size of the inputs and outputs to a system as a whole (based on requirements) then we can approximate the size of the processing task. The theory, which explains the above practitioners's observations and a major premise of this thesis, 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. The key, then, is to capture the essential structural complexity of the data at the system interface.
3.3.2. Input Events

When an event occurs in the environment, it is usually presented to the information system in the form of a transaction. This transaction contains a number of attributes which are logically related to one or more entities and their attributes in the database. A first step towards the measurement of processing complexity would be to simply count the number of system entities referenced by the transaction. Depending upon the specific implementation, this processing may be more or less difficult. But the knowledge of which system entities must be referenced is contained within the overall logical data structure (E-R-E chart).

The E-R-E diagram also contains basic information regarding the linkages among the entities. These linkages should provide clues as to which entities will be referenced by a transaction. The complexity of these linkages leads to potential processing complexity required to record an event and also to extract the information once stored.

3.3.3. Output Events

Recording the business events in the database is only one part of the processing required. Presumably, decision makers will want to know about the event at some point in time and at some level of aggregation. This introduces retrieval requirements. For each entity there is at least one simple retrieval processing requirement. For example, an inquiry into a customer's account status. There will most likely be more complex ones such as relating one entity set to another. The amount of processing logic for information retrieval will depend upon the actual number of entities that the
Determining processing difficulty for retrievals presents special problems. For the same E-R-E diagram, it is possible to have two sets of inquiry requirements which vary greatly in complexity. The issue centers around deciding on how much functionality to give the software. For now, it is sufficient to assume away the problem by defining which queries or reports are to be supported and which ones are not. In general, however, a retrieval can be compared to an input transaction which has to access specific entities. We can consider a query request by a user as another event to the system. The only difference is that the query is an extraction request. A retrieval event will still have to access various entities in the database in order to collect the necessary data to satisfy the query. The processing complexity introduced by a query event could be counted in the same way that an update event would be counted. However, a problem arises when we try to determine the effect of the systems' internal data structure complexity on processing. Halstead (1977) established that there is a relationship between the number of operators plus the number of operands and program length (and hence effort required to implement a specification). If we consider that operands are actually the data that a program must operate on, then by counting all data that a program accesses, we should be able to estimate program length. The problem is that we may not know the number of data elements in a data structure that will actually be used in a program. For example, a program accessing a data structure containing 20 elements may actually use only 5 of these elements. According

---

An interesting point emerges from the concept of unimplemented queries. These potential queries may provide a clue to the huge maintenance costs experienced in the software industry today. "Creeping functionality" and new releases may be no more than just implementing new portions and providing access to the overall E-R-E diagram that was discussed earlier.
to Halstead, program length is a function of 5 not 20. Two programs may access the same entity during event processing yet may differ in size depending upon the actual elements referenced. It may, nevertheless, be possible to "estimate" the number of elements to be referenced by considering the "class" of program being written. For example, in an on-line update program it is likely that data manipulation will occur at the element level. This implies that a high percentage of accessible data elements will actually be used. Alternatively, in a file interface program individual elements are less likely to be manipulated. To summarize, knowledge of the function of a program may give us insight into the kind and extent of data manipulation that may occur.

3.3.4. Functionality

There are some basic functions which we would expect an information system to provide in order to manipulate the entities and relationships in the system. The level of functionality or capabilities provided by the software can vary greatly but can at least be specified very early in the development process. Some examples of these are:

1. Entity occurrences will have to be added, deleted and have attribute values changed.
2. If an entity is added then we need to know certain attributes before the entity is allowed to exist in the data base.
3. A single or group of entity occurrences must be locatable and information provided to users of the system.
4. A listing of entities in various logical sequences should be available.

The point is that all of these processing requirements are common to all entity types. Only the specific attributes should affect the amount of processing logic to be specified. This kind of processing logic may be referred to as "entity maintenance". Each entity will likely require a menu screen for adds, changes, and deletes. Within
each menu item another screen will be required to handle all the attributes along with the necessary code to ensure data validity.

The most important aspect of the E-R-E diagram is that it can be one of the first documents produced during analysis. This thesis assumes that if calibrated correctly E-R-E diagrams can be used as the basis for estimating further work.

3.4. E-R-E AND FUNCTION POINTS

If the E-R-E approach to estimating is to be useful it must be usable earlier in the development process than current techniques. From the above discussion we can begin to see where function point estimating fits in. For example, each entity will require a number of screens for "maintenance" purposes. However the number of screens are just shadows of the underlying data complexity. It makes more sense to focus on the entities and events. By measuring entities, relationships and events we can obtain a reasonable approximation to the number of function points that the software will have to provide. The following table shows how the E-R-E is the more general case of the popular Function Point approach to estimating.

<table>
<thead>
<tr>
<th>Entities</th>
<th>Relationships</th>
<th>Events</th>
</tr>
</thead>
<tbody>
<tr>
<td>logical internal files, external files</td>
<td>logical internal files</td>
<td>input transaction types, external outputs, inquiries</td>
</tr>
</tbody>
</table>

It can be seen from the above Table that the E-R-E approach is also more
Theoretical Framework / 53

parsimonious than the Function point approach. The reason for this is that a detailed
evaluation of Albrecht's approach reveals at least two redundant measures. Demarco
[1982] observed this same phenomenon in his own attempt to estimate system effort
as a function of his Bang metric. His "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..." (pg. 83, bold added).

This thesis has taken entities, relationships, and events as the principal indicators of
requirements size. These concepts are now integrated into a theoretical model.

3.5. FORMALIZING REQUIREMENTS SIZE

3.5.1. Events, States and Transforms

The ideal situation is where we have complete information about the events and
inter-relations that exist at the system boundary. If it is assumed that we have been
able to accurately represent a system's requirements in terms of events at the system
boundary, entity sets, known relationships among entity sets and entity occurrences, and
controls or laws which specify allowable combinations of one or more events, then it
is possible to describe the statics and dynamics of a system with the use of set theory and matrix notation (see e.g. Bunge 1977). The discussion below represents an initial approach towards rigorously specifying the concepts developed previously in this chapter.

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:

$$E = [e_1,...,e_n]$$

The event vector, $E$ is comprised of two sub-vectors, $E^I$ and $E^O$ 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 values of entity attributes 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 $E^I$ and $S$ are the inputs into the matrix $T$ which transforms $E^I$ and $S$ into the new system state vector, $S'$, and possibly creates new output events at the system boundary $E^O$. The transformation $T$ is what we usually refer to as code. This model of an information system is shown below in Figure 3.3.
$E^i$ = input event space 
$S$ = system state space 
$E^o$ = output event space 
$S'$ = new system state space 
$T$ = process transformation
If the assumption regarding isomorphic transformations holds, then we would expect that the size of the transformation $T$ reflects the size 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 3.3 we can see that a first approximation of requirements size 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 data elements involved 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\textsuperscript{17}.

To better understand the potential complexity of $T$ it is helpful to project forward to a

\textsuperscript{17} 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.
fully implemented information system. At system start-up $S$ is comprised of a number of defined sets, $s_1...s_n$. Some are empty; these will be used to track events over some time period, e.g. a transaction file. Some are given members by humans at system initiation. These initial values can be considered as master files reflecting the current state of the environment, e.g. opening balances in asset accounts, existing clients, etc. The processing of the transaction events against the master files naturally keeps the later sets concurrent with a system's environment. What are important, however, are the linkages, i.e. the mappings among these sets.

Consider two subsets of $S$, $s_1$ and $s_2$, within a system. The sets $s_1$ and $s_2$ are mutually exclusive if any event, $e_i$ occurring in the system's environment requires that either $s_1$ must be modified in some way (e.g. add, change, or delete an element) or $s_2$ must be modified, but not both. No effort is required by humans (or machines) to record the event in both places. An additional requirement for mutual exclusivity is that the state of set $s_2$ in no way influences the possible changes to set $s_1$.

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 size of an individual input event can now be
defined as the number of set changes in $S$ that must be made given the event $e_i$.

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 size of an event $e_i$ can be defined as the square of the number of sets involved in the event.

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 $\hat{E}'$, $\hat{E}^0$, and $\hat{S}$. Essentially, these estimates represent data flows to and from the system processes and can be used to estimate $T$. This estimate of $T$ is referred to as $\hat{T}$.

3.6. SUMMARY

This chapter started out with the proposition that requirements size is the driving force behind the effort to build software. It claims that requirements size can be estimated early in the system development life cycle 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 that must be drawn from this chapter is that software processes are driven by the data. Without data there are no processes. While it is

\[\text{18}^\text{An issue which may be raised at this point is that a "good" design would minimize the number of set changes, possibly to one. The position of this thesis is that even with the "best" design, complex events will affect more than one set.}\]
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.

This chapter has 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. It has shown how measurements of requirements are available earlier in the development process than other measures such as Function Points and Lines of code. It has also shown how they are more parsimonious than these existing estimating approaches because they occur at a higher level of abstraction. What remains to be seen is if these measures of system requirements size are better, more reliable predictors of effort. To address this question, it is necessary to move away from theory into the real world where it can be investigated empirically. The remainder of the thesis is dedicated to this task.
CHAPTER 4. EMPIRICAL RESEARCH DESIGN

The objective of this chapter is to move from theory into the empirical domain. The preceding chapter has established the theoretical justification for using measures of requirements size as an early measure of system size. The purpose in doing so is that transforming system requirements into a working information system is believed to be the principal driving force behind effort. Three issues emerge from Chapter 3 which together form the purpose of the remainder of the thesis. The first is the extent to which measures of system requirements can explain design size. The second is the extent to which design measures can explain process or code size. The third is the extent to which effort can be predicted from any of these sizing metrics. Therefore the empirical portion of this thesis consists of three distinct steps: The first step is to establish the relationship between measures of $\hat{E}$ and $\hat{S}$ available at the design phase and process size, $\hat{T}$. The unit of analysis in this step is the individual program as programs can be considered as design decisions by developers. The second step is to then establish the relationship between requirements size and design size. In this second step the unit of analysis moves to the system level because during analysis it is the system which is the conceptual entity of interest. The first two steps are carried out in reverse order of system development to ensure that the source code is an accurate reflection of design and that design is an accurate reflection of requirements. Finally, once this linkage is established it is then possible to introduce the resource consumption variable and relate units of requirements and code size to units of labour.

This chapter proceeds in the following manner. In the first section, a research model graphically presents the concept of requirements size causing effort at both the
conceptual and empirical levels. By combining Figures 3.1, 3.2 and 3.3 from Chapter 3, this basic model expands to include the theory of requirements transformation and the corresponding sources of effort. This section also explores issues of construct validity and reliability and further defines the specific metric linkages, connecting measures of requirements with measures of a working system.

The second section presents a general strategy of how the research model may be empirically tested. This strategy calls for: 1) the systematic measurement and evaluation of working information systems. This is achieved by reverse engineering existing systems back to their design and requirements specifications and 2) construction of a software metrics database for model testing. To more fully articulate the concepts of entities, relationships, events and process transformations this section includes an example system which was reverse engineered manually. The issue of reliability and computational tractability identifies the need to automate the expert code analysis process. The design, construction and implementation of such an automated tool is then presented along with the measures for input to the derived metrics database. Finally, a general regression model uses the measures identified in the example system to explain process size, $\hat{T}$.

4.1. RESEARCH MODEL

Research into software engineering and the system development process in general, has suffered from the lack of a nomological net, i.e. the theoretical connections among a set of concepts or constructs in which to discuss empirical phenomenon. Further, substantive research in software engineering has been carried out in the absence of
construct validation research. Scientific knowledge of the system development process will be advanced, and our ability to estimate resource consumption, if and only if: construct valid relationships are established between the system desired and the one defined for development, and between the design and coding effort expected and actual work hours expended. Furthermore, the relationship from our defined requirements to the amount of resources consumed in the process, must then be empirically validated.

Schwab [1980] dichotomizes research into two separate phases of 1) construct validation research and 2) substantive research. Schwab defines construct validity as "...the correspondence between a construct (conceptual definition of a variable) and the operational procedure to measure that construct" (pp. 5-6). In contrast, substantive research refers to research that attempts to establish "...relationships between independent and dependent variables" (pg. 4).

One depiction of both the theoretical and empirical relationships among the independent and dependent variables of interest appears in Figure 4.1. This figure is interpreted as follows. At the conceptual level there is a causal relationship between the size of the requirements of the desired system and the amount of resource consumption necessary to implement the requirements, mitigated by two factors: kinds of tools used and the experience or skill level of the system builders. These effects are depicted as F. The system requirement size construct is further defined as including both system statics and system dynamics (S,E). In Figure 4.1 the independent variable, I, size of system requirements, causally affects the dependent variable D, which is the amount of resources consumed in the process. It is important to realize that the
Figure 4.1: Research Model
constructs at this level cannot be measured directly.

At the empirical level, the two components of I, system statics and dynamics, are operationalized as entities and relationships, and input events and output events respectively. In Figure 4.1 the measurable system requirements are labelled as I'. Similarly, the dependent variable, resource consumption, D, is operationalized as the number of work hours, D', expended to build the system. It is the covariation between our operationalization of a system's requirements, I', and the measures of effort to produce the working system, D', that is of substantive interest. Finally, two major factors affecting the amount of effort expended, level of tool use and level of personnel experience, are represented by the F' arrow. In this figure these effects have not been specifically operationalized. The model presented here is a general one which may be operationalized within specific empirical domains in future field work.

Referring to Figure 4.1 again, the scientist emphasizes the vertical relationships between both the desired and actual requirements and between the resources consumed and work hours. If measures of both constructs are valid, then the empirical relationship measuring the actual requirements and effort expended provide valid research. The practitioner, however, is satisfied if there is construct validity only on the vertical relationship between the measured dependent variable, i.e. the resources consumed and the work hours. Here the relationship between measured requirements and measured effort is enough to satisfy considerations of managerial control of the development process.

Both practitioners and academics have an interest in construct validity. In this case
systems developers are interested in accounting for the variance primarily in the dependent variable. In other words they are interested in predicting the amount of effort to develop a given system, i.e. their interest is in the relationship between \( l' \) and \( D \) of Figure 4.1. Thus the difference in approaches between academics and practitioners lies not on the choice of the independent and dependent variables, but rather on the amount of emphasis placed on the validity of the construct measures. While academics place equal value on the validity of the measures of both the independent and dependent constructs, the practitioner places more emphasis on predicting or estimating the size of the dependent variable, in this case the amount of resources consumed in building an information system.

4.1.1. Detailed Research Model

Figure 4.2 presents these concepts of construct validation and substantive research into a more useful context and establishes the detailed linkage between requirements size and effort.

To understand this detailed linkage it is useful to review the discussion on the process and the product of system development developed in Chapter 3. The process of systems development is conceptualized as consuming resources during: analysis of information requirements, design of specifications, and development or coding of specifications into an operational system. Each effort area takes as input some definition of the problem space, or size, and converts this definition into another representation form. The working IS is the result of transformations from each of the previous phases (see eg. Kottemann and Konsynski 1984). These transformations take conceptual reality
Figure 4.2: Detailed Research Model
Empirical Research Design / 67

in the minds of users through analysis, design, coding and a final translation into machine representation via the use of compilers or interpreters. Each of the transformations, except the last one, require human effort to understand and structure the problem space.

If the principle of top-down decomposition and stepwise refinement is applied, then each of the units at one level will subsequently expand to many units at a lower level. At the analysis stage of system development, the units that are available to count consist of the real world objects that are to be modelled within the information system, as well as knowledge of the events affecting those objects that will occur at the system boundary.

In Figure 4.2 both the constructs and their respective empirical measures can be observed. At the conceptual level we are interested in the size of: the system requirements, the logical design, and the working system. Construct validity between all three of the depicted vertical relationships is necessary to make scientific progress. Here, the static and dynamic properties of system requirements are operationalized as entities and relationships, and input events and output events respectively. \( I_1 \) and \( I'_1 \) correspond to \( I \) and \( I' \) in Figure 4.1. \( D_1 \), the output of the design phase and \( I_2 \), input to the coding phase are equivalent as are their operationalizations \( D'_1 \) and \( I'_2 \). In this figure \( I_1 \) is shown to cause \( D_1 \) through the intervening variable \( IV_1 \), effort. The size of the logical model represents the result of the transformation from system requirements. It is theorized that the size of this logical model maintains the size and complexity of the initial requirements through the design phase. Furthermore, these properties hold through the coding phase which eventually produces the size of the
At the empirical level $I_1$ is operationalized as discussed above. $D'_1$ and $I'_2$ are operationalized as the static and dynamic properties of the logical model. These measures are analogous to function points. For simplicity, $D'_2$, the size of the working system, is measured as $D'_2$, source lines of code. Alternately, Halstead [1977], McCabe [1976] or other program complexity metrics could be used at this point.

The reason that the mapping between measures of requirements and measures of code is important is that the relationships among the units are traceable. This discussion started by claiming that there is a causal relationship between size of requirements and resource consumption. However these units are not functionally comparable. It is necessary first to establish the properties of requirements which cause the production of source code. Once the functional relationships between requirements and code are established, then units of requirements can be used as predictors of development effort.

4.1.2. Metric Linkages

The mapping between requirements and code is shown in Figure 4.3. While Figure 4.2 simply showed a single empirical link between the I's and D's, Figure 4.3 shows in greater detail the specific measurable linkages from requirements through to source code. Figure 4.3 depicts how the operational measures of size are functionally related. Prior research has focused primarily on measurements at the design and code level (see e.g. Conte et al. 1986), and has developed empirical relationships between these measures and the eventual code of installed systems (Albrecht 1979; Albrecht and
Figure 4.3: Metric Linkages
Gaffney 1983; Rubin 1983, 1985; Jones 1986; Symons 1988). The empirical findings, discussed in Chapter 6, are the result of reverse engineering the code back to the design metrics shown in this figure. The regression equation then uses these counts to predict code size.

In order to map backwards from function point metrics to requirement metrics it is necessary to introduce semantic information into the reverse engineering process. If we assume that system designers decompose a requirement specification into processes (programs, modules) that deal with or hide some aspect of the specification (see eg. Parnas 1975), then it is reasonable to classify a system's programs according to the function that they perform. If we further assume that the basis for functional decomposition reflects a data flow orientation to design, typical of transaction processing systems, then we may conclude that events, identified as requirement units, would require separate processes to handle the dynamics of each event.

The information necessary to determine a program's function is embedded in its source code. At the program level certain language keywords and patterns of variable usage provide clues to the function of the program. For example, a program which has a high content of screen I/O and file manipulation statements will most likely be an online transaction capture program. Once classified correctly, semantic inferences may be made as to the number of entities, relationships and events that exist in a requirements specification by looking at the distribution of programs by program class. The proposition is that this information can be used to predict the amount of code required to implement a new requirements specification.
A limitation of the empirical portion of this research is that it deals primarily with the relationship between design and code size, i.e. at the program level. The extension to include the relationship between requirements and design is theoretical at this point and not fully operationalized. Future studies with larger data sets will be able to fully operationalize these concepts and test this theory.

4.2. SYSTEM DEVELOPMENT MEASUREMENT AND EVALUATION

Previous discussion laid out the conceptual framework for this study. Now we turn to the task of testing the model empirically. To achieve this, a general strategy for collection and analysis of field data is necessary\textsuperscript{19}. A top level data flow diagram of system development measurement and evaluation appears in Figure 4.4. The Systems Development process bubble is seen as consuming the resources of system developers and users, producing the working systems while generating a number of Project Details such as amount of resource consumption, personnel involved, tools and methods used and other demographics. The System Development Process bubble produces working systems as tangible outputs consisting of source code and data definitions. Two measurement process bubbles are required to produce the Derived Metrics database: The Expert Code Analyser and the Project Detail Extraction. The Expert Code Analyzer process bubble, either automated or manual, takes as input completed working systems and generates a number of software metrics shown later in Figure 4.12. From the perspective of this process a system is defined as a collection of integrated programs and their corresponding data definitions that, together, have an identifiable function.

\textsuperscript{19} Several authors have identified and discussed the general need for a software metrics collection and analysis group within a system development environment and research is underway to automate the entire process (see e.g. Basili and Rombach [1988], and De Marco [1982]).
Figure 4.4: System Development Measurement and Evaluation
within a company. The output from this process updates the Metrics database. Project Parameters can be extracted by either an advanced project control system or, in its absence, trained research personnel. Finally, the Metric Analysis consists of statistical routines which provide management with information which improves their understanding and estimating capabilities within their own development environment.

The advantages of reversing the code over using existing documentation lies in the accuracy of the documents. In the field it is common for documentation to lag the software. For example, added requirements may not be reflected in the requirements document. From a theoretical perspective it is useful to think of reverse engineering as reverse transformations. Hence it is assured that the analysis and design documents are true representations, i.e. without modifications, of the final working system, subject only to the interpretations of the researcher. The reversing operations, however, are objective and reproducible. Details of the rules applied to reverse engineer the code are contained in the Code Analyzer, discussed later.

4.2.1. Calibration

Calculating regression co-efficients from the reverse engineered systems is just the first step towards being able to predict resource consumption early in the development process. Once relationships have been established between requirements or design metrics and code for a given environment, it is then possible to introduce effort and other project factors into the estimation model. This is achieved by extracting resource consumption data from the project details and regressing these units against requirements, design or code units. Hence, the resulting regression weights are
calibrated to a specific system development environment consisting of a unique mixture of people skills and applied technology.

4.2.2. Measures of $\hat{E}$, $\hat{S}$ and $\hat{I}$ From the Code

From Figure 4.3 above it is possible to define a number of specific measures for events and system statics. These appear below. Following these is the measure of process size used and the justification for the measure.

4.2.2.1. System Dynamic Measurements: $\hat{E}$

1. Inputs: $\hat{E}^1$
   a. Input events = screens
   b. Input event size = screen variables

2. Outputs: $\hat{E}^0$
   a. Output events = reports
   b. Output event size = report variables

4.2.2.2. System Static Measurements: $\hat{S}$

1. Entities = files accessed
2. Entity size = fields available
3. Relationships = temporary files used (projections) +
   file cross references (joins)
4.2.2.3. Process Measurement: 

An issue raised in Chapter 2 is that code may be counted in various ways. The most common method is to count text lines of source code while ignoring comment lines [see Boehm 1981]. However, this approach is not without its drawbacks. Even within the same language a line of code may vary greatly with respect to the amount of work or function performed. For example a nested "IF" statement is more difficult to code than a simple assignment statement. Ideally, to control for these variances in individual lines, code should be broken down into its operator and operand symbols in order to obtain a more rigorous measure of program length. However, without the automated code counting program this approach is not computationally feasible. Instead, the more crude but commonly accepted method of counting text lines was used as a measure of process size for the pilot data. The development and testing of a code analyzer to alleviate the counting issue is the subject of a following section.

4.3. EXAMPLE SYSTEM

An example system will be used to describe the concepts developed above and in Chapter 3. This system has not been artificially created: it is a system which is currently in use within a company. It was not chosen because it reflected the theory well and as such some portions of the theory may not be represented. Rather, it was chosen for no other reason than its name starts with 'A' which placed it on top of the field sample pile. The source code from the company was obtained from its

See Halstead [1977] for definitions of program length, program volume, and program information content. Also see Fitzsimmons and Love [1978] and Shen et al. [1983] for independent validations of these measures.
production library which contains the current versions of all its working systems.

The example below begins with a system's requirements statement and traces the system's development down to the process level. The system's requirements contain the basic specification of system level entities, relationships and events. These are presented graphically. Then, the system's menu structure is presented using a Warnier-Orr type process hierarchy. Next, the process hierarchy is analyzed using data-flow diagrams (DFD's) which are exploded down to the lowest process level until each process bubble represents a single program. At that level it is possible to accurately count the software in various ways. In the conclusion to Chapter 3 it was stated that the independent variable for predicting process size is its data. Therefore, data flows are counted in and out of each program. Two distinctive data flows are counted. The first represent data flows across the human-machine boundary, i.e. attribute values of events occurring at the system boundary. The second represent the flows between each process and the system's static space, i.e. between the program and system storage.

4.3.1. System Description

The Asset Disposal System (ADS) is a relatively small menu-driven transaction processing and management reporting application system. Its function is to maintain inventory details on assets which the company has decided to sell as well as to record information about their sale when eventually sold to customers. There are two separate classes of assets about which the system must maintain information, capital assets and scrap assets. The first major event of interest to the system occurs when management decides to sell an asset. Information about the asset is then recorded into the system.
by a user via an online terminal. Further, decisions are not irrevocable, i.e. management can change its mind and decide to remove any asset up for sale or change any details on assets already up for sale. The second major event of interest is the sale of an asset to a purchaser. Sales events require the recording of the asset(s) involved in the sale and information about the purchaser and financial terms of the sale. Finally, a number of management reports are required which are available to users by selecting from menus presented at an online terminal. In total, 13 separate reports are requested by management. These reports range from simple listing of transactions (events) to sales and profit cross-tabulations.

The analysis of the above system requirements produces the first document, Figure 4.5, depicting the major entities, relationships and events involved in the system. Reporting requirements are not shown. In Figure 4.5 the two classes of assets are represented as the entity sets Capital Assets ASDX0001 and Scrap Assets ASDX0002. The decision to sell events are depicted as the relationship created between management and the company’s assets. The decision to not sell an asset, i.e. to reverse a decision, is represented by the 1:n relationship between management and the Asset entity sets. This is equivalent to stating that an asset may be involved in more than one event. Sales events are shown as the relationship created between a customer and one or more assets or asset classes via the entity set Sales Orders. Each sales order is made by one customer but each customer may be involved in many sales events. Each sales order may involve one or more assets from either asset class. These relationships are represented by the "includes" diamonds. The reporting requirements consist of extracting information from the three entity sets in various combinations. The ADS system can be

---

21 For the remainder of the example a numerical or capitalized reference specifies the actual name assigned to the system object.
summarized as having three kind of input events: two decisions and one sales, three entity sets: two asset classes and one sale orders, and thirteen output events: reports which convey information about all entities and relationships that exist within the system. It will be seen how each entity, relationship and event will be implemented as either a file, a data flow, a process, or user interface.

In Figure 4.6 the process structure is shown. Each line contains a numerical reference and description of a program. The first screen a user sees is the Main Menu. From this menu the user may select one of eight alternatives. The first three programs in Figure 4.6, 1100-1130, deal with the three events identified Figure 4.5. The remainder of the menu selections allow the user to choose which reports are to be printed out and where. Menu item 4080 in Figure 4.6 executes four programs in sequence producing two reports without further user interaction. Menu item 1020 produces a report sub-menu from where a user can select one of nine different reports. Each report selection executes a single program which may prompt a user for further report parameters. The linkage between these programs and the system files will now be shown in the DFD's.

Figure 4.7 contains the top level view of the system from a data flow perspective. There are six nodes in this diagram. The first box, User, represents the immediate system environment. The second node, process bubble 1000, contains the 21 sub-processes identified in Figure 4.6 and corresponds to the transformation matrix $\mathbf{T}$ in Figure 3.3 from the last chapter. The line between the User and process bubble 1000 represents all data flowing across the man-machine boundary. All input and output events occur across this system boundary which is depicted as a weighted arc between
Figure 4.6: Asset Disposal System Processes

Main Menu 1000

1100 Online Maintain Cap asset Detail

1110 Online Maintain Scrap Asset Detail

1130 Online Maintain Sales Orders

4080 Batch Reports

4100 Create Temp file

4110 S.O. Report Cap.

4120 S.O. Print Scr.

4130 Reset Flag

4140 Discrepancy Report

4160 Cap. Asset Status Report

9999 Set Printer

1020 Report Sub-Menu


4020 Disposal Income Rpt.

4030 S.O. Summary Rpt.

4040 ASD Status Rpt.

4050 Used Tie Sales Rpt.

4060 Unsold Asset Rpt.

4070 Scrap Sales Rpt.

4090 GL Account Rpt.

Figure 4.7: Top Level DFD

User

3,13,682

1000
Asset Disposal System

Capital Assets
ASDX0001

Scrap Assets
ASDX0002

Sales Orders
ASDX0003

S.O. Print Temp
ASDX9001
the two nodes. This arc corresponds to the \( E \) vector from Chapter 3. Three numbers define this arc. The first indicates that 3 input events occur at the boundary. The second indicates that 13 output events, i.e., reports are generated by the system. The third, 682 is an actual variable count of this flow. It includes all screen and report variables and no distinction is made at this point regarding direction of flow. Although, in general screen variables tend to be input to the system and report variables necessarily come out of the system. This value will play a key roll later when estimates of process size are made. In brief, it is our first good estimate of system dynamics.

The remaining nodes in Figure 4.7 are the system's data stores. These correspond to \( S \) and \( S' \) from Chapter 3. They also correspond to the three entities Capital Assets, Scrap Assets and Sales Orders. A fourth data store, Sales Order Temporary Print file, is used by some of the report programs as a buffer and is included primarily for the sake of completeness.

When the process bubble 1000 is exploded down to its next level, Figure 4.8 emerges. Each process bubble in this diagram represents a program which is accessible from the Main Menu program. As in Figure 4.7 the weighted arcs between the User boxes and each process represent the data-flows across the man-machine boundary, although these numbers now represent "micro-events" at the program level. For example, the "3,0,53" arc flowing into program 1100 should be interpreted as 3 screens, 0 reports, and 53 data elements. There are three on-line update programs available in the system. It is not surprising to learn that each update program corresponds exactly to each of the major events described in Figure 4.5. The added detail that this diagram provides is information regarding which data stores are accessed by each program and the size and direction of each data flow into each process.
Figure 4.8: Asset Disposal System - Main Menu Processes
Figure 4.9 contains the detail of the Reports Sub-Menu 1020 process. Where the User has some control over a program’s execution the connecting arc has a double headed arrow. The weighted arcs across the system’s boundary again represent the number of variables transferred between the system and its Users and the "micro-events" that occur. Figure 4.10 contains the processes executed from the batch report main-menu item.

Finally, Figure 4.11 has been included to show a subtle but important aspect of estimating process size. Within process 4160 a temporary hold file is created to reorganize the data coming in from the other data stores. This adds additional code which is not apparent from a higher level of abstraction. The added information of temporary files comes only after significant design work has been carried out. Hence it is expected that given this information an estimate of process size would increase in accuracy. This is indicative of the iterative nature of estimate refinement.

4.4. AUTOMATING THE EXPERT CODE ANALYSER

In order to establish unbiased, reliable measurements from the systems, it is necessary to solve the problem of the relative expertise among the measurers. It is believed that the problem of measurement reliability is solvable by the use of an automated code analyzer which can be used to reverse engineer installed systems.
Figure 4.10: Batch Report Run 4080
Figure 4.11: Asset Status Report 4160
4.4.1. Reliability

While Figures 4.1 and 4.2 have provided an overall framework for research, the issue of measurement reliability is central to the estimating problem. Reliability refers to the extent that various measures of code, or design function points, are consistently measured by one researcher or analyst to the next and from one system to the next. Problems with function point analysis were succinctly reviewed and articulated by Symons (1988) who indicated that the very process of measurement was fraught with difficulties.

"The method is not as easy to apply in practice as it first appears... For some time to come therefore it will be best in any one organization if all measurements are supervised by one objective, experienced function point analyst. Such an analyst should accumulate and document cases and derive general rules ... which will help ensure consistency and objectivity in function point analysis. (pp. 9,10)

Briefly, two major criticisms can be leveled at function point analysis:

1. It is unlikely that the function point weights and complexity adjustments are generalizable outside of a particular project data set. No theory has yet been proposed as to why a particular complexity scale is appropriate. This leads to the conclusion that all scales used in function point analysis are derived from empirical relationships found in a specific project database.

2. Assessing the complexity level of each major function type and determining overall processing complexity adjustments is subjective and may vary substantially from one analyst to the next.

To overcome these shortcomings one alternative is to first measure each individual system development environment to ascertain specific definitions of low, medium and high complexity. This may be achieved by obtaining a distribution of, for example, the number of data elements on each input screen. Low complexity could then be defined as the bottom third of the distribution, while high complexity may be defined over the
top third. However, with the use of automated methods it is unnecessary to categorize function types as low, medium or high as the continuous regression co-coefficient can be used directly. This would remove some of the inherent subjectivity involved with complexity classification.

4.4.2. The Automated Code Analyser

It became apparent very early in this research project that manual analysis of completed systems was time consuming, tedious and prone to researcher error. This motivated the design and development of an automated tool. It differs significantly from traditional code counters which are limited to counting lines of code and number of tokens. Essentially, the code Analyser is an implementation of the concepts developed in this and other chapters plus an implementation of Software Science theory [Halstead 1977] in a Fourth Generation language. The tool is written in FOCUS and hence is potentially portable to over 5000 MIS development environments in North America.22

4.4.2.1. Tool Development

Prior to construction, 75 FOCUS programs were counted manually using the reverse engineering approach discussed in the previous example.23 Rules were established to identify each program’s main function, identify major sections of code, count variables and classify variable usage. These rules were later implemented, and modified by experience, in the parsing and aggregation routines of the Analyser.

22 See Misra and Jalics [1988] for a discussion on the difficulties in using non-procedural languages to solve procedural problems
23 The counting was performed by the author
The construction of the Analyser began in early July 1988 and was iteratively tested, first against simple programs and then more complex code. By the beginning of August the Analyser had sufficient knowledge of FOCUS code to be run against the original 75 pilot programs. This revealed three sources of differences between the manual and automated measures:

1. bugs in the Analyser itself
2. manual counting errors in the pilot data.
3. code understanding and interpretation differences

The first source of error was dealt with easily. In the second case, the Analyser did a better job than the original manual counting. The third source of difference reflects a change in the knowledge and understanding of the FOCUS language by this researcher. This was brought about by building an Analyser written in the same language as the object of analysis. Hence, the manual vs automated results differ slightly as a result of different rules being applied in slightly different circumstances. As it turns out these changes are an improvement over the manual rules. In short, the manual process overlooked some key features. This process is discussed separately under Pilot Validation in Chapter 5.

4.4.2.2. Tool Description

The Code Analyser was implemented in PC/FOCUS 3.0 on an AT class micro-computer. Source code and data definitions for each completed system are loaded from diskette or imported over a communications link into separate sub-directories under each firm's name. The tool generates a list of systems for each company and corresponding lists
of FOCEXCS and MASTER FILE DEFINITIONS to be analyzed. These lists are then processed by the parsing routines which update the SYSTEMS database. For each program the main parser is invoked. The Code Analyzer scans each program's source code and produces the following outputs (see Figure 4.12):

1. Program classification
2. Program length: Text lines (excluding comments)
3. Halstead’s software science metrics:
   a. Number of unique operands: \( n_1 \)
   b. Total number of operands: \( N_1 \)
   c. Number of unique operators: \( n_2 \)
   d. Total number of operators: \( N_2 \)
   e. Program length: \( N = N_1 + N_2 \)
   f. Program vocabulary: \( n = n_1 + n_2 \)
   g. Program volume: \( V = N_{\log_2 n} \)
4. McCabes cyclomatic complexity metric
5. Number of screen images and number of input and output data elements on each screen
6. Number of reports and output variables
7. Number of files accessed
8. Number of projections and joins performed on tables
9. Distribution of language keyword usage.

4.4.2.3. Parsing Strategy

Each line of text is read in sequentially by the text parser. Blank lines and comments are omitted from further analysis, although the comment lines are latter scanned to pick up programmer and date information. Depending on what the line of code contains, various positional logic switches are turned on or off. The initial parse uses
Source Code and Data Definition from Working Systems

Code Analyser

Metrics to predict
New System Development:
- Function Points
- Bang Metrics
- Requirements Size

Metrics to predict
Maintenance:
- Halstead
- McCabe
- others

Figure 4.12: Outputs from Code Analyzer
blank as a delimiter. This generates a token which may be anywhere from 1 to 80 characters long and may contain one or more embedded tokens. This token is used in a table lookup of FOCUS keywords. If found then the token is an operator. Otherwise, if the token has the properties of a well formed operand then it is an operand. If both these two rules fail then further decomposition is performed on the token. A series of rules are applied to the complex token to break it down to a recognizable item. When a new chunk is parsed out, it becomes the current token and is passed to the top of the rules stack. New rules are easily incorporated into the structure with only a linear increase in processing time. An error trap routine waits at the bottom of the rule hierarchy to catch unidentified tokens. This routine is triggered when none of the preceding rules fire. It traps the tokens and writes them out to a log file which can then be looked at by a human after processing. The algorithm is recursive so that any complex token will eventually be decomposed into a simple interpretable one. Even after a simple token is found its interpretation depends on its position relative to preceding code. This complexity identifies a current flaw in the Analyser.

The parser makes only one pass through the code. It can only determine or interpret a token in the context of code that precedes the token. The Analyser does not look forward in the code and hence more subtle interpretations are not possible. For example, a variable appearing in a TABLE request can either be classified as being involved in the production of a report or actually appearing in a report. The information necessary to make this distinction does not appear in the code until the

\[2^4\] This technique proved useful in debugging the Analyser as new or special cases were uncovered when new programs were analyzed. It also discovered that some programs had been scrambled during transmission upload or download. These programs were deleted from the study.
end of the report program. Here, backwards classification would be necessary, a capability that the current version does not possess.\textsuperscript{25}

4.4.2.4. Program classification

The top level structure of the program classification follows standard data processing form. The subordinate classes reflect variations on this theme:

1. Control programs
   a. Menus
   b. Job control
   c. O/S interface
   d. Routing

2. Input programs
   a. Online interactive update
   b. Online interactive update with report
   c. Batch file update
   d. Batch file update with report

3. Output programs
   a. Batch reports
   b. Reports with user selection
   c. File output

4.4.2.5. Token counting rules

Tokens fall into two main categories - operators and operands. Tokens identified as being part of the FOCUS language or built-in procedures are operators. All remaining tokens fall into the category of data and the Code Analyser treats them as operands.\textsuperscript{25} This is one instance where the automated Code Analyzer would differ from manual analysis.
The Analyser classifies operands into one of 23 alternatives depending on its content and context within the program. The operand coding scheme appears in Appendix A.

4.4.2.6. Known limitations

1. The INCLUDE statement: The INCLUDE statement is a compile time instruction which brings in chunks of source code into programs. At present the Analyser does not bring these code chunks in for analysis in the context of the calling program but considers them as separate programs. When these code chunks are small re-useable routines such as input edit checks or printer setup, moderate usage has negligible effect on overall counting. However, when these code chunks are substantial in size, contain important file accessing information, and are used extensively, the situation is more problematic. Programs may be classified incorrectly, or their lengths may be underestimated and design characteristics may be hidden.

The issue is best understood by considering that at one level the program is the unit of analysis. Operationally, a program is counted as a file which contains a number of source instructions. If a program is physically distributed over several files then the unit of analysis breaks down. The same situation would arise at the system level if all programs were not present for analysis.

2. Compiled MODIFY: At present a problem exists in finding all the JOIN structures used by update programs that have been compiled. FOCUS requires that compiled MODIFY statements be run from a driver program which contains all the file access and join information. The problem is similar to the one above where two separate programs, a small driver and a larger body, are conceptually the same function. Several solutions exist to combine programs such
as these but none have yet been implemented.

3. Indirect file reference: If a file being accessed does not match a known master file definition name then this master’s information, clearly, can not be attached to the program. This situation may arise if complex indirect file referencing is used, such as global macro text substitution during run time.

The limitations identified above all revolve around the inability to logically collect related pieces of code at the program level while avoiding double counting. Several solutions are on the drawing boards to dynamically attach logically related chunks of code but these are being left for future development.

4.4.3. Systems Database

The entities and relationships in the Systems Database appear in Figure 4.13, with further detail in Appendix C.

The internal structure of the SYSTEMS database is basically hierarchical with a company as the root instance. Below company are systems, which are parents of programs and data definitions. Data definitions are cross-referenced to programs when encountered in the code. Below programs are lines of code which are the parents of operators and operands. There is a network type cross linkage to the Projects file. The Projects file contains project details regarding resource consumption, tasks performed, dates, and other project demographics.
Figure 4.13: Systems Database
4.4.4. Metrics Analysis

From the operationalizations provided previously the regression model to predict process size at the program level is:

\[ \hat{T} = fn(Screens, \ Input\ Variables, \ Output\ Variables \ Master\ files\ accessed, \ fields\ accessed) \]

At the system level the model becomes:

\[ \hat{T} = fn(Input\ events, \ Output\ events, \ Master\ files, \ Segments\ within\ master\ files, \ Total\ number\ of\ fields) \]

These models are tested against sample data described in Chapters 5 and 6.

4.5. HYPOTHESES

As discussed in this and previous chapters, the construct assumed to be the driving force behind effort is the size of the requirements. Clearly, other factors will have their effects during the process of development but these will have their effect on the base complexity. If, in fact, system requirements and design are the most important constructs to measure then it follows that:

1. The variance in requirements size and design size will explain a significant percentage of the variance in development effort measured at the system level.
2. Variance in \( \hat{E}' \) and \( \hat{S}' \) will explain a significant percentage of the variance in \( \hat{T}' \) at both the program and system level.
Additionally, due to the iterative nature of system development and estimating it is expected that:

3. As more information about $\hat{E}'$ and $\hat{S}'$ is obtained an increase in the accuracy of $\hat{T}'$ will be achieved.

4.6. SUMMARY

The research design consisted of three steps. The first was to establish a linkage between measures of design size and code size at the program level. The second step was to aggregate these parameters to the system level to link requirements size to design size. The third step required that a given system development environment have all its developed systems reverse engineered and then calibrated against resource consumption detail. This is to account for the wide differences amongst people skills, system or application type, and tool usage. In the next chapter the specific research environment is described where this is performed.
CHAPTER 5. THE EMPIRICAL STUDY

The purpose of this Chapter is to describe the field setting and field methods used in an initial test of the research model. This initial test focused on software systems written in a single language, within two development environments, and within the general class of "business data processing" as identified by De Marco [1982]. While this restriction will limit the generalizability of the findings to software systems developed in similar circumstances, there are a large number of development environments with similar characteristics in which to extend the research. Chapter 7 develops the extensions from this initial test.

This chapter first discusses the field setting and the data available for analysis. Second, it presents the results of pilot data analysis and validation against the Code Analyser. Third, it describes the data collection process and addresses a number of methodological issues.

Working in the field requires a great deal of patience and the ability to work as unobtrusively as possible. After all, any company agreeing to take part in a study is not guaranteed to receive any direct benefit. The researcher's presence consumes company resources, whether it is a single 30 second question or an explicit request for a time slot. Most critically, when resources become scarce, i.e. things get busy, research becomes secondary to the demands of the environment.
5.1. DATA SITE 1:

The first data site can be described as reasonably typical of modern system development shops. The DP Services employs nearly 80 people in a variety of jobs including clerical workers, 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, PLI) and 4th GL (FOCUS) languages are used. The staff turnover is comparatively low, with most professional personnel being with the firm over 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 investigation. The hardware development environment was constant, e.g. same operating system, screen editors, over the development period of the systems analyzed.

5.1.1. The data set

The data from this site included 26 application systems written in FOCUS. All systems were developed by the Small Projects Group in the company between 1984 and 1988. The systems analyzed represent all of the systems developed by the company in FOCUS. In addition to the measures specified in the last chapter, data on each system include:

1. Business function/application type.
2. Programmer(s)
3. Hours to build: analysis, design, implementation and maintenance.

The 26 systems represented nearly 800 FOCUS programs and over 62,000 FOCUS lines
of code (LOC). Details on the data collection procedure appear in a following section. In 4 systems, resource consumption data was not available as these hours were buried in other billing numbers such as user support, the Information Centre or another larger project. The hours data represent only the time spent building and maintaining the systems and do not reflect the ongoing support and training required by users. Hence the hours data are under-estimates of the actual labour resources spent on the systems.

5.1.2. Pilot Investigation

For the pilot study a small sample of the available data were analyzed manually before the automated tool was built. Two systems were manually counted in detail at the program level following the methodology described in Chapter 4. These two systems contained about 11,000 FOCUS LOC written within 75 programs. One system was 2700 LOC and 22 programs while the other was 8300 LOC and 53 programs. All programs were designed and written by the same senior systems analyst. The two systems were functionally similar in that they were basically menu driven by users. The users typically enter transactions from a terminal and may select a number of reports by specifying report parameters. Mean and variance tests were performed to determine if length of program was system dependent. These tests were not rejected allowing pooling of the programs. Analysis of this data set appears in Wrigley and Dexter [1988]. The strength of the relationships found in these 75 programs indicate there exists a stable relationship at the program level between operationalizations of the theory and LOC produced.
Additionally, hours and total FOCUS LOC were manually counted for 8 of the systems. Hours were extracted from the company's automated project control system. Hours for this sample range from 50 to 450 while LOC range from 300 to 8700. From this small sample there is evidence that hours and LOC are strongly related.

5.1.3. Pilot Validation

Prior to analyzing the sample data with the Code Analyser it was necessary to ensure that it generated the same metrics as the manual method. A problem immediately arose in comparing the manual pilot counts with the automated counts.

The manual analysis was performed in January 1988. When the source code for these systems was captured in machine readable form during July 1988, both systems had undergone revisions, one very minor and one more substantial. The added functionality to the second system resulted in it being 10% larger in terms of LOC. New reporting functions, several new data elements and new update routines were added. Also, several of the larger update programs had been converted to compiled modules for run-time efficiency. The combined modifications resulted in the system being expanded from 55 to 81 programs.

Many of the new programs resulted from certain update programs having been split into two programs. The update programs required that a separate small driver routine be split off. They had been together in the original pilot analysis and been split into two code chunks during system modification due to a restriction in FOCUS which requires that compile modules contain only MODIFY type statements. These driver
routines contained all file JOIN and file COMBINE information. Hence, the situation arose where some programs had a lot of sizing information yet no code body while their logically related programs had just code body with their data defined elsewhere. A current limitation of the Code Analyser does not allow for this situation to be dealt with in an automated way.  

For purposes of direct comparison with the pilot data the small driver programs were manually concatenated with their main program body. Also, only those programs appearing in both manual and automated counting were included in the validation. Comparison of both manual and automated counts appears in Appendix B, a - f, while the actual counts for both the independent and dependent variables for all programs appear in Appendix B, g - h. The simplest validation is to compare program length demographics. A more difficult task for the Analyser is to classify programs properly. Visual inspection of the two sets of counts shows very similar measures. The most difficult is to count the independent variables accurately. For this validation program by program visual checks were performed on the data in Appendix B.g and B.h. Differences that do exist are from three possible sources 1) manual error, 2) modifications made to the programs between January and July, and 3) different counting rules as discussed in Chapter 4. Finally, regressions were run to see if the differences that remained were important. A summary of the regression results appear below in Table 5.1.


Several solutions are on the drawing boards to dynamically attach logically related chunks of code.
Table 5.1: Pilot Regression - Explained Variance in Code Size

<table>
<thead>
<tr>
<th>Program Class</th>
<th>Manual</th>
<th>Automated</th>
</tr>
</thead>
<tbody>
<tr>
<td>Control</td>
<td>.48</td>
<td>.38</td>
</tr>
<tr>
<td>Updates</td>
<td>.87</td>
<td>.90</td>
</tr>
<tr>
<td>Outputs</td>
<td>.75</td>
<td>.74</td>
</tr>
</tbody>
</table>

One difference is noteworthy. The report variable is significant in the automated version. This is due to the different counting rules being used. Basically, in the manual analysis only those data elements actually appearing on a report were counted. The automated counting includes data elements involved in the production of the report as well as those actually appearing on the printed report.

Satisfied that the Automated Code Analyser was doing as good if not a better a job than the manual method, the source code from the remaining 26 systems in site 1 were collected and fed to the Code Analyser. Discussion of this collection process appears below while the analysis results appear in Chapter 6.
5.1.4. Data Collection

All systems written in FOCUS in the organization being studied were identified via discussions first with management and then with senior analysts. Reports from the project control system were used to cross check this list. Even at this stage the distinction between a project and a system became apparent. Several projects and even different systems could be logically grouped into a single system. The criteria used to call a group of programs a system was whether they could be attributed to an identifiable organization function, task or activity. A final list was constructed and used to locate the corresponding source code. In general all source code for a particular system was resident in its own library or machine. However, in some instances portions of systems resided in a production library. This entailed cross checking with system documentation and support personnel.

All source code and data definitions were written to tape and transported off site to U.B.C.'s computing facility. The contents of the tapes were loaded onto the University mainframe and then downloaded to an AT class micro-computer.

As discussed above, the Code Analyser was run first against the pilot data. After validation, the remaining systems were input to the Analyser and cross checking was done to ensure no duplicate programs existed. In several instances different program names were discovered to contain approximately the same source code. This came about due to some programmers leaving old copies of programs around. The duplication was discovered by careful visual inspection of the software metrics produced by the Analyser. Some programs were very similar as revealed by their metric
"signatures." Inspection of the original source code confirmed this. These programs were deleted from the sample after verbal confirmation by the programmers responsible for their production.

Originally the sample consisted of 28 systems. However, during cross checking two systems were deleted from the sample. It was discovered that one system was not a working system at all but an experimental collection of programs. A second system was deleted because it was written entirely by a contract analyst who was no longer with the firm and hence unavailable for verification questions. Additionally, it was discovered that the main update program for this system had been scrambled during data transmission from the data site. It was deemed unimportant to include this system in the sample. The 26 remaining systems comprised 793 programs. During data counting the Code Analyser found that an additional four programs had been scrambled in transport. As these programs were relatively small it was again deemed appropriate to drop them from the sample. Additionally, 19 programs were found to have missing master file definitions and were dropped from the sample. Hence the final sample size in data site 1 was 26 systems comprised of 770 programs. This data set was uploaded to the campus mainframe where the statistical package MIDAS was used.

5.1.4.1. Data Validity

Discussions were held with analysts responsible for each system to ensure that the program and master file descriptions extracted from the libraries represented complete pictures of the working systems. Where questions arose regarding possible errors in the data the analysts responsible for each system were contacted and shown what the
current assumptions were. Where errors or omissions were determined they were asked to supply the necessary data. This was done to verify that the data analyzed were accurate reflections of the systems’ status. It may be concluded that if another researcher went into the same environment the same measures would be produced.

There exists the possibility that some bias existed in the hours reported, exclusive to the "culture" of this particular environment. For example, some hours spent working on a system may be reported in general task classes and not against a specific project number. This may arise when a particular activity is applicable to more than one system. In this case hours may be reported against general department overhead and not to the specific systems. This is a threat only to the conclusions regarding the relationship between hours and other variables and therefore may reduce the internal validity of the observed relationship. However, it is unlikely that any field setting exists where there are no reporting confounds.

5.2. DATA SITE 2:

The second data site differed substantially from the first. It was a small consulting firm specializing in 4GL applications. The design and development team for all systems consisted of two persons only. Five systems consisting of over 1000 FOCUS programs were provided on floppy disks and loaded onto the campus micro. Hence data collection issues were minimal. This ease was probably due not only to similar hardware environments being used but also to a learning effect by this researcher by knowing the right questions to ask. The first data site represents not only the first test of the research model, but also a test and debugging of the research method. The feasibility
of conducting field research in this manner was born out by the relative speed of data collection and analysis in the second site.

Prior to constructing the Code Analyser one sample system was obtained for testing purposes. It was reasoned that the Code Analyser should be tested against code written by an independent source to ensure that the parsing rules were not site specific. This strategy proved invaluable as the consulting firm had written far more complex user interface code with a wider range of language usage.

5.3. DISCUSSION

In Chapter 4 limitations of the Code Analyser were identified. During the analysis of the data from site 2 one of the limitations became an issue. In site 1 INCLUDE statements (instructions that bring in re-usable code) were very few. They consisted primarily of PF KEY definitions, OS interface and printer routing, i.e. they were unclassified and did not affect the calling program's counting or classification. However, in site 2 the Analyser's limitation was more serious. Over 200 programs called in re-usable code. These code chunks contained critical information the Analyser needed to reverse engineer the systems properly. The situation was faced of whether or not to upgrade the Analyser or to limit the analysis of site 2 data. The later alternative was chosen with knowledge that the Analyser tool was not yet completely portable to different development environments.
5.4. OTHER MEASUREMENTS FROM THE DATA

In addition to the theory testing portion of the research, a number of descriptive statistics will be of interest to practitioners. The Automated Code Analyser allows an in depth view of the code which was previously computationally infeasible. Examples of the kinds of outputs the Analyser can produce appear below and will be addressed in the next chapter.

1. Comparison of systems and programs completed by the same personnel. This is useful as an analytic control mechanism.
   b. Changes over time in programming style.
2. Branching complexity: McCabe’s cyclomatic complexity. This measure is important for predicting maintenance effort.
3. Mix of program types, e.g. number of reports, updates, menus.
4. Productivity of programmers: LOC/Hours
5. Comparison of code produced by users vs system developers.
6. Software Science measures of a 4GL.

5.5. SUMMARY

This chapter has described the field settings and methods used to perform an initial test of the research model. Results of the manual and automated counting methods show the automated counting to be both valid and reliable within the specific field site. Limitations of the tool do not make it completely portable at this time; however, some cross site comparisons are possible. The next chapter presents the findings from
analysing the field data with the automated tool.
CHAPTER 6. EMPIRICAL FINDINGS

This chapter presents the findings from the empirical portion of the research. Section 6.1 considers the program as the unit of analysis and establishes the relationship between design and code for the detailed design phase. It reports program demographics, tests the program level regression model defined in section 4.4.4, and tests for programmer effects. Section 6.2 deals with the system as the unit of analysis and establishes the relationship between design and code for the preliminary design phase. It reports aggregated system level measures, tests the system level regression model defined in section 4.4.4, then briefly discusses resource consumption and overall productivity. Section 6.3 presents the first step towards generalizing the results across development environments by comparing the data from site 1 with the data from site 2. Finally, section 6.4 summarizes and discusses the empirical findings.

6.1. UNIT OF ANALYSIS: PROGRAM LEVEL

6.1.1. Program Size

The distribution of program size for site 1 appears in Figure 6.1. Of the 770 programs in site 1, 408 are less than 60 LOC (about 1 page of code) while 258 are between 60 and 150 LOC. 104 programs are considered "large", being over 150 LOC. However, in terms of total percentage of code in the sample these numbers are misleading. The 408 small programs represent only 15% of the code. The 258 medium sized programs represent another 35% of the code. The larger programs, while only 14% of the program count, represent just over 50% of the code. It is these larger programs which
may be of greater interest to managers not only because they may consume more effort to construct but also because they may require greater maintenance effort in the future.

6.1.2. Program Class

The theory developed in Chapter 3 and applied in Chapter 4 predicts that a software system will have processes to handle input and output events. Referring to Table 6.1, ANOVA tests on the mean and variance of program size in each class indicates that three different populations of programs do exist and correspond to Figure 3.3 as described below.

6.1.2.1. Update Programs

Update programs correspond conceptually to the event input vector, E', of Figure 3.3. Of 770 programs 200 classified as updates. These contained 48% of the code and have a mean size more than double the other classes. However, this class represents more than simple update processes. As the Code Analyser identifies main design points within the code it is able to partition the update class into several sub-classes. The Analyser discovered that some output functions such as update control totals were embedded within the code. The sub-classes within the general update class are:

1. Online using CRT: n=125
   a. No report: n=107
   b. With report: n=18
2. Batch file update: n=75
   a. No report: n=61
<table>
<thead>
<tr>
<th>MIDPOINT</th>
<th>COUNT FOR 5LOC (EACH X=3)</th>
</tr>
</thead>
</table>
| 0.00     | 102 +xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx
Table 6.1: Program class differences

UNIVARIATE 1-WAY ANOVA

ANALYSIS OF VARIANCE OF LOC  N= 720 OUT OF 720

<table>
<thead>
<tr>
<th>SOURCE</th>
<th>DF</th>
<th>SUM OF SQRS</th>
<th>MEAN SQR</th>
<th>F-STATISTIC</th>
<th>SIGNIF</th>
</tr>
</thead>
<tbody>
<tr>
<td>BETWEEN</td>
<td>2</td>
<td>.11315 +7</td>
<td>.56575 +6</td>
<td>51.253</td>
<td>.0000</td>
</tr>
<tr>
<td>WITHIN</td>
<td>717</td>
<td>.79144 +7</td>
<td>11038.</td>
<td></td>
<td></td>
</tr>
<tr>
<td>TOTAL</td>
<td>719</td>
<td>.90459 +7</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

RANDOM EFFECTS STATISTICS
ETA= .3537 ETA-SQR= .1251 (VAR COMP= 2497.2 %VAR AMONG= 18.45)

EQUALITY OF VARIANCES: DF= 2, .85500 +6  F= 179.11  .0000

<table>
<thead>
<tr>
<th>TOPCLASS</th>
<th>N</th>
<th>MEAN</th>
<th>VARIANCE</th>
<th>STD DEV</th>
</tr>
</thead>
<tbody>
<tr>
<td>UPDATE</td>
<td>200</td>
<td>147.16</td>
<td>28518.</td>
<td>168.87</td>
</tr>
<tr>
<td>OUTPUT</td>
<td>368</td>
<td>65.000</td>
<td>5421.9</td>
<td>73.634</td>
</tr>
<tr>
<td>CTL</td>
<td>152</td>
<td>47.651</td>
<td>1651.8</td>
<td>40.642</td>
</tr>
<tr>
<td>GRAND</td>
<td>720</td>
<td>84.160</td>
<td>12581.</td>
<td>112.17</td>
</tr>
</tbody>
</table>
Thirty two of the update programs contained output reports. In general these latter programs are not database extraction reports per se but reports on transaction processing, or changes to the database. Hence they can be considered as programmed output events.

6.1.2.2. Output Programs

Output Programs correspond to the event output vector, $E^O$, of Figure 3.3. The sample contained 368 programs (48%) in this class but only 39% of the code. Clearly, reporting processes were more numerous yet smaller.

6.1.2.3. Control Programs

The third primary class, control programs, represent the overhead required to organize and execute programs in the other two classes. There were 152 programs in this class representing 12% of the code.

6.1.2.4. Unclassified Analysis

Fifty (6%) of the 770 programs were not classified by the Code Analyser, i.e. they did not contain any code chunks or keywords which could be identifiable as design decisions. These programs consisted mainly of small code stubs (eg. include files, utility programs, program function key definitions or printer routing). The total amount of
code in the unclassified programs was 685 LOC, or 1% of the total code from data site 1. These programs were discarded from further analysis leaving 720 programs of interest.

6.1.3. Regression Results

The results from running the regression model in Section 4.4.4 against the three classes of programs appear in Tables 6.2.1 - 6.2.3.

6.1.3.1. Regression Discussion

The Master Files Accessed component of the regression model has been expanded to include Segments in order to obtain a more complete measure of the complexity in the database accessed. Segments are a measure of the structural linkages within a Master file. FOCUS structures its databases in a three level hierarchy with Master at the top consisting of subordinate Segments which in turn consist of a number of atomic fields. Segments are not significant in the model as Masters and Segments are highly correlated. It is included here as it will be used later, in conjunction with Fields, to form a principal component measure of the systems' static space, S.

As can be seen from Table 6.2.1 the variables in the model explain 86% of the variance in update program size. Of the three variables, Masters, Segments and Fields only one explains a significant proportion of the variance. This is because these three variables are highly correlated. The distribution of residuals from this regression have a standard deviation of 63 LOC with a slight positive skew of 2.7 LOC. A plot of the
### Tabel 6.2.1: Regression results - Update class

#### LEAST SQUARES REGRESSION <1> CLASS:UPDATE

#### ANALYSIS OF VARIANCE OF 5.LOC N= 200 OUT OF 200

<table>
<thead>
<tr>
<th>SOURCE</th>
<th>DF</th>
<th>SUM SQRS</th>
<th>MEAN SQR</th>
<th>F-STAT</th>
<th>SIGNIF</th>
</tr>
</thead>
<tbody>
<tr>
<td>REGRESSION</td>
<td>7</td>
<td>.48723 +7</td>
<td>.69604 +6</td>
<td>166.45</td>
<td>.0000</td>
</tr>
<tr>
<td>ERROR</td>
<td>192</td>
<td>.80290 +6</td>
<td>4181.8</td>
<td></td>
<td></td>
</tr>
<tr>
<td>TOTAL</td>
<td>199</td>
<td>.56752 +7</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

MULT R= .92657 R-SQR= .85852 SE= 64.667

<table>
<thead>
<tr>
<th>VARIABLE</th>
<th>PARTIAL</th>
<th>COEFF</th>
<th>STD ERROR</th>
<th>T-STAT</th>
<th>SIGNIF</th>
</tr>
</thead>
<tbody>
<tr>
<td>CONSTANT</td>
<td>-3.3345</td>
<td>9.1546</td>
<td>-.36424</td>
<td>.7161</td>
<td></td>
</tr>
<tr>
<td>6. #SCREENS</td>
<td>.31890</td>
<td>15.304</td>
<td>3.2825</td>
<td>4.6623</td>
<td>.0000</td>
</tr>
<tr>
<td>7. SCREEN_IN</td>
<td>.38937</td>
<td>1.5461</td>
<td>.26395</td>
<td>5.8576</td>
<td>.0000</td>
</tr>
<tr>
<td>8. SCREEN_OUT</td>
<td>.58744</td>
<td>2.4872</td>
<td>.24728</td>
<td>10.058</td>
<td>.0000</td>
</tr>
<tr>
<td>9. OUTPUT_DATA</td>
<td>.29229</td>
<td>5.1447</td>
<td>.12148</td>
<td>4.2350</td>
<td>.0000</td>
</tr>
<tr>
<td>10. MASTERS</td>
<td>.25929</td>
<td>19.014</td>
<td>5.1112</td>
<td>3.7200</td>
<td>.0003</td>
</tr>
<tr>
<td>12. SEGMENTS</td>
<td>.07893</td>
<td>2.9489</td>
<td>2.6880</td>
<td>1.0971</td>
<td>.2740</td>
</tr>
<tr>
<td>11. FIELDS</td>
<td>.10313</td>
<td>.37831</td>
<td>.26332</td>
<td>1.4367</td>
<td>.1524</td>
</tr>
</tbody>
</table>
### Least Squares Regression (2) Class: Output

#### Analysis of Variance of 5.Loc N= 368 Out of 368

<table>
<thead>
<tr>
<th>Source</th>
<th>DF</th>
<th>Sum Squares</th>
<th>Mean Squares</th>
<th>F-Stat</th>
<th>Signif</th>
</tr>
</thead>
<tbody>
<tr>
<td>Regression</td>
<td>7</td>
<td>.15438 +7</td>
<td>.22055 +6</td>
<td>178.01</td>
<td>0.</td>
</tr>
<tr>
<td>Error</td>
<td>360</td>
<td>.44602 +6</td>
<td>1238.9</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Total</td>
<td>367</td>
<td>.19899 +7</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

MULT R= .88083 R-SQR= .77585 SE= 35.198

#### Variable

<table>
<thead>
<tr>
<th>Variable</th>
<th>Partial</th>
<th>Coeff</th>
<th>Std Error</th>
<th>T-Stat</th>
<th>Signif</th>
</tr>
</thead>
<tbody>
<tr>
<td>CONSTANT</td>
<td>-1.7931</td>
<td>4.3138</td>
<td>-.41567</td>
<td>.6779</td>
<td></td>
</tr>
<tr>
<td>6.#SCREENS</td>
<td>.19375</td>
<td>27.049</td>
<td>7.2184</td>
<td>3.7472</td>
<td>.0002</td>
</tr>
<tr>
<td>7.SCREEN_IN</td>
<td>.10099</td>
<td>6.5655</td>
<td>3.4090</td>
<td>1.9259</td>
<td>.0549</td>
</tr>
<tr>
<td>8.SCREEN_OUT</td>
<td>.18134</td>
<td>6.3111</td>
<td>1.8039</td>
<td>3.4987</td>
<td>.0005</td>
</tr>
<tr>
<td>9.OUTPUT_DATA</td>
<td>.82075</td>
<td>.88397</td>
<td>.32428 -1</td>
<td>27.259</td>
<td>0.</td>
</tr>
<tr>
<td>10.MASTERS</td>
<td>.16469</td>
<td>9.6389</td>
<td>3.0426</td>
<td>3.1680</td>
<td>.0017</td>
</tr>
<tr>
<td>12.SEGMENTS</td>
<td>.00932</td>
<td>.19926</td>
<td>1.1263</td>
<td>.17692</td>
<td>.8597</td>
</tr>
<tr>
<td>11.FIELDS</td>
<td>.18129</td>
<td>.33605</td>
<td>.96078 -1</td>
<td>3.4977</td>
<td>.0005</td>
</tr>
</tbody>
</table>
### LEAST SQUARES REGRESSION <3> CLASS: CONTROL

#### ANALYSIS OF VARIANCE OF 5.LOC  N = 152 OUT OF 152

<table>
<thead>
<tr>
<th>SOURCE</th>
<th>DF</th>
<th>SUM SQRS</th>
<th>MEAN SQR</th>
<th>F-STAT</th>
<th>SIGNIF</th>
</tr>
</thead>
<tbody>
<tr>
<td>REGRESSION</td>
<td>7</td>
<td>.14553 +6 20789.</td>
<td>28.816</td>
<td>.0000</td>
<td></td>
</tr>
<tr>
<td>ERROR</td>
<td>144</td>
<td>.10389 +6 721.46</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>TOTAL</td>
<td>151</td>
<td>.24942 +6</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

MULT R = .76385  R-SQR= .58346  SE = 26.860

<table>
<thead>
<tr>
<th>VARIABLE</th>
<th>PARTIAL</th>
<th>COEFF</th>
<th>STD ERROR</th>
<th>T-STAT</th>
<th>SIGNIF</th>
</tr>
</thead>
<tbody>
<tr>
<td>CONSTANT</td>
<td>9.2193</td>
<td>3.8833</td>
<td>2.3741</td>
<td>.0189</td>
<td></td>
</tr>
<tr>
<td>6. #SCREENS</td>
<td>.57950</td>
<td>21.871</td>
<td>8.5328</td>
<td>.0000</td>
<td></td>
</tr>
<tr>
<td>7. SCREEN_IN</td>
<td>.01243</td>
<td>.40625</td>
<td>.14911</td>
<td>.8817</td>
<td></td>
</tr>
<tr>
<td>8. SCREEN_OUT</td>
<td>.37864</td>
<td>4.6876</td>
<td>4.9093</td>
<td>.0000</td>
<td></td>
</tr>
<tr>
<td>9. OUTPUT_DATA</td>
<td>.00181</td>
<td>.49062</td>
<td>.21760 -1</td>
<td>.9827</td>
<td></td>
</tr>
<tr>
<td>10. MASTERS</td>
<td>.04587</td>
<td>3.0950</td>
<td>.55104</td>
<td>.5825</td>
<td></td>
</tr>
<tr>
<td>12. SEGMENTS</td>
<td>.00537</td>
<td>.30617</td>
<td>.64481 -1</td>
<td>.9487</td>
<td></td>
</tr>
<tr>
<td>11. FIELDS</td>
<td>-.02928</td>
<td>-.16777</td>
<td>.35149</td>
<td>.7257</td>
<td></td>
</tr>
</tbody>
</table>
residuals and the predicted values shows slight heteroscedasticity. This may indicate that a non-linear term in the model should be explored.

In Table 6.2.2 the model explains 78% of the variance in output program size. The SCREEN-IN variable falls out of significance, as we would expect, because these are output programs. SCREEN-IN are the number of data elements entered by the user. In this class it represents user control over report production which usually entails only one or two variables being entered by the user. Segments are again insignificant for the same reason stated above. The distribution of residuals from this regression have a standard deviation of 34 LOC with a slight positive skew of 2.9 LOC. As above, a scatter plot of the residuals against the predicted values shows slightly increasing variance.

The Control class, in Table 6.2.3 is more problematic. The model only explains 58% of the variance. As expected, the database related measures do not explain any significant variance. There are no data flowing in these programs so we would also not expect screen input variables to be significant. The only code in this class of programs consists of screen display and a few output variables evident by the number of screens (#SCREENS) and the number of data elements displayed (SCREENOUT) being highly significant. The significant constant term in Tables 6.2.3 indicates some minimum overhead in writing each program such as variable initialization or other set-up procedures. The distribution of residuals from this regression have a standard deviation of 26 LOC with a slight positive skew of 2.7 LOC. Again, the residuals show increasing variance when plotted against the predicted dependent variable.
6.1.3.2. Towards Parsimony

In the previous regressions the screen related variables are highly correlated as are the database related variables. To remove this multicollinearity the first principal component was extracted from the number of screens (#SCREEN) and the number of input data elements (SCREEN-IN), and from the number of logical database groups (SEGMENTS) and the number of fields in all segments (FIELDS). These principal components correspond conceptually to the notions of input event size, E', and system static space size, S, and given the labels INSIZE and DATASIZE. As a measure of output event size, E°, the amount of data flowing out of the systems (OUTPUT-DATA) is relabelled OUTSIZE. These three variables form a new reduced model and the results of this run appear in Table 6.3. For update and output programs all three independent variables were highly significant explaining 82% and 75% of the variance in program length respectively. Control programs, however, remain problematic. The single significant independent variable, INSIZE, explained only 37% of the variance. However, we are primarily interested in the more numerous and larger input and output programs. This more parsimonious model indicates that aggregate measures of input, output and data size can be used to predict program size during the detailed design phase of system development.

6.1.4. Programmer differences

In Site 1 a total of thirteen persons were involved in producing the 26 systems. 693 of the 770 programs could be uniquely attributed to a programmer from the documentation embedded within the source code. Of these 693, 638 (92%) were
### Table 6.3: Reduced Model

**Least Squares Regression**

#### Analysis of Variance of LOC

<table>
<thead>
<tr>
<th>Source</th>
<th>DF</th>
<th>SUM SQRS</th>
<th>MEAN SQR</th>
<th>F-Stat</th>
<th>Signif</th>
</tr>
</thead>
<tbody>
<tr>
<td>Regression</td>
<td>3</td>
<td>0.46818</td>
<td>0.15606</td>
<td>307.91</td>
<td>.0000</td>
</tr>
<tr>
<td>Error</td>
<td>196</td>
<td>0.99340</td>
<td>0.05068</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Total</td>
<td>199</td>
<td>0.56752</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

**Mult R= .90827, R-SQR= .82496, SE= 71.192**

<table>
<thead>
<tr>
<th>Variable</th>
<th>Partial Coeff</th>
<th>Std Error</th>
<th>T-Stat</th>
<th>Signif</th>
</tr>
</thead>
<tbody>
<tr>
<td>Constant</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>INSIZE</td>
<td>.33.298</td>
<td>7.8665</td>
<td>4.2328</td>
<td>.0000</td>
</tr>
<tr>
<td>OUTSIZE</td>
<td>.44716</td>
<td>8.1512</td>
<td>6.9990</td>
<td>.0000</td>
</tr>
<tr>
<td>DATASIZE</td>
<td>.24684</td>
<td>7.4600</td>
<td>5.5662</td>
<td>.0005</td>
</tr>
</tbody>
</table>

#### Analysis of Variance of LOC

<table>
<thead>
<tr>
<th>Source</th>
<th>DF</th>
<th>SUM SQRS</th>
<th>MEAN SQR</th>
<th>F-Stat</th>
<th>Signif</th>
</tr>
</thead>
<tbody>
<tr>
<td>Regression</td>
<td>3</td>
<td>0.15007</td>
<td>0.50022</td>
<td>372.22</td>
<td>0.</td>
</tr>
<tr>
<td>Error</td>
<td>364</td>
<td>0.48918</td>
<td>1343.9</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Total</td>
<td>367</td>
<td>0.19899</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

**Mult R= .86843, R-SQR= .75416, SE= 36.659**

<table>
<thead>
<tr>
<th>Variable</th>
<th>Partial Coeff</th>
<th>Std Error</th>
<th>T-Stat</th>
<th>Signif</th>
</tr>
</thead>
<tbody>
<tr>
<td>Constant</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>INSIZE</td>
<td>.53586</td>
<td>13.008</td>
<td>12.109</td>
<td>.0000</td>
</tr>
<tr>
<td>OUTSIZE</td>
<td>.81603</td>
<td>8.9919</td>
<td>26.936</td>
<td>0.</td>
</tr>
<tr>
<td>DATASIZE</td>
<td>.26675</td>
<td>4.6729</td>
<td>5.2805</td>
<td>.0000</td>
</tr>
</tbody>
</table>

#### Analysis of Variance of LOC

<table>
<thead>
<tr>
<th>Source</th>
<th>DF</th>
<th>SUM SQRS</th>
<th>MEAN SQR</th>
<th>F-Stat</th>
<th>Signif</th>
</tr>
</thead>
<tbody>
<tr>
<td>Regression</td>
<td>3</td>
<td>91829.</td>
<td>30610.</td>
<td>28.747</td>
<td>.0000</td>
</tr>
<tr>
<td>Error</td>
<td>148</td>
<td>15759</td>
<td>1064.8</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Total</td>
<td>151</td>
<td>24942</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

**Mult R= .60678, R-SQR= .36818, SE= 32.631**

<table>
<thead>
<tr>
<th>Variable</th>
<th>Partial Coeff</th>
<th>Std Error</th>
<th>T-Stat</th>
<th>Signif</th>
</tr>
</thead>
<tbody>
<tr>
<td>Constant</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>INSIZE</td>
<td>.58567</td>
<td>8.2710</td>
<td>8.7903</td>
<td>.0000</td>
</tr>
<tr>
<td>OUTSIZE</td>
<td>.00066</td>
<td>2.1907</td>
<td>8.0263</td>
<td>.9936</td>
</tr>
<tr>
<td>DATASIZE</td>
<td>-.05201</td>
<td>-.10222</td>
<td>-.63354</td>
<td>.5274</td>
</tr>
</tbody>
</table>
written by seven programmers. The remaining programs were written by persons on loan from other areas of the DP shop. The number of observations for these programmers are too small to analyze.

Table 6.4: Programmer by Program Class

<table>
<thead>
<tr>
<th>Person</th>
<th>Control</th>
<th>Outputs</th>
<th>Updates</th>
<th>Unclass</th>
<th>Total N</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>7</td>
<td>27</td>
<td>14</td>
<td>0</td>
<td>48</td>
</tr>
<tr>
<td>B</td>
<td>6</td>
<td>25</td>
<td>15</td>
<td>0</td>
<td>46</td>
</tr>
<tr>
<td>C</td>
<td>0</td>
<td>25</td>
<td>0</td>
<td>1</td>
<td>26</td>
</tr>
<tr>
<td>D</td>
<td>0</td>
<td>4</td>
<td>0</td>
<td>0</td>
<td>4</td>
</tr>
<tr>
<td>E</td>
<td>6</td>
<td>45</td>
<td>21</td>
<td>0</td>
<td>72</td>
</tr>
<tr>
<td>F</td>
<td>2</td>
<td>02</td>
<td>03</td>
<td>0</td>
<td>7</td>
</tr>
<tr>
<td>G **</td>
<td>33</td>
<td>84</td>
<td>46</td>
<td>3</td>
<td>166</td>
</tr>
<tr>
<td>H</td>
<td>2</td>
<td>3</td>
<td>5</td>
<td>0</td>
<td>12</td>
</tr>
<tr>
<td>I **</td>
<td>44</td>
<td>47</td>
<td>27</td>
<td>7</td>
<td>125</td>
</tr>
<tr>
<td>J</td>
<td>10</td>
<td>17</td>
<td>21</td>
<td>0</td>
<td>48</td>
</tr>
<tr>
<td>K **</td>
<td>31</td>
<td>38</td>
<td>38</td>
<td>26</td>
<td>133</td>
</tr>
<tr>
<td>L</td>
<td>0</td>
<td>0</td>
<td>02</td>
<td>0</td>
<td>2</td>
</tr>
<tr>
<td>M</td>
<td>2</td>
<td>0</td>
<td>0</td>
<td>2</td>
<td>4</td>
</tr>
</tbody>
</table>

Of the 638 programs, 424 were written by just three people, accounting for over 60% of the FOCUS code output. The distribution of programs written by programmer by program class appears in Table 6.4. The double asterisk beside programmer indicates sufficient sample size for further analysis.
The source code for 77 of the 770 programs contained no documentation identifying the programmer. These programs consisted of 1544 LOC or 2.5% of the sample code. Of these 77 only 11 were unclassified and consisted of 137 LOC. Of the remainder, 9 (113 LOC) were small menus, 8 (310 LOC) were small update programs and, interestingly, 49 (1120 LOC) were report programs. If we assume that lack of programmer documentation in a program indicates that it was written by a non-DP professional, i.e. an end-user, then this points to an interesting observation. Purportedly, 4GL languages are a boon to end-user computing (EUC). The data in this sample indicates that EUC is still very much limited to small extraction programs accessing existing databases. The more difficult aspects of building the core of systems must rely on the expertise of MIS professionals. Furthermore, of the total 4GL code written, less than 3% appears to have been generated by non-DP staff. However, this observation may be an artifact of the data collection method used in this data site. With the growing use of micro-computer to mainframe communication it may be the case that user written code resides on end user personal computers and is only executed against databases accessible through the mainframe. Due to the diverse nature of the data site it was not possible to pursue this observation further.

6.1.4.1. Effects on Program Size

Three programmers - C, I and K had sufficient data points in each program class to allow comparisons across classes. To test for programmer effects on program size three dummy variables were added to the regression model in Tables 6.2.1 - 6.2.3. This expanded model appears in Tables 6.5.1 - 6.5.3 for program classes Update, Output and Control respectively. As can be seen from Table 6.5.1, none of the programmer
Table 6.5.1: Programmer Differences - Updates

LEAST SQUARES REGRESSION <1> CLASS:UPDATE

ANALYSIS OF VARIANCE OF 5.LOC  N= 200 OUT OF 200

<table>
<thead>
<tr>
<th>SOURCE</th>
<th>DF</th>
<th>SUM SQRS</th>
<th>MEAN SQR</th>
<th>F-STAT</th>
<th>SIGNIF</th>
</tr>
</thead>
<tbody>
<tr>
<td>REGRESSION</td>
<td>10</td>
<td>.48770 +7</td>
<td>.48770 +6</td>
<td>115.49</td>
<td>.0000</td>
</tr>
<tr>
<td>ERROR</td>
<td>189</td>
<td>.79813 +6</td>
<td>4222.9</td>
<td></td>
<td></td>
</tr>
<tr>
<td>TOTAL</td>
<td>199</td>
<td>.56752 +7</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

MULT R= .92702  R-SQR= .85936  SE= 64.984

<table>
<thead>
<tr>
<th>VARIABLE</th>
<th>PARTIAL</th>
<th>COEFF</th>
<th>STD ERROR</th>
<th>T-STAT</th>
<th>SIGNIF</th>
</tr>
</thead>
<tbody>
<tr>
<td>CONSTANT</td>
<td>-.56162</td>
<td>10.092</td>
<td>-.55650 -1</td>
<td>.9557</td>
<td></td>
</tr>
<tr>
<td>6.#SCREENS</td>
<td>.32495</td>
<td>15.829</td>
<td>4.7237</td>
<td>.0000</td>
<td></td>
</tr>
<tr>
<td>7.SCREEN_IN</td>
<td>.38962</td>
<td>1.5483</td>
<td>5.8159</td>
<td>.0000</td>
<td></td>
</tr>
<tr>
<td>8.SCREEN_OUT</td>
<td>.57926</td>
<td>2.4563</td>
<td>9.7695</td>
<td>.0000</td>
<td></td>
</tr>
<tr>
<td>9.OUTPUT_DATA</td>
<td>.28794</td>
<td>.52755</td>
<td>4.1336</td>
<td>.0001</td>
<td></td>
</tr>
<tr>
<td>10.MASTERS</td>
<td>.26022</td>
<td>20.149</td>
<td>3.7050</td>
<td>.0003</td>
<td></td>
</tr>
<tr>
<td>12.SEGMENTS</td>
<td>.04247</td>
<td>1.7621</td>
<td>.58442</td>
<td>.5596</td>
<td></td>
</tr>
<tr>
<td>11.FIELDS</td>
<td>.12035</td>
<td>.46121</td>
<td>1.6666</td>
<td>.0973</td>
<td></td>
</tr>
<tr>
<td>45.PERSON_G</td>
<td>-.05359</td>
<td>-9.2850</td>
<td>-.73779</td>
<td>.4616</td>
<td></td>
</tr>
<tr>
<td>46.PERSON_I</td>
<td>-.06435</td>
<td>-13.543</td>
<td>-.88654</td>
<td>.3765</td>
<td></td>
</tr>
<tr>
<td>47.PERSON_K</td>
<td>.00193</td>
<td>.35394</td>
<td>.26528 -1</td>
<td>.9789</td>
<td></td>
</tr>
</tbody>
</table>
Table 6.5.2: Programmer Differences - Outputs

LEAST SQUARES REGRESSION <2> CLASS: OUTPUT

ANALYSIS OF VARIANCE OF 5.LOC N= 368 OUT OF 368

<table>
<thead>
<tr>
<th>SOURCE</th>
<th>DF</th>
<th>SUM SQRs</th>
<th>MEAN SQR</th>
<th>F-STAT</th>
<th>SIGNIF</th>
</tr>
</thead>
<tbody>
<tr>
<td>REGRESSION</td>
<td>10</td>
<td>.15651 +7</td>
<td>.15651 +6</td>
<td>131.55</td>
<td>0.</td>
</tr>
<tr>
<td>ERROR</td>
<td>357</td>
<td>.42473 +6</td>
<td>1189.7</td>
<td></td>
<td></td>
</tr>
<tr>
<td>TOTAL</td>
<td>367</td>
<td>.19899 +7</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

MULT R= .88688  R-SQR= .78655 SE= 34.492

<table>
<thead>
<tr>
<th>VARIABLE</th>
<th>PARTIAL COEFF</th>
<th>STD ERROR</th>
<th>T-STAT</th>
<th>SIGNIF</th>
</tr>
</thead>
<tbody>
<tr>
<td>CONSTANT</td>
<td>-4.9776</td>
<td>4.3501</td>
<td>-1.1442</td>
<td>.2533</td>
</tr>
<tr>
<td>6.#SCREENS</td>
<td>.19385</td>
<td>7.2635</td>
<td>3.7335</td>
<td>.0002</td>
</tr>
<tr>
<td>7.SCREEN_IN</td>
<td>.06550</td>
<td>3.3938</td>
<td>1.2402</td>
<td>.2157</td>
</tr>
<tr>
<td>8.SCREEN_OUT</td>
<td>.18621</td>
<td>1.7729</td>
<td>3.5809</td>
<td>.0004</td>
</tr>
<tr>
<td>9.OUTPUT_DATA</td>
<td>.82867</td>
<td>.31900</td>
<td>27.972</td>
<td>0.</td>
</tr>
<tr>
<td>10.MASTERS</td>
<td>.19715</td>
<td>3.1019</td>
<td>3.7977</td>
<td>.0002</td>
</tr>
<tr>
<td>12.SEGMENTS</td>
<td>-.04763</td>
<td>1.1740</td>
<td>-.90095</td>
<td>.3682</td>
</tr>
<tr>
<td>11.FIELDS</td>
<td>.20146</td>
<td>.96575</td>
<td>3.8862</td>
<td>.0001</td>
</tr>
<tr>
<td>45.PERSON_G</td>
<td>.15300</td>
<td>4.9511</td>
<td>2.9254</td>
<td>.0037</td>
</tr>
<tr>
<td>46.PERSON_I</td>
<td>-.09174</td>
<td>6.2143</td>
<td>-1.7406</td>
<td>.0826</td>
</tr>
<tr>
<td>47.PERSON_K</td>
<td>.09139</td>
<td>6.3504</td>
<td>1.7340</td>
<td>.0838</td>
</tr>
</tbody>
</table>
### Table 6.5.3: Programmer Differences - Control

#### LEAST SQUARES REGRESSION <3> CLASS:CTL

#### ANALYSIS OF VARIANCE OF 5.LOC  N= 152 OUT OF 152

<table>
<thead>
<tr>
<th>SOURCE</th>
<th>DF</th>
<th>SUM SQRS</th>
<th>MEAN SQR</th>
<th>F-STAT</th>
<th>SIGNIF</th>
</tr>
</thead>
<tbody>
<tr>
<td>REGRESSION</td>
<td>10</td>
<td>.16104 +6</td>
<td>16104.</td>
<td>25.691</td>
<td>.0000</td>
</tr>
<tr>
<td>ERROR</td>
<td>141</td>
<td>88381.</td>
<td>626.81</td>
<td></td>
<td></td>
</tr>
<tr>
<td>TOTAL</td>
<td>151</td>
<td>.24942 +6</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

**MULT R = .80352  R-SQR= .64565  SE = 25.036**

#### VARIABLE  PARTIAL  COEFF  STD ERROR  T-STAT  SIGNIF

<table>
<thead>
<tr>
<th>VARIABLE</th>
<th>PARTIAL</th>
<th>COEFF</th>
<th>STD ERROR</th>
<th>T-STAT</th>
<th>SIGNIF</th>
</tr>
</thead>
<tbody>
<tr>
<td>CONSTANT</td>
<td></td>
<td>23.335</td>
<td>4.6745</td>
<td>4.9921</td>
<td>.0000</td>
</tr>
<tr>
<td>6.#SCREENS</td>
<td>.62444</td>
<td>22.787</td>
<td>2.4004</td>
<td>9.4932</td>
<td>.0000</td>
</tr>
<tr>
<td>7.SCREEN_IN</td>
<td>.02201</td>
<td>.66811</td>
<td>2.5553</td>
<td>.26146</td>
<td>.7941</td>
</tr>
<tr>
<td>8.SCREEN_OUT</td>
<td>.39718</td>
<td>4.6232</td>
<td>.89964</td>
<td>5.1389</td>
<td>.0000</td>
</tr>
<tr>
<td>9.OUTPUT_DATA</td>
<td>-.04901</td>
<td>-1.2366</td>
<td>2.1225</td>
<td>-.58264</td>
<td>.5611</td>
</tr>
<tr>
<td>10.MASTERS</td>
<td>.08210</td>
<td>5.2344</td>
<td>5.3514</td>
<td>.97813</td>
<td>.3297</td>
</tr>
<tr>
<td>12.SEGMENTS</td>
<td>-.03334</td>
<td>-1.7634</td>
<td>4.4518</td>
<td>-.39612</td>
<td>.6926</td>
</tr>
<tr>
<td>11.FIELDS</td>
<td>-.02810</td>
<td>-.14930</td>
<td>.44720</td>
<td>-.33385</td>
<td>.7390</td>
</tr>
<tr>
<td>45.PERSON_G</td>
<td>-.19397</td>
<td>-14.266</td>
<td>6.0764</td>
<td>-2.3478</td>
<td>.0203</td>
</tr>
<tr>
<td>46.PERSON_I</td>
<td>-.36021</td>
<td>-24.968</td>
<td>5.4456</td>
<td>-4.5850</td>
<td>.0000</td>
</tr>
<tr>
<td>47.PERSON_K</td>
<td>-.30728</td>
<td>-22.956</td>
<td>5.9871</td>
<td>-3.8343</td>
<td>.0002</td>
</tr>
</tbody>
</table>
dummy variables are significantly different from zero. This indicates that the other independent variables, representing measures of design, are the principle contributors to program length. In Table 6.5.2 programmer C appears to write slightly larger output programs (p < .004), however the coefficient is only 14 LOC so this may not be too meaningful for management. An interesting phenomenon occurs in control programs. Collectively, the three programmers, C, I, K, tend to produce smaller control programs than the other programmers combined. Since control programs represent only 11% of all code produced this may not be important from a project management perspective. However, from a personnel management perspective insights of this kind may assist in allocating training budgets or task assignments. Due to the letter of understanding between the researcher and the company involved the identities of the programmers will remain confidential.

6.7.4.2. Test For Learning Effects

One potential confounding variable lies in programmer learning effects over time. To address this possibility, the date that each program was written was extracted from the source code. This information was available for 626 programs. The period 1985-1988 was converted to a day displacement from January 1, 1985. For example, January 01, 1986 = 366, January 01, 1987 = 731 etc. The date that each program was written was converted to the above scale. In general programs were written within a short time period. This DATE variable was then correlated with all variables in the regression model by each programmer. The hypothesis that there was a change over time in the regression variables was not supported although some minor differences did emerge.
For programmer G there is some evidence that programs were becoming smaller over the three year period: \( R(date, LOC) = -0.167 \) (\( p < .04 \)). While programmer I included more variables in report programs: \( R = 0.16 \) (\( p < .03 \)), programmer J had no discernable change over time.

The research design in chapter 4 called for the personnel variables and system type to be held constant. As the staff were experienced analysts and FOCUS programmers and the kinds of systems being built were similar the above results are what we would expect. Based on the evidence, and looking at the regression model alone, it is safe to conclude that the programmers in the sample were tackling similar kinds of programming tasks in a consistent manner. However, the automated tool offered a unique opportunity to investigate programming style and a few interesting observations emerged.

A learning effect was detected for programmer I. Here the richness of language usage, or vocabulary, increased significantly over time (\( p < .01 \)). and the density of the code (number of code tokens per line) also increased (\( p < .03 \)). Conversely, the density of programmer K's code decreased over time (\( p < .01 \)). These anecdotes give some indication as to the managerial insights made possible by the Code Analyser.

6.2. UNIT OF ANALYSIS: SYSTEM LEVEL
6.2.1. Demographics

Measures of system sizes for the 26 systems from site 1 appear in Table 6.6. A histogram of system sizes in terms of LOC shows the same general shape as the distribution of program size observed at the program level.

6.2.2. Metric Linkage at System Level

For the 26 systems three measures of design size predict eventual total LOC very well. Number of screens, number of reports and number of segments in the database together explain 94% of the variance in this sample. However, it is premature to claim that the model is validated because several large systems could create this unusually striking result. The regression line is being forced through the large systems which contain most of the variance. Ideally, a larger data set would have data points uniformly distributed over the range. Details of this regression appear in Table 6.7. The large negative intercept is not meaningful in this context as there are no observations close to zero, and extending the regression line past the small data set is inappropriate.

6.2.3. Resource Consumption

The 22 systems with hours data required 11,365 hours to develop, enhance and
Table 6.6: System Level Size Measures

<table>
<thead>
<tr>
<th>System</th>
<th>LOC</th>
<th>Prog</th>
<th>Mast</th>
<th>Segm</th>
<th>Fields</th>
<th>Scr-</th>
<th>Rep-</th>
<th>Devel</th>
<th>Main</th>
<th>Total</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>Progs</td>
<td>Maste</td>
<td>Segms</td>
<td>Fields</td>
<td>Scr-</td>
<td>Rep-</td>
<td>Hours</td>
<td>Hours</td>
<td>Hours</td>
</tr>
<tr>
<td>1</td>
<td>3145</td>
<td>22</td>
<td>4</td>
<td>10</td>
<td>126</td>
<td>33</td>
<td>13</td>
<td>206</td>
<td>5</td>
<td>211</td>
</tr>
<tr>
<td>2</td>
<td>9509</td>
<td>81</td>
<td>11</td>
<td>13</td>
<td>109</td>
<td>101</td>
<td>30</td>
<td>607</td>
<td>0</td>
<td>607</td>
</tr>
<tr>
<td>3</td>
<td>1243</td>
<td>34</td>
<td>3</td>
<td>3</td>
<td>18</td>
<td>12</td>
<td>23</td>
<td>59</td>
<td>0</td>
<td>59</td>
</tr>
<tr>
<td>4</td>
<td>6335</td>
<td>52</td>
<td>17</td>
<td>30</td>
<td>245</td>
<td>42</td>
<td>25</td>
<td>2009</td>
<td>536</td>
<td>2545</td>
</tr>
<tr>
<td>5</td>
<td>1598</td>
<td>16</td>
<td>1</td>
<td>7</td>
<td>58</td>
<td>33</td>
<td>5</td>
<td>missing</td>
<td></td>
<td></td>
</tr>
<tr>
<td>6</td>
<td>3076</td>
<td>41</td>
<td>10</td>
<td>18</td>
<td>153</td>
<td>36</td>
<td>24</td>
<td>869</td>
<td>153</td>
<td>1022</td>
</tr>
<tr>
<td>7</td>
<td>4273</td>
<td>49</td>
<td>14</td>
<td>18</td>
<td>174</td>
<td>42</td>
<td>21</td>
<td>427</td>
<td>0</td>
<td>427</td>
</tr>
<tr>
<td>8</td>
<td>2046</td>
<td>31</td>
<td>7</td>
<td>10</td>
<td>69</td>
<td>28</td>
<td>10</td>
<td>59</td>
<td>11</td>
<td>70</td>
</tr>
<tr>
<td>9</td>
<td>594</td>
<td>20</td>
<td>5</td>
<td>5</td>
<td>89</td>
<td>3</td>
<td>10</td>
<td>63</td>
<td>0</td>
<td>63</td>
</tr>
<tr>
<td>10</td>
<td>476</td>
<td>14</td>
<td>4</td>
<td>4</td>
<td>41</td>
<td>15</td>
<td>5</td>
<td>35</td>
<td>0</td>
<td>35</td>
</tr>
<tr>
<td>11</td>
<td>1623</td>
<td>10</td>
<td>1</td>
<td>2</td>
<td>8</td>
<td>10</td>
<td>5</td>
<td>missing</td>
<td></td>
<td></td>
</tr>
<tr>
<td>12</td>
<td>9444</td>
<td>84</td>
<td>30</td>
<td>44</td>
<td>349</td>
<td>59</td>
<td>32</td>
<td>3749</td>
<td>624</td>
<td>4373</td>
</tr>
<tr>
<td>13</td>
<td>1383</td>
<td>21</td>
<td>2</td>
<td>4</td>
<td>19</td>
<td>31</td>
<td>12</td>
<td>165</td>
<td>0</td>
<td>165</td>
</tr>
<tr>
<td>14</td>
<td>865</td>
<td>18</td>
<td>9</td>
<td>9</td>
<td>133</td>
<td>12</td>
<td>5</td>
<td>missing</td>
<td></td>
<td></td>
</tr>
<tr>
<td>15</td>
<td>3371</td>
<td>47</td>
<td>15</td>
<td>22</td>
<td>179</td>
<td>24</td>
<td>7</td>
<td>292</td>
<td>0</td>
<td>292</td>
</tr>
<tr>
<td>16</td>
<td>304</td>
<td>10</td>
<td>1</td>
<td>1</td>
<td>26</td>
<td>8</td>
<td>5</td>
<td>34</td>
<td>0</td>
<td>34</td>
</tr>
<tr>
<td>17</td>
<td>635</td>
<td>13</td>
<td>2</td>
<td>2</td>
<td>24</td>
<td>7</td>
<td>1</td>
<td>54</td>
<td>0</td>
<td>54</td>
</tr>
<tr>
<td>18</td>
<td>734</td>
<td>17</td>
<td>3</td>
<td>5</td>
<td>54</td>
<td>14</td>
<td>9</td>
<td>50</td>
<td>0</td>
<td>50</td>
</tr>
<tr>
<td>19</td>
<td>744</td>
<td>18</td>
<td>3</td>
<td>3</td>
<td>58</td>
<td>16</td>
<td>7</td>
<td>missing</td>
<td></td>
<td></td>
</tr>
<tr>
<td>20</td>
<td>1343</td>
<td>25</td>
<td>3</td>
<td>4</td>
<td>61</td>
<td>11</td>
<td>12</td>
<td>97</td>
<td>0</td>
<td>97</td>
</tr>
<tr>
<td>21</td>
<td>1861</td>
<td>46</td>
<td>2</td>
<td>3</td>
<td>40</td>
<td>29</td>
<td>27</td>
<td>161</td>
<td>17</td>
<td>178</td>
</tr>
<tr>
<td>22</td>
<td>2267</td>
<td>41</td>
<td>6</td>
<td>15</td>
<td>181</td>
<td>0</td>
<td>32</td>
<td>502</td>
<td>200</td>
<td>702</td>
</tr>
<tr>
<td>23</td>
<td>1651</td>
<td>27</td>
<td>4</td>
<td>5</td>
<td>21</td>
<td>25</td>
<td>10</td>
<td>96</td>
<td>0</td>
<td>96</td>
</tr>
<tr>
<td>24</td>
<td>1194</td>
<td>20</td>
<td>3</td>
<td>4</td>
<td>118</td>
<td>16</td>
<td>9</td>
<td>91</td>
<td>39</td>
<td>130</td>
</tr>
<tr>
<td>25</td>
<td>1029</td>
<td>12</td>
<td>1</td>
<td>2</td>
<td>39</td>
<td>13</td>
<td>6</td>
<td>67</td>
<td>0</td>
<td>67</td>
</tr>
<tr>
<td>26</td>
<td>1327</td>
<td>20</td>
<td>1</td>
<td>10</td>
<td>118</td>
<td>26</td>
<td>1</td>
<td>90</td>
<td>0</td>
<td>90</td>
</tr>
</tbody>
</table>
Table 6.7: Predictors of System Size

LEAST SQUARES REGRESSION

ANALYSIS OF VARIANCE OF 6.LOC N= 26 OUT OF 26

<table>
<thead>
<tr>
<th>SOURCE</th>
<th>DF</th>
<th>SUM SQRS</th>
<th>MEAN SQR</th>
<th>F-STAT</th>
<th>SIGNIF</th>
</tr>
</thead>
<tbody>
<tr>
<td>REGRESSION</td>
<td>3</td>
<td>.14473 +9</td>
<td>.48244 +8</td>
<td>118.85</td>
<td>.0000</td>
</tr>
<tr>
<td>ERROR</td>
<td>22</td>
<td>.89304 +7</td>
<td>.40593 +6</td>
<td></td>
<td></td>
</tr>
<tr>
<td>TOTAL</td>
<td>25</td>
<td>.15366 +9</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

MULT R= .97051 R-SQR= .94188 SE= 637.13

<table>
<thead>
<tr>
<th>VARIABLE</th>
<th>PARTIAL</th>
<th>COEFF</th>
<th>STD ERROR</th>
<th>T-STAT</th>
<th>SIGNIF</th>
</tr>
</thead>
<tbody>
<tr>
<td>CONSTANT</td>
<td>-782.34</td>
<td>224.51</td>
<td>-3.4847</td>
<td>.0021</td>
<td></td>
</tr>
<tr>
<td>MASTERS</td>
<td>.79428</td>
<td>152.42</td>
<td>6.1320</td>
<td>.0000</td>
<td></td>
</tr>
<tr>
<td>SCREENS</td>
<td>.87651</td>
<td>65.967</td>
<td>8.5402</td>
<td>.0000</td>
<td></td>
</tr>
<tr>
<td>REPORT</td>
<td>.48757</td>
<td>43.652</td>
<td>2.6193</td>
<td>.0157</td>
<td></td>
</tr>
</tbody>
</table>
maintain. The distribution of effort was as follows:

<table>
<thead>
<tr>
<th>Phase</th>
<th>Hours</th>
<th>Proportion</th>
</tr>
</thead>
<tbody>
<tr>
<td>Original development:</td>
<td>7500</td>
<td>66%</td>
</tr>
<tr>
<td>Enhancements:</td>
<td>2280</td>
<td>20%</td>
</tr>
<tr>
<td>Maintenance:</td>
<td>1585</td>
<td>14%</td>
</tr>
<tr>
<td>Total:</td>
<td>11365</td>
<td>100%</td>
</tr>
</tbody>
</table>

The total code produced in these 22 systems was 57,000 yielding an overall productivity of 7.6 LOC/Hour for original development and dropping to 5.0 LOC/Hour when enhancement and maintenance hours are included. However, since it is impossible to determine the exact amount of code re-use and possible hours omission these productivity figures may be artificially high.

As a side issue here we see some evidence of surprisingly low maintenance costs for systems built around a fourth generation language. This supports the general practitioners’ position (and 4GL vendors’ position) that 4GLs require less maintenance. An alternative explanation to this conclusion, however, is that the systems in this sample were developed recently and maintenance costs may be yet to emerge. Longitudinal tracking of these systems would bear out this observation.

The relationship between total development hours (original development + enhancements) and lines of code appears in Figure 6.2. The linear regression of hours against LOC results in an $R^2$ of .61. In the absence of the outlier at 750 hours and 9500 LOC
Figure 6.2: Development Hours vs. Lines of Code

SCATTER PLOT
N= 22 OUT OF 30 6.LOC VS. 29.TOTDEVHR

LOC
9509.0 *
7607.2 *
5705.4 *
3803.6 *
1901.8 **
0. *

0. 749.80 1499.6 2249.4 2999.2 TOTDEVHR 3749.0
not only does there appear to be a slight curvature to the relationship but the regression line improves the $R^2$ to .87. The outlier may be due to unreported hours against the system or heavy use of reusable code.

When Development hours were regressed against the three predictors of system size, MASTERS, SCREENS and REPORTS, an $R^2$ of .80 results. However, Masters is the only variable of significance. Screens and Reports drop out of the model. This may indicate that reasonable estimates of development hours are possible when an aggregate measure of data, i.e. master files, is available for simple transaction processing systems written in FOCUS. This observation, it is stressed again, is specific to this one data site and no extendability of the regression co-efficients is suggested or warranted.

6.3. UNIT OF ANALYSIS: FIRM LEVEL

The preceding discussion focused on a single development environment. The findings indicate that the model and method can be used by the firm to better understand its own system development efforts. Of theoretical and practical interest is to determine if the model and method are extendable into other environments. As a first step towards this goal the source code from data site 2 was analyzed with the Code Analyer. Due to the limitations identified earlier only rough comparisons are possible at this time.

The two firms in the study are widely different in function, size, and mission. It would be surprising to find any similarities in the systems they develop unless there is some underlying generalizability to the model and method proposed in this thesis.

Comparison of the two data sites are as follows: The distributions of program size for
Empirical Findings / 137

all programs appears in Figure 6.1 for site 1 and Figure 6.3 for site 2. The means of these two distributions are not statistically different as revealed by an F-test (p > .35). Although the variances are significantly different, this difference could be misleading because the sample size is so large. This also may be an instance where the variances differ significantly but not meaningfully. An alternate plausible explanation is that the distribution having the larger variance comes from the data site with many programmers. We would expect more variance when more people are involved in system building due to any latent design and programming styles. However, this is a post hoc explanation for an observed phenomenon. Program classification similarities appear in both firms with only slightly different frequency distributions. This is likely due to the same language, i.e. FOCUS, being applied to similar kinds of problems. More importantly it shows that system builders use the tool consistently. The similarity in program size distribution indicates initial support for the position that this model and method are generalizable to other FOCUS development environments. It is expected that differences would appear only in the regression parameters when programmer style and application area have their influence on the process of system building.

6.4. SUMMARY AND DISCUSSION

Based upon the regression results presented in section 6.1 we may conclude that: given an accurate detailed design, which specifies the major input and output functions for programs, good predictions of code size are possible, explaining 86%, 78% and 58% of the variance in code size for update, output and control programs respectively. Further, section 6.2 has shown that: given an accurate preliminary design document specifying major system functions and objects such as screens, reports and master files,
<table>
<thead>
<tr>
<th>MIDPOINT</th>
<th>COUNT FOR 5.00C (EACH X = 4)</th>
</tr>
</thead>
<tbody>
<tr>
<td>0.00</td>
<td>58</td>
</tr>
<tr>
<td>25.00</td>
<td>137</td>
</tr>
<tr>
<td>50.00</td>
<td>332</td>
</tr>
<tr>
<td>75.00</td>
<td>200</td>
</tr>
<tr>
<td>100.00</td>
<td>104</td>
</tr>
<tr>
<td>125.00</td>
<td>52</td>
</tr>
<tr>
<td>150.00</td>
<td>37</td>
</tr>
<tr>
<td>175.00</td>
<td>22</td>
</tr>
<tr>
<td>200.00</td>
<td>17</td>
</tr>
<tr>
<td>225.00</td>
<td>22</td>
</tr>
<tr>
<td>250.00</td>
<td>9</td>
</tr>
<tr>
<td>275.00</td>
<td>14</td>
</tr>
<tr>
<td>300.00</td>
<td>13</td>
</tr>
<tr>
<td>325.00</td>
<td>3</td>
</tr>
<tr>
<td>350.00</td>
<td>2</td>
</tr>
<tr>
<td>375.00</td>
<td>3</td>
</tr>
<tr>
<td>400.00</td>
<td>7</td>
</tr>
<tr>
<td>425.00</td>
<td>4</td>
</tr>
<tr>
<td>450.00</td>
<td>1</td>
</tr>
<tr>
<td>475.00</td>
<td>0</td>
</tr>
<tr>
<td>500.00</td>
<td>2</td>
</tr>
<tr>
<td>525.00</td>
<td>2</td>
</tr>
<tr>
<td>550.00</td>
<td>0</td>
</tr>
<tr>
<td>575.00</td>
<td>1</td>
</tr>
<tr>
<td>600.00</td>
<td>0</td>
</tr>
<tr>
<td>625.00</td>
<td>1</td>
</tr>
<tr>
<td>650.00</td>
<td>1</td>
</tr>
<tr>
<td>675.00</td>
<td>0</td>
</tr>
<tr>
<td>700.00</td>
<td>0</td>
</tr>
<tr>
<td>725.00</td>
<td>0</td>
</tr>
<tr>
<td>750.00</td>
<td>0</td>
</tr>
<tr>
<td>775.00</td>
<td>0</td>
</tr>
<tr>
<td>800.00</td>
<td>0</td>
</tr>
<tr>
<td>825.00</td>
<td>1</td>
</tr>
<tr>
<td>850.00</td>
<td>0</td>
</tr>
<tr>
<td>875.00</td>
<td>0</td>
</tr>
<tr>
<td>900.00</td>
<td>0</td>
</tr>
<tr>
<td>925.00</td>
<td>0</td>
</tr>
<tr>
<td>950.00</td>
<td>0</td>
</tr>
<tr>
<td>975.00</td>
<td>0</td>
</tr>
<tr>
<td>1000.00</td>
<td>0</td>
</tr>
</tbody>
</table>

TOTAL 1045 (INTERVAL WIDTH = 25.000)
it is possible to predict overall system size in terms of LOC. These measures of preliminary design size together explain 94% of the variance in code size. This study has not empirically demonstrated the relationship between requirements size and design size. This remains for future work. It has shown that measures of design size explain 80% of the variance in resource consumption for this data set. However, due to the lack of measures of code re-use, motivation or skill level, extension to new development work is unwarrented.

6.4.1. Generalizability

6.4.1.1. System Size

The systems examined in both data sites fall into the category of relatively small transaction processing systems. The results indicate that the linear models employed do quite well in explaining code size. However, our a priori notions of size and complexity lead us to expect that increases in requirements and design size increases complexity and hence should result in a non-linear increase in code size. Examination of the residuals in section 6.1.3 indicates that non-linear terms may be missing from the model. To address the possibility that code size increases non-linearly with increased size of design, and to make the sizing method more generalizable to other settings, second order terms for each of the independent variables were added to the model. This expanded model did not explain any more variance in code size than the simpler linear model. Further, various combinations of interaction terms were tried but did not improve the results. It is anticipated, however, that larger systems will require these second order terms to account for the effects of requirements complexity.
Another alternative for dealing with the non-linearity issue would require performing a log transform on each of the variables. However, in the data set in this study many observations on the independent variables had zero values making the log transform infeasible. When a log transform was taken on just the dependent variable, LOC, and the regression models rerun, $R^2$'s of .59, .62, and .61 resulted for the update, output and control classes respectively. From a practical perspective, however, the log transformed model yields uninterpretable coefficients. This moves away from the objective of simplicity.

6.4.1.2. Model Validation

The generalizability of the regression model for predicting new system development has not been assessed. This would require longitudinal tracking of the system development process which is beyond the scope of this work but may be addressed in future research. Another level of generalization is when a sample of the population can be used to generalize to the remaining systems in the population. This requires that the model be validated against a second set of data. To address this issue a split-half double cross validation was performed on each class of program. This validation entailed the following procedure:

1. Randomly split the sample in half for each program class,
2. Use the first half of the sample to generate regression coefficients,
3. Use the coefficients to predict the second half of the sample,
4. Correlate the predicted values with the actual values,
5. Use the second half of the sample to generate a second set of regression coefficients,
6. Use these coefficients to predict the first half of the sample,
7. Correlate the predicted values with the actual values.

When the above procedure was performed on the data from site 1 correlations pairs
Empirical Findings / 141

of (.93, .90), (.80, .90), (.75, .76) were obtained for the update, output, and control classes respectively, indicating a high degree of internal homogeneity in the data. However, this procedure was carried out primarily for methodological reasons rather than strict model validation. As the sample data represented the entire population of systems it was expected that cross validation would provide these results. In other settings it may be the case that only a sample from the population of systems can be measured. In these instances the possibility of sample bias may influence the regression coefficients. Some measure of this sampling error would be achieved with the above described double cross validation technique. Although some would argue that adjusted R^2 or other shrinkage formulas would achieve the same result (see e.g. Murphy [1983]).

6.4.1.3. Reverse Engineering

Measures of systems design in this thesis were derived from the source code. The design units used are typical of business transaction processing systems. For systems occupying the remaining domain of Figure 2.2, such as scientific or function strong systems, it will likely be the case that design units other than screens, reports etc. would need to be reversed out of the software to reflect design size. Here we would have to assess the outputs from the system analysis and design tools used in each application area. For example, if object oriented design were used to model a system then the design units would have to reflect that orientation.

Moving from design specification back to requirements specification is a more difficult problem. Semantic inferences from the design specification would be required such as
aggregating a number of processes to represent a conceptual input or output event. Further, the identification of entities and relationships in the code would likely require domain knowledge to be effective. Extensive validation between human analysts and the machine reversed requirements would be needed.

6.4.1.4. Forward Engineering

The usefulness of this research approach is limited in application areas where formalized design specification exist and forward engineering compilers are available. For example, communication protocols can be specified in terms of state transition diagrams and their high level design specified in languages such as FDT ESTELLE. From a design specification modelled in ESTELLE, the ESTELLE compiler can generate PASCAL source code. Here the effort to go from design to implementation is minimal. The bulk of the effort expended is going from requirements to design specification. This situation may eventually arise in all application areas as more powerful languages and tools are developed.

6.4.1.5. Hours Data

Several limitations exist in being able to relate hours to the production of software. These come from two distinct sources:

1. Reusable code: Without actual process tracing it is difficult to know what portion of the code was generated from scratch and what portion was imported from similar programs;

2. Personnel differences: Although the results show that programmer style and
learning effects do not significantly affect the size relationship between design and code, this study cannot demonstrate that efficiency differences do or do not exist. While past conventional wisdom suggests that programmer efficiency can vary by a factor of 10, more data is needed to address this. Additionally, as no measures were taken of personnel motivation or actual skill level it would be premature to extend the relationship between hours and code to new development even within the same environment. However, the methodology is now in place to obtain these measures in the future.

6.4.1.6. Language

An issue which is legitimately raised is whether the method and measures used in this research are specific to FOCUS, are generalizable to other 4GL languages, or are generalizable across language generations. While it is premature to say anything conclusive in this area, the issue may be addressed by comparing findings by other researchers and judge whether they are similar enough to entertain the argument that the model and method demonstrated in this thesis have the potential of being generalizable across languages.

Halstead [1977] first theorized and then empirically showed that an interesting relationship exists between the length of a program, measured in the number of tokens, and the number of unique operators and operands used to write the program. He determined that a program's length, N, could be estimated by $\hat{N}$, based on the following formula:
\[ \hat{N} = \log_2 n_1 + n_2 \log_2 n_2 \]

where:

\[ n_1 = \text{number of unique operators (language keywords)} \]
\[ n_2 = \text{number of unique operands (variables)} \]

This surprising relationship is similar to being able to predict the length of a story written in English given only the number of unique verbs and unique nouns in the story.

The correlation found between these two measures for a sample of published PL/1 and Fortran programs, primarily scientific, varied between .83 and .89 depending on who did the research and the language used. If a similar relationship is found in another language (and of presumably higher level) from an independent sample of programs written for a different application area then something more profound and more generalizable is being observed.

The correlation between \( \hat{N} \) and \( N \) for the 703 programs from data site 1 is .89 using simple linear regression. However, what is more revealing is that Halstead’s data shows a curvilinear relationship as program size increases. A plot of the 703 programs shows a similar phenomenon.

Perhaps a more meaningful and powerful comparison would be to compare Halstead’s result at the system level instead of at the program level. The rationale for this is that the programs Halstead looked at were essentially self contained, defined functions.
Similarly, it seems reasonable to consider entire systems as self contained, stable units with defined interfaces. A separate analysis program was written to aggregate Halstead's metrics to the system level. This entailed integrating all of the variables across all programs in each system to arrive at unique variable usage. For example, if VARX was used uniquely in 5 different programs it should only be counted as 1 unique occurrence at the system level. For the 26 systems at data site 1 the correlation between $\hat{N}$ and $N$ is .916 and the scatter plot, shown in Figure 6.4, shows similar behaviour. $\hat{N}$ regressed against $N$ yields an $R^2$ of .84 with the constant term not significantly different from zero. These results indicate that computer languages are much more similar than the ordinal scale of "generations" would lead us to believe. It shows that programs can be broken down and described by their operators and operands.

What the previous discussion does not address is whether the Function Point like measures are language independent. Here, the argument is much easier to demonstrate. Function point measures proposed by Albrecht [1979], De Marco [1982], Jones [1986] and others were and are based on 3rd generation languages, yet they were developed to be used at the design phase, which should be independent of the implementation language. This research has now extended this use into a 4GL language. The data elements counted by the Analyser are linked to requirement specifications and are contained within a program by a design decision. Screen images are the result of design decisions to deal both with data entry and information extraction requirements. Master files are the result of requirements for retained data on some real world entity. Nowhere is the implementation language considered in these units. It may be the case, however, that design orientation such as data flow may be implemented more easily in certain languages relative to others. Taking this one step further, it may also be the
Figure 6.4: Calculated vs. Actual length

N = 26 OUT OF 26  NHAT VS. LENGTH

NHAT

34666.

27733.

20800.

13866.

6933.2

0.

0. 13450. 26901. LENGTH

6725.2 20176. 33626.
case that certain application areas, in this instance transaction processing systems, may naturally take a data flow approach to design. The issue is now removed from considerations of language dependence to a more substantial issue of generalizability across application areas and across design orientations. Future research could address these issues.

6.4.2. Maintenance

In the systems analyzed the main source of system maintenance effort was found to come from users requesting changes or additions to the data elements within the system files. This information was found in the requirements definition and change log of completed systems. These data element changes led to modifications in the data entry routines and in report layouts. Of the systems analyzed maintenance effort resulted both from new field definition and from range or integrity checks on existing fields. While the actual value of the maintenance effort attributable to these changes is beyond the scope of this thesis, one retrospective metric for requirements volatility can be tentatively offered as:

\[
\frac{\text{Number of data element changes}}{\text{Total number of data elements}}
\]

A confidence interval may be constructed around the mean requirements volatility from completed projects. This will give project managers some idea of the potential for requirements changes in their environment as well as the impact of the changes on resource consumption.

An observation from this field research is that there is ambiguity between what people
consider to be "projects" and "systems". Projects should be considered as a measure of resource consumption while information systems are the products of one or more projects. Projects have defined life cycles but it is not necessarily the case that information systems do. Information systems are objects that continue to be used, to grow and be maintained. The problem is that information systems are not stable units. Information systems can be merged with others, or be can redefined. Conceptually, information systems should continue to exist and change as long as the parent organization does. However, specific implementations of information systems may come and go with changes in technology or other environmental change. Hence the whole concept of maintenance may have to be rethought to recognize that systems continue to grow and be modified to meet continual environmental change. The model presented in Figure 3.3 assumes the input and output vectors are stable. However, if these vectors change, i.e. new events occur, then a system must change in order to remain ecologically viable. These new events can be considered as second order environmental change. First order change is that which system software is designed to deal, i.e. the changes within and among the entities and relationships in the retained data model. Second order change involves the definition of new entities, new relationships and new events.

6.4.3. System Decomposition

Average program length in Site 1 is 85 LOC with a lognormal appearance to the distribution. The bulk of the programs appear to be between 40 and 100 LOC. This indicates that, in general, design proceeds down to the level where a unit may be implemented in 1-2 pages of code. On rare occasions programs become quite lengthy
to a maximum of nearly 1000 LOC. This may be for one of several reasons. First, unbalanced design may lead to a programmer being faced with a piece too large to be easily implemented. The converse side to this is that for some kinds of update or reporting situations the complexity of the task quickly rises beyond the language's ability. This also may be due to the design tradeoffs made in the database structure.

An important observation and conclusion can be drawn from this study. If we assume that a given requirements specification may be implemented in many different ways then we would expect to see no consistent relationship, across systems, among the requirements, the design, and the size of code needed to implement those requirements. Each person or system builder would have their own way of solving a given specification. If, on the other hand, size is predictable from requirements and programmer differences are not large then this strongly suggests that human beings follow a reasonably well structured process in problem solving. They may be more or less efficient in carrying out the process, but the process is similar. With respect to computer programming, this phenomenon was first observed and articulated by Halstead [1977] pg. 15:

This finding gains significance when it is remembered that, for every way in which an algorithm can be implemented in agreement with equation (2.7)\(^2\), there are an infinite number of ways in which an equivalent version could be written. This suggests that the human brain obeys a more rigid set of rules than it has been aware of...

\[ N = n_1 \log_2 n_1 + n_2 \log_2 n_2 \]
CHAPTER 7. SUMMARY AND RESEARCH DIRECTIONS

The purpose of this chapter is to summarize the contribution of the research and identify fruitful areas for future investigation, both extending directly from the thesis as well as those ideas raised while conducting the study.

7.1. THESIS SUMMARY

The first issue raised by this thesis was the problem of system sizing. The central problem in estimating resource consumption for information system development is to determine the size of the information system as early in the life cycle as possible. This thesis has argued that the size of requirements transform into size of design, which in turn transform into size of code, and that properties of system requirements remain structurally isomorphic through design into implementable code.

Second, while current estimating models exist, their generalizability has not been demonstrated. It is unlikely that these models can be general without individual site calibration due to wide ranging environmental differences of technology, experience, skills and application area. For a realistic estimating environment to exist, knowledge of past system development efforts is crucial. In order to achieve this it is insufficient to have the knowledge resident in project managers heads as they may either forget or change jobs, but instead it is correct to use a methodology which captures and maintains a database of system metrics.

Third, a prototype research instrument has been implemented to resolve the problems
of computational tractability and measurement reliability. While the regression coefficients found in Chapter 6 are not generalizable beyond the data, the methodology for data capture and individual site calibration is. As it stands, with a few minor enhancements, the automated Code Analyser can be applied to any FOCUS development environment and used to construct regression coefficients for each MIS department.

Fourth, the model and tool have been used to measure two development environments. These findings provide an initial validation of the model and tool, showing that the parsimoneous model can be used to explain program and system size within a limited data set. The specific system development environments have been benchmarked which will allow for longitudinal tracking of individual system changes as well as overall productivity changes that may result from new development methods and technologies.

7.1.1. Empirical Limitations

Naturally, there are a large number of influences on the development effort outside the scope of this empirical work. This thesis recognizes their existence but cannot hope to measure or control all of them. The trade-off made by conducting this field research has been between construct validity and internal validity. The measures that are made of requirements size have high construct validity. The field setting introduces threats to internal validity by the existence of possible confounding variables, i.e. other factors influencing effort which cannot be measured or controlled. Each of the previous studies, identified in Chapter 2, found some common effects and some unique effects. It is likely that each new investigation will also find common as well as unique effects,
reflecting individual environmental differences. While these seemingly unique effects may be considered as such, they may in fact be part of a common factor which has yet to be identified. However, there are simply insufficient studies to begin to converge on an exhaustive set. Rather than to identify all factors, this research has chosen to hold the tools and development environment constant, provide analytic control for the personnel involved and focus on what is believed to be the principal driving force behind effort, namely requirements size. By establishing a stable, causal linkage between requirements, code, and effort it will then be possible to move into other development environments and have a solid ground on which to gather data.

7.2. **DIRECT EXTENSIONS**

The first objective is to extend this research approach into a larger sample of systems and companies, initially within the same language and then to others. The ability to understand the history of a system development environment is an important management goal. The knowledge to construct reasonable estimates of resource consumption is resident in the minds of the systems development staff. These people know the history of system development, the skills each person holds and special circumstances surrounding each system developed. Unfortunately, when senior systems personnel leave, their knowledge leaves with them. Even if staff turnover is low people have selective recall and imperfect evaluation of prior development efforts. However, part of this knowledge base is also resident in the completed systems. They posses the culmination of all effort expended to build them. This thesis has taken the position that analysis of these systems can contribute to management’s understanding of their own development environment. The reasons that this is not yet common practice include:
1. History takes time to develop. Only within the past decade have information systems become widespread. Now, companies possess portfolios of software built in-house. This software can be considered as a database of systems containing valuable information;

2. Manual analysis of software is infeasible. With the use of automated tools, such as the one designed and developed in this thesis, collection of data on past and current systems becomes practical.

Information about past development exists in software and can be extracted and used by management to improve project control:

1. Knowledge about how different programmers and designers have built different systems is useful for resource allocation. It makes sense to assign people to tasks they are best suited to perform.

2. Automated source code analysis allows complex programs to be identified. Maintenance effort will likely result from these programs.

3. Program length as a function of analysis and design information is useful for predicting new development.

4. Calibration of development environments: The objective here is to provide a firm with an understanding of its own people and tool use. Calibration is necessary so that a baseline of productivity can be established. It is anticipated that direct calibration will be required for each firm until a large enough data set is available for generalizations into new environments based upon attributes of those environments.

5. After sufficient external validity is established it will be feasible to make normative statements about the estimating and system development process, i.e.
objects and interactions which are found to consistently affect development can be identified early and managed accordingly.

Extending this method into other languages would require expanding the Code Analyser. Modifications to the low level parsing routines would be required so that languages such as Cobol, PL/1, Fortran or 4th generation languages other than FOCUS could be recognized. This would entail 1) replacing the language Keyword dictionary, 2) changing the parsing delimiters and syntax assumptions, and 3) adding recognition logic for major code chunks. Further, as the FOCUS language evolves enhancements to the existing Analyser will be needed.

7.3. EXPERT BEHAVIOUR

One fruitful area of investigation is the process of estimate construction by practising experts as the bulk of estimating work is done from experience. Experts in the field do quite well even to have an estimate of effort come within several hundred percent of actual effort expended, considering the potential range of software development costs. What this points to is that project managers base their estimates on characteristics of system size and that these characteristics should be measurable. Here we may conjecture that estimators actually perform some mental transformation from the system's requirements into some units (such as LOC) and then apply a productivity rule of thumb. Hence it would be fruitful to engage in observation of practising system developers in a real system feasibility decision situation. Within this category of research several different investigation strategies are feasible. The objective of observing practising system developers in an estimating situation is to understand their problem solving
behaviour and the techniques used to: a) understand the scope of a system; b) obtain user developer interactions, information elicitation and issue clarification; c) understand and process environmental signals.

1. One approach in this area would be to create an example system requirements specification and ask 20 practising analysts to derive an estimate of labour resources needed to implement the example system. The verbal protocols of the analysts could be used to determine which aspects of the problem domain caused more difficulty. The limitations of this approach are that i) the problem domain is artificial, and ii) the complexity of the system must be very simple, i.e. analysts must be able to construct an initial design within a few hours.

2. A second approach would be to move into a live situation. For this approach to be successful it would be necessary to obtain the cooperation of a company about to embark upon a system development project. Ideally, user and developer teams would be brought together into a Group Decision Support situation with the stated objectives of i) capturing system requirements, and ii) establishing a cost estimate. The GDSS environment would allow for online capture of estimating behaviour. The Delphi technique could be used to focus the process and converge on agreement. An integrating extension of this approach would be to allow analysts access to their database of past projects in order to obtain accurate cost data.

7.4. MODELLING

Integration of System Static and System Dynamic modelling: Within the area of conceptual modelling, improvements are needed in the way in which systems are represented conceptually. This thesis has argued that measures of system requirements
are actually measures of complexity. It is this complexity which induces effort in humans to understand, decompose, structure, and otherwise translate into a working machine artifact. The methods we use to model a system affect fundamentally our ability to structure complexity. Further work is needed to incorporate all dimensions of complexity within a unified methodology. This may entail introducing aspects of object oriented analysis and design, modelling the dynamic behaviour (events) of those objects or group of objects (entities) all within a static structural model of the system.

7.5. MANAGEMENT

A central problem facing the software development industry is the lack of consistent record keeping on system development projects. Even when records are kept, a second problem originates from the kinds of records kept. If data on development projects are collected, it is invariably not the right kind of data to construct a meaningful baseline for any estimating model. The problem stems from the fact that data is collected with individual project management and control as the objective and not towards gaining a broader perspective by collecting data with a research framework in mind. In order to obtain a research perspective data must be collected on the units of development work so that these units may be compared across systems.

Many project management systems allow developers to record hours spent on a particular task within their system development methodology. This approach, however, does not identify functional aspects of the software. For a meaningful estimating environment to exist, the units which form the basis for estimating must be the units against which effort is reported. Essentially, the problem is that project management
systems attempt to control the process of system development but not the product. For example, if the estimating approach uses Function Points to size a project then hours must be reported against the delivered function. Likewise if entities, relationships and events are used to form a project estimate then work should be reported as effort is expended analyzing, designing, and implementing those system objects. Additionally, specifics about the activities performed should be noted so that conditions causing further complexity can be examined. Finally, specifics on tool use must be reported for each system development in order to test for productivity gains (or losses) accompanying the installation of new development technologies. Reporting on this basis is crucial for project control.

7.6. **CLOSING REMARKS**

The measurement model and method developed in this thesis assumes that a valid representation of system requirements is available. It does not include such organizational factors as the political decision making process, cross departmental data flows, stability of the environment of the parent organization, structural dynamics, to name a few. It is assumed for this initial study that these factors influence the requirements definition and, mutatis mutandis, result in requirements being more or less complex. Much more work is needed to be able to estimate system size based on system objectives, organizational context and organizational complexity in general. This research represents one small step in that direction.
REFERENCES


APPENDIX A: OPERAND CODING SCHEME
OPERAND CODING SCHEME: AUG.08.1988
CLIVE WRIGLEY

00-19 SCREEN I/O:

01 - DATA ENTRY VARIABLE
02 - DATABASE TURNAROUND VARIABLE
03 - DATABASE DISPLAY VARIABLE
04 - LOCAL VARIABLE DISPLAY
05 - LOCAL VARIABLE TURNAROUND
06 - CHARACTER STRING
07 - POSITIONAL CONTROL
08 - GRAPHIC ATTRIBUTE

20-39 REPORT OUTPUT:

20 - DATABASE VARIABLE
21 - DATABASE VARIABLE IN HEADING
22 - LOCAL VARIABLE IN HEADING
23 - COLUMN HEADING
24 - REPORT HEADINGS
25 - PRINTER POSITIONING

40-49 FILENAMES:

40 - MASTER FILENAME
41 - TEMPORARY FILENAME
42 - INCLUDE FILENAME

50-59 INTERNAL DATA MANIPULATION:

50 - DATABASE VARIABLE MANIPULATION
51 - LOCAL VARIABLE MANIPULATION
52 - NUMERICAL CONSTANT
53 - STRING CONSTANT

60-69 LOGICAL:

60 - LABEL
99 - UNCLASSIFIED
### Appendix B: Pilot Validation - Program Demographics

#### a. Manual Counting: LOC

<table>
<thead>
<tr>
<th></th>
<th>N</th>
<th>MINIMUM</th>
<th>MAXIMUM</th>
<th>MEAN</th>
<th>STD DEV</th>
</tr>
</thead>
<tbody>
<tr>
<td>Control</td>
<td>13</td>
<td>30.000</td>
<td>88.000</td>
<td>65.615</td>
<td>20.855</td>
</tr>
<tr>
<td>Updates</td>
<td>18</td>
<td>15.000</td>
<td>740.00</td>
<td>240.28</td>
<td>226.58</td>
</tr>
<tr>
<td>Outputs</td>
<td>43</td>
<td>10.000</td>
<td>440.00</td>
<td>142.16</td>
<td>94.810</td>
</tr>
</tbody>
</table>

#### b. Automated Counting: LOC

<table>
<thead>
<tr>
<th></th>
<th>N</th>
<th>MINIMUM</th>
<th>MAXIMUM</th>
<th>MEAN</th>
<th>STD DEV</th>
</tr>
</thead>
<tbody>
<tr>
<td>Control</td>
<td>13</td>
<td>29.000</td>
<td>87.000</td>
<td>62.615</td>
<td>22.221</td>
</tr>
<tr>
<td>Updates</td>
<td>18</td>
<td>15.000</td>
<td>910.00</td>
<td>277.39</td>
<td>280.39</td>
</tr>
<tr>
<td>Outputs</td>
<td>43</td>
<td>10.000</td>
<td>427.00</td>
<td>131.05</td>
<td>91.879</td>
</tr>
</tbody>
</table>

#### c. Manual Program Classification: Two Sample T-Test

<table>
<thead>
<tr>
<th></th>
<th>UPDATE</th>
<th>REPORT</th>
<th>TEST STATISTIC</th>
<th>DF</th>
<th>SIGNIF</th>
</tr>
</thead>
<tbody>
<tr>
<td>MEAN</td>
<td>240.28</td>
<td>142.16</td>
<td>T= 2.4008</td>
<td>59</td>
<td>.0195</td>
</tr>
<tr>
<td>VAR</td>
<td>51340.</td>
<td>8989.0</td>
<td>F= 5.7114</td>
<td>18</td>
<td>.0000</td>
</tr>
<tr>
<td>N</td>
<td>13</td>
<td>43</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th></th>
<th>UPDATE</th>
<th>REPORT</th>
<th>TEST STATISTIC</th>
<th>DF</th>
<th>SIGNIF</th>
</tr>
</thead>
<tbody>
<tr>
<td>MEAN</td>
<td>240.28</td>
<td>142.16</td>
<td>T= 2.8726</td>
<td>54</td>
<td>.0058</td>
</tr>
<tr>
<td>VAR</td>
<td>51340.</td>
<td>8989.0</td>
<td>F= 20.668</td>
<td>42</td>
<td>.0000</td>
</tr>
<tr>
<td>N</td>
<td>13</td>
<td>43</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

#### d. Automated Program Classification: Two Sample T-Test

<table>
<thead>
<tr>
<th></th>
<th>UPDATE</th>
<th>REPORT</th>
<th>TEST STATISTIC</th>
<th>DF</th>
<th>SIGNIF</th>
</tr>
</thead>
<tbody>
<tr>
<td>MEAN</td>
<td>277.39</td>
<td>131.05</td>
<td>T= 3.0791</td>
<td>59</td>
<td>.0031</td>
</tr>
<tr>
<td>VAR</td>
<td>78620.</td>
<td>8441.8</td>
<td>F= 9.3132</td>
<td>17</td>
<td>.0000</td>
</tr>
<tr>
<td>N</td>
<td>13</td>
<td>43</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th></th>
<th>UPDATE</th>
<th>REPORT</th>
<th>TEST STATISTIC</th>
<th>DF</th>
<th>SIGNIF</th>
</tr>
</thead>
<tbody>
<tr>
<td>MEAN</td>
<td>131.05</td>
<td>62.615</td>
<td>T= 2.6462</td>
<td>54</td>
<td>.0106</td>
</tr>
<tr>
<td>VAR</td>
<td>8441.8</td>
<td>493.76</td>
<td>F= 17.097</td>
<td>42</td>
<td>.0000</td>
</tr>
<tr>
<td>N</td>
<td>43</td>
<td>13</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th></th>
<th>UPDATE</th>
<th>REPORT</th>
<th>TEST STATISTIC</th>
<th>DF</th>
<th>SIGNIF</th>
</tr>
</thead>
<tbody>
<tr>
<td>MEAN</td>
<td>277.39</td>
<td>62.615</td>
<td>T= 2.7426</td>
<td>29</td>
<td>.0103</td>
</tr>
<tr>
<td>VAR</td>
<td>78620.</td>
<td>493.76</td>
<td>F= 159.23</td>
<td>17</td>
<td>.0000</td>
</tr>
<tr>
<td>N</td>
<td>13</td>
<td>13</td>
<td></td>
<td></td>
<td>.9976</td>
</tr>
</tbody>
</table>
**Appendix B: Pilot Validation - Regression Results**

e. Manual Counting

**LEAST SQUARES REGRESSION: Update Programs**

**ANALYSIS OF VARIANCE OF 5.LOC N= 18 OUT OF 18**

<table>
<thead>
<tr>
<th>SOURCE</th>
<th>DF</th>
<th>SUM SQRS</th>
<th>MEAN SQR</th>
<th>F-STAT</th>
<th>SIGNIF</th>
</tr>
</thead>
<tbody>
<tr>
<td>REGRESSION</td>
<td>2</td>
<td>.75851 +6</td>
<td>.37925 +6</td>
<td>49.783</td>
<td>.0000</td>
</tr>
<tr>
<td>ERROR</td>
<td>15</td>
<td>.11427 +6</td>
<td>7618.1</td>
<td></td>
<td></td>
</tr>
<tr>
<td>TOTAL</td>
<td>17</td>
<td>.87278 +6</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

MULT R = .93224  R-SQR = .86907  SE = 87.282

<table>
<thead>
<tr>
<th>VARIABLE</th>
<th>PARTIAL COEFF</th>
<th>STD ERROR</th>
<th>T-STAT</th>
<th>SIGNIF</th>
</tr>
</thead>
<tbody>
<tr>
<td>CONSTANT</td>
<td>10.804</td>
<td>38.774</td>
<td>.27863</td>
<td>.7843</td>
</tr>
<tr>
<td>8.SCRNVARS</td>
<td>.89319</td>
<td>.21642</td>
<td>7.6927</td>
<td>.0000</td>
</tr>
<tr>
<td>15.MASTERS</td>
<td>.66891</td>
<td>14.179</td>
<td>3.4852</td>
<td>.0033</td>
</tr>
</tbody>
</table>

**LEAST SQUARES REGRESSION: Control Programs**

**ANALYSIS OF VARIANCE OF 5.LOC N= 13 OUT OF 14**

<table>
<thead>
<tr>
<th>SOURCE</th>
<th>DF</th>
<th>SUM SQRS</th>
<th>MEAN SQR</th>
<th>F-STAT</th>
<th>SIGNIF</th>
</tr>
</thead>
<tbody>
<tr>
<td>REGRESSION</td>
<td>1</td>
<td>2514.9</td>
<td>2514.9</td>
<td>10.230</td>
<td>.0085</td>
</tr>
<tr>
<td>ERROR</td>
<td>11</td>
<td>2704.2</td>
<td>245.83</td>
<td></td>
<td></td>
</tr>
<tr>
<td>TOTAL</td>
<td>12</td>
<td>5219.1</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

MULT R = .69417  R-SQR = .48187  SE = 15.679

<table>
<thead>
<tr>
<th>VARIABLE</th>
<th>PARTIAL COEFF</th>
<th>STD ERROR</th>
<th>T-STAT</th>
<th>SIGNIF</th>
</tr>
</thead>
<tbody>
<tr>
<td>CONSTANT</td>
<td>22.000</td>
<td>14.313</td>
<td>1.5371</td>
<td>.1525</td>
</tr>
<tr>
<td>8.SCRNVARS</td>
<td>.69417</td>
<td>3.0046</td>
<td>3.1985</td>
<td>.0085</td>
</tr>
</tbody>
</table>
LEAST SQUARES REGRESSION: Output Programs - Preliminary

ANALYSIS OF VARIANCE OF 4.LOC N= 43 OUT OF 43

<table>
<thead>
<tr>
<th>SOURCE</th>
<th>DF</th>
<th>SUM SQRS</th>
<th>MEAN SQR</th>
<th>F-STAT</th>
<th>SIGNIF</th>
</tr>
</thead>
<tbody>
<tr>
<td>REGRESSION</td>
<td>3</td>
<td>.23903 +6</td>
<td>79678.</td>
<td>26.900</td>
<td>.0000</td>
</tr>
<tr>
<td>ERROR</td>
<td>39</td>
<td>.11552 +6</td>
<td>2962.0</td>
<td></td>
<td></td>
</tr>
<tr>
<td>TOTAL</td>
<td>42</td>
<td>.35455 +6</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

MULT R= .82109 R-SQR= .67418 SE= 54.425

<table>
<thead>
<tr>
<th>VARIABLE</th>
<th>PARTIAL COEFF</th>
<th>STD ERROR</th>
<th>T-STAT</th>
<th>SIGNIF</th>
</tr>
</thead>
<tbody>
<tr>
<td>CONSTANT</td>
<td>-9.8091</td>
<td>20.688</td>
<td>-4.7415</td>
<td>.6380</td>
</tr>
<tr>
<td>5.SCRN</td>
<td>.41143</td>
<td>49.416</td>
<td>2.8190</td>
<td>.0075</td>
</tr>
<tr>
<td>9.FLDS</td>
<td>.19241</td>
<td>.51473</td>
<td>1.2245</td>
<td>.2281</td>
</tr>
<tr>
<td>7.RVAR</td>
<td>.74025</td>
<td>1.6813</td>
<td>6.8758</td>
<td>.0000</td>
</tr>
</tbody>
</table>

LEAST SQUARES REGRESSION: Output Programs - Detailed

ANALYSIS OF VARIANCE OF 4.LOC N= 43 OUT OF 43

<table>
<thead>
<tr>
<th>SOURCE</th>
<th>DF</th>
<th>SUM SQRS</th>
<th>MEAN SQR</th>
<th>F-STAT</th>
<th>SIGNIF</th>
</tr>
</thead>
<tbody>
<tr>
<td>REGRESSION</td>
<td>4</td>
<td>.26343 +6</td>
<td>65858.</td>
<td>27.464</td>
<td>.0000</td>
</tr>
<tr>
<td>ERROR</td>
<td>38</td>
<td>91123.</td>
<td>2398.0</td>
<td></td>
<td></td>
</tr>
<tr>
<td>TOTAL</td>
<td>42</td>
<td>.35455 +6</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

MULT R= .86197 R-SQR= .74299 SE= 48.969

<table>
<thead>
<tr>
<th>VARIABLE</th>
<th>PARTIAL COEFF</th>
<th>STD ERROR</th>
<th>T-STAT</th>
<th>SIGNIF</th>
</tr>
</thead>
<tbody>
<tr>
<td>CONSTANT</td>
<td>-3.4236</td>
<td>18.721</td>
<td>-1.8287</td>
<td>.8559</td>
</tr>
<tr>
<td>5.SCRN</td>
<td>.39640</td>
<td>42.388</td>
<td>2.6617</td>
<td>.0113</td>
</tr>
<tr>
<td>9.FLDS</td>
<td>.16168</td>
<td>.38422</td>
<td>1.0099</td>
<td>.3189</td>
</tr>
<tr>
<td>7.RVAR</td>
<td>.46058</td>
<td>.98873</td>
<td>3.1986</td>
<td>.0028</td>
</tr>
<tr>
<td>22.PANDJ</td>
<td>.45955</td>
<td>11.318</td>
<td>3.1897</td>
<td>.0029</td>
</tr>
</tbody>
</table>
Appendix B: Pilot Validation - Regression Results

f. Automated Counting

LEAST SQUARES REGRESSION: Update Programs

ANALYSIS OF VARIANCE OF 4.LOC  N= 18 OUT OF 18

<table>
<thead>
<tr>
<th>SOURCE</th>
<th>DF</th>
<th>SUM SQRS</th>
<th>MEAN SQR</th>
<th>F-STAT</th>
<th>SIGNIF</th>
</tr>
</thead>
<tbody>
<tr>
<td>REGRESSION</td>
<td>2</td>
<td>.12141</td>
<td>.60706</td>
<td>74.386</td>
<td>.0000</td>
</tr>
<tr>
<td>ERROR</td>
<td>15</td>
<td>.12241</td>
<td>8161.0</td>
<td></td>
<td></td>
</tr>
<tr>
<td>TOTAL</td>
<td>17</td>
<td>.13365</td>
<td>7</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

MULT R=.95310  R-SQR=.90841  SE= 90.338

<table>
<thead>
<tr>
<th>VARIABLE</th>
<th>PARTIAL</th>
<th>COEFF</th>
<th>STD ERROR</th>
<th>T-STAT</th>
<th>SIGNIF</th>
</tr>
</thead>
<tbody>
<tr>
<td>CONSTANT</td>
<td></td>
<td>20.846</td>
<td>.58970</td>
<td>.5642</td>
<td></td>
</tr>
<tr>
<td>6.SVAR</td>
<td>.89655</td>
<td>1.8907</td>
<td>7.8391</td>
<td>.0000</td>
<td></td>
</tr>
<tr>
<td>8.MAS</td>
<td>.67917</td>
<td>40.894</td>
<td>3.5837</td>
<td>.0027</td>
<td></td>
</tr>
</tbody>
</table>

LEAST SQUARES REGRESSION: Control Programs

ANALYSIS OF VARIANCE OF 4.LOC  N= 13 OUT OF 13

<table>
<thead>
<tr>
<th>SOURCE</th>
<th>DF</th>
<th>SUM SQRS</th>
<th>MEAN SQR</th>
<th>F-STAT</th>
<th>SIGNIF</th>
</tr>
</thead>
<tbody>
<tr>
<td>REGRESSION</td>
<td>1</td>
<td>2228.4</td>
<td>2228.4</td>
<td>6.6311</td>
<td>.0258</td>
</tr>
<tr>
<td>ERROR</td>
<td>11</td>
<td>3696.6</td>
<td>336.06</td>
<td></td>
<td></td>
</tr>
<tr>
<td>TOTAL</td>
<td>12</td>
<td>5925.1</td>
<td>1</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

MULT R=.61327  R-SQR=.37610  SE= 18.332

<table>
<thead>
<tr>
<th>VARIABLE</th>
<th>PARTIAL</th>
<th>COEFF</th>
<th>STD ERROR</th>
<th>T-STAT</th>
<th>SIGNIF</th>
</tr>
</thead>
<tbody>
<tr>
<td>CONSTANT</td>
<td></td>
<td>20.632</td>
<td>1.2081</td>
<td>.2523</td>
<td></td>
</tr>
<tr>
<td>6.SVAR</td>
<td>.61327</td>
<td>8.6632</td>
<td>2.5751</td>
<td>.0258</td>
<td></td>
</tr>
</tbody>
</table>
### Least Squares Regression: Output Programs - Preliminary

#### Analysis of Variance of 5[loc] N= 43 Out of 43

<table>
<thead>
<tr>
<th>Source</th>
<th>DF</th>
<th>Sum Sqr</th>
<th>Mean Sqr</th>
<th>F-Stat</th>
<th>Signif</th>
</tr>
</thead>
<tbody>
<tr>
<td>Regression</td>
<td>3</td>
<td>.14230 +6</td>
<td>47432.78</td>
<td>7.8637</td>
<td>.0003</td>
</tr>
<tr>
<td>Error</td>
<td>39</td>
<td>.23524 +6</td>
<td>6031.80</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Total</td>
<td>42</td>
<td>.37754 +6</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

\[
\text{MULT R} = .61393 \quad \text{R-SQR} = .37691 \quad \text{SE} = 77.665
\]

#### Variable Partial Coefficients

<table>
<thead>
<tr>
<th>Variable</th>
<th>Partial Coeff</th>
<th>Std Error</th>
<th>T-Stat</th>
<th>Signif</th>
</tr>
</thead>
<tbody>
<tr>
<td>Constant</td>
<td>9.9198</td>
<td>30.193</td>
<td>.32855</td>
<td>.7443</td>
</tr>
<tr>
<td>6 Screens</td>
<td>.35934</td>
<td>59.260</td>
<td>24.643</td>
<td>.0210</td>
</tr>
<tr>
<td>17 Fields</td>
<td>.34380</td>
<td>1.4506</td>
<td>.63443</td>
<td>.0277</td>
</tr>
<tr>
<td>10 Repvars</td>
<td>.18439</td>
<td>1.0608</td>
<td>.90544</td>
<td>.2485</td>
</tr>
</tbody>
</table>

### Least Squares Regression: Output Programs - Detailed

#### Analysis of Variance of 5[loc] N= 43 Out of 43

<table>
<thead>
<tr>
<th>Source</th>
<th>DF</th>
<th>Sum Sqr</th>
<th>Mean Sqr</th>
<th>F-Stat</th>
<th>Signif</th>
</tr>
</thead>
<tbody>
<tr>
<td>Regression</td>
<td>4</td>
<td>.28235 +6</td>
<td>70587.28</td>
<td>28.178</td>
<td>.0000</td>
</tr>
<tr>
<td>Error</td>
<td>38</td>
<td>95191. 0</td>
<td>2505.00</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Total</td>
<td>42</td>
<td>.37754 +6</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

\[
\text{MULT R} = .86479 \quad \text{R-SQR} = .74786 \quad \text{SE} = 50.050
\]

#### Variable Partial Coefficients

<table>
<thead>
<tr>
<th>Variable</th>
<th>Partial Coeff</th>
<th>Std Error</th>
<th>T-Stat</th>
<th>Signif</th>
</tr>
</thead>
<tbody>
<tr>
<td>Constant</td>
<td>4.9679</td>
<td>19.469</td>
<td>.25517</td>
<td>.8000</td>
</tr>
<tr>
<td>6 Screens</td>
<td>.51285</td>
<td>58.484</td>
<td>3.6825</td>
<td>.0007</td>
</tr>
<tr>
<td>17 Fields</td>
<td>.18069</td>
<td>.48556</td>
<td>1.1325</td>
<td>.2645</td>
</tr>
<tr>
<td>10 Repvars</td>
<td>.15548</td>
<td>.56972</td>
<td>.97025</td>
<td>.3381</td>
</tr>
<tr>
<td>49 Pandj</td>
<td>.77159</td>
<td>18.632</td>
<td>7.4771</td>
<td>.0000</td>
</tr>
</tbody>
</table>
Pilot Validation: Manual counting

<table>
<thead>
<tr>
<th>System</th>
<th>Program</th>
<th>LOC</th>
<th>Scrn</th>
<th>Svar</th>
<th>Rvar</th>
<th>Mast</th>
<th>Flds</th>
<th>Proj</th>
<th>Join</th>
<th>P+J</th>
</tr>
</thead>
<tbody>
<tr>
<td>ASD</td>
<td>ASDX1000</td>
<td>87</td>
<td>2</td>
<td>6</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td></td>
<td>ASDX1020</td>
<td>73</td>
<td>2</td>
<td>5</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td></td>
<td>ASDX1100</td>
<td>88</td>
<td>3</td>
<td>53</td>
<td>0</td>
<td>1</td>
<td>26</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td></td>
<td>ASDX1110</td>
<td>295</td>
<td>7</td>
<td>157</td>
<td>0</td>
<td>1</td>
<td>26</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td></td>
<td>ASDX1130</td>
<td>352</td>
<td>12</td>
<td>84</td>
<td>0</td>
<td>2</td>
<td>51</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td></td>
<td>ASDX4010</td>
<td>92</td>
<td>0</td>
<td>0</td>
<td>24</td>
<td>2</td>
<td>51</td>
<td>0</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td></td>
<td>ASDX4020</td>
<td>116</td>
<td>0</td>
<td>0</td>
<td>12</td>
<td>3</td>
<td>77</td>
<td>0</td>
<td>2</td>
<td>2</td>
</tr>
<tr>
<td></td>
<td>ASDX4030</td>
<td>169</td>
<td>1</td>
<td>3</td>
<td>22</td>
<td>3</td>
<td>77</td>
<td>0</td>
<td>2</td>
<td>2</td>
</tr>
<tr>
<td></td>
<td>ASDX4040</td>
<td>71</td>
<td>0</td>
<td>0</td>
<td>19</td>
<td>2</td>
<td>51</td>
<td>0</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td></td>
<td>ASDX4050</td>
<td>92</td>
<td>0</td>
<td>0</td>
<td>28</td>
<td>2</td>
<td>51</td>
<td>0</td>
<td>2</td>
<td>2</td>
</tr>
<tr>
<td></td>
<td>ASDX4060</td>
<td>52</td>
<td>0</td>
<td>0</td>
<td>19</td>
<td>1</td>
<td>26</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td></td>
<td>ASDX4070</td>
<td>205</td>
<td>1</td>
<td>5</td>
<td>50</td>
<td>2</td>
<td>51</td>
<td>0</td>
<td>2</td>
<td>2</td>
</tr>
<tr>
<td></td>
<td>ASDX4080</td>
<td>31</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td></td>
<td>ASDX4090</td>
<td>141</td>
<td>1</td>
<td>3</td>
<td>19</td>
<td>3</td>
<td>77</td>
<td>0</td>
<td>2</td>
<td>2</td>
</tr>
<tr>
<td></td>
<td>ASDX4100</td>
<td>346</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>6</td>
<td>75</td>
<td>4</td>
<td>0</td>
<td>4</td>
</tr>
<tr>
<td></td>
<td>ASDX4110</td>
<td>140</td>
<td>0</td>
<td>0</td>
<td>62</td>
<td>2</td>
<td>75</td>
<td>1</td>
<td>1</td>
<td>2</td>
</tr>
<tr>
<td></td>
<td>ASDX4120</td>
<td>191</td>
<td>0</td>
<td>0</td>
<td>62</td>
<td>2</td>
<td>75</td>
<td>1</td>
<td>1</td>
<td>2</td>
</tr>
<tr>
<td></td>
<td>ASDX4130</td>
<td>15</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>3</td>
<td>76</td>
<td>1</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td></td>
<td>ASDX4140</td>
<td>180</td>
<td>1</td>
<td>3</td>
<td>20</td>
<td>3</td>
<td>77</td>
<td>2</td>
<td>3</td>
<td>5</td>
</tr>
<tr>
<td></td>
<td>ASDX4150</td>
<td>150</td>
<td>1</td>
<td>3</td>
<td>20</td>
<td>3</td>
<td>77</td>
<td>1</td>
<td>2</td>
<td>3</td>
</tr>
<tr>
<td></td>
<td>ASDX4160</td>
<td>52</td>
<td>0</td>
<td>0</td>
<td>16</td>
<td>2</td>
<td>51</td>
<td>0</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td></td>
<td>ASDX9999</td>
<td>33</td>
<td>1</td>
<td>4</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>CAP</td>
<td>CAPX1000</td>
<td>87</td>
<td>2</td>
<td>6</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td></td>
<td>CAPX1010</td>
<td>72</td>
<td>2</td>
<td>5</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td></td>
<td>CAPX1030</td>
<td>69</td>
<td>2</td>
<td>5</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td></td>
<td>CAPX1040</td>
<td>73</td>
<td>2</td>
<td>5</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td></td>
<td>CAPX1050</td>
<td>67</td>
<td>2</td>
<td>5</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td></td>
<td>CAPX1060</td>
<td>76</td>
<td>2</td>
<td>5</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td></td>
<td>CAPX1070</td>
<td>67</td>
<td>2</td>
<td>5</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td></td>
<td>CAPX1080</td>
<td>76</td>
<td>2</td>
<td>5</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td></td>
<td>CAPX2010</td>
<td>691</td>
<td>11</td>
<td>314</td>
<td>0</td>
<td>4</td>
<td>61</td>
<td>0</td>
<td>3</td>
<td>3</td>
</tr>
<tr>
<td></td>
<td>CAPX2020</td>
<td>740</td>
<td>11</td>
<td>325</td>
<td>0</td>
<td>4</td>
<td>61</td>
<td>0</td>
<td>3</td>
<td>3</td>
</tr>
<tr>
<td></td>
<td>CAPX2030</td>
<td>486</td>
<td>6</td>
<td>83</td>
<td>0</td>
<td>4</td>
<td>61</td>
<td>0</td>
<td>3</td>
<td>3</td>
</tr>
<tr>
<td></td>
<td>CAPX2080</td>
<td>232</td>
<td>1</td>
<td>3</td>
<td>0</td>
<td>1</td>
<td>53</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td></td>
<td>CAPX2090</td>
<td>176</td>
<td>1</td>
<td>8</td>
<td>0</td>
<td>1</td>
<td>53</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td></td>
<td>CAPX2100</td>
<td>422</td>
<td>6</td>
<td>114</td>
<td>0</td>
<td>4</td>
<td>61</td>
<td>0</td>
<td>3</td>
<td>3</td>
</tr>
<tr>
<td></td>
<td>CAPX2110</td>
<td>73</td>
<td>1</td>
<td>7</td>
<td>0</td>
<td>2</td>
<td>57</td>
<td>1</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td></td>
<td>CAPX3010</td>
<td>49</td>
<td>1</td>
<td>7</td>
<td>0</td>
<td>1</td>
<td>3</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td></td>
<td>CAPX3020</td>
<td>40</td>
<td>1</td>
<td>7</td>
<td>0</td>
<td>1</td>
<td>3</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td></td>
<td>CAPX3030</td>
<td>41</td>
<td>1</td>
<td>5</td>
<td>0</td>
<td>1</td>
<td>2</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td></td>
<td>CAPX3040</td>
<td>41</td>
<td>1</td>
<td>5</td>
<td>0</td>
<td>1</td>
<td>2</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td></td>
<td>CAPX4010</td>
<td>118</td>
<td>1</td>
<td>6</td>
<td>55</td>
<td>4</td>
<td>61</td>
<td>0</td>
<td>3</td>
<td>3</td>
</tr>
<tr>
<td></td>
<td>CAPX4020</td>
<td>115</td>
<td>1</td>
<td>7</td>
<td>55</td>
<td>4</td>
<td>61</td>
<td>0</td>
<td>3</td>
<td>3</td>
</tr>
<tr>
<td></td>
<td>CAPX4030</td>
<td>237</td>
<td>1</td>
<td>6</td>
<td>49</td>
<td>4</td>
<td>61</td>
<td>3</td>
<td>5</td>
<td>8</td>
</tr>
<tr>
<td></td>
<td>CAPX4040</td>
<td>11</td>
<td>0</td>
<td>0</td>
<td>5</td>
<td>1</td>
<td>3</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td></td>
<td>CAPX4050</td>
<td>11</td>
<td>0</td>
<td>0</td>
<td>5</td>
<td>1</td>
<td>3</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>System</td>
<td>Program</td>
<td>LOC</td>
<td>Scrn</td>
<td>Svar</td>
<td>Rvar</td>
<td>Mast</td>
<td>Flds</td>
<td>Proj</td>
<td>Join</td>
<td></td>
</tr>
<tr>
<td>--------</td>
<td>----------</td>
<td>-----</td>
<td>------</td>
<td>------</td>
<td>------</td>
<td>------</td>
<td>------</td>
<td>------</td>
<td>------</td>
<td></td>
</tr>
<tr>
<td>ASD</td>
<td>ASDX1000</td>
<td>87</td>
<td>2</td>
<td>6</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td></td>
</tr>
<tr>
<td>ASD</td>
<td>ASDX1020</td>
<td>73</td>
<td>2</td>
<td>5</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td></td>
</tr>
<tr>
<td>ASD</td>
<td>ASDX1100</td>
<td>88</td>
<td>3</td>
<td>52</td>
<td>0</td>
<td>1</td>
<td>26</td>
<td>0</td>
<td>0</td>
<td></td>
</tr>
<tr>
<td>ASD</td>
<td>ASDX1110</td>
<td>295</td>
<td>5</td>
<td>157</td>
<td>0</td>
<td>1</td>
<td>26</td>
<td>0</td>
<td>0</td>
<td></td>
</tr>
<tr>
<td>ASD</td>
<td>ASDX1130</td>
<td>455</td>
<td>13</td>
<td>107</td>
<td>0</td>
<td>2</td>
<td>51</td>
<td>0</td>
<td>0</td>
<td></td>
</tr>
<tr>
<td>ASD</td>
<td>ASDX4010</td>
<td>91</td>
<td>0</td>
<td>0</td>
<td>32</td>
<td>2</td>
<td>51</td>
<td>0</td>
<td>1</td>
<td></td>
</tr>
<tr>
<td>ASD</td>
<td>ASDX4020</td>
<td>116</td>
<td>0</td>
<td>0</td>
<td>29</td>
<td>3</td>
<td>77</td>
<td>1</td>
<td>2</td>
<td></td>
</tr>
<tr>
<td>ASD</td>
<td>ASDX4030</td>
<td>171</td>
<td>1</td>
<td>3</td>
<td>34</td>
<td>3</td>
<td>77</td>
<td>1</td>
<td>2</td>
<td></td>
</tr>
<tr>
<td>ASD</td>
<td>ASDX4040</td>
<td>73</td>
<td>0</td>
<td>0</td>
<td>30</td>
<td>2</td>
<td>51</td>
<td>0</td>
<td>1</td>
<td></td>
</tr>
<tr>
<td>ASD</td>
<td>ASDX4050</td>
<td>92</td>
<td>0</td>
<td>0</td>
<td>30</td>
<td>2</td>
<td>51</td>
<td>0</td>
<td>1</td>
<td></td>
</tr>
<tr>
<td>ASD</td>
<td>ASDX4060</td>
<td>54</td>
<td>0</td>
<td>0</td>
<td>19</td>
<td>1</td>
<td>26</td>
<td>0</td>
<td>0</td>
<td></td>
</tr>
<tr>
<td>ASD</td>
<td>ASDX4070</td>
<td>209</td>
<td>1</td>
<td>5</td>
<td>57</td>
<td>2</td>
<td>51</td>
<td>1</td>
<td>2</td>
<td></td>
</tr>
<tr>
<td>ASD</td>
<td>ASDX4080</td>
<td>31</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td></td>
</tr>
<tr>
<td>ASD</td>
<td>ASDX4090</td>
<td>141</td>
<td>1</td>
<td>3</td>
<td>28</td>
<td>3</td>
<td>77</td>
<td>1</td>
<td>2</td>
<td></td>
</tr>
<tr>
<td>ASD</td>
<td>ASDX4100</td>
<td>348</td>
<td>0</td>
<td>0</td>
<td>28</td>
<td>2</td>
<td>74</td>
<td>4</td>
<td>0</td>
<td></td>
</tr>
<tr>
<td>ASD</td>
<td>ASDX4110</td>
<td>194</td>
<td>0</td>
<td>0</td>
<td>126</td>
<td>2</td>
<td>75</td>
<td>1</td>
<td>1</td>
<td></td>
</tr>
<tr>
<td>ASD</td>
<td>ASDX4120</td>
<td>194</td>
<td>0</td>
<td>0</td>
<td>126</td>
<td>2</td>
<td>75</td>
<td>1</td>
<td>1</td>
<td></td>
</tr>
<tr>
<td>ASD</td>
<td>ASDX4130</td>
<td>15</td>
<td>0</td>
<td>0</td>
<td>2</td>
<td>2</td>
<td>74</td>
<td>1</td>
<td>0</td>
<td></td>
</tr>
<tr>
<td>ASD</td>
<td>ASDX4140</td>
<td>178</td>
<td>1</td>
<td>3</td>
<td>32</td>
<td>3</td>
<td>77</td>
<td>2</td>
<td>3</td>
<td></td>
</tr>
<tr>
<td>ASD</td>
<td>ASDX4150</td>
<td>150</td>
<td>1</td>
<td>3</td>
<td>28</td>
<td>3</td>
<td>77</td>
<td>1</td>
<td>2</td>
<td></td>
</tr>
<tr>
<td>ASD</td>
<td>ASDX4160</td>
<td>56</td>
<td>0</td>
<td>0</td>
<td>27</td>
<td>2</td>
<td>51</td>
<td>0</td>
<td>1</td>
<td></td>
</tr>
<tr>
<td>ASD</td>
<td>ASDX9999</td>
<td>34</td>
<td>1</td>
<td>5</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td></td>
</tr>
<tr>
<td>CAP</td>
<td>CAPC2010</td>
<td>869</td>
<td>10</td>
<td>32</td>
<td>0</td>
<td>6</td>
<td>67</td>
<td>0</td>
<td>5</td>
<td></td>
</tr>
<tr>
<td>CAP</td>
<td>CAPC2020</td>
<td>910</td>
<td>11</td>
<td>349</td>
<td>0</td>
<td>6</td>
<td>67</td>
<td>0</td>
<td>5</td>
<td></td>
</tr>
<tr>
<td>CAP</td>
<td>CAPC2030</td>
<td>552</td>
<td>6</td>
<td>102</td>
<td>0</td>
<td>6</td>
<td>67</td>
<td>0</td>
<td>5</td>
<td></td>
</tr>
<tr>
<td>CAP</td>
<td>CAPC2040</td>
<td>75</td>
<td>0</td>
<td>2</td>
<td>0</td>
<td>2</td>
<td>63</td>
<td>0</td>
<td>0</td>
<td></td>
</tr>
<tr>
<td>CAP</td>
<td>CAPC2050</td>
<td>75</td>
<td>0</td>
<td>2</td>
<td>0</td>
<td>2</td>
<td>61</td>
<td>0</td>
<td>0</td>
<td></td>
</tr>
<tr>
<td>CAP</td>
<td>CAPC2080</td>
<td>256</td>
<td>1</td>
<td>16</td>
<td>0</td>
<td>6</td>
<td>67</td>
<td>0</td>
<td>5</td>
<td></td>
</tr>
<tr>
<td>CAP</td>
<td>CAPC2090</td>
<td>208</td>
<td>1</td>
<td>14</td>
<td>0</td>
<td>6</td>
<td>67</td>
<td>0</td>
<td>5</td>
<td></td>
</tr>
<tr>
<td>CAP</td>
<td>CAPC2100</td>
<td>497</td>
<td>6</td>
<td>135</td>
<td>0</td>
<td>6</td>
<td>67</td>
<td>0</td>
<td>5</td>
<td></td>
</tr>
<tr>
<td>CAP</td>
<td>CAPC3010</td>
<td>45</td>
<td>3</td>
<td>6</td>
<td>0</td>
<td>1</td>
<td>3</td>
<td>0</td>
<td>0</td>
<td></td>
</tr>
<tr>
<td>CAP</td>
<td>CAPC3020</td>
<td>45</td>
<td>3</td>
<td>6</td>
<td>0</td>
<td>1</td>
<td>3</td>
<td>0</td>
<td>0</td>
<td></td>
</tr>
<tr>
<td>CAP</td>
<td>CAPC3030</td>
<td>41</td>
<td>3</td>
<td>4</td>
<td>0</td>
<td>1</td>
<td>2</td>
<td>0</td>
<td>0</td>
<td></td>
</tr>
<tr>
<td>CAP</td>
<td>CAPC3040</td>
<td>41</td>
<td>3</td>
<td>4</td>
<td>0</td>
<td>1</td>
<td>2</td>
<td>0</td>
<td>0</td>
<td></td>
</tr>
<tr>
<td>CAP</td>
<td>CAPC3050</td>
<td>47</td>
<td>3</td>
<td>12</td>
<td>0</td>
<td>1</td>
<td>6</td>
<td>0</td>
<td>0</td>
<td></td>
</tr>
<tr>
<td>CAP</td>
<td>CAPX1000</td>
<td>85</td>
<td>2</td>
<td>6</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td></td>
</tr>
<tr>
<td>CAP</td>
<td>CAPX1010</td>
<td>83</td>
<td>2</td>
<td>5</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td></td>
</tr>
<tr>
<td>CAP</td>
<td>CAPX1030</td>
<td>75</td>
<td>2</td>
<td>6</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td></td>
</tr>
<tr>
<td>CAP</td>
<td>CAPX1040</td>
<td>73</td>
<td>2</td>
<td>6</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td></td>
</tr>
<tr>
<td>CAP</td>
<td>CAPX1050</td>
<td>66</td>
<td>2</td>
<td>5</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td></td>
</tr>
<tr>
<td>CAP</td>
<td>CAPX1060</td>
<td>75</td>
<td>2</td>
<td>5</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td></td>
</tr>
<tr>
<td>CAP</td>
<td>CAPX1070-</td>
<td>70</td>
<td>2</td>
<td>5</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td></td>
</tr>
<tr>
<td>CAP</td>
<td>CAPX2010</td>
<td>11</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>6</td>
<td>67</td>
<td>0</td>
<td>5</td>
<td></td>
</tr>
<tr>
<td>CAP</td>
<td>CAPX2020</td>
<td>11</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>6</td>
<td>67</td>
<td>0</td>
<td>5</td>
<td></td>
</tr>
<tr>
<td>CAPX</td>
<td>10</td>
<td>100</td>
<td>1000</td>
<td>10000</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>---------------</td>
<td>------</td>
<td>------</td>
<td>------</td>
<td>-------</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>CAPX2030</td>
<td>44</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>CAPX2080</td>
<td>11</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>CAPX2090</td>
<td>18</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>CAPX2100</td>
<td>11</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>CAPX2110</td>
<td>73</td>
<td>1</td>
<td>10</td>
<td>0</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>CAPX2120</td>
<td>101</td>
<td>1</td>
<td>15</td>
<td>0</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>CAPX2130</td>
<td>68</td>
<td>1</td>
<td>9</td>
<td>0</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>CAPX2140</td>
<td>76</td>
<td>1</td>
<td>9</td>
<td>0</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>CAPX2150</td>
<td>89</td>
<td>1</td>
<td>3</td>
<td>16</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>CAPX3010</td>
<td>4</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>CAPX3020</td>
<td>4</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>CAPX3030</td>
<td>4</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>CAPX3040</td>
<td>4</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>CAPX3050</td>
<td>4</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>CAPX3110</td>
<td>51</td>
<td>1</td>
<td>3</td>
<td>4</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>CAPX3120</td>
<td>51</td>
<td>1</td>
<td>3</td>
<td>4</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>CAPX3130</td>
<td>51</td>
<td>1</td>
<td>3</td>
<td>4</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>CAPX3140</td>
<td>51</td>
<td>1</td>
<td>3</td>
<td>4</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>CAPX3150</td>
<td>51</td>
<td>1</td>
<td>3</td>
<td>4</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>CAPX4010</td>
<td>114</td>
<td>1</td>
<td>6</td>
<td>42</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>CAPX4020</td>
<td>122</td>
<td>1</td>
<td>7</td>
<td>44</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>CAPX4030</td>
<td>245</td>
<td>1</td>
<td>6</td>
<td>112</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>CAPX4040</td>
<td>11</td>
<td>0</td>
<td>0</td>
<td>8</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>CAPX4050</td>
<td>11</td>
<td>0</td>
<td>0</td>
<td>8</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>CAPX4060</td>
<td>10</td>
<td>0</td>
<td>0</td>
<td>6</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>CAPX4070</td>
<td>10</td>
<td>0</td>
<td>0</td>
<td>6</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>CAPX4080</td>
<td>17</td>
<td>0</td>
<td>0</td>
<td>16</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>CAPX5010</td>
<td>62</td>
<td>1</td>
<td>4</td>
<td>2</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>CAPX5020</td>
<td>196</td>
<td>1</td>
<td>10</td>
<td>19</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>CAPX5030</td>
<td>63</td>
<td>0</td>
<td>0</td>
<td>14</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>CAPX5040</td>
<td>59</td>
<td>0</td>
<td>4</td>
<td>0</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>CAPX5050</td>
<td>25</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>CAPX6010</td>
<td>71</td>
<td>1</td>
<td>4</td>
<td>42</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>CAPX6020</td>
<td>106</td>
<td>1</td>
<td>4</td>
<td>45</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>CAPX6030</td>
<td>62</td>
<td>1</td>
<td>4</td>
<td>32</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>CAPX6040</td>
<td>147</td>
<td>1</td>
<td>4</td>
<td>72</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>CAPX6050</td>
<td>254</td>
<td>1</td>
<td>4</td>
<td>101</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>CAPX6060</td>
<td>171</td>
<td>1</td>
<td>4</td>
<td>82</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>CAPX6070</td>
<td>160</td>
<td>1</td>
<td>4</td>
<td>82</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>CAPX6080</td>
<td>59</td>
<td>1</td>
<td>4</td>
<td>22</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>CAPX6090</td>
<td>427</td>
<td>1</td>
<td>3</td>
<td>96</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>CAPX6100</td>
<td>405</td>
<td>1</td>
<td>3</td>
<td>96</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>CAPX6110</td>
<td>116</td>
<td>1</td>
<td>3</td>
<td>36</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>CAPX6120</td>
<td>157</td>
<td>1</td>
<td>6</td>
<td>59</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>CAPX6130</td>
<td>74</td>
<td>1</td>
<td>4</td>
<td>44</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>CAPX6140</td>
<td>50</td>
<td>1</td>
<td>4</td>
<td>29</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>CAPX6150</td>
<td>99</td>
<td>1</td>
<td>3</td>
<td>24</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>CAPX6160</td>
<td>109</td>
<td>1</td>
<td>3</td>
<td>41</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>CAPX7010</td>
<td>70</td>
<td>1</td>
<td>4</td>
<td>42</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>CAPX7020</td>
<td>33</td>
<td>1</td>
<td>4</td>
<td>0</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>CAPX7040</td>
<td>144</td>
<td>0</td>
<td>0</td>
<td>91</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>CAPX7050</td>
<td>254</td>
<td>0</td>
<td>0</td>
<td>137</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>CAPX7060</td>
<td>190</td>
<td>0</td>
<td>0</td>
<td>120</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>CAPX7070</td>
<td>190</td>
<td>0</td>
<td>0</td>
<td>120</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>---</td>
<td>---</td>
<td>---</td>
<td>---</td>
<td>---</td>
<td>---</td>
<td>---</td>
<td>---</td>
<td>---</td>
<td>---</td>
<td></td>
</tr>
<tr>
<td>CAPX7080</td>
<td>62</td>
<td>0</td>
<td>0</td>
<td>29</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td></td>
<td></td>
</tr>
<tr>
<td>CAPX7090</td>
<td>52</td>
<td>1</td>
<td>4</td>
<td>20</td>
<td>2</td>
<td>53</td>
<td>0</td>
<td>1</td>
<td></td>
<td></td>
</tr>
<tr>
<td>CAPX8010</td>
<td>63</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td></td>
<td></td>
</tr>
<tr>
<td>CAPX8020</td>
<td>34</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td></td>
<td></td>
</tr>
<tr>
<td>CAPX9999</td>
<td>29</td>
<td>1</td>
<td>5</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

*NEW PROGRAMS*
FILENAME = SYSTEMS, SUFFIX = FOC,$
SEGNAME = COMPANY, SEGTYPE = S1,$
    FIELDDNAME = COMPANY(NAME, ALIAS = FIRM, FORMAT = A8, FIELDTYPE = I,$
SEGNAME = SYSNAMES, PARENT = COMPANY, SEGTYPE = S1,$
    FIELDDNAME = SYSTEM(NAME, ALIAS = SYSTEM, FORMAT = A8, FIELDTYPE = I,$
SEGNAME = PROJNAME, PARENT = SYSNAMES, SEGTYPE = S1,$
    FIELDDNAME = PROJID, ALIAS = , FORMAT = A20, FIELDTYPE = I,$
SEGNAME = PROJINFO, PARENT = PROJNAME, SEGTYPE = KU,
    CRSEGNAME = PROJINFO, CRKEY = PROJID, CRFILE = PROJECTS,$
SEGNAME = PROGRAMS, PARENT = SYSNAMES, SEGTYPE = S1,$
    FIELDDNAME = PROGRAMNAME, ALIAS = FOCNAME, FORMAT = A8, FIELDTYPE = I,$
SEGNAME = BUILDER, PARENT = PROGRAMS, SEGTYPE = S1,$
    FIELDDNAME = PROGRAMMER, FORMAT = A8, FIELDTYPE = I,$
SEGNAME = LINES, PARENT = PROGRAMS, SEGTYPE = S0,$
    FIELDDNAME = LOC, FORMAT = A80,$
SEGNAME = OPERANDS, PARENT = LINES, SEGTYPE = S0,$
    FIELDDNAME = OPERAND, FORMAT = A12,$
    FIELDDNAME = OPERAND_TYPE, FORMAT = I2,$
SEGNAME = VOCAB, PARENT = LINES, SEGTYPE = S0,$
    FIELDDNAME = KEYWORD, FORMAT = A12,$
SEGNAME = FILES, PARENT = PROGRAMS, SEGTYPE = S1,$
    FIELDDNAME = FILENAME, FORMAT = A8, FIELDTYPE = I,$
    FIELDDNAME = #FILEDS, ALIAS = , FORMAT = I5,$
    FIELDDNAME = #SEGMENTS, ALIAS = , FORMAT = I5,$
    FIELDDNAME = #INDEXES, ALIAS = , FORMAT = I2,$
    FIELDDNAME = #FILES, ALIAS = , FORMAT = I2,$
    FIELDDNAME = F_LENGTH, ALIAS = , FORMAT = I4,$
SEGNAME = MASTERS, PARENT = SYSNAMES, SEGTYPE = S1,$
    FIELDDNAME = MASTERNAME, FORMAT = A8, FIELDTYPE = I,$
    FIELDDNAME = NUM_FILEDS, ALIAS = FIELDS, FORMAT = I5,$
    FIELDDNAME = NUM_SEGMENTS, ALIAS = SEGMENTS, FORMAT = I5,$
    FIELDDNAME = NUM_INDEXES, ALIAS = INDEXES, FORMAT = I2,$
    FIELDDNAME = NUM_FILES, ALIAS = , FORMAT = I2,$
    FIELDDNAME = FIELD_LENGTH, ALIAS = TOTAL_LENGTH, FORMAT = I4,$
MASTER FILE DEFINITION: PROJECTS

DATE WRITTEN: JUL.18.1988 BY: C.D. WRIGLEY
MODS:

FILENAME = PROJECTS,SUFFIX = FOC

SEGNAME = PROJINFO,SEGTYPE = S1
FIELDNAME = PROJID,ALIAS = PROJID,FORMAT = A20,FIELDTYPE = I,$
FIELDNAME = PROJNAME,ALIAS = ,FORMAT = A40,$

SEGNAME = SYSTEMS,PARENT = PROJINFO,SEGTYPE = S1,$
FIELDNAME = SYSTEM_NAME,ALIAS = ,FORMAT = A12,FIELDTYPE = I,$

SEGNAME = SYSINFO,PARENT = SYSTEMS,SEGTYPE = KU,
CRSEGNAME = SYSNAMES,CRFILE = SYSTEMS,CRKEY = SYSTEM_NAME,$

SEGNAME = RESOURCE,PARENT = PROJINFO,SEGTYPE = S1,$
FIELDNAME = RESOURCEID,ALIAS = ,FORMAT = A20,$
FIELDNAME = SKILL_LEVEL,ALIAS = ,FORMAT = I2,$

SEGNAME = TASKS,PARENT = PROJINFO,SEGTYPE = S1,$
FIELDNAME = TASKID,ALIAS = ,FORMAT = A12,FIELDTYPE = I,$

SEGNAME = WORKDONE,PARENT = TASKS,SEGTYPE = S1,$
FIELDNAME = WORK_UNITS,ALIAS = ,FORMAT = I4,$

SEGNAME = TOOLS,PARENT = PROJINFO,SEGTYPE = U,$
FIELDNAME = LANGUAGE,ALIAS = ,FORMAT = A8,$
FIELDNAME = METHODOLOGY,ALIAS = ,FORMAT = A20,$
FIELDNAME = HARDWARE,ALIAS = ,FORMAT = A20,$
FIELDNAME = SAD_TOOLS,ALIAS = ,FORMAT = A20,$
APPENDIX D: CODE ANALYSER SOURCE CODE
* PROGRAM: SYSLIST

* THIS EXEC TAKES TWO INPUTS:
* THE FIRST IS NAME OF THE COMPANY TO BE USED AS AN IDENTIFIER
* IN THE SYSTEMS DATABASE.
* THE SECOND IS THE PATHNAME OF A DIRECTORY
* WHERE THE COMPANY'S SYSTEMS TO BE REVERSE ENGINEERED ARE LOCATED.
* ITS GOOD POLICY TO USE THE DIRECTORY NAME
* AS THE SYSTEM NAME. FUTURE MODS WILL DO THIS AUTOMATICALLY.
* THE PROGRAM CONSTRUCTS A LIST OF SYSTEMS AND STORES THIS LIST IN
* SYSLIST.DIR IN THE COMPANY'S ROOT DIRECTORY
* AUG.08.88 CLIVE WRIGLEY

************************************************************************

DOS ERASE FOCSTACK.FTM
-RUN
-SET &STACK=ON;
-DOS FILEDEF COMPANY DISK D:COMPANY.PTR
-DOS FILEDEF SYSLIST DISK D:SYSLIST.DIR

-FIRST GET THE LOCATION OF THE SYSTEMS TO BE COUNTED
-TYPE ENTER COMPANY NAME (DIRECTORY CONTAINING THE SYSTEMS.)
-PROMPT &&COMPANYA8.ENTER ABREVIATED NAME OF COMPANY (8 CHARS).
-PROMPT &&DRIVE.ENTER DRIVE AND PATH WERE COMPANY DATA IS STORED.
-SET &&PATH = &&DRIVE | | &&COMPANY;

-DOS CD &&PATH
-* store the company name for future reference
-WRITE COMPANY &&COMPANY

-TYPE CONSTRUCTING LIST OF SYSTEMS IN: &&PATH
-EXEC GETDIRS PATH=&&PATH
-RUN
DOS C:
-RUN

************************************************************************
PROGRAM: SYSDIRS

* THIS EXEC TAKES TWO INPUTS:
* THE FIRST IS NAME OF THE COMPANY TO BE USED AS AN IDENTIFIER
* IN THE SYSTEMS DATABASE.
* THE SECOND IS THE PATHNAME OF A DIRECTORY
* WHERE THE COMPANY'S SYSTEMS TO BE REVERSE ENGINEERED ARE LOCATED.
* THE PROGRAM CONSTRUCTS LISTS OF FOCEXEC AND MASTER FILE DEFINITIONS
* FOR EACH NAMES FOUND IN SYSLIST.DIR IN THE COMPANY'S DIRECTORY
* THESE LISTS ARE STORED IN FEXLIST.DIR AND MASLIST.DIR IN THE SUB-
* DIRECTORIES CONTAINING THE ACTUAL SOURCE CODE.
* ADDITIONALLY THE COMPANY AND SYSTEM NAME ARE STORED IN SYSNAME.PTR
* AUG.08.88 C.D. WRIGLEY

DOS ERASE FOCSTACK.FTM
-RUN
-SET &STACK=ON;
-DOS FILEDEFF COMPANY DISK D:COMPANY.PTR
-DOS FILEDEFF SYSLIST DISK D:SYSLIST.DIR

*FIRST GET THE LOCATION OF THE SYSTEMS TO BE COUNTED
-TYPE ENTER COMPANY NAME (DIRECTORY CONTAINING THE SYSTEMS.)
-PROMPT &&COMPANYA8.ENTER ABREVIATED NAME OF COMPANY (8 CHARS).
-PROMPT &&DRIVE.ENTER DRIVE AND PATH WERE COMPANY DATA IS STORED.
-SET &&PATH = &&DRTVE
11
-DOS CD &&PATH
-DOS STATE D:SYSLIST.DIR
-IF &RETCODE NE 0 THEN GOTO NOSYS;
-DOS CD &&PATH
-GETSYS
-READ SYSLIST &&SYSNAMEA8.
-IF &IORETURN NE 0 THEN GOTO ENDSYS;
-* CHANGE DIRECTORY TO SYSNAME
-SET &&SYSPATH = &&PATH || \ &&SYSNAME;
-DOS CD &&SYSPATH
-* STORE THE SYSTEM NAME FOR LATER ACCESS
-TYPE STORING NAME FOR &&SYSNAME SYSTEM IN &&SYSPATH
-DOS FILEDEFF SYSNAME DISK D:SYSNAME.PTR
-WRITE SYSNAME &&COMPANY
-WRITE SYSNAME &&SYSNAME

-TYPE CONSTRUCTING FOCEXEC AND MASTER LISTS FOR: &&SYSNAME
-DOS CD &&SYSPATH
-EXEC GETNAMES EXTENSION='MAS'
-EXEC GETNAMES EXTENSION='FEX'
-DOS CD &&PATH
-GOTO GETSYS

-NOSYS
-TYPE **** NO SUBDIRECTORIES FOUND FOR: &COMPANY. TERMINATING RUN.
-GOTO EXIT
-ENDSYS
-RUN
DOS C:
-RUN
* PROGRAM: GETDIRS

* THIS PROGRAM CONSTRUCTS A LIST OF SUB-DIRECTORIES IN THE PATH
* SPECIFIED BY THE CALLING PROCEDURE. THE FILE TEMP.DIR IS CREATED IN THE
* CURRENT DRIVE AND DIRECTORY ALONG WITH SYSLIST.DIR
* CLIVE WRIGLEY JULY.11.88

-DEFAULTS &DRIVE='D:'
-DEFAULTS &RTNDRIVE='C:'
DOS &DRIVE
DOS CD &PATH
DOS DIR *.
>TEMP.DIR
-Run

-SET &SYSLIST = 'D:SYSLIST.DIR';
-DOS FILEDEF SYSLIST DISK &SYSLIST
-DOS FILEDEF DIRLIST DISK TEMP.DIR
-TOP
-READ DIRLIST &FILENAME.A8. &EXTENSION.A5. &DIR.A5.
-IF &IORETURN NE 0 THEN GOTO DONE;
-IF &DIR NE '<DIR>' OR (&FILENAME EQ '.' OR '..') THEN GOTO TOP;
-WRITE SYSLIST &FILENAME
-GOTO TOP
-DONE
DOS ERASE TEMP.DIR
DOS &RTNDRIVE
-Run
**PROGRAM: GETNAMES**

*• THIS PROGRAM CONSTRUCTS A LIST OF DOS FILENAMES HAVING THE EXTENSION
*• SPECIFIED BY THE CALLING PROCEDURE. THE FILE TEMP.DIR IS CREATED IN THE
*• CURRENT DRIVE AND DIRECTORY ALONG WITH XXXLIST.DAT
*• CLIVE WRIGLEY JULY.11.88

-DEFAULTS &DRIVE='D:'
-DEFAULTS &RTNDRIVE='C:'
DOS &DRIVE
DOS DIR *.&EXTENSION >TEMP.DIR
-RUN
-SET &FILELIST = &EXTENSION || 'LIST.DIR';
-DOS FILEDEF FILELIST DISK &FILELIST
-DOS FILEDEF DIRLIST DISK TEMP.DIR
-TOP
-IF &IORETURN NE 0 THEN GOTO DONE;
-IF &FILE1 EQ ' ' THEN GOTO TOP;
-SET &FILENAME = &FILE1 || &FILE2;
-WRITE FILELIST &FILENAME
-GOTO TOP
-DONE
DOS ERASE TEMP.DIR
DOS &RTNDRIVE
-RUN
* PROGRAM TO COUNT MASTERFILE DEFINITIONS
* &MASNAME IS PASSED FROM SYSCOUNT
* MASTER FILE LIST IS IN D:MASLIST.DIR
* SYSTEM NAME IS IN D:SYSNAME.PTR OF THE CURRENT DIRECTORY
* CLIVE WRIGLEY JUL.08.88
* MODS: SE.08.88 CHANGED NAME OF LOG FILE TO UPDATE.LOG

-DOS FILEDEF SYSNAME DISK D:SYSNAME.PTR
FILEDEF SYSTEMS DISK C:SYSTEMS.FOC
FILEDEF LOGFILE DISK C:UPDATE.LOG APPEND

* CHECK MASTERFILE DEMOGRAPHICS
DOS D;
CHECK FILE &MASNAME HOLD
DOS C:

TABLE FILE HOLD
COUNT SEGNAME AND CNT.FIELDNAME AND SUM.SKEYS BY FILENAME
ON TABLE HOLD AS HOLD2
END

MODIFY FILE SYSTEMS
FIXFORM ON HOLD2 MASTERNAMES/8 NUM_SEGMENTS/5 NUM_FIELDS/5 NUM_INDEXES/2
GOTO MASADD

CASE AT START
  COMPUTE STUPID/A8 = ";
GOTO FINDFIRM
ENDCASE

CASE FINDFIRM
  TYPE "*
  FIXFORM ON SYSNAME COMPANYNAME/8
  FIXFORM ON SYSNAME SYSTEMNAME/8
  MATCH COMPANYNAME
    ON NOMATCH TYPE "ADDING COMPANY: <COMPANYNAME TO DATABASE"
    ON NOMATCH TYPE ON LOGFILE "ADDING COMPANY: <COMPANYNAME TO DATABASE"
    ON NOMATCH INCLUDE
  GOTO FINDSYS
ENDCASE

CASE FINDSYS
  TYPE "*
  MATCH SYSTEMNAME
    ON NOMATCH TYPE "ADDING SYSTEM: <SYSTEMNAME TO DATABASE"
    ON NOMATCH TYPE ON LOGFILE "ADDING SYSTEM: <SYSTEMNAME TO DATABASE"
    ON NOMATCH INCLUDE
ENDCASE

CASE MASADD
  MATCH MASTERNAMES
    ON MATCH TYPE "DUPLICATE MASTER: <SYSTEMNAME <MASTERNAMES"
    ON MATCH TYPE ON LOGFILE "DUPLICATE MASTER: <SYSTEMNAME <MASTERNAMES"
    ON MATCH REJECT
    ON NOMATCH TYPE ON LOGFILE "ADDING MASTER: <SYSTEMNAME <MASTERNAMES"
ON NOMATCH TYPE "ADDING MASTER <MASTERNAME"
ON NOMATCH INCLUDE
ENDCASE
DATA
END
PGM: PGMCOUNT.FEX

* VERSION AS OF AUG.07.88

* PRINCIPAL CHANGES:
* 1) OPERANDS ARE CATEGORIZED ACCORDING TO THE OPERAND CODING SCHEME IN FOCUS\ANALYSER\OPCODES.DOC
* 2) TOKENS ARE STORED BELOW LOC AND NOT IN ANY ORDER AS OPPOSED TO BELOW PROGRAMNAME AND IN ORDER.
* 3) TOKEN COUNTING IN NOT DONE ANY MORE. THIS FUNCTION IS HANDLED BY 'TABLE' REQUESTS

* MODS: SE.07.88 FILENAME MAY CONTAIN SEGMENT INFO.THIS IS PARSED OUT EG. MODIFY FILE NAME.SEG

* SYSTEMNAME IS STORED IN SYSNAME.PTR IN THE CURRENT DIRECTORY &FOCEXEC IS PASSED FROM SYSCOUNT
* CLIVE WRIGLEY JULY.88

DOS FILEDEF SYSNAME DISK D:SYSNAME.PTR
DEFAULTS &OUTFILE1=C:SYSTEMS.FOC
DEFAULTS &LOOKUP1=C:FOCWORDS.FOC
SET &FOCFILE = 'D:' || &FOCEXEC || '.FEX';

THIS SECTION READS IN A TEXTFILE 'FOCFILE' INTO A FOCUS FILE 'SYSTEMS'
* EACH LINE OF TEXT IN TEXTFILE IS CHOPPED INTO TOKENS
* TABLE LOOK UP IS PERFORMED TO DETERMINE IF TOKENS ARE FOCUS KEYWORDS (OPERATORS) OR DATA REFERENCES (OPERANDS).
* TOKENS AND LOC ARE ADDED TO THE SYSTEMS FILE

FILEDEF SYSTEMS DISK &OUTFILE1
FILEDEF TEXTFILE DISK &FOCFILE
FILEDEF FOCWORDS DISK &LOOKUP1
FILEDEF LOGFILE DISK C:UPDATE.LOG APPEND

MODIFY FILE SYSTEMS

FIXFORM ON TEXTFILE LOC/80
GOTO PARSER

CASE AT START
COMPUTE
   MAXLOC/I4 = 9999;
   LOCOUNT/I4 =0;
   TABLEFLAG/I1 = 0;
   CRTFLAG/I1 = 0;
   GOTOFLAG/I1 = 0;
   FILEFLAG/I1 =0;
   LEFTSIDE/I1 = ;
   COMMLEN/I1 = 2;
   COMMENT/A2 ='.**';
   QUOTE/A1 = HEXBYTE(34,QUOTE);
   APOSTROPHE/A1 = HEXBYTE(39,APOSTROPHE);
   QUOTEPOS/I2 = ;
   CHARPOS/I2 = 0;
   OPENQUOTE/I1 =0;
CASE FINDSYS
    FIXFORM ON SYSNAME COMPANYNAME/8
    FIXFORM ON SYSNAME SYSTEMNAME/8
    MATCH COMPANYNAME
        ON MATCH TYPE "PROCESSING COMPANY: <COMPANYNAME"
        ON NOMATCH TYPE "ADDING NEW COMPANY: <COMPANYNAME"
        ON NOMATCH INCLUDE
    MATCH SYSTEMNAME
        ON MATCH TYPE "PROCESSING SYSTEM: <SYSTEMNAME"
        ON NOMATCH TYPE "ADDING NEW SYSTEM: <SYSTEMNAME"
        ON NOMATCH INCLUDE
    GOTO FINDPROG
ENDCASE

CASE FINDPROG
    FIXFORM PROGRAMNAME
    MATCH PROGRAMNAME
        ON MATCH TYPE ON LOGFILE
            "DUPLICATE PROGRAM: <SYSTEMNAME <PROGRAMNAME"
        ON MATCH TYPE "DUPLICATE PROGRAM: <PROGRAMNAME"
        ON MATCH REJECT
        ON MATCH GOTO EXIT
    ON NOMATCH TYPE ON LOGFILE
        "PARSING PROGRAM: <SYSTEMNAME <PROGRAMNAME"
        ON NOMATCH TYPE "PARSING PROGRAM: <PROGRAMNAME"
        ON NOMATCH INCLUDE
ENDCASE

CASE PARSER
    -* TYPE " "
    -* TYPE "<LOC"
    -* TYPE "<"

    COMPUTE
        LOC/A80 = LJUST(80,LOC,LOC);
        LOCLEN/I2 = ARGLEN(80,LOC,LOCLEN);
        LOCBUF/A80 = LOC;
LEFTSIDE/I4 = 1;
TOKCOUNT/I2 = 0;
LOCOUNT/I4 = LOCOUNT + 1;

IF LOCOUNT GT MAXLOC THEN GOTO ABORTPGM;

* IGNORE BLANK LINES AND COMMENTS
IF (LOCLEN EQ 0) OR (LOC CONTAINS COMMENT) OR (LOC EQ '-') THEN GOTO ENDCASE;

* INCLUDE THE LOC INTO THE DATABASE
NEXT LOC
ON NONEXT INCLUDE

* FOCUS BUG CAUSES THIS BRANCHING CLUGE CDW. AU.09.88
GOTO PARSE2
ENDCASE

CASE PARSE2
  * IN THE EVENT THAT STRINGS EXTEND OVER 2 OR MORE LINES
  IF OPENQUOTE THEN GOTO PATCHIT;

  * SET FLAGS TO INDICATE THE CONTEXT OF THE CODE
  COMPUTE
    TABLEFLAG/I1 = IF (LOC CONTAINS 'TABLE ') AND (LOC OMITS QUOTE OR APOSTROPHE)
    THEN 1
    ELSE (IF TABLEFLAG EQ 1
      THEN (IF LOC CONTAINS 'END' AND LOCLEN LE 4
      THEN 0
      ELSE TABLEFLAG)
    ELSE TABLEFLAG);

    MATCHFLAG/I1 = IF (LOC CONTAINS 'MATCH ') AND (LOC CONTAINS 'FILE ')
    AND (LOC OMITS QUOTE OR APOSTROPHE)
    THEN 1
    ELSE (IF MATCHFLAG EQ 1
      THEN (IF LOC CONTAINS 'END' AND LOCLEN LE 4
      THEN 0
      ELSE MATCHFLAG)
    ELSE MATCHFLAG);

    CRTFLAG/I1 = IF LOC CONTAINS 'CRTFORM '
    THEN 1
    ELSE (IF LOC CONTAINS 'TYPE '
      THEN 0);

    COMBINEFLAG/I1 = IF LOC CONTAINS 'COMBINE ' AND LOC CONTAINS 'FILES ' AND LOC OMITS QUOTE OR APOSTROPHE
    THEN 1
    ELSE (IF COMBINEFLAG EQ 1 AND LOC CONTAINS 'AS '
      THEN 0);

    *TYPE TBL= <TABLEFLAG CRT= <CRTFLAG FF= <FILEFLAG MAT= <MATCHFLAG*
    *TYPE COM= <COMBINEFLAG*
* LINE MAY REFER TO A MASTER FILE
IF LOC CONTAINS 'FILE' OR 'JOIN' OR 'TABLE'
THEN GOTO FILEPARSER;

GOTO EASYSTUFF
ENDCASE

******************************************************************************
* FILEPARSER IS CALLED WHEN AN INPUT LINE LOOKS SUSPICIOUSLY
* LIKE IT MAY HAVE SOME MASTERFILE INFORMATION IN IT
******************************************************************************

CASE FILEPARSER
    COMPUTE
        NEWLINE/A80 = LOC;
        NEWLEN/I2 = LOCLEN;
    GOTO GETFILE
ENDCASE

CASE GETFILE
    IF NEWLEN EQ 0 THEN GOTO EASYSTUFF;
    COMPUTE
        TOKEN/A80 = GETTOK(NEWLINE,NEWLEN,1',80,TOKEN);
        TOKENLEN/I2 = ARGLEN(80,TOKEN,TOKENLEN);
        NEWLINE = SUBSTR(NEWLEN,NEWLINE,TOKENLEN+1,NEWLEN,80,NEWLINE);
        NEWLINE/A80 = LJST(80,NEWLINE,NEWLINE);
        NEWLEN/I2 = ARGLEN(80,NEWLINE,NEWLEN);
        PERIODPOS/I2 = POSI(TOKEN,TOKENLEN,'.',1,PERIODPOS);
        MASTERNAME = IF PERIODPOS EQ 0
                        THEN SUBSTR(80,TOKEN,1,8,8,MASTERNAME)
                        ELSE SUBSTR(80,TOKEN,1,PERIODPOS-1,8,MASTERNAME);

        "TYPE "MASTERNAME = <MASTERNAME"
        MATCH MASTERNAME
            ON MATCH GOTO CHECKFILE
            ON NOMATCH GOTO GETFILE
        ENDCASE

CASE CHECKFILE
    COMPUTE
        FILENAME = MASTERNAME;

    MATCH FILENAME
        ON MATCH CONTINUE
        ON NOMATCH COMPUTE
            #FIELDS = D.NUM_FIELDS;
            #SEGMENTS = D.NUM_SEGMENTS;
            #INDEXES = D.NUM_INDEXES;
            #FILES = D.NUM_FILES;
            F_LENGTH = D.FIELD_LENGTH;
        ON NOMATCH INCLUDE
            ON NOMATCH TYPE "INCLUDING FILENAME <FILENAME"
                GOTO GETFILE
            ENDCASE

******************************************************************************
* START OF MAIN TOKEN PARSING LOOP
CASE EASYSTUFF
  COMPUTE
    TOKEN/A80 = " ";
    LOC/A80 = JJUST(80,LOC,LOC);
    LOCLEN/I2 = ARGLEN(80,LOC,LOC);
    TOKEN/A80 = GETTOK(LOC,LOCLEN,1,'80',TOKEN);
    TOKENLEN/I2 = ARGLEN(80,TOKEN,TOKENLEN);
    TOKCOUNT/T2 = TOKCOUNT+1;
  IF TOKCOUNT GT 99 THEN GOTO ABORTLINE;
  *
 -done with the line
  IF TOKENLEN EQ 0 THEN GOTO ENDCASE;
  GOTO LOOKITUP
ENDCASE

* do a table lookup to see if the token is a focus keyword
CASE LOOKITUP
  COMPUTE
    KEYWORD/A12 = " ";
    KEYWORD/A12 = IF TOKENLEN LE 12 THEN SUBSTR(TOKEN,TOKENLEN,12,KEYWORD)
     ELSE SUBSTR(TOKEN,TOKENLEN,12,KEYWORD);
    KEYTEST = FIND(KEYWORD IN FOCWORDS);
    IF KEYTEST EQ 0 THEN GOTO MEDIUMSTUFF ELSE GOTO PARSEKEY;
ENDCASE

--- OPERATORS -------------------------------

PARSEKEY IS CALLED when the input token matches a focus keyword

CASE PARSEKEY
  COMPUTE
    LEFTSIDE/I4 = IF KEYWORD EQ ' = ' THEN 0;
    GOTOFLAG/I1 = IF KEYWORD CONTAINS 'GOTO' OR 'CASE' THEN 1;
    FILEFLAG/I1 = IF KEYWORD
      EQ 'FILE' OR 'FILES'
      OR ((TABLEFLAG EQ 0 AND MATCHFLAG EQ 0) AND
        (KEYWORD IS 'AS' OR 'IN'))
      OR (COMBINEFLAG EQ 1 AND KEYWORD IS 'AND')
    THEN 1;
    NEXT KEYWORD
    ON NONEXT INCLUDE
  * on nonext type " KEYWORD=<KEYWORD FILEFLAG=<FILEFLAG"
    GOTO REMOVE
ENDCASE

--- OPERANDS -------------------------------

THIS IS WHERE OPERANDS ARE PROCESSED: once they're found.
* depending on its context an operand may be one of many things

CASE CODEOPERAND
  COMPUTE
    OPERAND_TYPE = IF FILEFLAG EQ 1 THEN 40
ELSE (IF GOTOFLAG EQ 1 THEN 60
ELSE (IF TABLEFLAG EQ 1 OR MATCHFLAG EQ 1
THEN (IF LOCBUF CONTAINS ' HOLD ' 
THEN 41
ELSE (IF MATCHFLAG EQ 1 
THEN 50
ELSE 20))
ELSE (IF LOCBUF CONTAINS '-INCLUDE ' 
THEN 42
ELSE (IF TOKEN GT ':' 
THEN 50
ELSE (IF TOKEN CONTAINS '&' 
THEN 51
ELSE 52)))));

GOTO PARSEOP
ENDCASE

.* ADD TO THE DATABASE
CASE PARSEOP

COMPUTE
TOKTEMP/A80 = ";
SLASHPOS/I2 = POSIT(TOKEN,TOKENLEN, '/', 1, SLASHPOS);
TOKTEMP/A80 = IF (SLASHPOS GT 0) AND (OPERAND TYPE NE 7) 
THEN SUBSTR(TOKENLEN,TOKEN,1,SLASHPOS-1,SLASHPOS-1,TOKTEMP) 
ELSE TOKEN;
TEMPLLEN/I2 = ARGLEN(80,TOKTEMP,TEMPLLEN);
OPERAND/A12 = SUBSTR(80,TOKTEMP,1,TEMPLLEN,12,OPERAND);
FILEFLAG = 0;
GOTOFLAG = 0;
IF OPERAND EQ "" THEN GOTO REMOVE;

NEXT OPERAND
ON NONEXT INCLUDE
.* ON NONEXT TYPE "OPERAND = < OPERAND TYPE = < OPERAND_TYPE"
GOTO REMOVE
ENDCASE

.* UTILITY TO REMOVE TOKEN FROM LOC
CASE REMOVE

COMPUTE
LOC/A80 = IF LOCLEN EQ TOKENLEN 
THEN 
ELSE SUBSTR(80,LOC,TOKENLEN+1, LOCLEN, 80,LOC);
LOC/A80 = LJUST(80,LOC,LOC);
LOC/A80 = IF KEYWORD EQ 'TYPE' THEN ""||LOC||"
LOCLEN/I2 = ARGLEN(80,LOC,LOCLEN);
GOTO EASYSTUFF
ENDCASE

*****************************************************************
-• WHEN THE TOKEN ISN'T STRAIGHTFORWARD SOME MORE LOGIC IS INVOKED
-• TO ASCERTAIN WHAT'S IN THE LINE
-*************************** MEDIUM STUFF ***********************

CASE MEDIUMSTUFF
* DEAL WITH QUOTES AND APOSTROPHES - A ROYAL PAIN IN THE ASS
  IF POSIT(TOKEN,TOKENLEN,QUOTE,1,QUOTEPOS) EQ 1 OR 2 THEN GOTO PQUOTE;
  IF POSIT(TOKEN,TOKENLEN,APOSTROPE,1,QUOTEPOS) EQ 1 THEN GOTO PAPOST;

  * IF THERE ARE NO SPECIAL CHARCTERS IN TOKEN THEN ITS AN OPARAND
  IF TOKEN OMITS '(' OR ')' OR ',' OR ';' OR '=' OR '-' OR '+' OR '[' OR '!' OR '@' OR '<' OR '/'
  THEN GOTO CODEOPERAND;

  * NOW THINGS ARE BIT TRICKIER: SCREEN OUT DELIMETERS OR PAD
  COMPUTE CHARPOS/2 = POSIT(TOKEN,TOKENLEN,')',1,CHARPOS);
  IF CHARPOS NE 0 THEN GOTO BLANKOUT;
  COMPUTE CHARPOS/2 = POSIT(TOKEN,TOKENLEN,')',1,CHARPOS);
  IF CHARPOS NE 0 THEN GOTO BLANKOUT;
  COMPUTE CHARPOS/2 = POSIT(TOKEN,TOKENLEN,')',1,CHARPOS);
  IF CHARPOS NE 0 THEN GOTO BLANKOUT;
  COMPUTE CHARPOS/2 = POSIT(TOKEN,TOKENLEN,')',1,CHARPOS);
  IF CHARPOS NE 0 THEN GOTO BLANKOUT;
  COMPUTE CHARPOS/2 = POSIT(TOKEN,TOKENLEN,')',1,CHARPOS);
  IF CHARPOS NE 0 THEN GOTO PADIT;
  COMPUTE CHARPOS/2 = POSIT(TOKEN,TOKENLEN,')',1,CHARPOS);
  IF CHARPOS NE 0 THEN GOTO CHECKLABEL;
  COMPUTE CHARPOS/2 = POSIT(TOKEN,TOKENLEN,')',1,CHARPOS);
  IF CHARPOS NE 0 THEN GOTO PADIT;
  COMPUTE CHARPOS/2 = POSIT(TOKEN,TOKENLEN,')',1,CHARPOS);
  IF CHARPOS NE 0 THEN GOTO PADIT;
  COMPUTE CHARPOS/2 = POSIT(TOKEN,TOKENLEN,')',1,CHARPOS);
  IF CHARPOS NE 0 THEN GOTO PADIT;

  * WHAT IS THIS STUFF?
  GOTO HARDER
  ENDCASE

  * THE "\" CHARACTER PRESENTS SPECIAL PROBLEMS. IT CAN BE EITHER A
* FORMATTING DELIMITER OR A DIVISION OPERATION DEPENDING ON WHICH SIDE
* OF THE "=" SIGN IT LIES.
CASE SLASHIT
  IF LEFTSIDE EQ 1 THEN GOTO CODEOPERAND;
  GOTO PADIT
ENDCASE

* DIALOGUE MANAGER STATEMENTS HAVE A - IN THE FIRST COLUMN
CASE CHECKLABEL
  IF POSIT(LOCBUF,80,'-',1,CHARPOS) NE 1 THEN GOTO PADIT;
  COMPUTE
    OPERAND TYPE = 60;
  GOTO PARSEOP
ENDCASE

* APOSTROPHES LOC='BLAH...
CASE PAPOST
  COMPUTE
    NEWLINE/A80 = SUBSTR(LOCLEN,LOC,2,LOCLEN,80,NEWLINE);
    QUOTEPOS/I2 = POSIT(NEWLINE,80,APOSTROPHE,1,QUOTEPOS);
    TOKEN/A80 = SUBSTR(LOCLEN, LOC, 1,QUOTEPOS + 1,80,TOKEN);
    TOKENLEN/I2 = ARGLEN(80,TOKEN,TOKENLEN);
    OPERAND TYPE = IF TABLEFLAG EQ 1 THEN 23 ELSE 53;
  GOTO PARSEOP
ENDCASE

* QUOTES LOC="BLAH..
CASE PQUOTE
  *TYPE "PQUOTE"
  COMPUTE
    QUOTEPOS = POSIT(TOKEN,TOKENLEN,QUOTE,1,QUOTEPOS);
    LOC/A80 = ""||LJUST(80,SUBSTR(LOCLEN,LOC,QUOTEPOS + 1,LOCLEN,80,NEWLINE),LOC);
    LOCLEN/I2 = ARGLEN(80,LOC,LOCLEN);
    CARPOS/I2 = POSIT(LOC,LOCLEN,LCARROT,LCARPOS);
    QUOTEPOS/I2 = POSIT(SUBSTR(LOCLEN,LOC,2,LOCLEN,80,NEWLINE),LOCLEN-1,QUOTE,1,QUOTEPOS);
    OPENQUOTE/I1 = IF QUOTEPOS EQ 0 THEN 1 ELSE 0;
    IF CARPOS EQ 2 THEN GOTO GETSVARS
    ELSE (TP CARPOS GT 2 THEN GOTO GETSTRING);
  
  IF CARPOS EQ 2 THEN GOTO GETSVARS
  ELSE (IF CARPOS GT 2 THEN GOTO GETSTRING);

* JUST CHARACTERS LEFT IN STRING
  COMPUTE
    TOKEN/A80 = IF QUOTEPOS GT 0
    THEN SUBSTR(LOCLEN, LOC, 1, QUOTEPOS + 1,80,TOKEN)
    ELSE LOC;
    TOKENLEN/I2 = ARGLEN(80,TOKEN,TOKENLEN);
    OPERAND TYPE = IF TABLEFLAG EQ 1
    THEN (IF TOKEN CONTAINS ' &'
    THEN 22
    ELSE 24)
    ELSE (IF TOKEN CONTAINS ' &'
    THEN 4
    ELSE 6);
GOTO PARSEOP
ENDCASE

- PARSE ON THE "<" DELIMITER
CASE GETSTRING
  COMPUTE
  TOKEN/A80 = GETTOK(LOC,LOCLEN,1,"<",80,TOKEN);
  TOKENLEN/I2 = ARGLEN(80,TOKEN,TOKENLEN);
  ADJUST/I2 = 1;
  OPERAND TYPE = IF TABLEFLAG EQ 1
    THEN (IF TOKEN CONTAINS '&'
      THEN 22
      ELSE 24)
    ELSE (IF TOKEN CONTAINS ' &'
      THEN 4
      ELSE 6);
GOTO QUOTE IT
ENDCASE

- MOST OF THE STUFF BELOW IS BECAUSE FOCUS STRING HANDLING ROUTINES
  - ARE NOT WELL DOCUMENTED AND SOMETIMES DO MYSTERIOUS THINGS
  - INPUT LOC = "<VAR BLAH" OR "<VAR<VAR"
CASE GETSVARS
  COMPUTE
  CARPOS/I2 = POSIT(LOC,LOCLEN,'>',1,CARPOS);
  LOC/A80 = IF CARPOS GT 0 THEN
    OVRLAY(LOC,LOCLEN,' ',1,CARPOS,LOC);
  TOKEN/A80 = GETTOK(LOC,LOCLEN,1,' ',80,TOKEN);
  TOKEN/A80 = OVRLAY(TOKEN,TOKENLEN,1,1,TOKEN);
  TOKEN/A80 = LJUST(80,TOKEN,TOKEN);
  TOKENLEN/I2 = ARGLEN(80,TOKEN,TOKENLEN);
  TOKEN/A80 = IF POSIT(TOKEN,TOKENLEN,QUOTEPOS) GT 1
    THEN OVRLAY(TOKEN,TOKENLEN,QUOTEPOS,QUOTEPOS);
  TOKENLEN/I2 = ARGLEN(80,TOKEN,TOKENLEN);
  REVERSED/A80 = REVERSE(TOKENLEN,TOKEN,REVERSED);
  ADJUST/I1 = 2;
  IF POSIT(REVERSED,TOKENLEN,'<',1,CARPOS) LT TOKENLEN-2
    THEN GOTO STRIPVAR;
  GOTO STRIPATTR
ENDCASE

- TOKEN="<VAR<VAR" OR "<VAR<VAR"
CASE STRIPVAR
  COMPUTE
  TOKEN/A80 = '<'"|GETTOK(LOC,LOCLEN,2,"<",80,TOKEN);
  TOKENLEN/I2 = ARGLEN(80,TOKEN,TOKENLEN);
GOTO STRIPATTR
ENDCASE

- WE NOW FINALLY HAVE A TOKEN CONTAINING SOMETHING RESEMBLING A SCREEN
- VARIABLE. UNFORTUNATELY WITH ALL THE FANCY GRAPHICS OUTPUTS THERE MAY
BE SOME EMBEDDED GRAPHICS CONTROL CHARIS. WE GET RID OF THESE HERE
GRAPHICS CONTROL LOOK LIKE <.AA.

CASE STRIPATTR
  TYPE 'STRIPATR1 TOKEN = <TOKEN TOKENLEN = <TOKENLEN REVERSED = <REVERSED'
  TYPE 'STRIPATR1 LOC = <LOC LOCLEN = <LOCLEN'
    IF (TOKEN GE '/' AND TOKEN LT ':') OR
      (TOKEN CONTAINS '+' OR '-' )
      THEN GOTO SCRNPOS;
  IF (TOKEN GE '<A') OR (TOKEN LE '<' AND TOKEN GE '<&')
    THEN GOTO SCRNVAR;

  COMPUTE
    REVERSED/A80 = ' ';
    REVERSED/A80 = REVERSE(TOKENLEN,TOKEN,REVERSED);
    ENDPERIOD/I2 = POSIT(REVERSED,TOKENLEN,'.',1,ENDPERIOD);
    OPERANDTYPE = 8;
  TYPE 'ENDPERIOD = <ENDPERIOD'
  IF ENDPERIOD EQ 1 THEN GOTO QUOTEIT;

  COMPUTE
    SPLITFLAG/I2 = IF TOKEN CONTAINS 'T.' OR 'D.' THEN 1 ELSE 0;
    SPLITPOS/I2 = IF SPLITFLAG EQ 1
      THEN TOKENLEN-ENDPERIOD-1
      ELSE TOKENLEN-ENDPERIOD+1;
    TOKEN/A80 = SUBSTR(TOKENLEN,TOKEN,1,SPLITPOS,80,TOKEN);
    TOKENLEN/I2 = ARGLEN(80,TOKEN,L0CNLENDPERIOD);
    NEWLINE/A80 = SUBSTR(LOCLEN,LOC,SPLITPOS + 2,LOCLEN,80,NEWLINE);
    LOC/A80 = '"' || TOKEN || '<' || NEWLINE;
    LOCLEN = ARGLEN(80,LOC,LOCLEN);
  TYPE 'STRIPATR2 TOKEN = <TOKEN TOKENLEN = <TOKENLEN SPLITPOS = <SPLITPOS'
  TYPE 'STRIPATR2 LOC = <LOC LOCLEN = <LOCLCN LEN NEWLINE = <NEWLINE'
    GOTO QUOTEIT
ENDCASE

CASE SCRNPOS
  TYPE 'SCRNPOS'
    COMPUTE
      OPERANDTYPE = IF TABLEFLAG EQ 1 THEN 25 ELSE 7;
    GOTO QUOTEIT
ENDCASE

CASE SCRNVAR
  TYPE 'SCRNVAR'
    COMPUTE
      OPERANDTYPE = IF TABLEFLAG EQ 1 THEN 21 
      THEN 21
      ELSE (IF TOKEN CONTAINS '<D.'
        THEN (IF TOKEN CONTAINS '&.' THEN 4 ELSE 3)
        ELSE (IF TOKEN CONTAINS '<T.'
            THEN (IF TOKEN CONTAINS '&.'
              THEN 5
            ELSE 2)
            ELSE (IF TOKEN CONTAINS '&.'
              THEN 4
            ELSE 5))
        ELSE 2)
    END
ELSE (IF CRTFLAG EQ 1
THEN 1
ELSE 3)));

· TYPE "OPERAND _TYPE= <OPERAND _TYPE"
  COMPUTE
    ADJUST = IF TOKEN CONTAINS '<D.' OR '<T.'
    THEN 5 ELSE 3;
· DROP THE <D., <T. OR <
  TOKEN/A80 = IF TOKEN CONTAINS '<D.' OR '<T.'
  THEN SUBSTR(TOKENLEN,TOKEN,4,TOKENLEN,80,TOKEN)
  ELSE SUBSTR(TOKENLEN,TOKEN,2,TOKENLEN,80,TOKEN);

  TOKENLEN/I2 = ARGLEN(80,TOKEN,TOKENLEN);
· TYPE "LOC= <LOC TOKEN= <TOKEN ADJUST= <ADJUST"
  GOTO QUOTE IT
ENDCASE

CASE QUOTE IT
  COMPUTE
    LOC/A80 = TOKEN ||
    '""|| SUBSTR(LOCLEN,LOC,TOKENLEN+ADJUST,
    LOCLEN,80,LOC);
    LOC/A80= OVRLAY(LOC,LOCLEN, ' ',1,TOKENLEN+1,LOC);
    LOCLEN/I2 = ARGLEN(80,LOC,LOCLEN);
  GOTO PARSEOP
ENDCASE

CASE BLANKOUT
  COMPUTE
    LOC/A80 = OVRLAY(LOC,LOCLEN, ' ',1,CHARPOS,LOC);
  GOTO EASYSTUFF
ENDCASE

CASE PADIT
  COMPUTE
    DELIMITER/A1= SUBSTR(TOKENLEN,TOKEN,CHARPOS,TOKENLEN,80,DELIMITER);
    NEWLINE/A80 = SUBSTR(LOCLEN,LOC,1,CHARPOS-1,80,NEWLINE)||'@';

    NEWLINE/A80 = NEWLINE||DELIMITER;
    NEWLINE/A80 = NEWLINE||'@' ||
    SUBSTR(LOCLEN,LOC,CHARPOS+1,LOCLEN,80,NEWLINE);
    NEWLEN/I2 = ARGLEN(80,NEWLINE,NEWLEN);
    NUPOS/I2 = POSIT(NEWLINE,NEWLEN,'@',1,NUPOS);
    NEWLINE/A80 = OVRLAY(NEWLINE,NEWLEN, ' ',1,NUPOS,NEWLINE);
    NEWLINE/A80 = OVRLAY(NEWLINE,NEWLEN, ' ',1,NUPOS+2,NEWLINE);
    LOC/A80 = NEWLINE;

  GOTO EASYSTUFF
ENDCASE

CASE HARDER
· TYPE "HARDER"
  IF OPENQUOTE EQ 1 THEN GOTO PATCHIT;
  GOTO GARBAGE
ENDCASE

CASE PATCHIT
· TYPE "PATCHED A LINE"
COMPUTE
   LOC/A80 = QUOTE||LOC;
   GOTO EASYSTUFF
ENDCASE

CASE GARBAGE
   TYPE ON LOGFILE "ERROR: GARBAGE = <TOKEN"
   GOTO REMOVE
ENDCASE

CASE ABORTLINE
   ^TE ON LOGFILE "ERROR: TOKEN COUNT EXCEEDS MAXIMUM. LINE ABORTED"
   TYPE ON LOGFILE "LOC = <LOCBUF"
ENDCASE

CASE ABORTPGM
   TYPE ON LOGFILE "ERROR: PROGRAM LENGTH EXCEEDS MAXIMUM. PROGRAM ABORTED"
   GOTO EXIT
ENDCASE

DATA
&FOCEXEC
END
EXEC PGMCLASS
  -TYPE FOCEXEC: PGMCLASS
  -RUN

EXEC PERSON
  -TYPE FOCEXEC: PERSON
  -RUN

EXEC FPOINTS
  -TYPE FOCEXEC: FPOINTS
  -RUN

EXEC HALTEMP
EXEC HALSTEAD
  -TYPE FOCEXEC: HALTEMP AND HALSTEAD
  -RUN

EXEC MCCABE
  -TYPE FOCEXEC: MCCABE
  -RUN

* NOW TAKE THE OUTPUT GENERATED FROM THE ABOVE PROGRAMS AND MERGE THEM
EXEC MERGE
  -TYPE MERGING FILES: PGMCLASS, FPOINTS, HALSTEAD, MCCABE, PERSON
  -RUN

* PRODUCE THE FINAL ASCII FILE
EXEC METRICP
  -TYPE GENERATING METRICS REPORT
  -RUN
DOS TIME
- PROGRAM: MERGE
- PURPOSE: MERGES SOFTWARE METRICS PRODUCED BY PGMCLASS, FPOINTS, HALSTEAD
    AND MCCABE INTO ONE FILE
- MAIN OUTPUT: METRICS.FTM
- WRITTEN: AU.10.88
- AUTHOR: C.D. WRIGLEY
- MODS: AU.22.88 MERGES PROGRAMMER ID FILE: PERSON
    AU.25.88. SYSNAMES AND FIRMNAMES TEXT MACROS

MATCH FILE PGMCLASS
    PRINT PGMCLASS AS 'CLASS'
    BY COMPANYNAME BY SYSTEMNAME BY PROGRAMNAME
RUN
FILE PERSON
    PRINT PROGRAMMER AS 'PERSON'
    BY COMPANYNAME BY SYSTEMNAME BY PROGRAMNAME
AFTER MATCH HOLD OLD-OR-NEW
RUN
FILE FPOINTS
    PRINT LOC AND SCRN AND SVARIN AND SVAROUT
    AND RVAR AND MAS
    AND FLDS AND SEGS AND IND AND VFILE AND FLEN AND PROJ AND JOIN
    BY COMPANYNAME BY SYSTEMNAME BY PROGRAMNAME
AFTER MATCH HOLD OLD-AND-NEW
RUN
FILE HALSTEAD
    PRINT N1 AND N2 AND ETA1 AND ETA2 AND LEN AND VOCAB AND VOL
    AND DSDEX AND NHAT
    BY COMPANYNAME BY SYSTEMNAME BY PROGRAMNAME
AFTER MATCH HOLD OLD-AND-NEW
RUN
FILE MCCABE
    PRINT KEYWORD AS 'MCABE'
    BY COMPANYNAME BY SYSTEMNAME BY PROGRAMNAME
AFTER MATCH HOLD AS METRICS OLD-OR-NEW

END
DOS ERASE FOCSORT.FTM
* PROGRAM: PGMCLASS

* CATEGORIZES PROGRAMS INTO CLASSES

* AU.01.88 C.D. WRIGLEY

* MOD: AU.18.88. INTRODUCED BITMAPPED KEYWORD USAGE APPROACH.

* A 6 BIT FIELD IS USED TO RECOGNIZE THE OCCURRENCE OF

* SPECIFIC FOCUS KEYWORDS.

DEFINE FILE SYSTEMS

CRTCLASS/I1 WITH LOC = IF LOC CONTAINS 'CRTFORM ' OR ' -CRTFORM ' THEN 1 ELSE 0;

MODCLASS/I1 WITH LOC = IF (LOC CONTAINS 'MODIFY ') AND (LOC OMITS '"') THEN 1 ELSE 0;

DATACLASS/I1 WITH LOC = IF (LOC CONTAINS 'FIXFORM ' OR 'FREEFORM ') AND (LOC OMITS '"') THEN 1 ELSE 0;

TABCLASS/I1 WITH LOC = IF (LOC CONTAINS 'TABLE ' OR 'TABLEF') AND (LOC CONTAINS ' FILE ') AND (LOC OMITS '"') THEN 1 ELSE 0;

ONTCLASS/I1 WITH LOC = IF LOC CONTAINS 'ON ' AND LOC CONTAINS 'TABLE ' AND (LOC CONTAINS 'HOLD ' OR 'SAVE ' OR 'SAVB ') AND (LOC OMITS '"') THEN 1 ELSE 0;

CONCLASS/I1 WITH LOC = IF (LOC CONTAINS 'EX ' OR 'EXEC ' OR 'RUN ') AND (LOC OMITS '"') AND (LOC OMITS 'FOCEXEC') AND (LOC OMITS '-RUN') AND (LOC NE 'RUN') THEN 1 ELSE 0;

END

TABLE FILE SYSTEMS

SUM CRTCLASS NOPRINT AND MODCLASS NOPRINT AND TABCLASS NOPRINT AND ONTCLASS NOPRINT AND CONCLASS NOPRINT AND DATACLASS NOPRINT AND

COMPUTE

BIT6/I6 = IF SUM.MODCLASS GT 0 THEN 100000 ELSE 0;

BIT5/I6 = IF SUM.CRTCLASS GT 0 THEN 10000 ELSE 0;

BIT4/I6 = IF SUM.DATACLASS GT 0 THEN 1000 ELSE 0;

BIT3/I6 = IF SUM.TABCLASS GT 0 THEN 100 ELSE 0;

BIT2/I6 = IF (SUM.ONTCLASS GT 0) AND (SUM.ONTCLASS EQ SUM.TABCLASS) THEN 10 ELSE 0;

BIT1/I6 = IF SUM.CONCLASS GT 0 THEN 1 ELSE 0;

COMPUTE
PGMCLASS/16 = BIT1+BIT2+BIT3+BIT4+BIT5+BIT6;

BY FIRM BY SYSTEMNAME BY PROGRAMNAME
SYSNAMES
FIRMNAMES
ON TABLE HOLD AS PGMCLASS
END
DOS ERASE FOCSORT.FTM
* PROGRAM: FPOINTS

* FPOINTS EXTRACTS FUNCTION POINT LIKE METRICS FROM THE DATABASE

* AU.10.88 C.D. WRIGLEY

* AU.22.88 SPLIT SVARS AND MAS COUNTING

* AU.24.88. SYSNAMES AND FIRM NAMES ARE TEXT MACROS

* SPECIFY LOGICAL CRITERIA IN DEFINE

DEFINE FILE SYSTEMS

JOINFLAG/I5 = IF (((LOC CONTAINS 'MATCH') AND (LOC CONTAINS 'FILE'))
   OR ((LOC CONTAINS 'JOIN ') AND (LOC OMIT 'CLEAR')))
   AND (LOC OMIT '”')
   THEN 1 ELSE 0;

PROJFLAG/I5 = IF LOC CONTAINS 'ON ' AND LOC CONTAINS ' TABLE ' AND
   (LOC CONTAINS ' HOLD ' OR ' SAVE ')
   THEN 1 ELSE 0;

END

MATCH FILE SYSTEMS

COUNT LOC AND
   FILENAME AS 'MAS' AND
   SUM.#FIELDS AS 'FLDS' AND
   SUM.#SEGMENTS AS 'SEGS' AND
   SUM.#INDEXES AS 'INDEX' AND
   SUM.#FILES AS 'FILE' AND
   SUM.F LENGTH AS 'FLEN'

BY FIRM BY SYSTEMNAME BY PROGRAMNAME

SYSNAMES

FIRMNAMES

RUN

FILE SYSTEMS

COUNT KEYWORD AS 'SCRN'
   IF KEYWORD EQ 'CRTFORM' OR 'CRTFORM'

BY FIRM BY SYSTEMNAME BY PROGRAMNAME

SYSNAMES

FIRMNAMES

AFTER MATCH HOLD OLD-OR-NEW

RUN

FILE SYSTEMS

COUNT OPERAND AS 'SVARIN'
   IF OPERAND TYPE EQ 1 OR 2 OR 5

BY FIRM BY SYSTEMNAME BY PROGRAMNAME

SYSNAMES

FIRMNAMES

AFTER MATCH HOLD OLD-OR-NEW

RUN

FILE SYSTEMS

COUNT OPERAND AS 'SVAROUT'
   IF OPERAND TYPE EQ 3 OR 4

BY FIRM BY SYSTEMNAME BY PROGRAMNAME

SYSNAMES

FIRMNAMES

AFTER MATCH HOLD OLD-OR-NEW

RUN

FILE SYSTEMS

COUNT OPERAND AS 'RVAR'
IF OPERAND TYPE IS-FROM 20 TO 22
BY FIRM BY SYSTEMNAME BY PROGRAMNAME
SYSNAMES
FIRMNAMES
AFTER MATCH HOLD OLD-OR-NEW
RUN
FILE SYSTEMS
SUM PROJFLAG AS 'PROJ' AND JOINFLAG AS 'JOIN'
BY FIRM BY SYSTEMNAME BY PROGRAMNAME
SYSNAMES
FIRMNAMES
AFTER MATCH HOLD AS FPOINTS OLD-OR-NEW
END
DOS ERASE FOCSORT.FTM
TABLE FILE SYSTEMS
  COUNT KEYWORD
    BY FIRM BY SYSTEMNAME BY PROGRAMNAME BY KEYWORD NOPRINT
SYSNAMES
FIRMNAMES
ON TABLE HOLD AS HOLD1
END

TABLE FILE SYSTEMS
  COUNT OPERAND
    BY FIRM BY SYSTEMNAME BY PROGRAMNAME BY OPERAND NOPRINT
SYSNAMES
FIRMNAMES
ON TABLE HOLD AS HOLD2
END

MATCH FILE HOLD1
  SUM KEYWORD AND CNT.KEYWORD
    BY COMPANYNAME BY SYSTEMNAME BY PROGRAMNAME
RUN
FILE HOLD2
  SUM OPERAND AND CNT.OPERAND
    BY COMPANYNAME BY SYSTEMNAME BY PROGRAMNAME
AFTER MATCH HOLD AS HALTEMP OLD-AND-NEW
END

DOS ERASE HOLD1. *
DOS ERASE HOLD2. *
DOS ERASE FOCSORT.FTM
TABLE FILE HALTEMP
PRINT E04 NOPRINT AND
   E05 NOPRINT AND
   E06 NOPRINT AND
   E07 NOPRINT AND
COMPUTE
   N1/I6 = E04;
   N2/I6 = E06;
   ETA1/I6=E05;
   ETA2/I6=E07;
   LEN/I6= N1 + N2;
   VOCAB/I6=ETA1 + ETA2;
   VOL/I6=LEN*(LOG(VOCAB)/LOG(2));
   DSDEX/D63  =  ETA2/N2;
   NHAT/I6 = (ETA1 * (LOG(ETA1)/LOG(2)))  +  (ETA2*(LOG(ETA2)/LOG(2)));
BY COMPANYNAME BY SYSTEMNAME BY PROGRAMNAME
ON TABLE HOLD AS HALSTEAD
END

DOS ERASE FOCSORT.FTM
DOS ERASE HALTEMP.FTM
DOS BEEP