Metrics For Object Oriented Software Design

by

Shyam R. Chidamber

M.S. University of Pittsburgh (1983)

Submitted to the Alfred P. Sloan School of Management in partial fulfillment of the requirements for the degree of

Doctor of Philosophy in Management
at the
Massachusetts Institute of Technology
June 1994

© Shyam R. Chidamber, 1994. All rights reserved.

The author hereby grants MIT permission to reproduce and to distribute copies of this document in whole or in part.

Signature of Author

Sloan School of Management
May 1994

Certified by

Chris F. Kemerer
Douglas Drane Associate Professor of Information Technology and Management
Dissertation Committee Chair

Accepted by

Birger Wernerfelt
Chairman, Ph.D. Committee
Sloan School of Management
Metrics For Object Oriented Software Design

by

Shyam R. Chidamber

Submitted to the Alfred P. Sloan School of Management
in partial fulfillment of the requirements for the degree of
Doctor of Philosophy in Management

Abstract

Object oriented programming promises to move software development and
maintenance from the current 'craft' environment to something more
closely resembling conventional engineering. In order to fulfill this promise,
it will require measures or metrics of the process. While software metrics are
a generally desirable feature in the software management function, they are of
special importance with a new technology such as the object-oriented
approach due to the need to train current and new software engineers in
generally accepted object-oriented principles.

This dissertation first presents theoretical work that builds a suite of metrics
for object-oriented design (OOD). In particular, these metrics are based upon
measurement theory and ontology. In addition, they incorporate the insights
of experienced object-oriented software developers. The proposed metrics are
mathematically evaluated against a widely-accepted list of seven software
metric evaluation criteria and the formal results of this evaluation are
presented.

A second goal of this work is to make these metrics machinable, i.e. build a
tool that extracts these metrics from commercially used class libraries. A
metrics extractor tool is built to extract metrics from C++ class libraries. The
metrics extractor is currently configured to collect metrics from C++ code in
UNIX workstation environments.

A third goal of this dissertation is to relate the metrics gathered on
commercial projects to performance indicators of managerial relevance such as
development effort, extent of reuse, and productivity in order to determine
the degree to which the proposed metrics are useful predictors of typical
economic considerations faced by MIS managers. In the final phase of this
dissertation, the organizational considerations of using metrics as part of the
development process are examined. The message to practicing managers is
that without adequate measures for OOD, organizations will find themselves
increasingly unable to achieve the full benefits of object-oriented technology.

Thesis Committee: 

Chris F. Kemerer, Chair
Stuart E. Madnick
Jack F. Rockart
D. Sriram (reader)
Acknowledgments

First and foremost, I would like to acknowledge my committee, who have been unstinting in their support and guidance throughout my stay at MIT. I could never have undertaken, let alone completed this dissertation without their encouragement. I owe a great to deal to Chris Kemerer, who has been my senior collaborator since the first semester. He helped to sharpen and focus my research over the years, while letting me define my own direction. He has also set a personal example of integrity and discipline in intellectual work. I will always cherish my collaboration with him. Stu Madnick has always amazed me with his most penetrating and insightful questions about my research, and constantly urged me to look at my own work dispassionately. Stu is also undoubtedly one of the best teachers at Sloan. Similarly, Jack Rockart has had a pivotal role in my doctoral education at MIT. No one can elucidate the connection between IT research and organizational relevance better than Jack. Since my first year, he has encouraged and supported me in conducting my research within the context of the 'real world'. Prof. Sriram helped me refine the early work in this dissertation, and served admirably as a reader for this manuscript. I have also benefited from the other faculty in the IT department. Wanda Orlikowski, Erik Brynjolfsson, Tom Malone, Rich Wang and Paul Resnick have all helped with feedback, encouragement and excellent teaching on many occasions.

Numerous other faculty have helped in shaping me as a researcher. John Carroll, John Henderson, N. Venkatraman and Paul Osterman from Sloan taught me research methods that will stand me in good stead for many years. No praise can be lavish enough for John Willett at Harvard, whose course in data analysis is probably the most interesting classroom experience that I have had. He made tedious statistics and arcane methodology come alive. I have also benefited from the substantial input that Professors Ron Weber and Yair Wand provided in the complex area of ontology and mathematical formulations of objects. They have added theoretical depth to my dissertation. The people at The Center for Information Systems Research have not only provided financial support for my research, but always been helpful in dealing with MIT bureaucracy. Judith Quillard, Debra Hoffman, Deb Small and Mary Bucci have become good friends over the years, and I am indebted to them for adding warmth and good cheer during many an afternoon, when the task at hand seemed onerous.

I would like to thank the many engineers and managers at the four different companies that participated in this research. In particular, David Nunn, Douglas Chen, Carmen Stratjean, Sesh Pratap, Carolyn Dubey, Lou Bershad, Joe Cascio, Suresh Nalluri, Colin Prosser, Bryan Thal, Yomi and Al-Noor
Ramji gave of their time, patience and indulgence. Since I spent many hours with them, I have gained from their practical insights and suggestions. Without them, this dissertation would have become a dry, theoretical exercise. Their openness, friendliness and professionalism lightened the burden of my field research. I am convinced as long as we have people like them, systems development will continue to be vibrant and exciting for a long time.

MIT is very special to me because of the fellowship that I have enjoyed with other graduate students. Seniors like Jay Cooprider, Andy Trice, Mike Epstein, Ben Bensou, Ajit Kambil and Mark Ackerman helped me through the process when I was the new kid on the block. Marc Weinstein, David Rabkin, Rob Fichman and Henry Kon provided sorely needed advice, encouragement and friendship without hesitation on an almost weekly basis. The tedium of graduate school would have been unbearable without them.

My family has been largely instrumental in my coming to MIT. My parents and sister always urged me to excel, and taught me to value honesty and hard work. They routinely sacrificed their needs to meet mine. Without them, I would have never made it to MIT. Usha's parents actively encouraged me, and gave a sense of support and stability that few people have the good fortune to count on when they embark on such an endeavor.

Of course none of this would have ever been possible without the staunchest ally of them all: my wife Usha. She willingly gave up her own career, and put up with the inevitable ups and downs of a Ph.D. student's moods and schedules without hesitation. My daughter Niranjani, who entered our life last year, has made the joy of finishing this dissertation immeasurably sweeter. My only regret is that my father is not here to share this with me.
This dissertation is dedicated to Sathya Sai Baba as a small token of my gratitude for his spiritual guidance and support.
Metrics For Object Oriented Software Design

Outline of the Dissertation

CHAPTER ONE: The Research Problem and Literature Review 8-42

Introduction .................................................................10
Overview of Dissertation ..................................................18
Previous Research ............................................................23
Bibliography .....................................................................38

CHAPTER TWO: Theoretical Development of the Metrics Suite 43-83

Theory Base for OOD Metrics ..................................................45
Viewpoint Collection .............................................................56
Metrics Evaluation Criteria ..................................................57
The Metrics Suite ...............................................................62
Summary ............................................................................78
Bibliography .....................................................................82

CHAPTER THREE: Tool Development and Empirical Data 84-102

Tool Development ...............................................................86
Empirical Data Collection .....................................................89
The WMC Metric Data .........................................................90
The DIT Metric Data ............................................................91
The NOC Metric Data ...........................................................93
The CBO Metric Data ...........................................................95
The RFC Metric Data ...........................................................97
The LCOM Metric Data .........................................................99
Summary of Managerial Results ................................................................. 100
Bibliography ............................................................................................... 102

CHAPTER FOUR: Metrics Validation ......................................................... 103-141
Introduction .............................................................................................. 105
Data Collection ......................................................................................... 106
Results ........................................................................................................ 113
Summary and Limitations ........................................................................... 136
Bibliography ............................................................................................... 140

CHAPTER FIVE: Transforming Software Development ................................. 142-157
Introduction .............................................................................................. 144
Background of the IS department at International Bank Corporation ....... 145
Transformation ........................................................................................ 146
Building Core Capability ........................................................................... 154
Bibliography ............................................................................................... 157

CONCLUDING COMMENTS ........................................................................... 158-161

APPENDIX A: Overview of Object Oriented Concepts ......................... 162-181
APPENDIX B: Tool Development Software .............................................. 182-201
APPENDIX C: Regression Analysis ........................................................... 202-242
CONSOLIDATED BIBLIOGRAPHY ............................................................... 243-252
Chapter One: The Research Problem and Literature Review
Chapter Outline

1.0 Introduction .............................................................................................................. 10
  1.1 The software challenge ....................................................................................... 10
  1.2 Managing software development ...................................................................... 13
    1.2.1 Increasing popularity of object technology .............................................. 14
    1.2.2 Measurement in managing software development .................................. 16
  1.3 Measurement of OO becomes an important problem ...................................... 17
2.0 Overview of Dissertation ..................................................................................... 18
  2.1 Goals of the dissertation .................................................................................... 20
  2.2 Primary Contributions ....................................................................................... 21
    2.2.1 Metrics that are rigorous and relevant ...................................................... 21
    2.2.2 Metrics that are information tools for designers ..................................... 21
    2.2.3 Metrics and tools that are a basis of IS managerial decision making ...... 22
  2.3 Outline of the rest of the dissertation .................................................................. 22
3.0 Previous Research .................................................................................................. 23
  3.1 Measurement of software complexity ............................................................... 24
    3.1.1 What is complexity and why is it important? ............................................ 24
    3.1.2 Categorization of Complexity Measures .................................................. 25
      3.1.2.1 Control Flow Measurement ................................................................ 26
      3.1.2.2 Information Flow Measurement ......................................................... 27
  3.2 Previous measurement in OO ............................................................................. 30
    3.2.1 Programming Style Guidelines .................................................................. 30
    3.2.2 Project Metrics ......................................................................................... 31
    3.2.3 Graph theoretic Measures .......................................................................... 32
    3.2.4 Special purpose OO metrics ...................................................................... 33
    3.2.5 Summary of shortcomings ......................................................................... 34
1.0 INTRODUCTION

1.1 The software challenge

Einstein once said that there must be fundamentally simplified explanations for everything, since God is not capricious or arbitrary. This faith in the underlying simplicity of the artifact of study is an intellectual cornerstone of the natural sciences. As Herbert Simon writes in *The Sciences of the Artificial*, "the central task of natural science is to make the wonderful, commonplace: to show that complexity, correctly viewed, is only a mask for simplicity" (Simon 1981). Unfortunately, Einstein's notion of underlying simplicity in the natural sciences seldom seems to apply in the sciences of the artificial (the science of man-made systems). As Joel Moses suggests, complexity is rife in all man-made systems (Moses 1990). Perhaps, God is a far better designer than humans can ever be. It is frequently argued that software systems are perhaps the exemplar of man-made systems and that complexity is an essential property of the artifact (Brooks 1975). Software is abstract and obeys no physical laws, unlike bridges or buildings that obey the laws of physics (Fairley 1985). Furthermore, the instinctive feel that humans have for physical artifacts is typically absent when they relate to software. A large software system is characterized by an enormous state space of possibilities, rendering it difficult to understand and appreciate. Simply put, software is hard.

Software is also expensive. Worldwide expenditure on software exceeded $140 billion in 1985 and it is estimated that it will be close to $450 billion by 1995 (Boehm 1987). Individual corporations are also spending heavily on software.
US corporations alone spent $42 billion in 1991\(^1\). Vitalari's survey of 142 IS operations in North America revealed that the average annual software budget is close to $32 million (Vitalari 1991). Even in organizations where expenses are low, software has considerable impact. Edward Yourdon reports that at the Jet Propulsion Laboratory, where only 10 to 15 per cent of expenses are devoted to software, it is "in series with what they do: if the software fails, the mission fails" (Yourdon 1992 p. 2). Indeed, there are spectacular examples of expensive software failure. A disruption of all electronic communications systems in 1990 that paralyzed cities stretching from Washington DC to Los Angeles was traced to three lines of software. In another incident during the 1980s, software errors in a particular brand of X-ray machines caused severe radiation burns to 5 patients and one fatality\(^2\). Individual organizations have also paid dearly. Blue Cross Blue Shield of Wisconsin, spent $200 million on a system that disbursed $60 million in overpayments and duplicate checks in one year\(^3\). The statistics and incidents emphasize that software is expensive to develop and deliver and that it can often become a major source of instability (financial or otherwise) to various stakeholders in the economy. Unlike ecological or airline disasters, the software problem is not restricted to isolated, spectacular failures. Capers Jones has documented that the average IS project in the US is one year late and 100% over budget (Jones 1991). Boehm estimates that a 20% improvement in software practice will be worth $90 billion to the world economy (Boehm 1987).

Despite its complexity and cost, software has become indispensable to modern organizations. Whether it is in the form of back room applications like payroll

---

\(^1\) Source: The Economist, October 10, 1992
\(^2\) From "Turning software from a black art into a science" (Business Week, October 25, 1991).
\(^3\) From "It's late, costly, incompetent-but try firing a computer system" (Business Week, Nov. 7, 1988)
processing, order entry and inventory control or front office applications like executive support systems, marketing databases and customer service systems (e.g. ASAP\textsuperscript{4}, OTISLINE\textsuperscript{5}) software is critical to the essential activities of an organization. In some service industries, software development is integral to the design and delivery of new products. At an MIT seminar on IT productivity, the chief economist of Morgan Stanley, Stephen Roach, remarked that modern financial instruments like synthetics and derivatives could not exist without computer software. Thus within 200 years of the first program written (by Lady Augusta Byron for Babbage's difference engine), software has become an important economic activity that is mission critical to modern organizations.

Unfortunately, the software development track record has been far from exemplary. Many researchers have suggested that software is constantly in a state of crisis, plagued by late deliveries, high costs and poor quality (Mcfarlan 1974; Brooks 1975; Zmud 1980; Boehm 1981; Abdel-Hamid and Madnick 1991; Rockart and Hofman 1992). The statistics reported are sobering (DeMarco 1982):

- Cost overruns of 100% to 200% are common.
- Most software projects are delivered late.
- 15% of all software projects are abandoned.
- A project can be considered a "success" if overruns are below 30%.
- End-users rarely read or use more than 75% of a software's functionality.

Faced with ever increasing demands for better products and services from end users (within and outside the organization), and beset with mediocre

\footnote{4 See "Baxter Healthcare Corporation: ASAP Express", Harvard Business School Case #9-188-080.}
\footnote{5 See "OTISLINE (A)", Harvard Business School Case #9-186-304.}
performance in systems delivery, IS managers have become increasingly concerned about improving software development. In some cases, this concern has driven firms to the drastic action of outsourcing systems development (Loh and Venkatraman 1992). The software development issue is of urgent interest to general managers as well. It has been noted that development of software systems will play an increasingly important role in an organization's ability to compete in the post-industrial global economy where executives have to manage using informational representations of the business (Haeckel and Nolan 1993). Thus, the stakes are high for both the IS function and the organization. Excellence in software development could mean the difference between prospering in the information economy and painful demise.

1.2 Managing software development

IS managers frequently adopt new technological solutions to improve software development within their organizations, but have met with mixed success. A few new technologies like higher level languages (HLL) and database management systems (DBMS) have delivered gains in terms of improved quality and timeliness of software delivery. However other technologies like Computer Aided Software Engineering (CASE) and artificial intelligence (AI) have had limited impact. Rockart and Hofman attribute this to the delicate balance of technology, management processes, people, structure and systems strategy in an organization. They contend that in order to make substantial improvements to systems delivery, organizations must pay adequate attention to all these factors simultaneously.

Leading edge IS organizations have understood this, and over the years have labored to advance up the ladder of stages from "chaotic" to "repeatable" to
"managed" processes that are outlined by the Software Engineering Institute's (SEI) Process Maturity Framework (Humphrey 1988). They are attempting to transform the largely craft nature of software development (with each line of code built by hand by a skilled worker) into a modern production process that ensures quality and improves efficiency. This requires skillful deployment of technology, processes improvements and organizational changes.

A significant challenge to IS managers is to avoid slipping down the SEI ladder while advancing along the technology dimension. Excellence in the IS function requires the unique ability to advance simultaneously along the technology and process maturity dimensions. This dissertation deals with the problem of improving the software development process while using a new technology called Object Technology, that presents both an opportunity and a threat to the equilibrium of an IS organization.

1.2.1 Increasing popularity of object technology

The Software Engineering Laboratory at NASA has heralded object technology (OT) as the most promising software technology to date (McGarry and Waligora 1991). OT (a term that encompasses object oriented programming, object oriented analysis, object oriented design, object oriented databases) is the latest in the series of software technologies that promise to alleviate the problems of cost, schedule and quality in software development6. Object technology is based on a different approach to organizing software applications. Instead of separating data and instructions and managing them independently, OT suggests combining the two into a coherent abstraction called an object. Objects and

6 This dissertation does not deal with object oriented databases, and OT is used primarily to refer to analysis, design and programming.
classes of objects can be used to represent key abstractions of an application, and by arranging them in imaginative ways a sophisticated system can be designed using simple building blocks. Appendix A and Chapter 2 contain more information on the concepts and terms associated with OT.

Proponents of OT proclaim that this approach to building software will transform the software world irrevocably (Cox 1990; Taylor 1990). They envisage a scenario where software will be assembled using standard objects available through extensive public, commercial and private libraries making software more reliable and flexible, yet less expensive. Dissenting opinions assert that OT is merely a repackaging of existing ideas and that it is yet another promise of a "silver bullet" that will fail to deliver promised gains (Page-Jones and Weiss 1989; Wasserman, Pircher and Muller 1989).

Major software vendors are aggressively producing OT products and there appears to be a band wagon effect developing for OT7. An industry consortium, The Object Management Group (OMG), estimates that the market for OT software will likely total $1 billion by 1996. While accurate estimates of acceptance of OT are unavailable, it can be suggested that many IS organizations are moving towards adding OT to their technology portfolio. For example, Al-Noor Ramji, the Executive Director of Swiss Bank Corporation's Information Systems Department says that OT is their "technology of choice for building new systems"8.

7 For example, IBM, Hewlett Packard, Microsoft, Borland, Sun Microsystems, Apple Computer, Texas Instruments all have OT products.
8 Personal communication in September 1993.
Fichman and Kemerer suggest that due to both community-wide and organizational barriers to adoption, OO may not become a dominant technology in large in-house IS organizations (Fichman and Kemerer 1992). Scholtz et al. and Stark also caution that while OO holds promise, there are significant hurdles to learning and reusability that individual organizations have to overcome before they can realize any benefits (Scholtz et al. 1993; Stark 1993). Whether OO will become a technology choice for most IS organizations remains to be seen, but preliminary evidence suggests that there is considerable interest in OT. However, it will require enlightened management to deliver significant cost-effective business systems\(^9\). Thus, OT represents a significant management challenge to IS organizations.

1.2.2 Measurement in managing software development

Peter Drucker has pointed out that measurement is an essential function of management. He emphasizes that in organizations, measurement is the information that guides behavior and performance (Drucker 1988). In Mason and Swanson’s view, measurement is the lens through which the manager sees organizational reality (Mason and Swanson 1981 p. 11). The existence of various accounting and financial measures attest to the importance of measurement as a vital tool for managing enterprises. In the IS domain, managers have a tradition of measuring variables like hardware down-time and CPU usage to manage hardware operations.

Unfortunately, software development has lacked this tradition. As Gill and Kemerer suggest, without adequate metrics, the ”tasks of planning and

---

\(^9\) See “Brooklyn Union Gas: OOPS on Big Iron”, Harvard Business School Case #9-192-144 for a case discussion of these issues.
controlling software development and maintenance will remain stagnant in a craft-type mode" (Gill and Kemerer 1991). Software has gradually begun to develop this vocabulary of measurement. For example, Function Points (Albrecht and Gaffney 1983) and McCabe's cyclomatic number (McCabe 1976) are software measurements that have wide currency. However, as Banker and Kemerer suggest in their principal-agent analysis of metrics, there is significant noise in the current measurement of software attributes and more disciplined work is needed in order to improve the collective understanding of the field (Banker and Kemerer 1992). Thus software measurement is an area that merits research attention.

1.3 Measurement of OO becomes an important problem

The focus on process improvement has increased the demand for software measures, or metrics with which to manage the software development process. The need for such metrics is particularly acute in the case of an organization adopting a new technology like OT, for which established practices have yet to be developed. Booch underscores the risk of regressing in the SEI ladder when dealing with OT and recommends rethinking the process of OO systems delivery (Booch 1993).

The OO approach centers around modeling the real world in terms of its objects, which is in contrast to older, more traditional approaches that emphasize a function-oriented view that separates data and procedures. Several theoretical discussions have speculated that OO approaches may even induce different problem-solving behavior and cognitive processing in the design process, (Booch 1991; Kim and Lerch 1991). Given the fundamentally different notions inherent in these two views, it is not surprising to find that software metrics developed
with traditional methods in mind, do not readily lend themselves to OO notions such as classes, inheritance, encapsulation and message passing (Wilde and Huit 1992).

The shortcomings of existing metrics and the need for new metrics especially designed for OO have been suggested by a number of researchers. Several have suggested that traditional metrics like Function Points or McCabe's metric may be inadequate for OO design and development (Banker, Kauffman and Kumar 1991; Tegarden, Sheetz and Monarchi 1992). In particular, the inadequacies of traditional metrics in aiding the maintenance of OO systems have been noted (Lejter, Meyers and Reiss 1992; Wilde and Huit 1992).

Therefore, given that current software metrics are subject to general criticisms and are seen as not supporting key OO concepts, it seems appropriate to develop a set, or suite of new metrics especially designed to measure unique aspects of the OO approach.

2.0 Overview of Dissertation

The motivation for this dissertation arose from the recognition that software (and OO software in particular) is a fascinating and increasingly important activity in organizations, and that the management of this activity poses interesting questions to researchers. Software has had a tradition of being a "black art", and the typical manager had little understanding or appreciation for this purely intellectual product. How should a manager deal with this activity? As suggested earlier, many researchers contend that this unique activity can be proactively managed, by better measurement of the development process.
Research in the measurement of both traditional and OO software (see Section 3) follows two broad streams, one that focuses on practical relevance and another that focuses on theoretical rigor. The key issues dealt with in the practical stream are ease of measurement, empirical validity and applicability to specific local environments. The theoretical stream has concerned itself with mathematical representations, models of software and analytical validity. Few researchers have attempted to bridge the two streams, despite the obvious merit of doing so. This could be attributed to a number of factors like the long time scale for such research, the paucity of data and the inter-disciplinary nature of research effort.

However, the benefits of such research are many, these include 1) improvement in organizational ability to manage software development activities (since metrics that are well designed will provide consistently accurate information), 2) providing the ability to benchmark alternative designs, enabling informed decision making (since ease of measurement will facilitate comparisons) 3) community wide learning of acceptable design principles (since metrics will have external validity) 4) building systematic understanding of software behavior (managers, like doctors, can become experts in recognizing the symptoms of illness, and taking evasive action\textsuperscript{10}) and 5) evolution of a discipline for software development.

The research for this dissertation is focused on integrating the practical and theoretical schools of software measurement, and make a substantial contribution to the literature in this area.

\textsuperscript{10} For a discussion on the analogy between managing software and medical practice see (Rettig 1993).
2.1 Goals of the dissertation

Criticisms of currently available metrics point towards the need for developing metrics that are firmly grounded in theory, and relevant to practitioners in organizations (Bilow 1992; Tegarden et al. 1992). Specifically, drawing from ontology and measurement theory, a goal of this research is to develop theoretically sound metrics that are also constructed with input from expert practitioners, through intensive field study of their design experiences.

A second goal of this work is to make these metrics "machinable" i.e. to build a tool that extracts these metrics with minimum human intervention. This addresses a long standing complaint that software metrics are often difficult and labor intensive to collect.

A third goal of this dissertation is to relate the metrics gathered on commercial projects, to performance indicators of managerial relevance such as development effort, extent of reuse, and productivity improvements. By systematically gathering data on the development and maintenance effort, and extent of reuse on a commercial application, and relating them to the metrics data gathered using the extractor tool, this research can determine the degree to which these metrics are useful predictors of typical economic considerations faced by IS managers.

Most research performed in the area of software metrics has tended to ignore the organizational consequences of incorporating metrics collection in the systems development process. Flamholtz points out that in the context of organizations, the role of measurement is not merely a technical one of representation; it has social and psychological dimensions as well (Flamholtz 1979). Recognizing this,
the last phase of the dissertation examines the use of metrics to manage the software development process in organizations.

2.2 Primary Contributions

2.2.1 Metrics that are rigorous and relevant

The set of metrics proposed in this dissertation is presented as the first empirically validated proposal for formal metrics for OOD. By bringing together the formalism of measurement theory, ontology, mathematical evaluation criteria, and empirical data from professional software developers working on commercial projects, this dissertation seeks to demonstrate the level of rigor required in the development of usable metrics for design of software systems. At a minimum, this metrics suite should lay the groundwork for a formal language to describe metrics for OOD. In addition, these metrics may also serve as a generalized solution for other researchers to rely on when seeking to develop specialized metrics for particular purposes or customized environments.

2.2.2 Metrics that are information tools for designers

This suite of metrics provides senior designers, who may not be completely familiar with the design details of an application, an indication of the integrity of the design. They can use it as a vehicle to address the architectural and structural consistency of the entire application. By using the metrics suite, they can identify areas of the application that may require more rigorous testing as well as areas that are candidates for redesign. Using the metrics in this manner, potential flaws and other leverage points in the design can be identified and dealt with earlier in the design-develop-test-maintenance cycle of an application.
Another benefit of using these metrics is the added insight gained into tradeoffs made between conflicting requirements such as increased reuse (via more inheritance) and ease of testing (via a less complicated inheritance hierarchy). Since there are many possible OO designs for the same application, these metrics can help in selecting one that is most appropriate to the goals of the organization (such as reducing the cost of development, testing and maintenance over the life of the application).

2.2.3 **Metrics and tools that are a basis of IS managerial decision making**

The suite of metrics proposed in this dissertation have already begun to be used in a few leading edge organizations in innovative ways. Sharble and Cohen report on how these metrics were used by Boeing Computer Services to evaluate different methodologies (Sharble and Cohen 1993). Two implementations of an example system, one using a responsibility based methodology and another using a data driven methodology were analyzed using these six metrics. Based on this analysis, Sharble and Cohen recommended the usage of responsibility based design methodology in the organization. This suggests an active interest on the part of the practitioner community in using well-constructed metrics as a basis for managerial decision-making.

2.3 **Outline of the rest of the dissertation**

The next chapter (chapter 2) discusses the theoretical development of a suite of metrics, while chapter 3 presents the tool designed to capture the proposed metrics, and data gathered from two different commercial organizations. Chapter 4 focuses on validation of the metrics suite, and presents relationships between the metrics and performance data from a financial services firm. Chapter 5 describes the crucial part metrics play in the transformation of
software development at the financial services firm. The dissertation also includes three appendices that contain an overview of object oriented concepts, the tool software programs and the regression data analysis.

3.0 Previous Research

Software engineering management refers to the collection of techniques and principles concerned with applying the rigor of engineering to the management of software projects. Fenton includes in this domain: costing, planning, analyzing, designing, implementing, testing and maintaining software systems along with the tools and techniques used to support and integrate them (Fenton 1991). It is an article of faith amongst researchers and practitioners alike that the only long term solution to the software crisis (poor quality systems that are delivered late and over budget) is the disciplined accumulation and application of scientific knowledge. Unfortunately, as Mary Shaw states in her article on the state of software research, software engineering is not yet a true discipline, although it does have the potential to become one (Shaw 1990).

Lord Kelvin is said to have stated that for any field of endeavor to be considered scientifically advanced, there must be a system of measurement that is part of the literature and practice of that field. However, the field lacks a common language of measurement, which deters progress (Brooks 1987; Fenton 1991; Gill and Kemerer 1991). Measurement is therefore a central conceptual issue in advancing the field of software engineering management. Tom DeMarco injects an urgency into the dialog that will appeal to IS managers, by emphasizing that "you cannot control what you cannot measure." (DeMarco 1982).
3.1 Measurement of software complexity

It has been argued in the previous sections that software is extraordinarily complex to design and manage, and measurement is a systematic way of building a discipline for the field. This section is devoted to examining the prior research work in the area of software complexity measurement, including measurement of OO software.

3.1.1 What is complexity and why is it important?

Webster's dictionary defines complexity as "the property of being complex", and complex as "intricate, knotty and consisting of several confusingly inter-connected parts". As the definition suggests, the concept of complexity is associated with the level of effort required in understanding something. Researchers have defined software complexity in a similar vein. Cook defines complexity as "a measure of how difficult it may be for an individual to test, maintain or program source code" (Cook 1982). Basili suggests that complexity is "a measure of the resources expended while interacting with a piece of software to perform a given task" (Basili 1980). Likewise, Curtis et al. define software complexity in terms of the psychological effort required to understand and work with a piece of software (Curtis et al. 1979). Fenton captures the inter-relatedness aspect in his definition that complexity denotes "the totality of all internal attributes" (Fenton 1991). Thus, complexity, like beauty, is in the eyes of the beholder.

However, the importance of complexity in software is widely recognized and agreed to by practitioners. For example, for software programs (code) it is frequently mentioned that:
• Less complex code is easier to test (Basili and Perricone 1984).
• Less complex code is cheaper to maintain (Banker et al. 1993).
• Less complex code takes less time to develop (Shen, Conte and Dunsmore 1983).
• Less complex code is of higher quality (Card and Agresti 1988).
• Less complex code has better functional performance (Munson and Khoshgoftar 1992).

These empirical findings suggest that 1) reducing complexity can reduce the cost of maintenance, which is often 80% of the life cycle cost of software (Fairley 1985), 2) lower complexity can also reduce the time taken to develop a software product, accelerating its time-to-market, and alleviating the applications backlog, 3) reducing complexity can reduce testing effort, an activity that can constitute 20% of the cost of software development, 4) reducing complexity can also result in higher quality, which translates to lower cost of responding to failures at customer sites, and higher customer satisfaction. Simply stated, complexity can significantly impact the productivity of the software development and maintenance. Thus, measurement of software complexity becomes a central question in software engineering management.

3.1.2 Categorization of Complexity Measures

Certain problems are more complex than others. For example, the traveling salesman problem is more complex than a sorting problem. Computer scientists have a well established vocabulary for cataloging problems of differing complexity. The traveling salesman problem belongs to the class of NP complete problems whereas the sorting problem is an \(O(\log n)\) problem. The categorization of a problem depends on the complexity of the algorithm used in its solution.
This is referred to as \textit{algorithmic complexity}, and it is generally accepted that development of a superior algorithm is a major contribution to the field.

However, in commercial systems development, IS managers are rarely confronted with the problem of ameliorating algorithmic complexity, which is a characteristic of the problem domain and not within their control. They are confronted with the more practical problem of decreasing implementation-induced complexity or \textit{controllable complexity}. There are two broad categories of measures for controllable complexity, control flow and information flow measures.

\textit{3.1.2.1 Control Flow Measurement}

Early work in software measurement is based on the view that the flow of control within a software program determines its complexity (McCabe 1976; Myers 1977; Hansen 1978; Basili and Hutchens 1983). A program is modeled as a directed graph with edges, nodes and line segments. A graph represents the control paths through a piece of software. The nodes of a graph are declarative statements in a program (e.g. \( X = Y, X = 5 \)), line segments show the flow of logic in the program (e.g. \( X = Y \) is executed after \( X = 5 \)). Edges of the graph denote branching in program control caused by typical IF-THEN-ELSE logic (e.g. If \( X \) is greater than 5, then stop, else continue). Drawing from the extensive theory of graphs\textsuperscript{11}, various measures of the graphical representation were proposed as measures of complexity\textsuperscript{12}. The most popular metric in this category of software measures is McCabe's cyclomatic number, which is computed by determining the number of edges and nodes in the graph that represents a program (McCabe 1976). All

\textsuperscript{11} The reader is referred to (Wilson 1972) for a comprehensive treatment of graph theory.
\textsuperscript{12} A comprehensive review of control flow metrics is presented in (Fenton 1991).
control flow measurements suffer from a common practical drawback: they can be determined only after the program is written. This diminishes their usefulness since it is widely believed that flaws detected early in the life-cycle of the product are less expensive to fix (Card and Agresti 1988).

3.1.2.2 Information Flow Measurement

The second category of software measurement is predicated on the notion that the flow of information within and between different partitions (or modules) of a software system determines its complexity. The "information flow complexity" metric is the earliest and most well known measure in this category (Henry and Kafura 1981). This metric is determined by counting the number of exchanges of data between modules. The possibility of determining the metric prior to completion of source code made this a major improvement over control flow measurement. Henry and Kafura also presented data from a commercial system, and demonstrated that information flow complexity was positively correlated to maintenance effort. Spurred by this evidence, other researchers also proposed metrics predicated on the concept of information flow as a measure of software complexity (Bieman and Debnath 1985; Hausen 1989; Shepperd 1990). The information flow concept has also contributed substantially to superior practice in traditional software development by elaborating on issues of modularity, reusability, structuredness, cohesiveness and coupling (Troy and Zweben 1981; Yourdon and Constantine 1991).

From a historical perspective, both control flow and informational flow metrics were framed by the issues relating to the type of software developed during the 1970s and 1980s. In the 1970s, software systems were relatively small, and programs were merely streams of instructions for the CPU to follow. Complexity
manifested itself in the form of branching and nesting of logic in the instructions. Simplifying the flow of logic was considered to be the fundamental issue, and control flow metrics were ideally suited to do this. Informational flow metrics were proposed in the 1980s, when software systems became larger and modules of software required data and information from the internals of other modules within the system. Complexity manifested itself in the unexpected (and inadvertent) interconnectedness between parts of a software system. The information flow metrics were specifically designed to capture this tendency.

Can traditional metrics be effective for OO, where software is a representation of the problem space? Perhaps, but they often fail to capture the nature of OO software. For example, OO abstractions (called classes) contain several methods, each of which is only a few lines long. There is very little control or information flow within the method. Traditional metric values for OO classes will imply low complexity. However, the number of methods and the number of inter-class variables used, often pose problems in testing and performance (Lejter et al. 1992). This aspect of complexity is not captured by traditional metrics. Another example of the inappropriateness of traditional measures is the failure to account for inheritance relationships between classes. The cyclomatic number fails to distinguish between classes that are at the root of the inheritance hierarchy and classes at the bottom of the hierarchy (leaf classes). Since the latter inherit properties from the former, understanding and testing leaf classes requires tacit knowledge of the root classes. The cyclomatic number for leaf classes does not account for this in any way. OO aspects are ignored by traditional metrics, since they were proposed with a procedural and data oriented view of software in mind. Booch expresses this emphatically by writing that "traditional complexity measures, better suited to early generation programming languages, have
minimal correlation with completeness and complexity in object-oriented systems, are therefore largely useless ..." (Booch 1993 p. 279).

Traditional metrics have also come under criticism on three other issues: measurement properties, validation and implementation. Prather, Kearney et al., and Weyuker all proposed that traditional software metrics do not possess appropriate mathematical properties, and consequently fail to display what might be termed normal predictable behavior (Prather 1984; Kearney et al. 1986; Weyuker 1988). For example, Weyuker has shown that many information flow metrics do not increase monotonically when two programs are combined. Metrics that fail to meet this fundamental property would imply that it takes fewer resources to design a larger piece of code than a smaller one, which is not consistent with practical experience. It has also been noted that most traditional metrics lack empirical validation, and are inconsistent, lacking in both discriminative and predictive power (Schneidewind 1992), Fenton also suggests that traditional metrics have ignored basic measurement principles resulting in measurements that do not preserve the intuitive ordinal relations that programmers require (Fenton 1992). Kemerer and Card and Agresti observe that some traditional metrics are difficult and labor intensive to collect (Kemerer 1993; Card and Agresti 1988). Consequently, traditional have metrics become a burden to collect and use. Kearney et al. express this succinctly, "the results of these (i.e. metric) explorations are difficult to interpret and provide only weak support for the use of complexity measures. Until, more comprehensive evidence is available, software complexity measures should be used very cautiously" (Kearney et al. 1986 p. 1044 ). Thus the combination of issues relating to appropriateness, measurement, validation and implementation has precipitated the need for new metrics for OO.
3.2 Previous measurement in OO

Given the significant interest, a number of metrics have been proposed for measuring the complexity of OO software. Broadly, the prior work in this area can be sub-divided into four categories: programming style guidelines, project metrics, metrics based on graph theory and special purpose metrics. Table 1.1 at the end of the chapter provides a summary of the extant OO metrics.

3.2.1 Programming Style Guidelines

Lieberherr and his colleagues present a formal attempt at defining the rules of correct object oriented programming style, building on concepts of coupling and cohesion that are used in traditional programming (Lieberherr, Holland and Riel 1988). There are two "laws" for good programming style labeled the "Weak Law of Demeter" and the "Strong Law of Demeter". Both laws relate to the definition of instance variables in a given class. If the strong law is followed, any changes to data structures will affect only methods declared in classes that define the data structures. Strict adherence to the strong law entails a penalty in terms of performance and increased size. The weak law represents a compromise between strict information hiding and performance. Numerous examples of good and bad programming styles are presented, but there are no concrete metrics proposed that can be used by managers on an ongoing basis.

Coplien suggests a number of rules of thumb for OO programming in C++ (Coplien 1993). Some of these rules could easily map into metrics that can be measured on OO software. For example, the number of parameters of a member function could be an useful metric. However, there other rules that are less amenable to easy measurement like "reduce the number of switch statements that select an action on the basis of a field". Without rigorous definitions for the terms
"action" and "field", this would be implementation dependent. Similarly, Meyers offers 50 rules for avoiding complexity in C++ programs (Meyers 1992). While these heuristics can be useful to individual programmers, they are a) not driven by theoretical arguments b) are language specific (limited to C++) and c) not measurable without a specially built C++ pre-processor.

The programming guidelines are useful tips for individual programmers, but they are uniformly focused on coding, not design of systems. The language specificity also limits broader applicability of this literature. More importantly, there are no formal or empirical proofs provided to substantiate the validity of these proposals. Consequently, these metrics do not offer consistent, collectable, trackable data that can relate to time, defects or overall costs of an OO project.

3.2.2 Project Metrics

Morris has presented proposals for OO metrics that are aimed at the project level of analysis (Morris 1988). His proposal for average method complexity, application granularity and object library effectiveness aggregate method, class level constructs to the project level. Morris's work is based on one in-depth case study of a commercial OO project, and represents a significant contribution to the literature. There are two concerns with Morris's proposals, 1) from a measurement point of view, aggregation of ordinal data is usually considered inappropriate and 2) any attempt at validation will require a large sample of projects, which is difficult to obtain. Henderson-Sellers's metrics are expressed in rigorous mathematical terms and measure the difficulty of class reuse (Henderson-Sellers 1991). However, Henderson-Sellers does not present empirical data or validation to substantiate their metrics. Another suggestion uses simple counts of objects and methods to develop and test a cost estimation model for OO development.
(Pfleeger and Palmer 1990). The cost model is similar to traditional cost estimation models and replaces lines of code with number of objects and methods. It has been noted that, while project level metrics can be valuable, there is inadequate collective knowledge about OO design to pursue metrics at this level of analysis at the present time (Kolewe 1993).

3.2.3 Graph theoretic Measures

Graph theoretic measures are based on the traditional notions of control flow in software. While traditional control flow metrics like McCabe do not account for OO concepts, a new set of graph-theoretic measures that specifically take inheritance relationships, information hiding and dynamic binding into consideration have been proposed (Sheetz, Whitmire). Sheetz et al. have suggested over 20 OO complexity metrics in a comprehensive measurement scheme that begins at the method level, and culminates at the system level. The rigor of the graph theoretic approach coupled with analytical evaluation of the metrics and automated data collection from an experimental project makes this research a valuable contribution to the literature. However, no empirical validation of the metrics are presented. Another criticism of this research is the inability of graph theory to adequately characterize encapsulation in OO systems (Bilow 1992).

3.2.4 Special purpose OO metrics

Several other researchers have proposed measures for specific OO concepts or applications. Moreau and Dominick suggest three metrics for OO graphical information systems, but do not provide formal, testable definitions (Moreau and Dominick 1989). Rajaraman and Lyu suggest a coupling metric, but it is specific to C++ programs (Rajaraman and Lyu 1992). Chen and Lu propose a set of
metrics based on subjectively assigned values of operational parameters (e.g. assigning a value of 0 to an integer operation, and 10 for a file operation) to measure complexity of an experimental system. These measures were regressed against the scores assigned to individual classes by designers, and found to be statistically valid. Besides the limitations of studying an experimental system, and subjective ratings of effort and operations, this study does not provide any data on possible multi-collinearity between different measures, and the measures are not based on any strong theoretical definitions of object oriented concepts.

Rising and Carliss have proposed, measured (using automated collection) and validated a single metric for the concept of information-hiding in commercial OO systems (Rising and Callis 1993). Their statistical analysis demonstrates that maintenance of software with low information hiding metric values is more expensive. However, the external validity of this study is limited, since the maintenance effort variable was operationalized as the effort required by university students to maintain the commercial system. Lake and Cook also propose metrics for measuring inheritance in OOD. Their metrics are based on specific C++ constructs and programming environments like the number of include files used (a UNIX and C++ specific mechanism). Though an automated collection tool and preliminary data from small C++ applications are presented, theoretical and analytical validation are not examined. Another drawback of both these studies is the single dimensional nature of the metrics, since only one aspect of OOD is examined.

3.2.5 Summary of shortcomings

The extant literature on OO software metrics suffers from one (or more) of the following shortcomings:
• The metrics lack strong theoretical bases.

• The metrics are not evaluated for adequate measurement properties.

• The metrics measure only one aspect of OO complexity.

• The metrics are not collectable through automatable means.

• The metrics have not been gathered on commercial systems.

• There is no data on metrics usage in commercial organizations.

As outlined in section 2 of this chapter, the goals of this dissertation are to propose metrics that avoid all of the above shortcomings and contribute to the literature in software engineering management by laying the foundation for a more formal, yet practical language for software measurement.
<table>
<thead>
<tr>
<th>Authors</th>
<th>Metric type</th>
<th>Theoretical basis</th>
<th>Empirical data</th>
<th>Analytical evaluation</th>
<th>Machinability</th>
<th>Contribution</th>
</tr>
</thead>
<tbody>
<tr>
<td>Libereherr et al. (1988)</td>
<td>Programming guidelines</td>
<td>structured design literature</td>
<td>not presented</td>
<td>none</td>
<td>not demonstrated</td>
<td>Weak and strong laws of Demeter that encourage better OO design</td>
</tr>
<tr>
<td>Morris (1988)</td>
<td>Project metrics (e.g. average method complexity)</td>
<td>extracted from traditional metrics and one case study</td>
<td>not presented</td>
<td>none</td>
<td>not demonstrated</td>
<td>Review of traditional metrics, field based study</td>
</tr>
<tr>
<td>Moreau and Dominick (1989)</td>
<td>special purpose metrics for graphical user interface systems</td>
<td>none presented from one project</td>
<td>none presented</td>
<td>not demonstrated</td>
<td>Three metrics for assessing complexity of graphical user interfaces</td>
<td></td>
</tr>
<tr>
<td>Pfleeger and Palmer (1990)</td>
<td>special metric</td>
<td>function points</td>
<td>one project</td>
<td>none</td>
<td>not demonstrated</td>
<td>Provides a candidate replacement to function points in GUI systems</td>
</tr>
<tr>
<td>Banker et al. (1991)</td>
<td>special purpose metric for CASE environments</td>
<td>function point literature from a commercial project</td>
<td>none</td>
<td>not demonstrated</td>
<td>Proposes object counts as a replacement for function points</td>
<td></td>
</tr>
<tr>
<td>Chidamber and Kemerer (1991)</td>
<td>design metrics for class complexity</td>
<td>ontology and measurement theory</td>
<td>not presented</td>
<td>against Weyuker's criteria</td>
<td>not demonstrated</td>
<td>Rigorously defined suite to measure OO design</td>
</tr>
<tr>
<td>Henderson-Sellers (1991)</td>
<td>special metric to measure reuse</td>
<td>not presented</td>
<td>none</td>
<td>not demonstrated</td>
<td>Rigorously defined metric to examine reuse in an application</td>
<td></td>
</tr>
<tr>
<td>Lake and Cook (1992)</td>
<td>special purpose metric to measure inheritance</td>
<td>none presented from a number of small C++ applications</td>
<td>none</td>
<td>commercial quality tool for gathering and reporting metric data</td>
<td>Integrated tool set for C++ environments to measure inheritance related metrics</td>
<td></td>
</tr>
</tbody>
</table>

**TABLE 1.1: SUMMARY OF EXTANT OBJECT ORIENTED METRICS**
<table>
<thead>
<tr>
<th>Author/s</th>
<th>Metric type</th>
<th>Theoretical basis</th>
<th>Empirical data</th>
<th>Analytical evaluation</th>
<th>Machinability</th>
<th>Contribution</th>
</tr>
</thead>
<tbody>
<tr>
<td>Meyers (1992)</td>
<td>Programming guidelines</td>
<td>programming styles</td>
<td>not presented</td>
<td>none</td>
<td>not demonstrated</td>
<td>Rules of thumb for avoiding complexity in programs</td>
</tr>
<tr>
<td>Rajaram and Lyu (1992)</td>
<td>special metric (specific to C++) to measure coupling</td>
<td>structured programming</td>
<td>not presented</td>
<td>none</td>
<td>not demonstrated</td>
<td>Analysis of issues relating to coupling in OOD</td>
</tr>
<tr>
<td>Sheetz et al. (1992)</td>
<td>graph theoretic metric for methods, classes and projects</td>
<td>graph theory, matrix algebra</td>
<td>from one experimental project</td>
<td>based on qualitative recommendations, with mention of Weyuker's criteria</td>
<td>demonstrated</td>
<td>Rigorous, practical, machinable metrics, but no empirical validation</td>
</tr>
<tr>
<td>Chen and Lu (1993)</td>
<td>special purpose metrics to measure class hierarchy, reuse, coupling and cohesion</td>
<td>none presented</td>
<td>from one experimental system</td>
<td>none</td>
<td>demonstrated</td>
<td>Metrics correlated positively to subjective evaluations of experts</td>
</tr>
<tr>
<td>Coplien (1993)</td>
<td>Programming guidelines</td>
<td>sociology of objects</td>
<td>not presented</td>
<td>none</td>
<td>not demonstrated</td>
<td>Tailored to C++ applications, avoidance of complexity</td>
</tr>
</tbody>
</table>

**TABLE 1.1: SUMMARY OF EXTANT OBJECT ORIENTED METRICS**
<table>
<thead>
<tr>
<th>Author/s</th>
<th>Metric type</th>
<th>Theoretical basis</th>
<th>Empirical data</th>
<th>Analytical evaluation</th>
<th>Machinability</th>
<th>Contribution</th>
</tr>
</thead>
<tbody>
<tr>
<td>Li and Henry</td>
<td>variants of MOOSE*</td>
<td>none presented</td>
<td>from experimental and commercial projects</td>
<td>none</td>
<td>research quality tool</td>
<td>MOOSE and variants explain variance in maintenance effort</td>
</tr>
<tr>
<td>(1993)</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Rising and Carliss</td>
<td>special purpose metric to measure information hiding</td>
<td>measurement theory</td>
<td>from a commercial ADA project</td>
<td>evaluated against analytical properties of Fenton</td>
<td>demonstrated</td>
<td>Proposed metric of information hiding correlates positively with maintenance effort</td>
</tr>
</tbody>
</table>

* MOOSE refers to the metrics proposed in this dissertation

**TABLE 1.1: SUMMARY OF EXTANT OBJECT ORIENTED METRICS**
4.0 BIBLIOGRAPHY


Myers, G. J. (1977) "An extension to the cyclomatic measure of program complexity." *SIGPLAN Notices*, 12, 10, 61-64.


Chapter Two: Theoretical Development of the Metrics Suite
Chapter Outline

1.0 Theory Base for OOD Metrics ................................................................. 45
  1.1 Measurement theory base ................................................................. 46
  1.2 Ontological Definitions ................................................................. 49
2.0 Viewpoint Collection ........................................................................ 56
3.0 Metrics Evaluation Criteria ............................................................. 57
4.0 The Metrics Suite ............................................................................. 62
  Metric 1: Weighted Methods Per Class (WMC) .................................... 62
    Definition ........................................................................................ 62
    Theoretical basis ........................................................................... 62
    Viewpoints .................................................................................... 63
    Analytical Evaluation of Weighted Methods Per Class (WMC) ......... 63
  Metric 2: Depth of Inheritance Tree (DIT) .......................................... 64
    Definition ........................................................................................ 64
    Theoretical basis ........................................................................... 64
    Viewpoints .................................................................................... 65
    Analytical Evaluation of Depth of Inheritance Tree (DIT) .............. 65
  Metric 3: Number of children (NOC) .................................................. 68
    Definition ........................................................................................ 68
    Theoretical basis ........................................................................... 68
    Viewpoints .................................................................................... 68
    Analytical Evaluation of Number Of Children (NOC) ................... 68
  Metric 4: Coupling between object classes (CBO) ............................. 70
    Definition ........................................................................................ 70
    Theoretical basis ........................................................................... 70
    Viewpoints .................................................................................... 70
    Analytical Evaluation of Coupling Between Objects (CBO) .......... 71
  Metric 5: Response For a Class (RFC) ................................................ 72
    Definition ........................................................................................ 72
    Theoretical basis ........................................................................... 72
    Viewpoints .................................................................................... 72
    Analytical Evaluation of Response for a Class (RFC) ................... 73
  Metric 6: Lack of Cohesion in Methods (LCOM) ................................ 74
    Definition ........................................................................................ 74
    Theoretical basis ........................................................................... 75
    Viewpoints .................................................................................... 75
    Analytical Evaluation of Lack Of Cohesion Of Methods (LCOM) ... 76
5.0 Summary .......................................................................................... 78
  5.1 The Metrics Suite ........................................................................... 78
  5.2 Summary of Analytical Results ...................................................... 79
  5.3 Booch OOD Steps .......................................................................... 81
6.0 Bibliography ..................................................................................... 82
1.0 Theory Base for OOD Metrics

While there are many object oriented design (OOD) methodologies, one that reflects the essential features of OOD is presented by Booch (Booch 1991)\(^1\). He outlines four major steps involved in the object-oriented design process:

1) Identification of Classes (and Objects) In this step, key abstractions in the problem space are identified and labeled as potential classes and objects.

2) Identify the Semantics of Classes (and Objects) In this step, the meaning of the classes and objects identified in the previous step is established, this includes definition of the life-cycles of each object from creation to destruction.

3) Identify Relationships between Classes (and Objects) In this step, class and object interactions, such as patterns of inheritance among classes and patterns of visibility among objects and classes (what classes and objects should be able to "see" each other) are identified.

4) Implementation of Classes (and Objects) In this step, detailed internal views are constructed, including definitions of methods and their various behaviors.

Whether the design methodology chosen is Booch's OOD or any of the several other methodologies, design of classes is consistently declared to be central to the OO paradigm. As deChampeaux et al. suggest, class design is the highest priority in OOD, and since it deals with the functional requirements of the system, it must occur before systems design (mapping objects to processors, processes) and program design (reconciling of functionality using the target languages, tools etc.) (deChampeaux et al. 1992). Given the importance of class design, the metrics outlined in this dissertation

\(^{20}\) For a comparison and critique of six different OO analysis and design methodologies see (Fichman and Kemerer 1992).
specifically are designed to measure the complexity in the design of classes\(^2\). The limitation of this approach is that possible dynamic behavior of a system is not captured. Since the proposed metrics are aimed at assessing the design of an object oriented system rather than its specific implementation, the potential benefits of this information can be substantially greater than metrics aimed at later phases in the life-cycle of an application. In addition, implementation-independent metrics will be applicable to a larger set of users, especially in the early stages of industry’s adoption of OO before dominant design standards emerge.

1.1 Measurement theory base

An object oriented design can be conceptualized as a *relational system*, which is defined by Roberts as an ordered tuple consisting of a set of *elements*, a set of *relations* and a set of *binary operations*. (Roberts 1979). More specifically, an object oriented design, \( D \), is conceptualized as a relational system consisting of object-elements (classes and objects), empirical relations and binary operations that can be performed on the object-elements. By starting with these definitions, the mathematical role of metrics as a mapping (or transformation) can be formally outlined. Notationally:

\[
D \equiv (A, R_1... R_n, O_1...O_m)
\]

where

\(A\) is a set of object-elements

\(R_1...R_n\) are empirical relations on object-elements of \(A\) (e.g., bigger than, smaller than, etc.)

\(2\) These are therefore static metrics, and they can be gathered prior to program execution.
$O_1...O_m$ are binary operations on elements of $A$ (e.g., combination).

A useful way to understand empirical relations on a set of object-elements is to consider the measurement of complexity. A designer generally has some intuitive ideas about the complexity of different object-elements, as to which element is more complex than another or which ones are equally complex. For example, a designer intuitively understands that a class that has many methods is generally more complex, ceteris paribus, than one that has few methods. This intuitive idea is defined as a viewpoint. The notion of a viewpoint was originally introduced to describe evaluation measures for information retrieval systems and is applied here to capture designer views (Cherniavsky and Lakhuty 1971). More recently, Fenton states that viewpoints characterize intuitive understanding and that viewpoints must be the logical starting point for the definition of metrics (Fenton 1991). An empirical relation is identical to a viewpoint, and the two terms are distinguished here only for the sake of consistency with the measurement theory literature.

A viewpoint is a binary relation $\geq$ defined on a set $P$ (the set of all possible designs). For $P$, $P'$, $P'' \in P$, the following two axioms must hold:

$$P \geq P' \text{ or } P' \geq P$$ (completeness: $P$ is more complex than $P'$ or $P'$ is more complex than $P$)

$$P \geq P', P' \geq P'' \Rightarrow P \geq P''$$ (transitivity: if $P$ is more complex than $P'$ and $P'$ is more complex than $P''$, then $P$ is more complex than $P''$)

i.e., a viewpoint must be of weak order (Roberts 1979).

To be able to measure something about an object design, the empirical relational system as defined above needs to be transformed to a formal
relational system. Therefore, let a formal relational system \( F \) be defined as follows:

\[
F = (C, S_1 \ldots S_n, B_1 \ldots B_m)
\]

\( C \) is a set of elements (e.g., real numbers)

\( S_1 \ldots S_n \) are formal relations on elements of \( C \) (e.g., \( >, <, = \))

\( B_1 \ldots B_m \) are binary operations on elements of \( C \) (e.g., \( +,-,\times \))

This required transformation is accomplished by a metric \( \mu \) which maps an empirical system \( D \) to a formal system \( F \). For every element \( a \in D, \mu(a) \in F \). It must be noted here that \( \mu \) preserves and does not alter the implicit notion underlying the empirical relations. The example below involving a set of school children illustrates the mapping between an empirical relational system and a formal relational system (Kaposi 1991):

<table>
<thead>
<tr>
<th>Empirical Relational System</th>
<th>Formal Relational System</th>
</tr>
</thead>
<tbody>
<tr>
<td>Heights of school children</td>
<td>Real Numbers</td>
</tr>
<tr>
<td>Relations: Equal or taller than</td>
<td>Relations: = or &gt;</td>
</tr>
<tr>
<td>Child ( P ) is taller than Child ( P' )</td>
<td>36 inch child &gt; 30 inch child</td>
</tr>
<tr>
<td>Binary Operations: Combination:</td>
<td>Binary Operations: +:</td>
</tr>
<tr>
<td>two children standing atop one another</td>
<td>add the real numbers associated with the two children</td>
</tr>
</tbody>
</table>

The empirical relation "Child \( P \) is taller than Child \( P' \)" in the above example is transformed to the formal relation "36 inch child > 30 inch child", enabling the explicit understanding of the heights of school children. The assumption in the argument for transformation of empirical relational systems to a formal empirical systems is that the "intelligence barrier" to understanding of the former is circumvented due to the transformation (Kriz 1988). In the

48
example of the school children the intelligence barrier is small, but the principle is that numerical representations produced by the transformation to formal systems help in better understanding the empirical system (see figure below).

![Diagram showing transformation from Empirical Relational System to Formal Relational System through Intelligence Barrier and Understanding, and back to Numerical Representation through Interpretation and Transformation]

In the example of the school children the intelligence barrier is small, but the principle is that numerical representations produced by the transformation to formal systems help in better understanding the empirical system. While the exercise of transformation may seem laborious for the simple example above, it can prove to be valuable in understanding complexity of software where the complexity relationships are not visible or not well understood (Kaposi 1991). Design of object oriented systems is a difficult undertaking in part due to the newness of the technology, and the consequent lack of formal metrics to aid designers and managers in managing complexity in OOD.

1.2 Ontological Definitions

The ontological principles proposed by Bunge in his "Treatise on Basic Philosophy" forms the basis of the concept of objects. While Bunge did not
provide specific ontological definitions for object oriented concepts, several recent researchers have employed his generalized concepts to the object oriented domain (Wand 1989, Wand and Weber 1990, Weber and Zhang 1991). Bunge's ontology has considerable appeal for OO researchers since it deals with the meaning and definition of representations of the world, which are precisely the goals of the object oriented approach (Parsons and Wand 1993). Consistent with this ontology, objects are defined independent of implementation considerations and encompass the notions of encapsulation, independence and inheritance. According to this ontology, the world is viewed as composed of things, referred to as substantial individuals, and concepts. The key notion is that substantial individuals possess properties. A property is a feature that a substantial individual possesses inherently. An observer can assign features to an individual, but these are attributes and not properties. All substantial individuals possess a finite set of properties; as Bunge notes, "there are no bare individuals except in our imagination" (Bunge 1977).

Some of the attributes of an individual will reflect its properties. Indeed, properties are recognized only through attributes. A known property must have at least one attribute representing it. Properties do not exist on their own, but are "attached" to individuals. On the other hand, individuals are not simply bundles of properties. A substantial individual and its properties collectively constitute an object (Wand 1989, Wand and Weber 1989). Therefore, an object is not simply a bundle of methods, but a representation of the application domain that includes the methods and instance variables that a designer assigns to that object. Another benefit of this stream of
research is that it provides a formal mathematical approach to dealing specifically with the key ideas of object orientation.

An object can be represented in the following manner:

\[ X = \langle x, p(x) \rangle \] where \( x \) is the substantial individual and \( p(x) \) is the finite collection of its properties.

\( x \) can be considered to be the token or name by which the object is represented in a system. In object oriented terminology, the instance variables\(^3\) together with its methods\(^4\) are the properties of the object (Banerjee et al. 1987).

Using these representations of objects, previous research has defined concepts like scope and similarity that are relevant to object oriented systems (Bunge 1977, Wand 1989). Following this tradition, this dissertation defines in the paragraphs below two important software design concepts for object classes, coupling and cohesion. Intuitively, coupling refers to the degree of interdependence between parts of a design, while cohesion refers to the internal consistency within parts of the design. All other things being equal, good software design practice calls for minimizing coupling and maximizing cohesiveness. It should be noted that these definitions are derived from the ontology of objects as opposed to other sources that have been graph-theory (e.g. McCabe (McCabe 1976)), information content (e.g. Halstead (Halstead 1977)) or structural attributes (e.g. Card and Agresti (Card and Agresti 1988)).

For further details on the appropriateness of the ontological approach the reader is referred to the comprehensive treatment of the subject in (Wand 1989), (Parsons and Wand 1993) and (Tagaki and Wand 1991).

\[^3\] An instance variable store a unique value in each instance of a class.
\[^4\] A method is an operation on an object that is defined as part of the declaration of the class.
**Coupling.** In ontological terms, "two objects are coupled if and only if at least one of them acts upon the other, \(X\) is said to act upon \(Y\) if the history of \(Y\) is affected by \(X\), where history is defined as the chronologically ordered states that a thing traverses in time" [41, p 547].

Let \(X = <x, p(x)>\) and \(Y = <y, p(y)>\) be two objects.

\[
p(x) = \{ M_X \} \cup \{ I_X \}
\]

\[
p(y) = \{ M_Y \} \cup \{ I_Y \}
\]

where \(\{ M_i \}\) is the set of methods and \(\{ I_i \}\) is the set of instance variables of object \(i\).

Using the above definition of coupling, any action by \(\{M_X\}\) on \(\{M_Y\}\) or \(\{I_Y\}\) constitutes coupling, as does any action by \(\{M_Y\}\) on \(\{M_X\}\) or \(\{I_X\}\). When \(M_X\) calls \(M_Y\), \(M_X\) alters the history of the usage of \(M_Y\); similarly when \(M_X\) uses \(I_Y\), it alters the access and usage history of \(I_Y\). Therefore, any evidence of a method of one object using methods or instance variables of another object constitutes coupling. Since objects of the same class have the same properties, two classes are coupled when methods declared in one class use methods or instance variables of the other class\(^5\).

**Cohesion.** Bunge defines similarity \(\sigma()\) of two things to be the intersection of the sets of properties of the two things [5, p 87]:

\[
\sigma(X,Y) = p(x) \cap p(y)
\]

Following this general principle of defining similarity in terms of sets, the degree of similarity of the methods within the object can be defined to be the intersection of the sets of instance variables that are used by the methods.

\(^5\) Note that this will include coupling due to inheritance.
This is an extension of Bunge's definition of similarity to similarity of methods. It should be clearly understood that instance variables are not properties of methods, but it is consistent with the notion that methods of an object are intimately connected to its instance variables.

\[ \sigma(M_1,M_2) = \{I_1\} \cap \{I_2\} \]

where \( \sigma(M_1,M_2) \) = degree of similarity of methods \( M_1 \) and \( M_2 \)

and \( \{I_1\} = \) set of instance variables used by method \( M_1 \).

Example: Let \( \{I_1\} = \{a,b,c,d,e\} \) and \( \{I_2\} = \{a,b,e\} \). \( I_1 \cap I_2 \) is non-empty, and \( \sigma(M_1,M_2) = \{a,b,e\} \).

The degree of similarity of methods relates both to the conventional notion of cohesion in software engineering, (i.e., keeping related things together) as well as encapsulation, that is, the bundling of methods and instance variables in an object class. The degree of similarity of methods can be viewed as a major aspect of object class cohesiveness. If an object class has different methods performing different operations on the same set of instance variables, the class is cohesive. This view of cohesion is centered on data that is encapsulated within an object and on how methods interact with data. It is proposed for object orientation as an alternative to other previous approaches, such as generalization-specialization cohesion or service cohesion as defined by Coad and Yourdon (Coad and Yourdon 1991).

**Complexity of an object.** Bunge defines complexity of an individual to be the "numerosity of its composition", implying that a complex individual has a large number of properties (Bunge 1977). Using this definition as a base, the complexity of an object class can be defined to be the cardinality of its set of properties.
Complexity of $\langle x, p(x) \rangle = \| p(x) \|$, where $\| p(x) \|$ is the cardinality of $p(x)$.

**Scope of Properties.** In simple terms, a class is a set of objects that have common properties (i.e. methods and instance variables). A designer develops an abstraction of the application domain by arranging the classes in a hierarchy. The inheritance hierarchy is a directed acyclic graph that can be described as a tree structure with classes as nodes, leaves and a root. In any application, there can be many possible choices for the class hierarchy. Design choices on the hierarchy employed to represent the application are essentially choices about restricting or expanding the scope of properties of the classes of objects in the application. Two design decisions which relate to the inheritance hierarchy can be defined. They are *depth of inheritance* of a class of objects and the *number of children* of the class.

\[
\text{Depth of Inheritance} = \text{depth of the class in the inheritance tree}
\]

The depth of a node of a tree refers to the length of the maximal path from the node to the root of the tree.

\[
\text{Number of Children} = \text{Number of immediate descendants of the class}
\]

Both these concepts relate to the ontological notion of scope of properties\(^6\), i.e., how far does the influence of a property extend? Depth of inheritance indicates the extent to which the class is influenced by the properties of its ancestors, and number of children indicates the potential impact on descendants. The depth of inheritance and number of children collectively indicate the genealogy of a class.

---

\(^6\) For formal mathematical definitions of scope of properties, see (Wand and Weber 1990).
Methods as measures of communication. In the object oriented approach, objects communicate primarily through message passing\(^7\). A message can cause an object to "behave" in a particular manner by invoking a particular method. Methods can be viewed as definitions of responses to possible messages (Banerjee et al. 1987). It is reasonable, therefore, to define a response set for a class of objects in the following manner:

Response set of a class of objects = \{set of all methods that can be invoked in response to a message to an object of the class\}

Note that this set will include methods outside the class as well, since methods within the class may call methods from other classes. The response set will be finite since the properties of a class are finite and there are a finite number of classes in a design. During the implementation and maintenance phases of systems development, the response set may change, since new object instantiations may create different communication links.

Combination of object classes. As Booch observes, class design is an iterative process involving sub-classing (creating new classes based on existing ones), factoring (splitting existing classes into smaller ones) and composition (or combination) that unites existing classes into one. The notion of sub-classing is well understood in OO design, but the semantics of combination are less clear. However, Bunge's ontology provides a basis for defining the combination of object classes. From the principle of additive aggregation of two (or more) things, the combination of two object classes results in another class whose properties are the union of the properties of the component classes.

\(^7\)While objects can communicate through more complex mechanisms like bulletin boards, a majority of OO designers employ message passing as the primary mechanism for communicating between objects (Booch 1991).
Let \( X = \langle x, p(x) \rangle \) and \( Y = \langle y, p(y) \rangle \) be two object classes, then \( X+Y \) is defined as \( \langle z, p(z) \rangle \) where \( z \) is the token with which \( X+Y \) is represented and \( p(z) \) is given by:

\[
p(z) = p(x) \cup p(y)
\]

For example, if a class \( \text{foo}_a \) has properties (i.e. methods and instance variables) \( a,b,c,d \) and class \( \text{foo}_b \) has properties \( a,l,m,n \) then \( \text{foo}_a + \text{foo}_b \) has properties \( a,b,c,d,l,m,n \). If \( \text{foo}_a \) and \( \text{foo}_b \) both have identical properties \( a,b,c,d \), then \( \text{foo}_a + \text{foo}_b \) will also have the same properties \( a,b,c,d \).

Designers' empirical operations of combining two classes in order to achieve better representation are formally denoted here as combination and shown with a \( + \) sign. Combination results in a single joint state space of instance variables and methods instead of two separate state spaces; the only definite result of combination of two classes is the elimination of all prior messages between the two component classes.

2.0 Viewpoint Collection

As defined earlier, a design encompasses the implicit ideas designers have about complexity. These viewpoints are the empirical relations \( R_1, R_2, ... R_n \) in the formal definition of the design \( D \). The viewpoints that were used in constructing the metrics presented in this dissertation were gathered from extensive collaboration with a highly experienced team of software engineers from a software development organization. This organization has used OOD in more than four large projects over a period of five years. Though the primary development language for all projects at this site was C++, the research aim was to propose metrics that were language independent.
3.0 **Metrics Evaluation Criteria**

Several researchers have recommended properties that software metrics should possess to increase their usefulness. For example, Basili and Reiter suggest that metrics should be sensitive to externally observable differences in the development environment, and must also correspond to intuitive notions about the characteristic differences between the software artifacts being measured (Basili and Reiter 1979). The majority of recommended properties are qualitative in nature and consequently, most proposals for metrics have tended to be informal in their evaluation of metrics.

Consistent with the desire to move metrics research into a more rigorous footing, it is desirable to have a formal set of criteria with which to evaluate proposed metrics. More recently, Weyuker has developed a formal list of desiderata for software metrics and has evaluated a number of existing software metrics using these properties (Weyuker 1988). These desiderata include notions of monotonicity, interaction, non-coarseness, non-uniqueness and permutation.

Weyuker's properties are not without criticism. Fenton suggests that Weyuker's properties are not predicated on a single consistent view of complexity (Fenton 1991). Zuse criticizes Weyuker on the grounds that her properties are not consistent with the principles of scaling (Zuse 1992). Cherniavsky and Smith suggest that Weyuker's properties should be used carefully since the properties may only give necessary, but not sufficient conditions for good complexity metrics (Cherniavsky and Smith 1991).

However, as Gustafson and Prasad suggest, formal analytical approaches subsume most of the earlier, less well-defined and informal properties and
provide a language for evaluation of metrics (Gustafson and Prasad 1991). Her list, while currently still subject to debate and refinement, is a widely known formal analytical approach, and is therefore chosen for this analysis. Finally, in the course of the analysis presented below further suggestions are offered on the relative appropriateness of these axioms for object oriented development.

Of Weyuker's nine properties, three will be dealt with only briefly here. Weyuker's second property, "granularity", only requires that there be a finite number of cases having the same metric value. Since the universe of discourse deals with at most a finite set of applications, each of which has a finite number of classes, this property will be met by any metric measured at the class level. The "renaming property" (Property 8) requires that when the name of the measured entity changes\(^8\), the metric should remain unchanged. As all metrics proposed in this dissertation are measured at the class level and, as none of them depend on the names of the class or the methods and instance variables, they also satisfy this property. Since both these properties are met, they will not be discussed further.

Weyuker's seventh property requires that permutation of elements within the item being measured can change the metric value. The intent is to ensure that metric values change due to permutation of program statements. This property is meaningful in traditional program design, where the ordering of if-then-else blocks could alter the program logic (and consequent complexity). In OOD, a class is an abstraction of the problem space, and the order of

\[^8\] Note, this property deals only with the name of the entity, and not the names associated with any of the internals of the entity.
statements within the class definition has no impact on eventual execution or use. For example, changing the order in which methods are declared does not affect the order in which they are executed, since methods are triggered by the receipt of different messages from other objects. In fact, Cherniavsky and Smith specifically suggest that this property is not appropriate for OOD metrics because "the rationales used may be applicable only to traditional programming" (Cherniavsky and Smith 1991, p. 638). Therefore, this property is not considered further. The remaining six properties are repeated below\(^9\).

**Property 1: Non-coarseness**

Given a class \( P \) and a metric \( \mu \), another class \( Q \) can always be found such that: \( \mu(P) \neq \mu(Q) \). This implies that not every class can have the same value for a metric, otherwise it has lost its value as a measurement.

**Property 2: Non-uniqueness (notion of equivalence)**

There can exist distinct classes \( P \) and \( Q \) such that \( \mu(P) = \mu(Q) \). This implies that two classes can have the same metric value, i.e., the two classes are equally complex.

**Property 3: Design details are important**

Given two class designs, \( P \) and \( Q \), which provide the same functionality, does not imply that \( \mu(P) = \mu(Q) \). The specifics of the class must influence the metric value. The intuition behind Property 3 is that even though two class designs

---

\(^9\) Readers familiar with Weyuker's work should note that the exclusion of these three properties makes the property numbers used here no longer consistent with the original property numbers. It should also be noted that Weyuker's definitions have been modified where necessary to use classes rather than programs.
perform the same function, the details of the design matter in determining the metric for the class.

**Property 4: Monotonicity**

For all classes P and Q, the following must hold: $\mu(P) \leq \mu(P+Q)$ and $\mu(Q) \leq \mu(P+Q)$ where $P + Q$ implies combination of P and Q\(^{10}\). This implies that the metric for the combination of two classes can never be less than the metric for either of the component classes.

**Property 5: Non-equivalence of interaction**

$\exists P, \exists Q, \exists R$, such that

$\mu(P) = \mu(Q)$ does not imply that $\mu(P+R) = \mu(Q+R)$.

This suggests that interaction between P and R can be different than interaction between Q and R resulting in different complexity values for P+R and Q+R.

**Property 6: Interaction increases complexity**

$\exists P$ and $\exists Q$ such that:

$\mu(P) + \mu(Q) < \mu(P+Q)$

The principle behind this property is that when two classes are combined, the interaction between classes can increase the complexity metric value.

\(^{10}\)It should be noted that P+Q is the combination of two classes, whereas $\mu(P) + \mu(Q)$ is the addition of the metric value of P and the metric value of Q.
Assumptions. Some basic assumptions made regarding the distribution of methods and instance variables in the discussions for each of the metric properties.

Assumption 1:

Let \( X_i \) = The number of methods in a given class \( i \).
\[ Y_i = \text{The number of methods called from a given method } i. \]
\[ Z_i = \text{The number of instance variables used by a method } i. \]
\[ C_i = \text{The number of couplings between a given class of objects } i \text{ and all other classes.} \]

\( X_i, Y_i, Z_i, C_i \) are discrete random variables each characterized by some general distribution function. Further, all the \( X_i \)s are independent and identically distributed (i.i.d.). The same is true for all the \( Y_i \)s, \( Z_i \)s and \( C_i \)s. This suggests that the number of methods, variables and couplings follow a statistical distribution that is not apparent to an observer of the system. Further, the observer cannot predict the variables, methods etc. of one class based on the knowledge of the variables, methods and couplings of another class in the system.

Assumption 2: In general, two classes can have a finite number of "identical" methods in the sense that a combination of the two classes into one class would result in one class's version of the identical methods becoming redundant. For example, a class "foo_one" has a method "draw" that is responsible for drawing an icon on a screen; another class "foo_two" also has a "draw" method. Now a designer decides to have a single class "foo" and combines the two classes. Instead of having two different "draw" methods the designer can decide to just have one "draw" method (albeit modified to reflect the new abstraction).
Assumption 3: The inheritance tree is "full", i.e., there is a root, intermediate nodes and leaves. This assumption merely states that an application does not consist only of stand-alone classes; there is some use of subclassing\(^{11}\).

4.0 THE METRICS SUITE

Metric 1: Weighted Methods Per Class (WMC)

Definition.

Consider a Class C\(_1\), with methods M\(_1\), \ldots, M\(_n\) that are defined in the class. Let c\(_1\), \ldots, c\(_n\) be the complexity\(^{12}\) of the methods. Then:

\[
\text{WMC} = \sum_{i=1}^{n} c_i
\]

If all method complexities are considered to be unity, then WMC = n, the number of methods.

Theoretical basis.

WMC relates directly to Bunge's definition of complexity of a thing, since methods are properties of object classes and complexity is determined by the cardinality of its set of properties. The number of methods is, therefore, a

\(^{11}\) Based on the data from sites A and B, this appears to be a reasonable assumption.

\(^{12}\) Complexity is deliberately not defined more specifically here in order to allow for the most general application of this metric. It can be argued that developers approach the task of writing a method as they would a traditional program, and therefore some traditional static complexity metric may be appropriate. This is left as an implementation decision, as the general applicability of any existing static complexity metric has not been generally agreed upon. Any complexity metric used in this manner should have the properties of an interval scale to allow for summation. The general nature of the WMC metric is presented as a strength, not a weakness of this metric as has been suggested elsewhere (Kalakota et al. 1993).
measure of class definition as well as being attributes of a class, since attributes correspond to properties\textsuperscript{13}.

\textbf{Viewpoints}

- The number of methods and the complexity of methods involved is a predictor of how much time and effort is required to develop and maintain the class.

- The larger the number of methods in a class the greater the potential impact on children, since children will inherit all the methods defined in the class.

- Classes with large numbers of methods are likely to be more application specific, limiting the possibility of reuse.

\textit{Analytical Evaluation of Weighted Methods Per Class (WMC)}

From assumption 1, the number of methods in class P and another class Q are i.i.d., this implies that there is a non-zero probability that $\exists$ Q such that $\mu(P) \neq \mu(Q)$, therefore property 1 is satisfied. Similarly, there is a non-zero probability that $\exists$ R such that $\mu(P) = \mu(R)$. Therefore property 2 is satisfied. The function of the class does not define the number of methods in a class. The choice of the number of methods is a design decision and independent of the functionality of the class, therefore Property 3 is satisfied. Let $\mu(P) = n_P$ and $\mu(Q) = n_Q$, then $\mu(P+Q) = n_P + n_Q - \delta$, where $\delta$ is the number of common

---

\textsuperscript{13} Note that this is one interpretation of Bunge's definition, since it does not include the number of instance variables in the definition of the metric. This is done for two reasons: 1) Developers expressed the view that methods are more time consuming to design than instance variables, and adding the instance variables to the definition will increase the noise in the relationship between this metric and design effort. 2) By restricting the metric to methods, the process of adding static complexity weights to methods will not detract from the comprehensibility of the metric.
methods between P and Q. Clearly, the maximum value of \( \partial \) is \( \min(nP, nQ) \). Therefore, \( \mu(P+Q) \geq nP+nQ - \min(nP, nQ) \). It follows that \( \mu(P+Q) \geq \mu(P) \) and \( \mu(P+Q) \geq \mu(Q) \), thereby satisfying Property 4. Now, let \( \mu(P) = n, \mu(Q) = n, \) and \( \exists \) a class R such that it has a number of methods \( \partial \) in common with Q (as per assumption 2) and \( \beta \) methods in common with P, where \( \partial \neq \beta \). Let \( \mu(R) = r \).

\[
\begin{align*}
\mu(P+R) &= n + r - \beta \\
\mu(Q+R) &= n + r - \partial
\end{align*}
\]

Therefore \( \mu(P+R) \neq \mu(Q+R) \) and Property 5 is satisfied. For any two classes P and Q, \( nP + nQ - \partial \leq nP + nQ \) i.e., \( \mu(P+Q) \leq \mu(P) + \mu(Q) \) for any P and Q.

Therefore, Property 6 is not satisfied\(^{14}\).

**Metric 2: Depth of Inheritance Tree (DIT)**

**Definition**

Depth of inheritance of the class is the DIT metric for the class. In cases involving multiple inheritance, the DIT will be the maximum length from the node to the root of the tree.

**Theoretical basis.**

DIT relates to Bunge's notion of the scope of properties. DIT is a measure of how many ancestor classes can potentially affect this class.

\(^{14}\) The implications of not satisfying Property 6 is discussed in the Summary section.
Viewpoints.

- The deeper a class is in the hierarchy, the greater the number of methods it is likely to inherit, making it more complex to predict its behavior\(^\text{15}\).

- Deeper trees constitute greater design complexity, since more methods and classes are involved.

- The deeper a particular class is in the hierarchy, the greater the potential reuse of inherited methods.

Analytical Evaluation of Depth of Inheritance Tree (DIT)

Per assumption 3, the inheritance hierarchy has a root and leaves. The depth of inheritance of a leaf is always greater than that of the root. Therefore, property 1 is satisfied. Also, since every tree has at least some nodes with siblings (per assumption 3), there will always exist at least two classes with the same depth of inheritance, i.e., property 2 is satisfied. Design of a class involves choosing what properties the class must inherit in order to perform its function. In other words, depth of inheritance is design implementation dependent, and Property 3 is satisfied.

When any two classes P and Q are combined, there are three possible cases\(^\text{16}\):

i) P and Q are siblings ii) P and Q are neither children nor siblings of each other and iii) one is the child of the other.

\(^{15}\) Interestingly, this has been independently observed by other researchers (Leijter et al. 1992).

\(^{16}\) A fourth case would involve multiple inheritance, and it can be shown that Property 4 is satisfied in this case also. Suppose A has two subclasses P and X, Q is a subclass of X and also a subclass of B. \(\mu(P) = 1\) and \(\mu(Q) = 2\). The combined class P+Q will be a subclass of X and B, and
Case i) P and Q are siblings

In this case, \( \mu(P) = \mu(Q) = n \) and \( \mu(P+Q) = n \), i.e. Property 4 is satisfied.

Case ii) P and Q are neither children nor siblings of each other.

If \( P+Q \) is located as the immediate ancestor to \( B \) and \( C \) (P's location) in the tree, the combined class cannot inherit methods from \( X \), however if \( P+Q \) is located as an immediate child of \( X \) (Q's location), the combined class can still inherit methods from all the ancestors of \( P \) and \( Q \). Therefore, \( P+Q \) will be in
located Q's location\textsuperscript{17}. In this case, \(\mu(P) = x, \mu(Q) = y\) and \(y > x\). \(\mu(P+Q) = y\), i.e., \(\mu(P+Q) > \mu(P)\) and \(\mu(P+Q) = \mu(Q)\) and Property 4 is satisfied.

iii) when one is a child of the other\textsuperscript{18}:

\[
\begin{align*}
&\text{In this case, } \mu(P) = n, \mu(Q) = n + 1, \text{ but } \mu(P+Q) = n, \text{ i.e. } \mu(P+Q) < \mu(Q). \\
&\text{Property 4 is not satisfied.}
\end{align*}
\]

Let P and Q' be siblings, i.e. \(\mu(P) = \mu(Q') = n\), and let R be a child of P. Then \(\mu(P+R) = n\) and \(\mu(Q'+R) = n + 1\), i.e., \(\mu(P+R)\) is not equal to \(\mu(Q'+R)\).

Therefore, Property 5 is satisfied. For any two classes P and Q, \(\mu(P+Q) = \mu(P)\) or \(= \mu(Q)\). Therefore, \(\mu(P+Q) \leq \mu(P) + \mu(Q)\), i.e., Property 6 is not satisfied.

\textsuperscript{17}If there are several intermediate classes between P and the common ancestor of P and Q, the combined class will still be located as an immediate child of X and also inherit (via multiple inheritance) from P's immediate ancestors.

\textsuperscript{18}This case is also representative of the situation where Q is a descendent, but not an immediate child of P.
Metric 3: Number of children (NOC)

Definition

NOC = number of immediate sub-classes subordinated to a class in the class hierarchy.

Theoretical basis.

NOC relates to the notion of scope of properties. It is a measure of how many sub-classes are going to inherit the methods of the parent class.

Viewpoints.

• Greater the number of children, greater the reuse, since inheritance is a form of reuse.

• Greater the number of children, the greater the likelihood of improper abstraction of the parent class. If a class has a large number of children, it may be a case of misuse of sub-classing.

• The number of children gives an idea of the potential influence a class has on the design. If a class has a large number of children, it may require more testing of the methods in that class.

Analytical Evaluation of Number Of Children (NOC)

Let P and R be leaves, \( \mu(P) = \mu(R) = 0 \), let Q be the root \( \mu(Q) > 0 \). \( \mu(P) \neq \mu(Q) \) therefore property 1 is satisfied. Since \( \mu(R) = \mu(P) \), Property 2 is also satisfied. Design of a class involves decisions on the scope of the methods declared within the class, i.e., the sub-classing for the class. The number of sub-classes
is therefore dependent upon the design implementation of the class. 

Therefore, Property 3 is satisfied.

Let P and Q be two classes with np and nQ sub-classes respectively (i.e., μ(P) = np and μ(Q) = nQ). Combining P and Q\textsuperscript{19}, will yield a single class with np + nQ - δ sub-classes, where δ is the number of children P and Q have in common. Clearly, δ is 0 if either np or nQ is 0. If Q is a sub-class of P, then P+Q will have np + nQ - 1 sub-classes. Therefore, in general the number of sub-classes of P+Q is np + nQ - β, where β = 1 or δ. Now, np + nQ - β ≥ np and np + nQ - β ≥ nQ. This can be written as: μ(P+Q) ≥ μ(P) and μ(P+Q) ≥ μ(Q) for all P and all Q. Therefore, Property 4 is satisfied\textsuperscript{20}. Let P and Q each have n children and R be a child of P which has r children. μ(P) = n = μ(Q). The class obtained by combining P and R will have (n-1) + r children, whereas a class obtained by combining Q and R will have n + r children, which means that μ(P+R) ≠ μ(Q+R). Therefore Property 5 is satisfied. Given any two classes P and Q with np and nQ children respectively, the following relationship holds:

μ(P) = np and μ(Q) = nQ

μ(P+Q) = np + nQ - δ

where δ is the number of common children. Therefore, μ(P+Q) ≤ μ(P) + μ(Q) for all P and Q. Property 6 is not satisfied.

\textsuperscript{19} The combination of two classes will result in the combined class located in the inheritance hierarchy at the position of the class with the greater depth of inheritance.

\textsuperscript{20} In cases where a class is both a parent and a grandparent of another class, this property will be violated. However, most OO environments will disallow this type of hierarchy.
Metric 4: Coupling between object classes (CBO)

Definition

CBO for a class is a count of the number of other classes to which it is coupled.

Theoretical basis.

CBO relates to the notion that an object is coupled to another object if one of them acts on the other, i.e., methods of one use methods or instance variables of another. As stated earlier, since objects of the same class have the same properties, two classes are coupled when methods declared in one class use methods or instance variables defined by the other class.

Viewpoints.

• Excessive coupling between object classes is detrimental to modular design and prevents reuse. The more independent a class is, the easier it is to reuse it in another application.

• In order to improve modularity and promote encapsulation, inter-object class couples should be kept to a minimum. The larger the number of couples, the higher the sensitivity to changes in other parts of the design, and therefore maintenance is more difficult.

• A measure of coupling is useful to determine how complex the testing of various parts of a design are likely to be. The higher the inter-object class coupling, the more rigorous the testing needs to be.
Analytical Evaluation of Coupling Between Objects (CBO)

As per assumption 1, there exist classes P, Q and R such that \( \mu(P) \neq \mu(Q) \) and \( \mu(P) = \mu(R) \) thereby satisfying properties 1 and 2. Inter-class coupling occurs when methods of one class use methods or instance variables of another class, i.e., coupling depends on the manner in which methods are designed and not on the functionality provided by P. Therefore Property 3 is satisfied. Let P and Q be any two classes with \( \mu(P) = n_P \) and \( \mu(Q) = n_Q \). If P and Q are combined, the resulting class will have \( n_P + n_Q - \partial \) couples, where \( \partial \) is the number of couples reduced due to the combination. That is \( \mu(P+Q) = n_P + n_Q - \partial \), where \( \partial \) is some function of the methods of P and Q. Clearly, \( n_P - \partial \geq 0 \) and \( n_Q - \partial \geq 0 \) since the reduction in couples cannot be greater than the original number of couples. Therefore,

\[
\begin{align*}
n_P + n_Q - \partial &\geq n_P \\
n_P + n_Q - \partial &\geq n_Q
\end{align*}
\]

i.e., \( \mu(P+Q) \geq \mu(P) \) and \( \mu(P+Q) \geq \mu(Q) \) for all P and Q. Thus, Property 4 is satisfied. Let P and Q be two classes such that \( \mu(P) = \mu(Q) = n \), and let R be another class with \( \mu(R) = r \).

\[
\begin{align*}
\mu(P+Q) &= n + r - \partial, \text{ similarly} \\
\mu(Q+R) &= n + r - \beta
\end{align*}
\]

Given that \( \partial \) and \( \beta \) are independent functions, they will not be equal, i.e., \( \mu(P+R) \) is not equal to \( \mu(Q+R) \), satisfying Property 5. For any two classes P and Q, \( \mu(P+Q) = n_P + n_Q - \partial \).

\[
\mu(P+Q) = \mu(P) + \mu(Q) - \partial \text{ which implies that} \\
\mu(P+Q) \leq \mu(P) + \mu(Q) \text{ for all P and Q.}
\]

Therefore Property 6 is not satisfied.
Metric 5: Response For a Class (RFC)

Definition

RFC = | RS | where RS is the response set for the class.

Theoretical basis

The response set for the class can be expressed as:

\[ RS = \{ M \} \cup \text{ all } i \{ R_i \} \]

where \( \{ R_i \} \) = set of methods called by method \( i \) and \( \{ M \} \) = set of all methods in the class

The response set of a class is a set of methods that can potentially be executed in response to a message received by an object of that class\(^{21}\). The cardinality of this set is a measure of the attributes of objects in the class. Since it specifically includes methods called from outside the class, it is also a measure of the potential communication between the class and other classes.

Viewpoints.

- If a large number of methods can be invoked in response to a message, the testing and debugging of the class becomes more complicated since it requires a greater level of understanding required on the part of the tester.

- The larger the number of methods that can be invoked from a class, the greater the complexity of the class.

\(^{21}\) It should be noted that membership to the response set is defined only up to the first level of nesting of method calls due to the practical considerations involved in collection of the metric.
• A worst case value for possible responses will assist in appropriate allocation of testing time.

*Analytical Evaluation of Response for a Class (RFC)*

Let $X_P = RFC$ for class $P$

$X_Q = RFC$ for class $Q$.

$X_P$ and $X_Q$ are functions of the number of methods and the external coupling of $P$ and $Q$ respectively. It follows from assumption 1 (since functions of i.i.d. random variables are also i.i.d.) that $X_P$ and $X_Q$ are i.i.d. Therefore, there is a non-zero probability that $\exists$ a $Q$ such that $\mu(P) \neq \mu(Q)$ resulting in property 1 being satisfied. Also there is a non-zero probability that $\exists$ a $Q$ such that $\mu(P) = \mu(Q)$, therefore property 2 is satisfied. Since the choice of methods is a design decision, Property 3 is satisfied. Let $P$ and $Q$ be two classes with RFC of $P = n_P$ and RFC of $Q = n_Q$. If these two classes are combined to form one class, the response for that class will depend on whether $P$ and $Q$ have any common methods. Clearly, there are three possible cases: 1) when $P$ and $Q$ have no common methods nor do their methods use any of the same methods, and therefore the combined class $P + Q$ will have a response set = $n_P + n_Q$. 2) when $P$ and $Q$ have methods in common, and the response set will smaller than $n_P + n_Q$. 3) when $P$ and $Q$ have no methods in common but some of methods used by methods of $P$ and $Q$ are the same, the response set will be smaller than $n_P + n_Q$. For both cases 2 and 3, $\mu(P + Q) = n_P + n_Q - \delta$, where $\delta$ is some function of the methods of $P$ and $Q$. Clearly, $n_P + n_Q - \delta \geq n_P$ and $n_P + n_Q - \delta \geq n_Q$ for all possible $P$ and $Q$. $\mu(P+Q) \geq \mu(P)$ and $\geq \mu(Q)$ for all $P$ and $Q$. Therefore, Property 4 is satisfied.
Let P and Q be two classes such that \( \mu(P) = \mu(Q) = n \), and let R be another class with \( \mu(R) = r \).

\[
\begin{align*}
\mu(P+Q) &= n + r - \partial, \text{ similarly} \\
\mu(Q+R) &= n + r - \beta
\end{align*}
\]

Given that \( \partial \) and \( \beta \) are independent functions, they will not necessarily be equal, i.e., \( \mu(P+R) \) is not necessarily equal to \( \mu(Q+R) \), satisfying Property 5. For any two classes P and Q,

\[
\mu(P+Q) = \mu(P) + \mu(Q) - \partial \text{ which implies that} \\
\mu(P+Q) \leq \mu(P) + \mu(Q) \text{ for all P and Q.}
\]

Therefore Property 6 is not satisfied.

**Metric 6: Lack of Cohesion in Methods (LCOM)**

**Definition**

Consider a Class \( C_1 \) with \( n \) methods \( M_1, M_2, ..., M_n \). Let \( \{I_i\} = \text{set of instance}\) variables used by method \( M_i \). There are \( n \) such sets \( \{I_1\}, ..., \{I_n\} \). Let \( P = \{(I_i, I_j) \mid I_i \cap I_j = \emptyset \} \) and \( Q = \{(I_i, I_j) \mid I_i \cap I_j \neq \emptyset \} \). If all \( n \) sets \( \{I_1\}, ..., \{I_n\} \) are \( \emptyset \) then let \( P = \emptyset \).

\[
\text{LCOM} = \begin{cases} 
|P| - |Q|, & \text{if } |P| > |Q| \\
0 & \text{otherwise}^{22}
\end{cases}
\]

Example: Consider a class C with three methods \( M_1, M_2 \) and \( M_3 \). Let \( \{I_1\} = \{a,b,c,d,e\} \) and \( \{I_2\} = \{a,b,e\} \) and \( \{I_3\} = \{x,y,z\} \). \( \{I_1\} \cap \{I_2\} \) is non-empty, but \( \{I_1\} \cap \{I_3\} = \emptyset \).

---

22 Note that the LCOM metric for a class where \( |P| = |Q| \) will be zero. This does not imply maximal cohesiveness, since within the set of classes with LCOM = 0, some may be more cohesive than others.
\{I3\} and \{I2\} \cap \{I3\} are null sets. LCOM is the (number of null intersections - number of non-empty intersections), which in this case is 1.

*Theoretical basis.*

This uses the notion of degree of similarity of methods. The degree of similarity for two methods $M_1$ and $M_2$ in class $C_1$ is given by:

$$\sigma() = |I_1| \cap |I_2|$$

where $|I_1|$ and $|I_2|$ are the sets of instance variables used by $M_1$ and $M_2$.

The LCOM is a count of the number of method pairs whose similarity is 0 (i.e. $\sigma()$ is a null set) minus the count of method pairs whose similarity is not zero. The larger the number of similar methods, the more cohesive the class, which is consistent with traditional notions of cohesion that measure the inter-relatedness between portions of a program. If none of the methods of a class display any instance behavior, i.e. do not use any instance variables, they have no similarity and the LCOM value for the class will be zero. The LCOM value provides a measure of the relative disparate nature of methods in the class. A smaller number of disjoint pairs (elements of set P) implies greater similarity of methods. LCOM is intimately tied to the instance variables and methods of a class, and therefore is a measure of the attributes of an object class.

*Viewpoints.*

- Cohesiveness of methods within a class is desirable, since it promotes encapsulation.

- Lack of cohesion implies classes should probably be split into two or more sub-classes.

- Any measure of disparities of methods helps identify flaws in the design of classes.
• Low cohesion increases complexity, thereby increasing the likelihood of errors during the development process.

**Analytical Evaluation of Lack Of Cohesion Of Methods (LCOM)**

Let $X_P = \text{LCOM}$ for class $P$

$X_Q = \text{LCOM}$ for class $Q$.

$X_P$ and $X_Q$ are functions of the number of methods and the instance variables of $P$ and $Q$ respectively. It follows from assumption 1 (since functions of i.i.d. random variables are also i.i.d.) that $X_P$ and $X_Q$ are i.i.d. Therefore, there is a non-zero probability that $\exists$ a $Q$ such that $\mu(P) \neq \mu(Q)$ resulting in property 1 being satisfied. Also there is a non-zero probability that $\exists$ a $Q$ such that $\mu(P) = \mu(Q)$, therefore property 2 is satisfied. Since the choice of methods and instance variables is a design decision, Property 3 is satisfied.

Suppose class $P$ has 3 methods $M_1$, $M_2$, $M_3$, and $M_2$ and $M_3$ use common instance variables, while $M_1$ has no common instance variables with $M_2$ and $M_3$. The LCOM for $P$ will be 1. Now, let another class $Q$ have 3 methods, all of which use common instance variables. The LCOM for $Q$ will be 0. When $P$ and $Q$ are combined, if the instance variables of $Q$ are the same as the variables used by $M_2$ and $M_3$, the LCOM for $P+Q$ will become 0, since the number of non-empty intersections will exceed the number of empty intersections. This implies that $\mu(P+Q) > \mu(Q)$, which violates Property 4. Therefore, LCOM does not satisfy Property 4\textsuperscript{23}.

Let $P$ and $Q$ be two classes such that $\mu(P) = \mu(Q) = n$, and let $R$ be another class with $\mu(R) = r$.

\textsuperscript{23} The author is indebted to the associate editor of the IEEE Transactions on Software Engineering for providing this example.
\[ \mu(P+Q) = n + r - \delta, \text{ similarly} \]
\[ \mu(Q+R) = n + r - \beta \]

Given that \( \delta \) and \( \beta \) are independent functions, they will not necessarily be equal. i.e., \( \mu(P+R) \neq \mu(Q+R) \), satisfying Property 5. For any two classes \( P \) and \( Q \), \( \mu(P+Q) = n_P + n_Q - \delta \). i.e.,

\[ \mu(P+Q) = \mu(P) + \mu(Q) - \delta \text{ which implies that} \]
\[ \mu(P+Q) \leq \mu(P) + \mu(Q) \text{ for all } P \text{ and } Q. \]

Therefore Property 6 is not satisfied\(^{24}\).

\(^{24}\) It can be shown for some cases that the number of disjoint sets will increase (i.e. \( \delta \) will be negative) when two classes are combined. Under these circumstances, property 6 will be satisfied.
5.0 SUMMARY

5.1 The Metrics Suite

The six metrics are designed to measure the three non-implementation steps in Booch's definition of OOD. Each metric is one among several that can be defined using Bunge's ontological principles. For example, the cardinality of the set of properties of an object (which will include both methods and instance variables) could be defined as a metric. But inclusion in the proposed suite is influenced by three additional criteria, as stated in the goals of the research: 1) ability to meet analytical properties 2) intuitive appeal to practitioners and managers in organizations and 3) ease of automated collection. The definition and meaning of the proposed metrics suite is summarized below.

<table>
<thead>
<tr>
<th>Metric</th>
<th>Definition/meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>Weighted Methods per Class (WMC)</td>
<td>This relates to the definition of complexity of an object. The number of methods and the static complexity of methods involved is predictor of the time and effort required to develop and maintain a class.</td>
</tr>
<tr>
<td>Depth of Inheritance (DIT)</td>
<td>A measure of how many ancestor classes can potentially affect a class. It is useful to have a measure of how deep a particular class is in the hierarchy so that the class can be designed to reuse inherited methods.</td>
</tr>
<tr>
<td>Number of Children (NOC)</td>
<td>A measure of how many sub classes are going to inherit the methods of a parent class. NOC gives an idea of the potential influence a class has on the design. If a class has a large number of children, it may require more testing of the methods in a class.</td>
</tr>
<tr>
<td>Coupling between Object Classes (CBO)</td>
<td>A count of the number of couplings with other classes. Any incidence of a method calling or accessing methods and instance variables of another class is a couple.</td>
</tr>
<tr>
<td>Response for a Class (RFC)</td>
<td>This is a cardinality of the set of all the methods (up to one level of nesting) of all possible methods that can execute in response to the arrival of a message to an object of this class.</td>
</tr>
<tr>
<td>Lack of Cohesion of Methods (LCOM)</td>
<td>This is a count of the number of disjoint method pairs minus the number of similar method pairs in a class. Two methods are disjoint if they have no common instance variables.</td>
</tr>
</tbody>
</table>

Table 2.1: The Proposed Metrics Suite
5.2 Summary of Analytical Results

All the metrics satisfy the majority of the properties prescribed by Weyuker, with one strong exception, Property 6 (interaction increases complexity). Property 6 is not met by any of the metrics in this suite. Weyuker's rationale for Property 6 is to allow for the possibility of increased complexity due to interaction. Failing to meet Property 6 implies that a complexity metric could increase, rather than reduce, if a class is divided into more classes. Interestingly, the experienced OO designers who participated in this study found that memory management and run-time detection of errors are both more difficult when there are a large number of classes to deal with. In other words, their viewpoint was that complexity can increase when classes are divided into more classes. Therefore, satisfying Property 6 may not be an essential feature for OO software design complexity metrics. From a measurement theoretic standpoint, a metric that meets property 6, cannot be an interval or a ratio scale. This means that such a metric cannot be used to make judgments like "class A is twice as complex as class B", which limits its appeal. Thus, not satisfying property 6 is beneficial, rather than detrimental to widespread usage of the metrics.

The only other violation of Weyuker's properties is in the case of the DIT and LCOM metrics. The DIT metric fails to satisfy Property 4 (monotonicity) only in cases where two classes are in a parent-descendent relationship. This is because the distance from the root of a parent cannot become greater than one of its descendants. In all other cases, the DIT metric satisfies Property 4\(^{25}\).

\(^{25}\)It is interesting to note that other authors have also observed difficulties in applying this particular property of Weyuker's. For example, see (Harrison 1988).
Also, under certain conditions of class combination, the LCOM metric can fail to satisfy this property as well. A summary of the analytical results is presented in the table below.

<table>
<thead>
<tr>
<th>Metric</th>
<th>P1</th>
<th>P2</th>
<th>P3</th>
<th>P4</th>
<th>P5</th>
<th>P6</th>
</tr>
</thead>
<tbody>
<tr>
<td>WMC</td>
<td>Met</td>
<td>Met</td>
<td>Met</td>
<td>Met</td>
<td>Met</td>
<td>Not met</td>
</tr>
<tr>
<td>DIT</td>
<td>Met</td>
<td>Met</td>
<td>Met</td>
<td>*</td>
<td>Met</td>
<td>Not met</td>
</tr>
<tr>
<td>NOC</td>
<td>Met</td>
<td>Met</td>
<td>Met</td>
<td>Met</td>
<td>Met</td>
<td>Not met</td>
</tr>
<tr>
<td>CBO</td>
<td>Met</td>
<td>Met</td>
<td>Met</td>
<td>Met</td>
<td>Met</td>
<td>Not met</td>
</tr>
<tr>
<td>RFC</td>
<td>Met</td>
<td>Met</td>
<td>Met</td>
<td>Met</td>
<td>Met</td>
<td>Not met</td>
</tr>
<tr>
<td>LCOM</td>
<td>Met</td>
<td>Met</td>
<td>Met</td>
<td>*</td>
<td>Met</td>
<td>Not met</td>
</tr>
</tbody>
</table>

P1= Non-coarseness  
P2=Non-uniqueness  
P3=Design dependence  
P4=Monotonicity  
P5=Non-equivalence of interaction  
P6=interaction increases complexity  

*=Met except for special cases

Table 2.2: Summary of Analytical Properties
5.3 Booch OOD Steps

The proposed metrics cover all aspects of OOD as outlined by Booch in the table below:

<table>
<thead>
<tr>
<th>Metric</th>
<th>Identification</th>
<th>Semantics</th>
<th>Relationships</th>
</tr>
</thead>
<tbody>
<tr>
<td>WMC</td>
<td>✓</td>
<td></td>
<td>✓</td>
</tr>
<tr>
<td>DIT</td>
<td>✓</td>
<td></td>
<td></td>
</tr>
<tr>
<td>NOC</td>
<td>✓</td>
<td></td>
<td></td>
</tr>
<tr>
<td>RFC</td>
<td></td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>CBO</td>
<td></td>
<td></td>
<td>✓</td>
</tr>
<tr>
<td>LCOM</td>
<td></td>
<td>✓</td>
<td></td>
</tr>
</tbody>
</table>

Table 2.3: Mapping of Metrics to Booch OOD Steps

WMC, DIT and NOC relate to the first step (definition of classes) in OOD since WMC is an aspect of the complexity of the class and both DIT and NOC directly relate to the layout of the class hierarchy. WMC and RFC capture how objects of a class may "behave" when they get messages. For example, if a class has a large WMC or RFC, it has many possible responses (since a potentially large number of methods can execute). The LCOM metric relates to the packaging of data and methods within a class definition provides a measure of the cohesiveness of a class. Thus WMC, RFC and LCOM relate to the second step (the semantics of classes) in OOD. A benefit of having a suite of metrics is that there is the potential for multiple measures of the same underlying construct\(^\text{26}\). The RFC and CBO metrics also capture the extent of communication between classes by counting the inter-class couples and

\(^{26}\) Another outcome of multiple measures is the statistical correlation between some metrics. For example, the RFC and WMC metric were highly correlated (Spearman rank correlation of 0.9) at both sites, while the NOC and LCOM had low correlation (less than 0.1). The median value of inter-metric correlations was 0.22 at Site A and 0.16 at Site B.


Chapter Three: Tool Development and Empirical Data
# Chapter Outline

1.0 Tool Development ........................................................................................................... 86
   1.1 Tool Architecture ........................................................................................................ 86
   1.2 The Message Server ................................................................................................... 87
   1.3 The Database Server ................................................................................................... 88
   1.4 The Metrics Extraction Program ................................................................................. 88

2.0 Empirical Data Collection ............................................................................................ 89

3.0 The WMC Metric Data ................................................................................................. 90
   3.1 Interpretation of Data .................................................................................................. 90

4.0 The DIT Metric Data ..................................................................................................... 91
   4.1 Interpretation of Data .................................................................................................. 92

5.0 The NOC Metric Data ................................................................................................... 93
   5.1 Interpretation of Data .................................................................................................. 94

6.0 The CBO Metric Data ................................................................................................... 95
   6.1 Interpretation of Data .................................................................................................. 96

7.0 The RFC Metric Data ................................................................................................... 97
   7.1 Interpretation of Data .................................................................................................. 98

8.0 The LCOM Metric Data ............................................................................................... 99
   8.1 Interpretation of Data .................................................................................................. 99

9.0 Summary of Managerial Results .................................................................................. 100

Bibliography ......................................................................................................................... 102
1.0 TOOL DEVELOPMENT

The importance of automated metrics measurement cannot be over emphasized. Lack of tools for metrics collection is one of the road blocks to widespread metrics use in organizations. Fenton recommends that automated data collection is essential for managing successful metrics programs in organizations (Fenton 1991). Since the focus of this research is to provide managers with useful information, an important goal in this research is to demonstrate that the proposed metrics can be collected automatically from large scale commercial systems.

1.1 Tool Architecture

The tool that was developed as part of this dissertation research is capable of capturing metrics from commercial C++ applications. The architecture supports collection over a network of UNIX workstations in a customized environment. A simpler version of the tool works in conjunction with a commercially available toolset, and can be deployed (with modifications) in any UNIX and C++ environment.

The figure below shows the architecture of the tool. There are three major elements in the architecture that enable the collection of metrics from a C++ library and deliver the results to the user. These are a) a message server that administers communications between different programs b) a database server that contains details of the library that is under analysis and 3) a metrics extractor program that computes the metrics and relays it to the user.
Figure 3.1: Architecture for automatic metrics extraction

The metrics extractor communicates to a database server using a custom message protocol (known as CLIPC)\(^1\). Any program on any UNIX machine in the same LAN can register with the message server (a program that arbitrates the messaging protocol), and send and receive messages (in a CLIPC format) from other programs that have registered with the message server. Currently the CLIPC protocol is proprietary to a specific company, but its principles of establishing communication between different programs is well understood, and can be applied in most UNIX environments.

1.2 The Message Server

A program registers with the message server by calling a specific CLIPC library function (for example see Appendix C). Once the program registers with the message server, it can send or receive messages from other programs that by calling functions that are declared under the CLIPC protocol. The message

\(^1\) For additional technical details see (Mottola 1992).
server and the CLIPC protocol provides a communication path across the entire LAN, enabling a program in one machine to communicate with another program dynamically on another machine. This capability would be analogous to a word processing program running on a personal computer being able to communicate during run time with a spreadsheet program that is executing on another personal computer in a different part of the building.

1.3 The Database Server

The database server is essentially a class browser that extracts class level information from code written in C++. The server preserves the class information in the form of a dictionary and passes this to any program that requests the information. Three steps are required prior to activation of the database server: 1) the C++ library is compiled 2) A UNIX utility (mikxref) is invoked to build a cross reference table and 3) Another UNIX utility (librwttool) is invoked to join different files together. Once the database server program is activated, it is ready to respond to any messages asking for class information about the library. Appendix C contains sample programs of the database server.

1.4 The Metrics Extraction Program

The metrics extraction program is initiated by the user (typically on a separate window on a machine). This program registers with the message server and asks the database server for a list of the classes in the library that is under analysis. It then queries the database server and obtains details on the member functions, instance variables and inheritance relationships for each of the classes in the library. The intermediate results of these queries are written to temporary files. During this time, it keeps the user updated on the
status of the queries to the database server to alert the user to any
communication errors. The intermediate files are then processed and results
are posted to both the screen and output files. Appendix C contains program
listings for the metrics program.

2.0 EMPirical DATA COLLECTION

The proposed metrics were then collected using automated tools at two
different organizations which will be referred to here as Site A and Site B.
Site A is a software vendor that uses OOD in their development work and has
a collection of different C++ class libraries. Metrics data from 634 classes from
two C++ class libraries that are used in the design of graphical user interfaces
(GUI) were collected. Both these libraries were used in different product
applications for rapid prototyping and development of windows, icons and
mouse-based interfaces. Reuse across different applications was one of the
primary design objectives of these libraries. These were typically used at Site
A in conjunction with other C++ libraries and traditional C-language
programs in the development of software sold to UNIX workstation users.
Metrics at this data site were extracted using the architecture described in the
earlier section on tool development.

Site B is a semiconductor manufacturer and uses the Smalltalk programming
language for developing flexible machine control and manufacturing
systems. Metrics were collected on the class libraries used in the
implementation of a computer aided manufacturing system for the
production of VLSI circuits. Over 30 engineers worked on this application,
after extensive training and experience with object orientation and the
Smalltalk environment. Metrics data from 1459 classes from Site B were
collected using a Smalltalk implementation of the metrics extractor (developed by professional programmers at the data site).

3.0 THE WMC METRIC DATA

The histograms and summary statistics from both sites are shown below:

![Histograms for WMC metric](image)

**Figure 3.2: Histograms for the WMC metric**

<table>
<thead>
<tr>
<th>Site</th>
<th>Metric</th>
<th>Median</th>
<th>Max</th>
<th>Min</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>WMC</td>
<td>5</td>
<td>106</td>
<td>0</td>
</tr>
<tr>
<td>B</td>
<td>WMC</td>
<td>10</td>
<td>346</td>
<td>0</td>
</tr>
</tbody>
</table>

**Table 3.1: Summary Statistics for the WMC metric**

3.1 Interpretation of Data.

The most interesting aspect of the data is the similarity in the nature of the distribution of the metric values at Site A and B, despite differences in i) the nature of the application, ii) the people involved in their design and iii) the languages (C++ and Smalltalk) used. This seems to suggest that most classes
tend to have a small number of methods (0 to 10), while a few outliers declare a large number of them. Most classes in an application appear to be relatively simple in their construction, providing specific abstraction and functionality.

Examining the outlier classes at Site A revealed some interesting observations. The class with the maximum number of methods (106) had no children and was at the root of the hierarchy, whereas another outlier class with 87 methods had 14 sub-classes and a total number of 43 descendants. In the first case, the class’s methods have no reuse within the application and, unless this is a generalized class that is reused across applications, the effort expended in developing this class will be a one-shot investment. However, the class with 87 methods has significant reuse potential within the application making increased attention to testing the methods in this class worthwhile, since the methods can have widespread use within the system.

4.0 THE DIT METRIC DATA

The histograms and summary statistics are shown below (all metric values are integers):

![Histograms for the DIT metric](image)

**Figure 3.3: Histograms for the DIT metric**
Table 3.2: Summary Statistics for the DIT metric

<table>
<thead>
<tr>
<th>Site</th>
<th>Metric</th>
<th>Median</th>
<th>Max</th>
<th>Min</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>DIT</td>
<td>1</td>
<td>8</td>
<td>0</td>
</tr>
<tr>
<td>B</td>
<td>DIT</td>
<td>3</td>
<td>10</td>
<td>0</td>
</tr>
</tbody>
</table>

4.1 Interpretation of Data.

Both Site A and B libraries have a low median value for the DIT metric. This suggests that most classes in an application tend to be close to the root in the inheritance hierarchy. By observing the DIT metric for classes in an application, a senior designer or manager can determine whether the design is "top heavy" (too many classes near the root) or "bottom heavy" (many classes are near the bottom of the hierarchy). At both Site A and Site B, the library appears to be top heavy, suggesting that designers may not be taking advantage of reuse of methods through inheritance\(^2\). Note that the Smalltalk application has a higher depth of inheritance due, in part, to the library of reusable classes that are a part of the language. For example, all classes are sub-classes of the class "object". Another interesting aspect is that the maximum value of DIT is rather small (10 or less). One possible explanation is that designers tend to keep the number of levels of abstraction to a manageable number in order to facilitate comprehensibility of the overall architecture of the system. Designers may be forsaking reusability through inheritance for simplicity of understanding. This also illustrates one of the advantages of gathering metrics of design complexity in that a clearer picture

\(^2\text{Of course, such occurrences may also be a function of the application. It is interesting to note, however, that this phenomenon appears to be present in both data sets, which represent relatively different applications and implementation environments.}\)
of the conceptualization of software systems begins to emerge with special attention focused on design tradeoffs. Examining the class at Site A with a DIT value of 8 revealed that it was a case of increasingly specialized abstractions of a graphical concept of control panels. The class itself had only 4 methods and only local variables, but objects of this specialized class had a total 132 methods available through inheritance. Designing this class would have been a relatively simple task, but the testing could become more complicated due to the high inheritance\(^3\). Resources between design and testing could be adjusted accordingly to reflect this.

5.0 THE NOC METRIC DATA

The histograms and summary statistics from both sites are shown below:

![Histograms](image)

**Figure 3.4: Histograms for the NOC metric**

\(^3\)Testers have frequently experienced that finding which method is executing (and from where) is a time consuming task (Leijter *et al.* 1992).
<table>
<thead>
<tr>
<th>Site</th>
<th>Metric</th>
<th>Median</th>
<th>Max</th>
<th>Min</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>NOC</td>
<td>0</td>
<td>42</td>
<td>0</td>
</tr>
<tr>
<td>B</td>
<td>NOC</td>
<td>0</td>
<td>50</td>
<td>0</td>
</tr>
</tbody>
</table>

Table 3.3: Summary Statistics for the NOC metric

5.1 Interpretation of Data

Like the WMC metric, an interesting aspect of the NOC data is the similarity in the nature of the distribution of the metric values at Site A and B. This seems to suggest that classes in general have few immediate children and that only a very small number of outliers have many immediate sub-classes. This further suggests that designers may not be using inheritance of methods as a basis for designing classes, as the data from the histograms show that a majority of the classes (73% at Site A and 68% at Site B) have no children. Considering the large sample sizes at both sites and their remarkable similarity, both the DIT and NOC data seem to strongly suggest that reuse through inheritance may not be being fully adopted in the design of class libraries, at least at these two sites. One explanation for the small NOC count could be that the design practice followed at the two sites dictated the use of shallow inheritance hierarchies\(^4\). A different explanation could be a lack of communication between different class designers and therefore that reuse opportunities are not being realized. Whatever the reason, the metric values and their distribution provide designers and managers with an opportunity to examine whether their particular design philosophy is being adhered to in

\(^4\) Some C++ designers at this site systematically avoid sub-classing in order to maximize operational performance.
the application. An examination of the class with 42 sub-classes at Site A was a GUI-command class for which all possible commands were separate sub-classes. Further, none of these sub-classes had any sub-classes of their own. Systematic use of the NOC metric could have helped to restructure the class hierarchy to exploit common characteristic of different commands (e.g. text commands, mouse commands etc.).

6.0 THE CBO METRIC DATA

The histograms and summary statistics from both sites are shown below:

![Histograms for the CBO metric](image)

**Figure 3.5: Histograms for the CBO metric**

<table>
<thead>
<tr>
<th>Site</th>
<th>Metric</th>
<th>Median</th>
<th>Max</th>
<th>Min</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>CBO</td>
<td>0</td>
<td>84</td>
<td>0</td>
</tr>
<tr>
<td>B</td>
<td>CBO</td>
<td>9</td>
<td>234</td>
<td>0</td>
</tr>
</tbody>
</table>

**Table 3.4: Summary Statistics for the CBO metric**
6.1 Interpretation of Data

Both Site A and Site B class libraries have skewed distributions for CBO, but the Smalltalk application at Site B has relatively high median values. One possible explanation is that contingency factors (e.g., type of application) are responsible for the difference. A more likely reason is the difference between the Smalltalk and C++ languages. Smalltalk requires virtually every interaction between run-time entities be done through message passing, while C++ does not. In Smalltalk, simple scalar variables (integers, reals, and characters) and control flow constructs like if, while, repeat statements are objects. Each of these invocations is performed via message passing which will be counted as an interaction in the CBO metric. Simple scalars will not be defined as C++ classes, and certainly control flow entities are not objects in C++. Thus, CBO values are likely to be smaller in C++ applications.

One interpretation that may account for both the similarity in the shape of the distribution and the higher values for Site B is that coupling between classes is an increasing function of the number of classes in the application. The Site B application has 1459 classes compared to the 634 classes at Site A. It is possible that complexity due to increased coupling is a characteristic of large class libraries. This could be an argument for a more informed selection of the scale size (as measured by number of classes) in order to limit coupling. The low median values of coupling at both sites suggest that at least 50% of the classes are self-contained and do not refer to other classes (including super-classes). Since a fair number of classes at both sites have no parents or

---

5 The author is indebted to an anonymous referee of the IEEE Transactions on Software Engineering who provided the following explanation.
no children, the limited use of inheritance may be also responsible for the small CBO values.

Examination of the outliers at Site B revealed that classes responsible for managing interfaces have high CBO values. These classes tended to act as the connection point for two or more sub-systems within the same application. At Site A, the class with the highest CBO value was also the class with the highest NOC value, further suggesting the need to re-evaluate that portion of the design. The CBO metric can be used by senior designers and project managers as a relative simple way to track whether the class hierarchy is losing its integrity, and whether different parts of a large system are developing unnecessary interconnections in inappropriate places.

7.0 THE RFC METRIC DATA

The histograms and summary statistics from both sites are shown below:

![Histograms for the RFC metric](image)

**Figure 3.6: Histograms for the RFC metric**
<table>
<thead>
<tr>
<th>Site</th>
<th>Metric</th>
<th>Median</th>
<th>Max</th>
<th>Min</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>RFC</td>
<td>6</td>
<td>120</td>
<td>0</td>
</tr>
<tr>
<td>B</td>
<td>RFC</td>
<td>29</td>
<td>422</td>
<td>3</td>
</tr>
</tbody>
</table>

Table 3.5: Summary Statistics for the RFC metric

7.1 Interpretation of Data

The data from both Site A and Site B suggest that most classes tend to able to invoke a small number of methods, while a few outliers may be most profligate in their potential invocation of methods. This reinforces the argument that a small number of classes may be responsible for a large number of the methods that executed in an application, either because they contain many methods (this appears to be the case at Site A) or that they call many methods. By using high RFC valued classes as structural drivers, high test coverage can be achieved during system test.

Another interesting aspect is the difference in values for RFC between Site A and B. Note that the median and maximum values of RFC at Site B are higher than the RFC values at Site A. As in the case of the CBO metric, this may relate to the complete adherence to object oriented principles in Smalltalk which necessitates extensive method invocation, whereas C++'s incremental approach to object orientation gives designers alternatives to message passing through method invocation\(^6\). Not surprisingly, at Site B high RFC value classes performed interface functions within the application.

---

\(^6\) RFC does not count calls to X-library functions and I/O functions like printf, scanf that are present in C++ applications. Similar functionality is obtained through interface classes in Smalltalk that are counted in the RFC calculations.
Since there are a number of classes that are stand-alone (i.e. no parents, no children, no coupling) the RFC values also tend to be low. Again, the metrics collectively and individually provide managers and designers a basis for examining the design of class hierarchies.

8.0 The LCOM Metric Data

The histograms and summary statistics from both sites are shown below:

![Histograms for the LCOM metric](image)

**Figure 3.7: Histograms for the LCOM metric**

<table>
<thead>
<tr>
<th>Site</th>
<th>Metric</th>
<th>Median</th>
<th>Max</th>
<th>Min</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>LCOM</td>
<td>0</td>
<td>200</td>
<td>0</td>
</tr>
<tr>
<td>B</td>
<td>LCOM</td>
<td>2</td>
<td>17</td>
<td>0</td>
</tr>
</tbody>
</table>

**Table 3.6: Summary Statistics for the LCOM metric**

8.1 Interpretation of Data

At both sites, LCOM median values are extremely low, indicating that at least 50% of classes have cohesive methods. In other words, instance variables seem to be operated on by more than one method defined in the class. This is
consistent with the principle of building methods around the essential data elements that define a class. The Site A application has a few outlier classes that have low cohesion, as evidenced by the high maximum value 200. In comparison, the Site B application has almost no outliers, which is demonstrated by the difference in the shape of the two distributions.

A high LCOM value indicates disparateness in the functionality provided by the class. This metric can be used to identify classes that are attempting to achieve many different objectives, and consequently are likely to behave in less predictable ways than classes that have lower LCOM values. Such classes could be more error prone and more difficult to test and could possibly be disaggregated into two or more classes that are more well defined in their behavior. The LCOM metric can be used by senior designers and project managers as a relatively simple way to track whether the cohesion principle is adhered to in the design of an application and advise changes, if necessary, at an earlier phase in the design cycle.

9.0 SUMMARY OF MANAGERIAL RESULTS

The data from two different commercial projects and subsequent discussions with the designers at those sites lead to several interesting observations that may be useful to managers of OOD projects. Designers may tend to keep the inheritance hierarchies shallow, forsaking reusability through inheritance for simplicity of understanding. This potentially reduces the extent of method reuse within an application. However, even in shallow class hierarchies it is possible to extract reuse benefits, as evidenced by the class with 87 methods at Site A that had a total of 43 descendants. This suggests that managers need to
proactively manage reuse opportunities and that this metrics suite can aid this process.

Another demonstrable use of these metrics is in uncovering possible design flaws or violations of design philosophy. As the example of the command class with 42 children at Site A demonstrates, the metrics help to point out instances where sub-classing has been misused. This is borne out by the experience of the designers interviewed at one of the data sites where excessive declaration of subclasses was common among engineers new to the OO paradigm. These metrics can be used to allocate testing resources. As the example of the interface classes at Site B (with high CBO and RFC values) demonstrates, concentrating test efforts on these classes may have been a more efficient utilization of resources.

In addition to the proposal and analytic test of theoretically-grounded metrics, this dissertation has also presented empirical data on these metrics from actual commercial systems. The implementation independence of these metrics is demonstrated in part through data collection from both C++ and Smalltalk implementations, two of the most widely used object oriented environments. These data are used to demonstrate not only the feasibility of data collection, but also to suggest ways in which these metrics might be used by managers. In addition to the usual benefits obtained from valid measurements, OO design metrics should offer needed insights into whether developers are following OO principles in their designs. This use of metrics may be an especially critical one as organizations begin the process of migrating their staffs toward the adoption of OO principles.
Using several of the metrics together can help managers and senior designers, who may be unable to review design materials for the entire application, to exercise some measure of architectural control over the evolution of an OO application. They could be by means of the WMC, DIT and NOC metrics to check whether the application is getting "top heavy" (i.e. too many classes at the root level declaring many methods) or using the RFC and CBO metrics check whether there are interconnections between various parts of the application that are unwarranted. As OO systems evolve, maintenance of the architectural integrity of an application becomes an important managerial responsibility, and this metrics suite could be used as a tool to meet this challenge.

BIBLIOGRAPHY


Chapter Four: Metrics Validation
Chapter Outline

1.0 Introduction .............................................................................................................105
2.0 Data Collection .......................................................................................................106
  2.1 Background of Data Site .......................................................................................106
  2.2 Systems Description .............................................................................................107
    2.2.1 TPM (Trade Position Manager) .................................................................107
    2.2.2 SLB (Securities Lending & Borrowing) ......................................................108
    2.2.3 FIS (Fixed Income Sales) ............................................................................109
  2.3 Data Sources .........................................................................................................109
    2.3.1 Source Code Control Systems (SCCS) .......................................................109
    2.3.2 Project Management Documents ...............................................................110
    2.3.3 Programmer Reports ....................................................................................110
    2.3.4 In house expert ratings .................................................................................111
  2.4 Data Definitions .....................................................................................................112
3.0 Results .....................................................................................................................113
  3.1 Exploratory Productivity Models .........................................................................113
    3.1.4 Exploring Individual Differences ................................................................123
  3.2 Exploratory Models for Rework and Reusability .................................................127
  3.3 Exploratory model for early design effort .............................................................131
4.0 Summary and Limitations .......................................................................................136
Bibliography ..................................................................................................................140
1.0 INTRODUCTION

One of the primary shortcomings of metrics proposals is the lack of examination of their ability to predict variables of managerial interest such as productivity, reusability and effort. Fenton remarks that metrics that do not demonstrate this are likely to be ignored by managers and designers alike (Fenton 1991 p 88). Another frequent issue with metrics validation is the proclivity for most researchers to conduct laboratory experiments with students. This is been noted to be detrimental to the progress of software engineering management, since laboratory experiments do not reflect the practical considerations that are typically encountered in professional organizations (Traub et al. 1989). Recognizing the validity of this argument, this chapter presents an examination of the predictive properties of the six metrics that have been proposed in this dissertation in a commercial setting.

Software complexity has been mentioned by researchers to have crucial impact on the typical managerial concerns of cost, quality and productivity (Banker et al. 1993; Basili and Perricone 1984; Card and Agresti 1988; Kemerer 1987; Kriebel and Moore 1980; Munson and Khoshgoftar 1992; Shen et al. 1983; Stabell and Forsund 1983). Since the proposed metrics are measurements of design complexity, variations in the following managerial variables are expected to be a function of the metrics:

- Productivity
- Rework Effort
- Reusability
- Design Effort
As Banker et al. state such an analysis will provide evidence of the statistical significance of complexity metrics as well as give practitioners useful information on the extent to which each complexity metric affects the dependent variables (Banker et al. 1993). This can lead to more informed design and resource allocation decisions in organizations.

2.0 DATA COLLECTION

The data collection needed for metrics validation requires active collaboration from sites that have surpassed the Level 1 stage of software development in the SEI maturity model\(^1\). In addition to having a strong tradition of software process improvement through measurement, the data site must be willing to make design documents, project plans, source code available for inspection so that multiple sources of data are available to measure variables like effort, productivity and reusability. It was therefore deemed appropriate to study several projects over a period of time at a single data site rather than undertake a cross-sectional study of many organizations. Other researchers who have chosen to validate metrics on commercial software systems have adopted a similar strategy (Gill and Kemerer 1991; Li and Henry 1993).

2.1 Background of Data Site

The Information Systems Division (ISD) of a European bank (referred to in this dissertation as The International Bank Corporation or IBC) was the data site that participated in final stage of this research project to validate the proposed metrics. IBC is an international bank that offers a wide range of

---

\(^1\) The SEI Maturity model is described in (Humphrey 1988).
modern financial instruments to clients on a world wide basis. Technological leadership is central to their competitive position in the financial services industry. The ISD staff (of over 130 people) is charged with the responsibility of providing computer hardware and software solutions that provide access to up-to-the-minute information, analytical capabilities and financial engineering tools to traders and portfolio managers in the bank. These solutions help to formulate portfolio structure, assess market risk factors, advise market makers, ensure liquidity of financial positions and competitive pricing of financial instruments. The strategic and financial implications of the transactions that are handled by these systems are significant, and therefore the software systems are planned, designed and tested with considerable care. The organization has had a tradition of collecting detailed data during various phases of the systems development process to facilitate management control of IS projects.

2.2 Systems Description

Three software systems developed by the ISD at IBC were analyzed as part of the study. These systems are used by financial traders to assist them in the buying, selling, recording and analysis of various financial instruments like stocks (or equities), bonds, options, derivatives and foreign exchange positions. All three projects were begun and completed in the period 1991 to 1994 and are used extensively by IBC traders in the US, Europe and Asia.

2.2.1 TPM (Trade Position Manager)

TPM is a key software system in the IS infrastructure at IBC. It is responsible for "straight through" trading, i.e. it enables typical computers from the trading desk to communicate to back-office mainframe computers that
maintain and consolidate the massive amount of data relating to the bank’s current and historical trading in financial instruments. TPM manages the physical and logical distribution of data among different computer systems, while providing users with a consistent single database for product risk assessment and position keeping. TPM was developed on a SUN SparcStation in C++ using the Booch OO design methodology. Phase I of the system comprised of several key subsystems that were designed by a team of 5 designers over a period of 8 months and released to the users in 1993. The system is currently being updated for increased functionality and connectivity by the same team. The classes that were analyzed as part of this study were drawn from Phase I of TPM.

**2.2.2 SLB (Securities Lending & Borrowing)**

The SLB system is an object oriented software system built to run on a NeXT workstation. The business mission of SLB is to help maximize the returns on trading positions by effectively coordinating the borrowing and lending of equity and equity linked instruments. The system assists securities traders in an investment bank to lend to and borrow from external counterparties (other financial institutions) to hedge their trading positions. The SLB system provides functionality to enter stock borrowings and loans into a database, checking market price of stocks, tracking pending loans and payments, and generating accounting and financial reports. The system provides GUI interfaces, graphing and analysis tools and the capability to communicate with other computer systems to access and consolidate data. The language used for developing SLB was Objective-C and a total of three people were assigned to work on a full-time on the design, testing and deployment of the system. The metrics data from SLB were extracted from textual design
artifacts prior to coding. These data reflect analysis and early design conditions of the system.

2.2.3 FIS (Fixed Income Sales)

FIS is a trading decision support system. It provides pricing, analytics, risk management and management information to the Fixed Income Trading & Sales teams in the investment bank. In addition to these functions, FIS automates the manual process of issuing trading tickets\(^2\), and passes the consolidated information to other mainframe computer programs for initiating electronic transfer of funds, archival and storage.

FIS is a large project that involved 8 designers and there are a number of different versions of the system that have been developed over the period 1991-1993. It was developed using Objective C and works primarily on NeXT workstations. Certain classes developed for FIS were considered to be reusable in other systems, and the data collected during this project is restricted to the classes that were reused in the SLB project.

2.3 Data Sources

2.3.1 Source Code Control Systems (SCCS)

SCCS is a source management system provided under the UNIX operating system which maintains records of changes made in files within a project. Records stating what the changes were, why and when they were made, and who made them are kept for each version. Previous versions can be

---

\(^2\) Traditionally, each time a trader completes a trade, the details of the commodity (stock, bond) and the transaction are recorded on a slip of paper known as a "ticket."
recovered, and different versions can be maintained simultaneously. Since all three systems studied at IBC employed the SCCS utility, detailed records of time, version, size and programmer usage were made available for this research\(^3\).

**2.3.2 Project Management Documents**

IBC has formal mechanisms for project initiation, approval and management. There are steering committees that oversee individual projects and several documents are released during different stages of a software products' life cycle. Two such documents are the requirements document and the systems analysis and design report. Both these documents contain information on the cost (in terms of time), early design descriptions of key classes that are used in the system. In addition the project manager in charge of each project releases detailed project plans that describe current design status, staffing levels and costs. These documents were made available for the research study to aid data collection.

**2.3.3 Programmer Reports**

In addition to the formal records maintained in SCCS and in project management documents, individual programmer activity reports were another source of data for this study. This helped in discounting personal holidays and time taken to attend tc activities that are not directly related to the project. In the case of all three SLB and FIS projects key designers

---

\(^3\) The reader is referred to UNIX reference manual (or the man SCCS UNIX command) for more details on SCCS.
reviewed their personal calendars to provide accurate estimates for their activities.

2.3.4 In house expert ratings

An important dependent variable of interest in this study is reusability of classes, since one of the proposed benefits of the OO approach is to reuse these key abstractions over several projects. At IBC, reuse is promoted and advocated by a reuse committee formed by members of various project teams under the stewardship of an infrastructure group within ISD. Two experts on OO design from this group rated the reusability of the FIS classes that were reworked and reused in the SLB project. The guidelines for assessing reuse were derived from recommendations made in the extant literature on reusable classes (Booch 1991; Booch 1993; Lamping 1993; Wasserman et al. 1989). The experts rated each class based on:

- Class relationships (to other classes in the system)
- The consistency of the class interface
- The ease of addition to the class
- The level of documentation available for the class

The following procedures were strictly followed to ensure that the reuse ratings of the two experts would yield a reliable measure of class reusability:

- Both experts did not know the designers of the FIS and SLB classes.
- Both experts had sufficient knowledge of the type of application being developed.
- Each class was rated by separately by both experts.

---

4 The author is extremely grateful to Professor John Carroll of the Behavioral and Policy Sciences section of the Sloan School of Management for designing these procedures.
• The experts went through a short training session together to facilitate the interpretation of the guidelines.

• The rating was on a 9 point scale: 5 labeled points with one intermediate point in between a pair of labeled points.

1 2 3 4 5 6 7 8 9
(1=Not reusable; 3 Reusable with considerable rework; 5 Reusable with moderate rework; 7 Reusable with minimal rework; 9 Reusable with no rework)

• Egregious dissimilarities in ratings by the two experts were revisited and re-rated.

2.4 Data Definitions

In addition to the six metrics WMC, DIT, NOC, RFC, CBO and LCOM the following variables were used in the analysis of the three systems in this study.

**SIZEL:** Size of class in lines of code

**PROD:** Productivity measured as SIZEL/work days, where work days are the number of calendar days (excluding weekends and holidays) spent on design and unit testing the code before returning it to the central repository.

**VERSN:** Version number of the system

**BA_SIZE:** Size of the batch of classes designed in the same time period by the same designer. For example, the BA_SIZE for a class would be 10, if 10 other classes were designed by the same individual in the same calendar time.

**CODER_2, CODER_3, CODER_4, CODER_5:** Dummy variables for the specific designers.
EFFORT: Amount of time taken to rework the class for use in SLB measured in hours

REUSE1: Expert 1's rating of reusability of the class on a scale of 1 to 9 (maximum reusability)

REUSE2: Expert 2's rating

REUSABA: Average of the reusability ratings of two experts. The variable REUSABA is calculated as $= 0.5 \times \text{REUSE1} + 0.5 \times \text{REUSE2}$.

3.0 RESULTS

The previous section described the variables that were measured from the three systems at IBC. Productivity, rework effort, design effort and reusability are the key dependent variables that are examined. In this section, three categories of regression models using these variables are presented, along with an interpretation of the various models.

3.1 Exploratory Productivity Models

Since software requires little in terms of tooling, capital, inventory, floor space (factors that are germane to the production of physical goods), the only resource is the time spent by the people designing, testing and maintaining the product. Consequently output has traditionally been measured by Lines of Code (LOC) or some modification like object lines of code (Wolverton 1974), (Jones 1978) and productivity is defined by the following ratio:

---

5 REUSE1 and REUSE2 have a first order correlation of 0.81 significant at the 0.001 level indicating high inter-expert reliability.
Productivity = (Lines of Code) / (Time taken to produce it)

Since reliable data on productivity was collected on TPM class design and maintenance, this is examined with the TPM data. Three different models of productivity were examined 1) design and maintenance\(^6\) by all programmers, 2) design of new classes by all programmers and 3) design and maintenance of the most prolific programmer in the TPM design team.

3.1.1 Design and Maintenance Productivity

The summary statistics for this data set is presented in the table below.

<table>
<thead>
<tr>
<th>Variable</th>
<th>Mean</th>
<th>Max</th>
<th>Min</th>
<th>Standard Deviation</th>
</tr>
</thead>
<tbody>
<tr>
<td>WMC</td>
<td>12.42</td>
<td>63</td>
<td>0(^7)</td>
<td>14.40</td>
</tr>
<tr>
<td>DIT</td>
<td>0.05</td>
<td>2</td>
<td>0</td>
<td>0.27</td>
</tr>
<tr>
<td>NOC</td>
<td>0.22</td>
<td>2</td>
<td>0</td>
<td>0.53</td>
</tr>
<tr>
<td>RFC</td>
<td>17.4</td>
<td>102</td>
<td>0</td>
<td>19.82</td>
</tr>
<tr>
<td>CBO</td>
<td>6</td>
<td>39</td>
<td>0</td>
<td>7.28</td>
</tr>
<tr>
<td>LCOM</td>
<td>17.58</td>
<td>122</td>
<td>0</td>
<td>27.59</td>
</tr>
<tr>
<td>PROD</td>
<td>16.93</td>
<td>198</td>
<td>0</td>
<td>25.83</td>
</tr>
<tr>
<td>SIZEL</td>
<td>199.17</td>
<td>1308</td>
<td>0</td>
<td>239.05</td>
</tr>
</tbody>
</table>

Table 4.1: Summary Statistics for TPM Design & Maintenance (N=93 classes)

\(^6\) Maintenance refers to changes made to pre-existing classes in the application prior to final release of the product.

\(^7\) Examination of the TPM system revealed that there several classes that were declared merely to hold data, and the metric values are 0 for such classes.
Examination of the first order Pearson correlations\(^8\) between the variables suggests that PROD is significantly positively correlated\(^9\) to SIZEL and CODER\(_5\) (dummy variable for a specific programmer). This suggests that these two variables are candidates for a base model for analysis of the marginal explanatory power of the six metrics i.e. the extent to which the metrics explain variance in the dependent variable over and beyond traditional measures. This approach is similar to the one adopted by Banker et al. in demonstrating the explanatory power of complexity metrics in a COBOL environment (Banker et al., 1993).

The metrics exhibit potential problems with multi-collinearity since RFC, CBO and WMC are highly (and significantly) correlated (greater than 0.8). This is not surprising since classes that have a large number of methods have a greater likelihood of calling methods from other classes causing an increase in the CBO and RFC counts. However, all other inter-metric correlations are insignificant.

Since data on the number of different classes that individual programmers were designing during the same work period were available, the BA\_SIZ variable was considered for inclusion in the regression model. A significant positive relationship between BA\_SIZ and PROD would indicate there are economies of scale in class design and maintenance, while a negative relationship would suggest the opposite. The VERSN variable was included to control for learning effects, since is possible that earlier versions of the system were less productive, due to a steep learning curve. The following

\(^8\) Appendix C contains the Pearson correlation coefficients for the variables in the study.
\(^9\) Level of significance is 0.05
regression model (N=93) was examined using PROD as the dependent variable:

\[
\text{PROD} = a + b1*\text{SIZEL} + b2*\text{BA}\_\text{SIZ} + b3*\text{VERSN} + b4*\text{CODER}\_2...+b8*\text{CODER}\_6
\]

The model (N=93) explains 70.9% of the variance in PROD and the Belsey Kuh Welsh test yields condition indices that are less than 15 indicating that the regression is not affected by problems of multi-collinearity (Belsley et al. 1980). However, the t-statistics indicate that a more parsimonious base model is the following (see Appendix C):

\[
\text{PROD} = 0.21 + 0.07*\text{SIZEL} + 39.11*\text{CODER}\_5
\]

\[
(t = 11.16) \quad (t = 6.33)
\]

This model yields an adjusted R-squared of 0.703 and both predictors are significant at the 0.05 level.

Since classes that have a large number of interconnections with other classes are likely to require greater of understanding of architectural details like the inheritance hierarchy and message passing, it is hypothesized that they will take more time to develop, test and modify. Consequently the productivity levels for such classes may be lower. To test this, the CBO metric (which is a measure of inter-class coupling) is added to the base model, and yields the following result (adjusted R-squared = 0.724, N=93)\(^{10}\):

\[
\text{PROD} = 3.17 + 0.07*\text{SIZEL} + 36.58*\text{CODER}\_5 - 0.56*\text{CBO}
\]

\[
(t = 11.89) \quad (t = 6.08) \quad (t = -2.82)
\]

---

\(^{10}\) The Belsey Kuh Welsh test indicates that the model does not suffer from the effects of possible multi-collinearity between the predictors.
These data suggest that higher CBO impacts PROD negatively. This confirms a long lasting result in software engineering that higher coupling is detrimental to productive design and maintenance. This lends added credibility to CBO as a legitimate measure of coupling in OO software. The negative 0.56 coefficient for CBO indicates that a unit increase in CBO decreases productivity by 0.56 lines of code per work day. This model also demonstrates that despite the overpowering influence of the programmer on productivity, the metric is able to predict additional variance. None of the other five complexity metrics added significant explanatory power.
3.1.2 Design Productivity

Since the proposed metrics are design related, it may be hypothesized that they may be better predictors of design productivity rather than both design and maintenance productivity. Therefore the next set of models were examined for only the newly designed classes in the TPM system. This "design only" data set has 45 data points and contains classes that are designed by all programmers in the TPM team. The table below provides the summary statistics for the data set.

<table>
<thead>
<tr>
<th>Variable</th>
<th>Mean</th>
<th>Max</th>
<th>Min</th>
<th>Standard Deviation</th>
</tr>
</thead>
<tbody>
<tr>
<td>WMC</td>
<td>9.27</td>
<td>63</td>
<td>0</td>
<td>11.60</td>
</tr>
<tr>
<td>DIT</td>
<td>0.04</td>
<td>2</td>
<td>0</td>
<td>0.3</td>
</tr>
<tr>
<td>NOC</td>
<td>0.07</td>
<td>2</td>
<td>0</td>
<td>0.33</td>
</tr>
<tr>
<td>RFC</td>
<td>13.82</td>
<td>102</td>
<td>0</td>
<td>18.27</td>
</tr>
<tr>
<td>CBO</td>
<td>4.51</td>
<td>39</td>
<td>0</td>
<td>7.21</td>
</tr>
<tr>
<td>LCOM</td>
<td>6.96</td>
<td>90</td>
<td>0</td>
<td>16.46</td>
</tr>
<tr>
<td>PROD</td>
<td>28.36</td>
<td>198</td>
<td>2.13</td>
<td>32.91</td>
</tr>
<tr>
<td>SIZEL</td>
<td>335</td>
<td>1308</td>
<td>66</td>
<td>263.69</td>
</tr>
</tbody>
</table>

Table 4.2: Summary Statistics for TPM Design (N=45 classes)
Similar to the model for design and maintenance productivity, the following base model was examined for this data set (adjusted $R^2 = 0.80$, N=45):

$$\text{PROD} = 18.22 + 0.07 \times \text{SIZEL} - 0.0 \times \text{BA\_SIZ} - 12.92 \times \text{COD\_D2} - 28.68 \times \text{COD\_D3} + 60.42 \times \text{COD\_D5}$$

All the regressors are significant at the 0.05 level. This suggests that, as in the case of the design and maintenance data, both SIZEL and COD\_D5 strongly influence PROD.

Since it likely that classes with a larger number of methods involve a larger (and more complex) state space of dynamic and semantic behavior, it is hypothesized that WMC will negatively influence productivity. Adding the WMC metric to the base model yields a model with higher explanatory power. The table below shows the regression results:

<table>
<thead>
<tr>
<th>Variable</th>
<th>co-eff</th>
<th>std. error</th>
<th>T-stat</th>
<th>p-value</th>
</tr>
</thead>
<tbody>
<tr>
<td>CONSTANT</td>
<td>21.01</td>
<td>5.14</td>
<td>4.09</td>
<td>0.005</td>
</tr>
<tr>
<td>SIZEL</td>
<td>0.09</td>
<td>0.01</td>
<td>8.53</td>
<td>0.005</td>
</tr>
<tr>
<td>BA_SIZ</td>
<td>-0.004</td>
<td>0.00</td>
<td>-4.34</td>
<td>0.005</td>
</tr>
<tr>
<td>COD_D2</td>
<td>-13.42</td>
<td>5.34</td>
<td>-2.51</td>
<td>0.02</td>
</tr>
<tr>
<td>COD_D3</td>
<td>-32.93</td>
<td>9.93</td>
<td>-3.32</td>
<td>0.005</td>
</tr>
<tr>
<td>COD_D5</td>
<td>41.98</td>
<td>8.82</td>
<td>4.76</td>
<td>0.005</td>
</tr>
<tr>
<td>WMC</td>
<td>-0.91</td>
<td>0.24</td>
<td>-3.84</td>
<td>0.005</td>
</tr>
</tbody>
</table>

Table 4.3: Regression model for the Design Data set
Despite the fact that the WMC metric is correlated to SIZEL (the co-efficient is +0.53), this model does not exceed the Belsey Kuh Welsh criterion for multicollinearity. The adjusted $R^2$ of the model is 0.852, and suggests that high WMC is detrimental to PROD, since the WMC coefficient is negative. Also, interestingly BA_SIZ (i.e. the size of other classes designed in the same time period) is negatively influencing PROD i.e. larger the number of classes designed in the same time period, lower the productivity. This suggests that there may be dis-economies of scale in the design of classes. And finally, SIZEL, though significant, has minimal impact on PROD, suggesting that size is not an important measure of OO design complexity.
3.1.3 Individual Programmer Productivity

The results from the previous two analyses indicate that a major predictor of productivity is the programmer. A question that then immediately arises is how well do the metrics in the proposed suite predict individual programmer effort? Since the TPM project had 6 different programmers, but had a significant majority of the classes designed by one programmer, a separate data set of the classes designed (or maintained) by this most prolific programmer in the team was created. The table below shows the summary statistics for this data set.

<table>
<thead>
<tr>
<th>Variable</th>
<th>Mean</th>
<th>Max</th>
<th>Min</th>
<th>Standard Deviation</th>
</tr>
</thead>
<tbody>
<tr>
<td>WMC</td>
<td>11.67</td>
<td>58</td>
<td>0</td>
<td>12.60</td>
</tr>
<tr>
<td>DIT</td>
<td>0.03</td>
<td>1</td>
<td>0</td>
<td>0.17</td>
</tr>
<tr>
<td>NOC</td>
<td>0.23</td>
<td>2</td>
<td>0</td>
<td>0.52</td>
</tr>
<tr>
<td>RFC</td>
<td>15.97</td>
<td>95</td>
<td>0</td>
<td>16.96</td>
</tr>
<tr>
<td>CBO</td>
<td>5.44</td>
<td>37</td>
<td>0</td>
<td>6.40</td>
</tr>
<tr>
<td>LCOM</td>
<td>18.77</td>
<td>122</td>
<td>0</td>
<td>26.51</td>
</tr>
<tr>
<td>PROD</td>
<td>12.22</td>
<td>52</td>
<td>0</td>
<td>11.61</td>
</tr>
<tr>
<td>SIZEL</td>
<td>150.26</td>
<td>848</td>
<td>0</td>
<td>157.36</td>
</tr>
</tbody>
</table>

Table 4.4: Summary Statistics for TPM Individual Programmer Design & Maintenance (N=66 classes)
Since size is a traditional explanatory variable for productivity, the base model for this data set was chosen to be:

\[ \text{PROD} = 1.6370 + 0.0102\times \text{SIZEL} \]

\[ (t=4.16) \quad (t=5.19) \]

The adjusted \( R^2 \) for this model is 0.318 (N=66), and SIZEL is significant at the 0.05 level.

Since complexity negatively influences productivity, it is expected that higher metric values (denoting higher complexity) would predict lower productivity. The metric variables were added to the base model, but only LCOM is a significant predictor. The final model is shown below (adjusted \( R^2 =0.43, \) N=66)

<table>
<thead>
<tr>
<th>Variable</th>
<th>co-eff</th>
<th>std. error</th>
<th>T-stat</th>
<th>p-value</th>
</tr>
</thead>
<tbody>
<tr>
<td>CONSTANT</td>
<td>2.35</td>
<td>0.40</td>
<td>5.81</td>
<td>0.0005</td>
</tr>
<tr>
<td>SIZEL</td>
<td>0.01</td>
<td>0.01</td>
<td>6.06</td>
<td>0.0005</td>
</tr>
<tr>
<td>LCOM</td>
<td>-0.03</td>
<td>0.01</td>
<td>-3.78</td>
<td>0.0003</td>
</tr>
</tbody>
</table>

Table 4.5: Regression model for the Individual Data set

The data suggest that some of the metrics that were significant in multiple programmer data sets are not significant in this case. Only the LCOM metric is a significant predictor of PROD. It appears that designing classes with low cohesion is detrimental to productivity for this individual. This could be attributable to the unique design habits of this most prolific programmer.
3.1.4 Exploring Individual Differences

The productivity analysis and the empirical data in Chapter 3 suggest that class design and maintenance are influenced by who performs the task. But do metric values vary significantly across different designers? A reliable method to determine statistically significant differences between designers is to use the analysis of variance (or ANOVA) technique. The six metrics are treated as dependent variables, and a categorical variable (CODERS) that signifies the designer is the independent variable in the model. ANOVA models were examined for the full data set and the design data set for the TPM system. The results for both models is shown below:
<table>
<thead>
<tr>
<th>Variable</th>
<th>Sum of Squares</th>
<th>Degree of freedom</th>
<th>Mean square error</th>
<th>F-Ratio</th>
<th>P value</th>
</tr>
</thead>
<tbody>
<tr>
<td>WMC</td>
<td>16468.3141</td>
<td>6</td>
<td>2744.71</td>
<td>14.08</td>
<td>0.0005</td>
</tr>
<tr>
<td>Error</td>
<td>16954.6859</td>
<td>87</td>
<td>194.88</td>
<td></td>
<td></td>
</tr>
<tr>
<td>DIT</td>
<td>0.6183</td>
<td>6</td>
<td>0.103</td>
<td>1.40</td>
<td>0.22</td>
</tr>
<tr>
<td>Error</td>
<td>6.3817</td>
<td>87</td>
<td>0.07</td>
<td></td>
<td></td>
</tr>
<tr>
<td>NOC</td>
<td>5.9668</td>
<td>6</td>
<td>0.99</td>
<td>3.59</td>
<td>0.003</td>
</tr>
<tr>
<td>Error</td>
<td>24.0332</td>
<td>87</td>
<td>0.27</td>
<td></td>
<td></td>
</tr>
<tr>
<td>RFC</td>
<td>32315.791</td>
<td>6</td>
<td>5385.96</td>
<td>14.64</td>
<td>0.0005</td>
</tr>
<tr>
<td>Error</td>
<td>31986.2086</td>
<td>87</td>
<td>367.65</td>
<td></td>
<td></td>
</tr>
<tr>
<td>CBO</td>
<td>3646.9924</td>
<td>6</td>
<td>607.83</td>
<td>11.56</td>
<td>0.0005</td>
</tr>
<tr>
<td>Error</td>
<td>4571.0076</td>
<td>87</td>
<td>52.54</td>
<td></td>
<td></td>
</tr>
<tr>
<td>LCOM</td>
<td>41457.76</td>
<td>6</td>
<td>6909.62</td>
<td>10.49</td>
<td>0.0005</td>
</tr>
<tr>
<td>Error</td>
<td>57293.2319</td>
<td>87</td>
<td>658.54</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Table 4.6: ANOVA for the six metrics across different designers (Full data set)
<table>
<thead>
<tr>
<th>Variable</th>
<th>Sum of Squares</th>
<th>Degree of freedom</th>
<th>Mean square error</th>
<th>F-Ratio</th>
<th>P value</th>
</tr>
</thead>
<tbody>
<tr>
<td>WMC</td>
<td>4637.46</td>
<td>5</td>
<td>927.49</td>
<td>7.21</td>
<td>0.0001</td>
</tr>
<tr>
<td>Error</td>
<td>5145.53</td>
<td>40</td>
<td>128.63</td>
<td></td>
<td></td>
</tr>
<tr>
<td>DIT</td>
<td>0.44</td>
<td>5</td>
<td>0.0889</td>
<td>1.00</td>
<td>0.43</td>
</tr>
<tr>
<td>Error</td>
<td>3.55</td>
<td>40</td>
<td>0.088</td>
<td></td>
<td></td>
</tr>
<tr>
<td>NOC</td>
<td>0.31</td>
<td>5</td>
<td>0.062</td>
<td>0.5294</td>
<td>0.75</td>
</tr>
<tr>
<td>Error</td>
<td>4.6897</td>
<td>40</td>
<td>0.117</td>
<td></td>
<td></td>
</tr>
<tr>
<td>RFC</td>
<td>10523.13</td>
<td>5</td>
<td>2104.62</td>
<td>6.59</td>
<td>0.0001</td>
</tr>
<tr>
<td>Error</td>
<td>12768.86</td>
<td>40</td>
<td>319.22</td>
<td></td>
<td></td>
</tr>
<tr>
<td>CBO</td>
<td>1207.90</td>
<td>5</td>
<td>241.58</td>
<td>4.84</td>
<td>0.0015</td>
</tr>
<tr>
<td>Error</td>
<td>1995.09</td>
<td>40</td>
<td>49.87</td>
<td></td>
<td></td>
</tr>
<tr>
<td>LCOM</td>
<td>5764.79</td>
<td>5</td>
<td>1152.95</td>
<td>5.52</td>
<td>0.0006</td>
</tr>
<tr>
<td>Error</td>
<td>8340.20</td>
<td>40</td>
<td>208.50</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Table 4.7: ANOVA for the six metrics across different designers (Design data)

The ANOVA suggests that for these data, the WMC, RFC, CBO and LCOM metrics are significantly different for the designers in the project. This is true for both design and maintenance of classes in the application. Performing a
multivariate analysis for both data sets suggests that as a group of variables, the suite of six metrics vary significantly by designer\textsuperscript{11}. It should be noted that this holds true for the entire sample of classes, not for differences between individual designers. Additional tests to explore differences between the average values of the suite of metrics, suggest that there are statistically significant differences in the LCOM averages between CODER_3, CODER_1, CODER_2 and CODER_5 for class design\textsuperscript{12}. Also, there were significant differences between CODER_4, CODER_2 and CODER_5 in LCOM averages for class design and maintenance. None of the other five metrics were found to have significant differences in average values between different designers. The pair-wise comparison of the LCOM metric for the five designers is shown in the table below (e.g. CODER_3 has higher LCOM than CODER_1):

<table>
<thead>
<tr>
<th></th>
<th>CODER_1</th>
<th>CODER_2</th>
<th>CODER_3</th>
<th>CODER_4</th>
<th>CODER_5</th>
</tr>
</thead>
<tbody>
<tr>
<td>CODER_1</td>
<td>•</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>CODER_2</td>
<td>NS</td>
<td>•</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>CODER_3</td>
<td>&gt;</td>
<td>&gt;</td>
<td>•</td>
<td></td>
<td></td>
</tr>
<tr>
<td>CODER_4</td>
<td>NS</td>
<td>&lt;</td>
<td>NS</td>
<td>•</td>
<td></td>
</tr>
<tr>
<td>CODER_5</td>
<td>NS</td>
<td>NS</td>
<td>&lt;</td>
<td>&lt;</td>
<td>•</td>
</tr>
</tbody>
</table>

Table 4.8: Pair-wise comparison of designers for the LCOM metric

\textsuperscript{11} The Wilkes Lamda F-statistic, Pillai trace and Hotelling-Lawley trace all suggest this at the 0.005 level of significance.

\textsuperscript{12} Classes designed by CODER_3 on average have higher LCOM values than CODER_5, CODER_2 or CODER_1. This result is significant at the 0.05 level.
3.2 Exploratory Models for Rework and Reusability

Since one of the major benefits of the OO paradigm is the ability to reuse key classes across applications, reusability is a key variable of managerial interest. Reusable classes can be a key to faster development cycles. In theory, a designer could assemble pre-existing classes, and build a new system in a fraction of the time it would take to build it anew. However, it is rare that a class created in one application would be readily reusable in another without modest rework. This rework effort was recorded for the classes from the FIS project that were reused for the SLB project. The rework effort was measured in the number of hours spent by the SLB programmer to modify the class for use in the SLB project. Also as mentioned in the earlier section, each of these classes were rated for their reusability by two independent experts. The summary statistics for these variables is presented in the table below:
<table>
<thead>
<tr>
<th>Variable</th>
<th>Mean</th>
<th>Max</th>
<th>Min</th>
<th>Standard Deviation</th>
</tr>
</thead>
<tbody>
<tr>
<td>WMC</td>
<td>19.86</td>
<td>31</td>
<td>2</td>
<td>7.46</td>
</tr>
<tr>
<td>DIT</td>
<td>1.14</td>
<td>2</td>
<td>1</td>
<td>0.36</td>
</tr>
<tr>
<td>NOC</td>
<td>0.07</td>
<td>2</td>
<td>0</td>
<td>0.38</td>
</tr>
<tr>
<td>RFC</td>
<td>38.79</td>
<td>93</td>
<td>8</td>
<td>23.37</td>
</tr>
<tr>
<td>CBO</td>
<td>8.61</td>
<td>22</td>
<td>4</td>
<td>4.37</td>
</tr>
<tr>
<td>LCOM</td>
<td>28.32</td>
<td>387</td>
<td>0</td>
<td>89.20</td>
</tr>
<tr>
<td>LINES</td>
<td>98.82</td>
<td>153</td>
<td>38</td>
<td>25.97</td>
</tr>
<tr>
<td>REUSABA</td>
<td>6.48</td>
<td>90</td>
<td>1.5</td>
<td>2.34</td>
</tr>
<tr>
<td>EFFORT</td>
<td>5.68</td>
<td>40</td>
<td>0.15</td>
<td>8.97</td>
</tr>
</tbody>
</table>

Table 4.9: Summary statistics for the FIS data set (N=28 classes)

The two dependent variables of interest in this data set are EFFORT and REUSABA, and since LINES is the traditional independent variable, it was chosen as the base model for this analysis (MODEL 1A below). Since classes with higher inheritance, lower cohesion or large number of inter-class connections may require additional effort to customize in another system (e.g. duplicating methods called from other classes that are no longer available in the new system) it is hypothesized that high DIT and high LCOM may explain
higher rework effort. Likewise low DIT, RFC and LCOM may explain higher levels of reusability. To test these relationships, the following regression models were analyzed.

MODEL 1A: EFFORT = a + b*LINES

MODEL1B: EFFORT = a + b1*LINES + b2*DIT

MODEL1C: EFFORT = a + b1*LINES + b2*DIT + b3*LCOM

MODEL2A: REUSABA = a + b*LINES

MODEL2B: REUSABA = a + b1*LINES + b2*DIT + b3*LCOM + b4*RFC
The table below shows the regression results:

<table>
<thead>
<tr>
<th>Model</th>
<th>Variable</th>
<th>co-eff</th>
<th>std. error</th>
<th>T-stat</th>
<th>p-value</th>
</tr>
</thead>
<tbody>
<tr>
<td>1A</td>
<td>CONSTANT</td>
<td>13.21</td>
<td>6.75</td>
<td>1.96</td>
<td>0.06</td>
</tr>
<tr>
<td>R² = 0.05</td>
<td>LINES</td>
<td>-0.08</td>
<td>0.07</td>
<td>-1.15</td>
<td>0.26</td>
</tr>
<tr>
<td>1B</td>
<td>CONSTANT</td>
<td>-2.62</td>
<td>8.56</td>
<td>-0.3</td>
<td>0.7</td>
</tr>
<tr>
<td>R² = 0.20</td>
<td>LINES</td>
<td>-0.05</td>
<td>0.06</td>
<td>-0.8</td>
<td>0.41</td>
</tr>
<tr>
<td></td>
<td>DIT</td>
<td>11.58</td>
<td>4.40</td>
<td>2.6</td>
<td>0.01</td>
</tr>
<tr>
<td>1C</td>
<td>CONSTANT</td>
<td>-8.78</td>
<td>7.81</td>
<td>-0.1</td>
<td>0.92</td>
</tr>
<tr>
<td>R² = 0.33</td>
<td>LINES</td>
<td>-0.08</td>
<td>4.01</td>
<td>-1.5</td>
<td>0.13</td>
</tr>
<tr>
<td></td>
<td>DIT</td>
<td>12.36</td>
<td>0.01</td>
<td>3.0</td>
<td>0.005</td>
</tr>
<tr>
<td></td>
<td>LCOM</td>
<td>0.04</td>
<td>0.05</td>
<td>2.5</td>
<td>0.01</td>
</tr>
<tr>
<td>2A</td>
<td>CONSTANT</td>
<td>7.13</td>
<td>0.79</td>
<td>3.97</td>
<td>0.005</td>
</tr>
<tr>
<td>R² = 0.01</td>
<td>LINES</td>
<td>-0.01</td>
<td>0.02</td>
<td>-0.37</td>
<td>0.71</td>
</tr>
<tr>
<td>2B</td>
<td>CONSTANT</td>
<td>11.45</td>
<td>1.25</td>
<td>9.10</td>
<td>0.0005</td>
</tr>
<tr>
<td>R² = 0.77</td>
<td>LINES</td>
<td>0.01</td>
<td>0.01</td>
<td>1.57</td>
<td>0.12</td>
</tr>
<tr>
<td></td>
<td>DIT</td>
<td>-4.13</td>
<td>0.64</td>
<td>-6.39</td>
<td>0.0005</td>
</tr>
<tr>
<td></td>
<td>RFC</td>
<td>-0.04</td>
<td>0.01</td>
<td>-3.35</td>
<td>0.002</td>
</tr>
<tr>
<td></td>
<td>LCOM</td>
<td>-0.01</td>
<td>0.002</td>
<td>-3.77</td>
<td>0.001</td>
</tr>
</tbody>
</table>

Table 4.10: Regression results for the FIS data set
The models suggest that LINES is not a useful predictor of EFFORT or REUSABA\textsuperscript{13}. In contrast, the DIT, LCOM and RFC metrics have significant prediction properties. Of these three metrics, only RFC is significantly correlated (the co-efficient is +0.59) to LINES, however all the models pass the Belsey Kuh Welsh test for multi-collinearity. DIT and LCOM explain over 30\% of the variance in rework effort, while DIT, LCOM and RFC collectively explain over 75\% of the variance in reusability of classes. Of these three metrics, DIT appears to have the most influence on EFFORT and REUSABA. High level classes (low DIT values) take substantially less time to rework, and are more reusable across projects. The data suggest that classes with high DIT, RFC and LCOM metric values are likely to require more rework, and therefore possibly less reusable.

3.3 Exploratory model for early design effort

All the models tested thus far have utilized data gathered from source code from the TPM and FIS projects. The SLB project data set is different from the others in that the metrics data has been gathered prior to writing any code. The metrics are obtained from the output of a JAD (joint application design) session, where major application classes are specified, but not coded. Besides the metrics data, the amount of time spent (in hours) to specify the high level design of each of the classes was also collected. The table below provides the usual summary statistics.

\textsuperscript{13} Examination of a possible quadratic relationship between LINES and EFFORT or REUSABA did not yield any significant results (see Appendix C for the models).
<table>
<thead>
<tr>
<th>Variable</th>
<th>Mean</th>
<th>Max</th>
<th>Min</th>
<th>Standard Deviation</th>
</tr>
</thead>
<tbody>
<tr>
<td>WMC</td>
<td>6.44</td>
<td>22</td>
<td>1</td>
<td>3.76</td>
</tr>
<tr>
<td>DIT</td>
<td>1.89</td>
<td>3</td>
<td>1</td>
<td>0.75</td>
</tr>
<tr>
<td>NOC</td>
<td>0.85</td>
<td>11</td>
<td>0</td>
<td>2.35</td>
</tr>
<tr>
<td>RFC</td>
<td>9.85</td>
<td>42</td>
<td>1</td>
<td>8.34</td>
</tr>
<tr>
<td>CBO</td>
<td>1.33</td>
<td>8</td>
<td>0</td>
<td>2.15</td>
</tr>
<tr>
<td>LCOM</td>
<td>3.89</td>
<td>83</td>
<td>0</td>
<td>16.02</td>
</tr>
<tr>
<td>EFFORT</td>
<td>6.85</td>
<td>40</td>
<td>0.3</td>
<td>11.37</td>
</tr>
</tbody>
</table>

Table 4.11: Summary Statistics for SLB (N=27 classes)

It was hypothesized that DIT would largely influence EFFORT, since classes lower in the inheritance hierarchy are simple (their behavior would be largely inherited from super-classes) and would take less effort to design. DIT is the independent variable in the base model. Unfortunately, since WMC, LCOM, RFC and CBO are highly (and significantly) correlated, addition of any two out of these four metrics together in the same model results in failure of the Belsey Kuh Welsh test. Therefore only regression models with one metric variable in addition to the base model is meaningful. The following regression models were examined for this data set, and all cases, the models are free from the effects of serious multi-collinearity.
BASE: $\text{EFFORT} = a + b_1 \times \text{DIT}$

MODEL A: $\text{EFFORT} = a + b_1 \times \text{DIT} + b_2 \times \text{WMC}$

MODEL B: $\text{EFFORT} = a + b_1 \times \text{DIT} + b_2 \times \text{RFC}$

MODEL C: $\text{EFFORT} = a + b_1 \times \text{DIT} + b_2 \times \text{CBO}$

MODEL D: $\text{EFFORT} = a + b_1 \times \text{DIT} + b_2 \times \text{LCOM}$

The table below shows the regression results:
<table>
<thead>
<tr>
<th>Model</th>
<th>Variable</th>
<th>co-eff</th>
<th>std. error</th>
<th>T-stat</th>
<th>p-value</th>
</tr>
</thead>
<tbody>
<tr>
<td>BASE</td>
<td>CONSTANT</td>
<td>14.72</td>
<td>5.89</td>
<td>2.40</td>
<td>0.01</td>
</tr>
<tr>
<td>R^2 = 0.03</td>
<td>DIT</td>
<td>-4.16</td>
<td>2.90</td>
<td>-1.4</td>
<td>0.16</td>
</tr>
<tr>
<td>A</td>
<td>CONSTANT</td>
<td>12.03</td>
<td>5.00</td>
<td>2.40</td>
<td>0.02</td>
</tr>
<tr>
<td>R^2 = 0.33</td>
<td>DIT</td>
<td>-9.41</td>
<td>4.88</td>
<td>-3.27</td>
<td>0.005</td>
</tr>
<tr>
<td></td>
<td>WMC</td>
<td>1.95</td>
<td>0.57</td>
<td>3.41</td>
<td>0.005</td>
</tr>
<tr>
<td>B</td>
<td>CONSTANT</td>
<td>13.92</td>
<td>4.70</td>
<td>2.96</td>
<td>0.01</td>
</tr>
<tr>
<td>R^2 = 0.39</td>
<td>DIT</td>
<td>-8.47</td>
<td>2.56</td>
<td>-3.31</td>
<td>0.005</td>
</tr>
<tr>
<td></td>
<td>RFC</td>
<td>0.91</td>
<td>0.23</td>
<td>3.93</td>
<td>0.005</td>
</tr>
<tr>
<td>C</td>
<td>CONSTANT</td>
<td>17.09</td>
<td>4.91</td>
<td>3.48</td>
<td>0.005</td>
</tr>
<tr>
<td>R^2 = 0.35</td>
<td>DIT</td>
<td>-7.74</td>
<td>2.59</td>
<td>-2.97</td>
<td>0.01</td>
</tr>
<tr>
<td></td>
<td>CBO</td>
<td>3.25</td>
<td>0.91</td>
<td>3.58</td>
<td>0.005</td>
</tr>
<tr>
<td>D</td>
<td>CONSTANT</td>
<td>16.62</td>
<td>5.58</td>
<td>2.98</td>
<td>0.01</td>
</tr>
<tr>
<td>R^2 = 0.16</td>
<td>DIT</td>
<td>-5.76</td>
<td>2.81</td>
<td>-2.05</td>
<td>0.05</td>
</tr>
<tr>
<td></td>
<td>LCOM</td>
<td>0.29</td>
<td>0.13</td>
<td>2.16</td>
<td>0.04</td>
</tr>
</tbody>
</table>

Table 4.12: Regression results for the SLB data set
DIT is not significant in the base model, but significant in all other models. This suggests that DIT has a moderating influence on EFFORT i.e. its impact is felt only in presence of other variables. The correlation between WMC, LCOM, RFC and CBO precluded the addition of any three (or more) metrics together in the same model. This is unfortunate from the point of view of statistical prediction, but it should be emphasized that three out of the four models containing two of the metrics explain in excess of 30% of the variance in design and analysis effort. This itself provides considerable information to managers, a) classes that make use of inheritance (i.e. high DIT) take less time to design, b) classes that are highly cohesive (low LCOM) take less time to design, c) classes that are highly coupled (high CBO) take more time to design, d) classes that call many methods (high RFC) take more time to design and e) classes with more methods (high WMC) take more time to design.
4.0 Summary and Limitations

The table below summarizes the situations in which the proposed metrics explain variance in the dependent variables of productivity, design effort, rework effort and reusability.

<table>
<thead>
<tr>
<th>Metric</th>
<th>Data Set</th>
<th>N</th>
<th>Impact</th>
</tr>
</thead>
<tbody>
<tr>
<td>WMC</td>
<td>TPM (Design only)</td>
<td>93</td>
<td>Significant predictor of design productivity</td>
</tr>
<tr>
<td></td>
<td>SLB</td>
<td>27</td>
<td>Significant predictor of early design effort</td>
</tr>
<tr>
<td>DIT</td>
<td>FIS</td>
<td>28</td>
<td>Significant predictor of rework</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>Significant predictor of reusability</td>
</tr>
<tr>
<td>RFC</td>
<td>FIS</td>
<td>28</td>
<td>Significant predictor of reusability</td>
</tr>
<tr>
<td></td>
<td>SLB</td>
<td>27</td>
<td>Significant predictor of early design effort</td>
</tr>
<tr>
<td>CBO</td>
<td>TPM (Design and Maintenance)</td>
<td>93</td>
<td>Significant predictor of productivity</td>
</tr>
<tr>
<td></td>
<td>SLB</td>
<td>27</td>
<td>Significant predictor of early design effort</td>
</tr>
<tr>
<td>LCOM</td>
<td>TPM (Individual Programmer)</td>
<td>66</td>
<td>Significant predictor of productivity</td>
</tr>
<tr>
<td></td>
<td>FIS</td>
<td>28</td>
<td>Significant predictor of rework</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>Significant predictor of reusability</td>
</tr>
</tbody>
</table>

Table 4.13: Summary of metrics impact

As the data demonstrate, 5 of the 6 metrics (the exception is NOC) appear to predict key variables of interest to MIS managers. One explanation for the fact that the NOC metric does not appear to be a significant predictor in any of the
models is that the very few classes in these three systems have any appreciable number of children. Designers at IBC espouse the view that classes should have a limited number of sub-classes. It is possible that at other data sites, this metric could play a role in predicting key dependent variables. In the TPM data, WMC, CBO and LCOM explain variance in productivity of design and maintenance in addition to the usual contextual variables such as individual coder, version of the system. This demonstrates that the metrics have predictive power on the margin over and beyond traditional measures.

The metrics also predict new design efforts. As the FIS and SLB data demonstrate, the DIT, RFC, LCOM, WMC and CBO metrics have greater impact than size does on the amount of time it takes to rework or design a class. More importantly it gives a manager better understanding of what the impact of design decisions (such as communication with other classes or subclassing) have on the effort required. Since reuse is one of the advertised benefits of the OO approach, the final model from the SLB system analysis is of high current interest. Designing reusable classes implies certain design tradeoffs. For example, classes with high DIT take less time to design, but longer to rework and reuse. Using these models, managers could decide the relative worth building a class for reuse.

The summary statistics also indicate that metric values vary widely from one system to another (e.g. mean RFC in the SLB project is 9.85 while mean RFC for FIS is 38.79). As suggested earlier in Chapter 3, this could be due to the inherent nature of the application, the language employed or the work practices followed during development. TPM is a large system written in C++ using UNIX tools, while FIS is an Objective-C system written using
NeXTStep, whereas the SLB system is pre-code design information. The implication for managers is that bad choices of tools, languages and work practices can lead to increase in controllable complexity and lead to lowered productivity, increased rework or lowered reusability of classes. Also, the low values of DIT and NOC indicate that reuse opportunities (via inheritance) are perhaps being compromised in favor of comprehensibility of the overall architecture of the application.

A certain degree of caution needs to be exercised in the interpretation and use of the metrics are levers of managerial control. There is multi-collinearity between the six metrics (particularly between the WMC, RFC and CBO metrics), that suggest that a smaller set of hybrid metrics would have the same degree of predictive power and yield superior parameter estimates. However, at this early stage in the development and adoption of OOD, these metrics compensate for the multi-collinearity problem by being more meaningful to designers and managers in their work. The data sets from the SLB and FIS projects have small sample sizes, which reduces the statistical power and reliability of the estimates. Additionally all the data sets are drawn from a single organization that develops software systems for highly specialized applications in the financial services industry, this would limit the applicability of the results to other general OO environments. Lastly, as the TPM data sets demonstrate the contextual variables in a project such as work practices and individual ability have large impact on the productivity of OO design. The metrics must necessarily be used to assist rather than assess individual programmers.

This set of six proposed metrics is presented as the first empirically validated proposal for formal metrics for OOD. By bringing together the formalism of
measurement theory, Bunge's ontology, Weyuker's evaluation criteria and empirical data from commercial projects, this dissertation seeks to demonstrate the level of rigor required in the development of usable metrics for design of software systems. Of course, there is no reason to believe that the proposed metrics will be found to be comprehensive, and further work could result in additions, changes and possible deletions from this suite. However, the suite provides coverage for all three of Booch's steps for OOD and, at a minimum, this metrics suite should lay the groundwork for a formal language to describe metrics for OOD. In addition, these metrics may also serve as a generalized solution for other researchers to rely on when seeking to develop specialized metrics for particular purposes or customized environments. It is hoped that this research helps in moving software development management (OO software in particular) towards a strong theoretical base should help to provide a basis for significant future progress.
BIBLIOGRAPHY


Chapter Five: Transforming Software Development
Chapter Outline

1.0 Introduction....................................................................................................................144
2.0 Background of the IS department at International Bank Corporation.........................................................145
3.0 Transformation..............................................................................................................146
  3.1 The IT Triad ..............................................................................................................148
    3.1.1 Object Technology..............................................................................................148
    3.1.2 Rapid Application Development......................................................................149
    3.1.3 Measurement......................................................................................................150
      Measurement based charge-back system.................................................................150
      Competitive Benchmarking......................................................................................151
      Creation of new metrics working group.................................................................151
    3.1.4 Delivering Key Products....................................................................................153
  3.2 Automating the Triad...............................................................................................154
4.0 Building Core Capability............................................................................................154
Bibliography......................................................................................................................157
1.0 INTRODUCTION

Hitherto the dissertation has dealt with the theoretical development, commercial measurement and statistical validation of object oriented metrics. The empirical data from the two large commercial systems (presented in Chapter 3) demonstrated that senior designers or project managers could use the metrics to preserve architectural integrity. But can IS managers use such metrics for more broad based decision making and control? The answer is unequivocally in the affirmative. Leading edge organizations have already begun to use the proposed metrics to determine internal IS policy. One example is Boeing Aerospace, a leading US manufacturer of commercial and military aircraft. The Advanced Concepts Group at Boeing designed two systems (that performed identical functions) with two different methodologies. Based on the metric evaluations for the two systems that corroborated their experience with maintenance costs, one methodology was chosen to be the corporate standard (Sharble and Cohen 1993). In addition to the enhanced capability for resource estimation and planning, the value of the proposed suite of metrics also lies in the implicit discipline that it brings to the abstract business of software development.

However, metrics and measurements are part of a larger management agenda for improvement within IS organizations. The "metrics mind set" is a powerful agent for facilitating change. This chapter describes how the IS department of an investment bank transformed its systems development from a IBM COBOL operation to an UNIX-client-server shop in 3 years. The key enablers for this swift change were rapid application development (RAD), object orientation (OO) and measurement. The diagram below, an adaptation
from (Rockart and Hofman 1992) outlines the key issues that frame the discussion.

![Diagram](image)

**Figure 5.1: A Managerial Perspective on Transforming Software Development**

2.0 **BACKGROUND OF THE IS DEPARTMENT AT INTERNATIONAL BANK CORPORATION**

The International Bank Corporation (IBC) is a large financial services that has historically enjoyed a rock-solid reputation in the international banking community. The Information Systems Division (ISD) within IBC is based in London, and provides the investment banking operations with computer support. The business and technology environment until the 1980s can be best characterized as stable. The "gentlemen's club" investment banking
culture in the Europe was far removed from the frenetic trading pits of Chicago or New York. This stability was mirrored within ISD, with IBM mainframe technology as the only computing platform and COBOL as the dominant software technology.

The business environment changed dramatically in the mid 1980s with large scale computerization of the stock market in the UK, global competition, fast paced growth in Asian markets and the advent of brand new (and highly profitable) financial instruments. Likewise on the technology front, the value proposition of high end PCs, workstations and networks challenged the economics of continuing with IBM mainframe systems. Financial services was being redefined for a new world of global, electronic commerce\textsuperscript{1}, and IBC had to respond quickly or face steady, inevitable decline.

\section*{3.0 Transformation}

Change began at the top of the organization. In late 1989, a new CEO was appointed to meet the substantial challenges that faced the company. The new CEO brought in several senior managers from outside the company, and outlined a strategic vision to become a leading international wholesale bank (one that offers a full range of products and services). While the future of IS in many organizations has been declared extinct (Dearden 1987, Hopper 1990) or one that is totally outsourced (Huber 1993, Lacity and Hirschheim 1993), senior managers at IBC believed that without a strong IS department, their strategic vision would be in jeopardy. A new Director for ISD was appointed

\textsuperscript{1} For a provocative view of IT mediated financial services see "Banker's Trust's 2020 Vision" in \textit{The Economist}, March 26, 1994
in 1991 to revamp the IS function and reverse the IS department's tendency to provide old, overpriced technology. In the ensuing shake-out, the IS department shrank from 180 to 120 people. The IS strategy was to become the most competitive provider of technological solutions to its users, and build a reputation as the premier user of IT in the UK financial services community.

IBC participates in diverse financial markets, ranging from merchant banking instruments to the fast paced equities, bonds and foreign exchange trading. Each product category calls for a different strategy and differing IT requirements. However, the need to share information between different applications, and between different users required an integrated approach. The new ISD Director identified clearly that the choice of the IT infrastructure was a senior management decision, and a pivotal component of the firm's IT strategy. Since there were substantial investments in mainframe technology, the firm did not have the luxury of a "clean slate." The decision was made to switch to SUN SPARC 2 workstations and UNIX operating systems for front-office real time, on-line systems but continue to use IBM mainframes for typical back office applications. Gradually on a pay-as-you-go basis, the two worlds would be integrated so that a seamless, flexible resource would be available to users.

Implicitly, the Strategic Alignment Model (SAM) proposed in (Henderson and Venkatram 1993) was followed at IBC. This model calls for strategic, functional and cross-functional alignments between business and technology. The IS department was organized around specific businesses of IBC (e.g. Equities, Capital Markets & Treasury), steering committees comprising of IS and line managers oversaw IS projects. In 1991, three major components of the IT capability were identified to meet the increasing demand from users for
extensive analytics, information and superior display and interface capabilities. Object Technology, Rapid Application Development (RAD) and Measurement formed the triad, based on their potential for immediate impact on the productivity.

3.1 The IT Triad

3.1.1 Object Technology

Object technology was considered to be a major component of IT capability for several reasons. First, it offered a modeling technique that could be used during RAD sessions to reduce the semantic gap that traditionally exists between users and designers. Key abstractions (or classes) in the application are determined in conjunction with users, moreover these abstractions are labeled in business terminology. For example, in a Trade Position Manager system, a key class Trade has subclasses called BondTrade, FxTrade that capture the specific attributes of these financial instruments. Second, OT promises the elusive goal of reuse within and between applications. While some of these reuse expectations remain unfulfilled, the lure of inexpensive, reusable code was (and indeed remains) a potent lure. Thirdly, OO was suitable for development of GUI interfaces, which were urgently needed for the front office systems used by users on the trading floor. Lastly, a US-based strategic ally was a leading edge user of OO on the NeXT work station environment. These prior positive experiences with OO were hoped to be duplicated in London.
3.1.2 Rapid Application Development

Rapid Application Development was chosen as a crucial building block at IBC based on independent research reports that RAD costs are lower over the lifecycle of a software system (Hekmatpour 1987, Gordon and Bieman 1993). RAD relies on intensive structured interaction between users and designers to determine systems requirements. A team of systems and business experts attend workshop sessions, and a model of the business problem is produced. The model is further refined in subsequent sessions if required. The output of a RAD is a scoping document that contains the project charter, objectives, measures and priorities. This document is jointly authored and issued by the ISD designers and the business users. The RAD methodology followed at IBC is similar to the Joint Application Design (JAD) methodology pioneered at IIM².

Each RAD session is conducted using OO analysis and design techniques, and in addition to the scoping document, key application abstractions are mapped to class cards that are the early design artifacts for the system. These class cards are produced regardless of whether the final system is written in an OO language or in traditional COBOL. At IBC, all systems are conceived as OO software systems. This was a conscious decision on the part of IS management for two reasons: 1) to promote OO literacy across the board within the department, and 2) to minimize the number of different methodologies and process techniques that people have to understand, appreciate and adopt. The objective is to reap economies of scale in learning in software development.

²See (Davidson 1993) for an exploratory analysis of JAD.
3.1.3 Measurement

The ISD management motto has been "you cannot control what you cannot measure." Consequently, senior IS managers are constantly measuring both the process and the products that they are responsible for. While many organizations readily admit to benefits of measurements, little is done in this area. By contrast, IBC has adopted this a significant management tool. The Director of ISD says "our people are free to raise questions about specific metrics, but they cannot question whether we need to measure."

Measurement based charge-back system

ISD operates as a profit center within IBC, and follows a charge back system for the services it provides to users from different strategic business units (SBUs) within the bank. Key SBUs that "buy" ISD services include the Equities, Capital Markets & Treasury, Bond Trading and Foreign Exchange Trading. Based on requirements, an estimate of function points of the software system is made. Client SBUs are charged for ISD services on the basis of this metric. Standard charges are negotiated based on administrative, infrastructure and human resource costs.

Since the revenue into ISD is based strictly on function points, each manager and designer is keenly aware of this measurement for the project that they are currently engaged in. In 1993 bonuses for ISD personnel were based in part on their function point contribution. This graphically underscores the importance of measurements within ISD.
Competitive Benchmarking

IBC has an on-going contract with a leading IT consulting firm to benchmark IS performance vis-a-vis other leading IS departments. Each year, the consulting firm examines all IS projects undertaken by ISD, and reports on the size, productivity, quality, defect rate, on-time delivery in comparison to other IS departments. In addition to these figures, comparisons are made on other significant measures such as time-pressure of work, requirements stability and staff turnover. The benchmarking report is issued twice a year, and has wide circulation within ISD and IBC.

In addition to pin-pointing specific IS weaknesses, the benchmarking is a goal setting opportunity. In 1991, the stated goal to was be within the top 10% of the firms studied by the consulting firm. This has been achieved three years in a row, and the current goal is to be within the top 1%. The quality, productivity and delivery indices are common knowledge within ISD, and the numbers serve to reinforce the commitment to excellence in those dimensions.

Creation of new metrics working group

Until 1993 the responsibility to report and analyze metrics resided with a single manager within ISD. This manager tracked function points, user reported errors, bug fixes and productivity estimates for each project and reported them in a monthly IS report that had wide circulation. This created awareness that such measures were considered to be important by higher level managers. Despite the wide circulation of this data, the commitment and collection of metrics varied across different project groups. The quality of
the reporting was highly contingent on the level of commitment of individual project managers.

This centralized reporting changed in 1993 to a more grass roots level program with the creation of a Metrics Working Group that comprises of members from each of the different IS projects. The goal of the Metrics Working Group, as stated by one of its members is to "make metrics a habit, not a chore." Each committee member is responsible for periodic collection, analysis and reporting to the committee and the Director of ISD. Another crucial function served by each committee member was educating others within their project groups about the value and usefulness of metrics. They also served as a conduit for voicing concerns about how metrics were being used and interpreted by senior management. The work of the committee assures consistency and reliability of metrics reporting, in addition to scanning, proposing and experimenting with new metrics proposals.3

One specific concern relates to the use of function points for measuring OO software. Since function points implicitly assume file structures, forms and data entry as the key elements of software, its appropriateness to OO software on UNIX machines has been questioned within ISD. This concern has prompted an examination of possible replacements for function points. The Metrics Working Group is formulating the notion of "task points" that can easily map to OO software, and the metrics proposed in this dissertation are integral to this calculation. The six metrics proposed in this dissertation will be reported on all future IS projects.

3 The validation of the metrics proposed in this dissertation was sponsored by one of the members of the metrics working group.
3.1.4 Delivering Key Products

Since 1991, two large new products have been delivered for worldwide use within IBC using the triad described above. Both are important systems in the IS infrastructure. One system is responsible for "straight through" trading *i.e.* it enables typical computers from the trading desk to communicate to back-office mainframe computers that maintain and consolidate the massive amount of data relating to the bank's current and historical trading in financial instruments. It manages the physical and logical distribution of data among different computer systems, while providing users with a consistent single database for product risk assessment and position keeping. Another system is a trading decision support system which provides pricing, analytics, risk management and management information to the Fixed Income Trading & Sales teams in the investment bank. In addition to these functions, it automates the manual process of issuing reports, and passes the consolidated information to other mainframe computer programs for initiating electronic transfer of funds, archival and storage.

Based on these two systems, several smaller systems for meeting specific needs have been built. These smaller systems tend to reuse the class libraries and the design principles adopted in the two large systems and are usually built by a small team of two designers within a period of six months. The Associate Director who spearheaded the development of the straight through trading system suggests that the reuse approach is at two levels, component level and design level. Component reuse is mainly the reuse of small classes (that provide universal functions like Time, Date, Screen Icons) while design level reuse is the borrowing of the architectural ideas that embody a system. But the Associate Director says that "the real challenge is achieving reuse of
intermediate level components, large piece parts of systems that can be employed within different architectures.” In this regard, the large scale reuse that was targeted remains an elusive goal.

3.2 Automating the Triad

The three elements of the IT Triad are collectively referred within IBC as the ISRAD approach. Since all new projects, and a growing number of maintenance work follow ISRAD, a set of automated tools are currently under development to support this. In early 1994, ISD entered into an alliance with a small startup company to build the required tools. This tool kit will capture all information from the RAD sessions, help with early design of application classes, gather project metrics (including the metrics proposed in this dissertation) and issue reports. The tool kit will be available to all members of ISD and also have wide circulation within IBC.

4.0 BUILDING CORE CAPABILITY

Senior managers, reflecting on the successful transformation within ISD, suggest that it was possible due to several factors: strong leadership, commitment, a high focus on education, aggressive implementation of technology and deploying the resources required to get the job done. One manager made an analogy to the remarkable military success of the US and its allies in the 1992 Gulf war. Plan carefully, choose the appropriate mechanisms, take the time to gather consensus and then deploy massive resources if you wish to accomplish the task in a short period of time.

Based on the needs of the business, IBC decided early on that IT was a required core capability to be a major player in the financial services business.
It then went about building that core capability and chose specific process technologies. The core capability has resulted in key or "core" products that are an essential part of the IS infrastructure. The core products form the basis for delivering quick, custom solutions to the various businesses within IBC. This framework is shown in the diagram below:

![Diagram showing the framework for deployment of core capability]

**Figure 5.2: Framework for Deployment of Core Capability**

In addition to the technology investment and the management system, one other factor that makes a substantial difference in the development and deployment of this core capability: the knowledge and skills present within ISD.

There is a significant emphasis placed on higher education and training of ISD personnel. Recent hires have Masters and often doctoral level education in computer science and business. Current employees are encouraged to
obtain Masters degrees on a part-time basis. In addition to this, there are a wide variety of video and computer-assisted courses on a wide spectrum of subjects, ranging from OO design methods to the fundamentals of bond trading. These are augmented by making Best Practice Notes, journals, reports, training sessions and conference attendance easily accessible. A recent in-house course on OO fundamentals concludes with a university-level examination. Participants who pass the exam get a small (but not insignificant) financial bonus. These efforts are all directed towards forming an IS department that has a high degree of knowledge of both technology and financial services, and motivated to seek new solutions.

While the future still poses many challenges to the IS function, the IBC example demonstrates that it is indeed possible to adjust and thrive in the business and technology environment of the 1990s. It also demonstrates the importance of the role of measurement in transforming software development.
BIBLIOGRAPHY


Concluding Comments
This dissertation research was motivated by the recognition that software
development is an interesting and increasingly important activity in
organizations. It is also predicated on the idea that well constructed and
meaningful metrics can help organizations to plan, evaluate and learn new
software technologies. However, the majority of extant literature on software
metrics is either ad hoc and lacks theoretical rigor, or is theoretical and not
informed by the practical situations that IS managers confront. The agenda
for the dissertation was to attempt to bridge the gap between theory and
practice by proposing metrics that are both theoretically sound and practically
relevant. By bringing together the formalism of ontology, mathematical
evaluation criteria and field work in commercial organizations, this
dissertation claims to have narrowed that gap.

There are several avenues for further research in this area. An immediate
next step would be to analyze data gathered from multiple organizations in
order to determine whether or not the metrics have broad explanatory power.
Also, an examination of the relationship between complexity metrics and
quality indicators is certainly warranted. It would be both interesting and
useful to find out whether the metrics can predict variables such as number
of defects and mean-time between failures. Another possibility might be the
derivation of project-level metrics. There is an urgent need within
organizations for size and resource estimation metrics for object oriented
systems. Using the proposed metrics as a basis, a composite measure for an
entire project can be formulated. This would help IS managers design metric-
based charge-back mechanisms for their services.

It is important to remember that software development is still a new activity
for most organizations, and that there are significant gaps in people's
comprehension of large, commercial software systems. Being an immature field, there is a need for both traditional scientific development and practical organizational solutions. Measurement is a common ground between these two interests. Since building a discipline calls for establishing a common vocabulary of measurement, and IS managers urgently require metrics to manage their software development activities, research in software measurement contributes to the long term development of the field while helping practicing managers achieve their immediate objectives.

However, there are significant risks in misinterpreting metrics research, and IS managers need to exercise caution before using metrics to judge their designers. Metrics are best used as a language for dialog between software professionals and managers. This should be their only logical use for the foreseeable future. It is therefore appropriate to end this dissertation with this cautionary note.

In medieval Europe, the clergy invented and promoted the use of church bells as a time-keeping device so that the laity would be periodically reminded of the Almighty. As the bells tolled on through the decades, this significance of time-keeping fell from its lofty perch. It began to be used primarily to channel and synchronize man's productive capacities at the local farm and factory. Eventually the euphonic tones of stately church bells were replaced by the urgent stridency of factory whistles. An instrument of God had degenerated to become an instrument of Mammon. Likewise, albeit on a smaller scale, it would be a tragedy if this research were used to legitimize the measurement and evaluation of programming staff with the sole intention of increasing the economic output of their labors. This research is not predicated on a narrow vision of software development as a purely economic
activity. Rather, the premise is that software development is a heroic expression of human brilliance and intelligence. The building of meaningful and useful artifacts from the proverbial ether is a sterling example of our ability to experience the divine act of creation. Software is, and should continue to be a labor of love. This dissertation has been aimed to lessen some of the toil of that labor, not to destroy the love.
Appendix A: Overview of Object Oriented Concepts
Outline

1.0 Introduction .................................................................................................................. 164
2.0 Concepts of Object Orientation .................................................................................. 165
  2.1 Basic Object concepts ................................................................................................. 165
  2.2 Essential Elements of OO ......................................................................................... 166
  2.3 Additional Elements of OO ...................................................................................... 167
  2.4 Short history of OO .................................................................................................... 169
    2.4.2 SIMULA ................................................................................................................. 170
    2.4.3 ADA ....................................................................................................................... 170
    2.4.3 The Smalltalk 80 environment .............................................................................. 171
    2.4.4 C++ ......................................................................................................................... 173
3.0 Glossary of OO terminology ....................................................................................... 175
4.0 Bibliography ................................................................................................................ 180
1.0 INTRODUCTION

The Software Engineering Laboratory at NASA recently heralded object technology (OT) as the most promising software technology to date (McGarry and Waligora 1991). OT (a term that includes object oriented programming, object oriented analysis, object oriented design, object oriented databases) is the latest in the series of process technologies that promise to alleviate the problems of cost, schedule and quality in systems delivery. Proponents of OT proclaim that this approach will transform applications development to assembly of standard objects that are available through extensive public, private and commercially available libraries. Not surprisingly, object orientation (OO) has become a buzz-word of the software industry in the 1990s. As witnessed with other new technologies, there seems to be a entire new vocabulary devoted to describing OO terminology. Confusion with the vocabulary is further compounded by the range of meanings for the terms and concepts associated with this technology.

This appendix provides explanations to key concepts in OO, and presents the reader with the basic vocabulary associated with the technology. Section 2 provides an introduction to the concept of objects and classes, explains certain essential and additional elements of object orientation, and outlines a short history of the development of the paradigm. Section 3 provides a glossary of OO terms and concepts adapted from Taylor and Semaphore (Taylor 1990, Semaphore 1993).
2.0 CONCEPTS OF OBJECT ORIENTATION

2.1 Basic Object concepts

The central principle of object orientation is that software should be a representation of the problem space, rather than simply a list of instructions for a computer to follow to solve a given problem. Since a model of the problem space rarely delineates data from the operations performed on the data, OO requires that data and instructions be integrated into a single abstraction called an object. An object is an element that represents an aspect of the problem space, contains information (instance variables) and instructions (methods) that enable the object to respond to requests (messages) from other objects. This is a major departure from traditional programming paradigms that separate instructions (or programs) from data, and manage them independently.

Individual objects are unique since they have a unique names and data values. However several objects can belong to the same class, i.e. they share a common definition of instance variable types and methods. A class is a template or prototype for a collection of objects. For example, car is a class, and my_car, your_car, their_car, john's_car are objects that belong the class car. The class car defines instance variables like price, year, number of occupants, and methods like depreciation, insurance premium that operate on the instance variables. The object my_car will have unique values for price, year, number of occupants and will use the methods depreciation, insurance premium that are defined in car to respond to requests from other objects.

Objects and classes of objects are the basic building blocks of OO software. The power of the OO paradigm is the ease with which an application can be modeled using objects, and the ability to add abstractions incrementally. With the
accumulation of classes and objects over time, new application development can become assembly of pre-existing objects and classes.

2.2 Essential Elements of OO

Recognizing that users may find it difficult to discern the relative importance of different OO concepts, several researchers have suggested essential features for an object oriented system. Unfortunately, these lists are not in agreement\(^1\). This dissertation uses Thomas's list, which includes encapsulation, inheritance, message passing and dynamic binding as the essential features of object orientation (Thomas 1989).

*Encapsulation* refers to the practice of placing the attributes of an object and all valid operations on those attributes in one self-contained package, thereby protecting the state of an object from other objects. The manipulation of the state (i.e. the data values) must be private and of restricted scope. The only way to access the data values is through the invocation of special member functions (or methods) that are part of the object itself.

*Inheritance* allows a programmer to design classes of objects that are specializations of other classes of objects. For example, consider the class of *cars* as a derived class of the super-class of *vehicles*. A car object, in addition to having its own special properties, inherits the properties of the class *vehicles*.

---

\(^1\) Thomas suggested that encapsulation, inheritance, message passing and dynamic binding are the key features in object orientation (Thomas 1989). Pascoe defined four elements that an OOPL must support: information hiding, data abstraction, dynamic binding and inheritance (Pascoe 1986). Loy suggested only three essential features for object orientation: encapsulation, inheritance and classification (Loy 1990). Booch provides a more generic list which includes abstraction, encapsulation, modularity and hierarchy (Booch 1991).
Unlike traditional structured design software, objects do not make function calls. Instead they communicate through messages. The receiver of a message invokes a member function or method in response to it. This mechanism of message passing is another unique feature of OO.

Lastly, dynamic binding refers to the ability to add new objects and classes to the inheritance hierarchy, without having to recompile existing code. With dynamic binding, it becomes possible to invoke operations on an object without realizing the actual type of the object until run-time. Despite the performance loss due to the run-time binding mechanism, Cox, and Thomas insist that this feature is what makes it possible to build truly modular systems, where new components can be added with ease to pre-existing ones (Thomas 1989, Cox 1990).

2.3 Additional Elements of OO

New OO languages (like Objective-C, Eiffel) and improved versions of older languages and environments offer features that go further in terms of reducing the conceptual gap between the programmer and the computer. These are features that would help by removing some of the problems associated with managing and promoting the OO philosophy of abstraction and reuse at every possible level. Specifically, these are over-loading, garbage collection, multiple inheritance and component libraries.

Over-loading refers to the ability to define new implementations for the same function for different classes of objects. For example, the '+' operator can be defined differently for lists, stacks or integers. It would then be possible for adding, subtracting, incrementing complex structures with the same ease as simpler structures like integers (a programmer can write a statement like:

\[ \text{List}_\text{object } A + \text{List}_\text{object } B = \text{List}_\text{object } C \]
Integer C, since the + operator is implemented differently for the class List_object.

Users can invoke traditional operators (e.g. +, -) and functions (e.g. sort, print) without knowing the details about their internals. Overloading also makes it possible for designers of different objects to use names and operators without worrying whether their implementation conflicts with those of other objects.

A major issue in any software design is memory management. In lower level languages (like assembly) the programmer is responsible for the task of acquiring and managing system memory. Higher level languages, provide better facilities for some of the tasks associated with memory management (e.g., C provides the malloc and calloc functions). Yet memory management errors are common during software development. More significantly, these errors are often insidious and can be difficult to trace. Reducing the potential for this, is a significant step in easing the responsibilities placed on programmers. Garbage collection refers to the facility to recover chunks of memory that are no longer in use and make them available to other programs that may need them. If this is done automatically by the language, the programmer can be absolved of this difficult and error-prone responsibility. Certain languages and environments provide this facility (e.g. Smalltalk).

As discussed earlier, one of the major benefits of OO design is the ability for classes of objects to inherit properties (i.e. methods) from a super-class. An advanced feature is allowing for a class to inherit methods from more than one super-class. For example, objects of the class red cars can inherit methods from the class of red painted parts as well as the class of cars. This feature is referred to as multiple inheritance. Multiple inheritance allows the programmer more flexibility to build classes, economy of expression and promotes increased reuse of methods. Stefik and Bobrow suggest that "multiple inheritance allows
increased brevity in specifications by increasing the ability to share descriptions" (Stefik and Bobrow 1986 p 46).

Many OO proponents envision a market place of object components, where users can mix and match objects from different commercial libraries along with their own custom-built ones. While this market place may not manifest itself in the immediate future, the concept of an object library is already a reality. "ADA components" and "C++ Booch components" are available in the commercial marketplace. Providing a library of objects, which contain ready-made implementations for frequently used operations, can be valuable to any project. If the OO environment itself provides this facility, it could potentially decrease the cost (in terms of time and money) of an application. For example, the Smalltalk environment provides a library of over 200 reusable component objects.

While OO has gained increasing popularity recently, this technology has had a significant gestation period. OO concepts were introduced by computer scientists in the early 1970s, and it has roots in the development of programming languages.

2.4 Short history of OO

Shaw suggests that the development of programming paradigms has been largely motivated by the dual concerns for cost (of development and maintenance) and quality of software (Shaw 1984). Techniques that improve abstraction, modularity and reuse have been regarded as successful steps towards addressing these two concerns (Meyer 1988, Booch 1991). OO is the latest step in this continuing quest for improving abstraction, modularity and reuse. Some of the key concepts in OO like objects, classes, messages and
inheritance have its origins in the development of programming languages like SIMULA, ADA, Smalltalk and C++. The next four sub-sections deal briefly with the motivation for and the contributions made by SIMULA, ADA, Smalltalk and C++.

2.4.2 SIMULA

The concept of objects was first introduced in SIMULA in the early 1970s. SIMULA (short for simulation language) is a descendent of ALGOL 60 in the genealogy of computer languages. Like other simulation languages (e.g. GPSS, SIMSCRIPT), it was specifically designed to be able to describe physical systems with greater ease than traditional FORTRAN or LISP. One of the key concepts that SIMULA introduced was that of an object which consists of a header (identifying the object), a data structure for storage and an action sequence to perform operations on the data structure and to interact with other objects. A motivating factor for objects was to reduce the conceptual distance between computer representation and the problem space. Data and program segments were aggregated into objects to better represent the physical world, since "data and actions are very closely related" (Birnstiel et al. 1979, p 25.). Many present day OO languages like Smalltalk, CLOS have borrowed concepts from SIMULA.

2.4.3 ADA

ADA (a standard programming language proposed by the US Department of Defense), represents the idea that through the adoption of a programming language or environment, we can achieve interoperability and modularity and provide a problem solving domain that is more suited to the way human beings think. Booch contends that ADA helps break the "Von-Neuman mind set" and lets us deal with our solutions in our actual problem space (Booch 1986). The
ADA programming standard was a response to the problems caused by the having over 400 different languages on mission critical DOD systems. ADA has no flavors or dialects, since the language standard is strictly enforced by the DOD.

ADA actively supports modularity, abstraction, concurrency and exception handling. As a result, solutions become more readable, reliable and more maintainable. ADA provides a set of rich constructs for describing primitive objects and operations, and in addition offers a package construct with which we may build and enforce our own abstractions. Each package is a collection of computational resources, which may encapsulate data types, data objects, sub-programs, or even other packages. There is a facility called tasks for routing messages, for performing concurrent actions i.e. ADA supports encapsulation and message passing. However ADA does not support the notion of inheritance or dynamic binding.

2.4.3 The Smalltalk 80 environment

Smalltalk 80 is an integrated programming environment, which was a result of nearly a decade of research by the Software Concepts Group at the Palo Alto Research Center (PARC). One of the central concepts of this effort was the development of object oriented programming (Goldberg and Robson 1983). Smalltalk borrows extensively from Common LISP, which was also developed at PARC. Smalltalk is more than a language, it is an entire programming environment. The programming language, supporting tools (text editors, linkers, debuggers), and the operating system itself are an integrated, tightly coupled whole which reside within the same virtual address space. The entire environment is accessible to the programmer, including the operating system.
The programmer can even manipulate the internal representations of objects in the environment, to suit their requirements.

Smalltalk provides an extraordinary amount of power and flexibility to the programmer. This may be perhaps the most complete object-oriented language, since it adheres to the object principle exhaustively. Literally everything is an object - not only large, infrequently used high-level entities like windows, directories, and projects but also low-level entities like integers and stack frames. Even the declaration of the class is an object, which can be accessed through messages.

An object in Smalltalk is private data and a collection of procedures that can access that data. The data is private to the object, and cannot be accessed without help from the object’s procedures. The procedures are public to all consumers, who access the object by writing message expressions. A message expression tells an object what it should do. The object responds to a message by selecting and then performing a procedure whereby all objects of this class perform this message. This procedure is called a method, and the command carried by the message is called a selector. The destination for a message is resolved only at run-time i.e. messages are dynamically bound. Smalltalk also supports inheritance between classes, so that a sub-class can inherit methods from its super-class.

Smalltalk provides automatic garbage collection, which means the lifetime of the object is determined by the system not by the programmer. This is done by the system, when the object is no longer referenced. Automatic garbage collection frequently entails a performance penalty, and Smalltalk applications often require more memory and more processing power than other OO environments.
2.4.4 C++

C++ (developed at AT&T Bell Laboratories) was designed to actively support the principles of object oriented programming, without losing compatibility with C (Stroustrup 1988). This is because of the large body of C language code that AT&T Bell Laboratories had developed for their switching systems at considerable cost. Also, C++ was an attempt to repair some of the long standing flaws of C, particularly its lax treatment of data types. C++ still retains the high machine efficiency and portability that are the hallmarks of C.

C++ extends the struct mechanism (which allows only data members) to allow for declaring classes that have private, protected and public member functions and data. Private data and functions are available only within the class, protected members to all sub-classes, and public members to all possible classes. Thus, C++ provides the ability to ensure compete encapsulation of data. C++ supports message passing by using a mechanism similar to a function call invocation (for example: obj.method() ). The C++ construct of virtual functions allows for dynamic binding. When a virtual member function is invoked, the function essentially passes this message to the appropriate sub-class at run-time. The language allows for sub-classes that can inherit methods from super-classes by a providing a derived class construct. A class can be declared to be derived from one or more (in C++ version 2.0 or greater) classes.

While C++ allows for operator overloading, it does not provide a garbage collection facility. However, C++ allows for objects to be dynamically allocated on heap memory and referred to by memory location in addition to allowing static allocation of space for variables. A constructor function is automatically run on declaration of an object, which can then get space from the heap section of
memory. A destructor operator is automatically run when the object is no longer required, which can free the allocated space. The programmer has to specifically create the constructor and destructor with the appropriate memory managing code and place it along with the class definition. The C++ language does not provide a built-in class libraries, but a number of commercial and public domain C++ libraries are currently available.

Numerous other flavors of OO languages and environments have emerged over the past few years, and the prospective OO adopter has many choices. While this appendix does not contain a comprehensive review of alternatives, it provides a starting point for further exploration of the possibilities afforded by OO. Table 1 provides a comparison of several of the popular OO programming languages.
<table>
<thead>
<tr>
<th><strong>3.0 GLOSSARY OF OO TERMINOLOGY</strong></th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>abstract class</strong></td>
</tr>
<tr>
<td><strong>abstract data type</strong></td>
</tr>
<tr>
<td><strong>argument</strong></td>
</tr>
<tr>
<td><strong>base class</strong></td>
</tr>
<tr>
<td><strong>binding</strong></td>
</tr>
<tr>
<td><strong>browser</strong></td>
</tr>
<tr>
<td><strong>C++</strong></td>
</tr>
<tr>
<td><strong>class</strong></td>
</tr>
<tr>
<td><strong>class hierarchy</strong></td>
</tr>
<tr>
<td><strong>constructor</strong></td>
</tr>
<tr>
<td><strong>data abstraction</strong></td>
</tr>
<tr>
<td>Term</td>
</tr>
<tr>
<td>------------------</td>
</tr>
<tr>
<td>data member</td>
</tr>
<tr>
<td>data type</td>
</tr>
<tr>
<td>delegation</td>
</tr>
<tr>
<td>derived class</td>
</tr>
<tr>
<td>destructor</td>
</tr>
<tr>
<td>dynamic binding</td>
</tr>
<tr>
<td>encapsulation</td>
</tr>
<tr>
<td>garbage collection</td>
</tr>
<tr>
<td>information hiding</td>
</tr>
<tr>
<td>inheritance</td>
</tr>
<tr>
<td>instance</td>
</tr>
<tr>
<td>instance variable</td>
</tr>
<tr>
<td>Term</td>
</tr>
<tr>
<td>--------------------</td>
</tr>
<tr>
<td>interoperability</td>
</tr>
<tr>
<td>member function</td>
</tr>
<tr>
<td>method</td>
</tr>
<tr>
<td>modular programming</td>
</tr>
<tr>
<td>multiple inheritance</td>
</tr>
<tr>
<td>object</td>
</tr>
<tr>
<td>object oriented</td>
</tr>
<tr>
<td>OMG</td>
</tr>
</tbody>
</table>
overloading

The assignment of multiple meanings to the same method name, allowing a single message to perform different functions depending on which object receives it and what parameters accompany it. The language automatically selects the appropriate meaning by noting the recipient of the message and checking the number and types of the parameters. For example, a method `foo` can be overloaded by having several `foo` functions like `foo(parameter_1)`, `foo(parameter_2)` or `foo(parameter_3, parameter_4).

override

A language mechanism where a method for a subclass augments or replaces a method for the superclass.

parameter

A data element that is included in the invocation of a message. Also referred to as arguments.

polymorphism

An ability to refer to different implementations of a feature or function by the same name. For example, a class `document` could have a method called `print`, and another class `spreadsheet` could also have a method called `print`. The specific `print` method that executes during run time depends on whether a `document` or a `spreadsheet` object is currently active.

procedure

A sequence of instructions to a computer.

SIMULA

A computer language developed in the 1960s by the Norwegian Computer Center for simulating real world processes.

single inheritance

This refers to the case when a class is an immediate subclass of only one superclass.

Smalltalk

An OO language developed at Xerox PARC in the 1970s.

static binding

The association of a name with a class that is done at compile time.
<table>
<thead>
<tr>
<th>Term</th>
<th>Definition</th>
</tr>
</thead>
<tbody>
<tr>
<td>structured programming</td>
<td>This refers to the collection of techniques that are designed to increase the rigor of software development. Modularity and functional decomposition are important aspects of structured programming.</td>
</tr>
<tr>
<td>subclass</td>
<td>A class that is often a special class of another class. For example, car is a special case i.e. subclass of vehicle.</td>
</tr>
<tr>
<td>subroutine</td>
<td>A program that is defined as a separate, and used for a special purpose by other programs.</td>
</tr>
<tr>
<td>superclass</td>
<td>A class that is the higher in the class hierarchy that another class. Vehicle is the superclass of car.</td>
</tr>
<tr>
<td>variable</td>
<td>A storage place within an object for a data element. The data element could be a built-in type offered by the language, or it can be a reference to other objects.</td>
</tr>
</tbody>
</table>
4.0 BIBLIOGRAPHY


Table 1: Comparison of languages

<table>
<thead>
<tr>
<th></th>
<th>Encapsulation</th>
<th>Message passing</th>
<th>Inheritance</th>
<th>Dynamic binding</th>
<th>Operator overloading</th>
<th>Garbage collection</th>
<th>Multiple inheritance</th>
<th>Component libraries</th>
</tr>
</thead>
<tbody>
<tr>
<td>Smalltalk 80</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>No</td>
<td>Yes</td>
</tr>
<tr>
<td>C++</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>No</td>
<td>Yes</td>
<td>No</td>
</tr>
<tr>
<td>Ada</td>
<td>Yes</td>
<td>Yes</td>
<td>No</td>
<td>No</td>
<td>No</td>
<td>Yes</td>
<td>No</td>
<td>Yes</td>
</tr>
<tr>
<td>CLOS</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>No</td>
<td>Yes</td>
<td>No</td>
</tr>
<tr>
<td>Objective C</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>No</td>
<td>Yes</td>
<td>No</td>
<td>Yes</td>
</tr>
</tbody>
</table>
Appendix B: Tool Development Software
CLIPC_STATUS clipc_array_addstring A((CLIPC_ARRAY array, char *string));
CLIPC_STATUS clipc_array_addmem A((CLIPC_ARRAY array, int len, char *mem));
CLIPC_STATUS clipc_array_addchar A((CLIPC_ARRAY array, char c));
CLIPC_STATUS clipc_array_addint A((CLIPC_ARRAY array, int num));
CLIPC_STATUS clipc_array_addfloat A((CLIPC_ARRAY array, double num));
CLIPC_STATUS clipc_array_adddict A((CLIPC_ARRAY array, CLIPC_DICT val));
CLIPC_STATUS clipc_array_addarray A((CLIPC_ARRAY array, CLIPC_ARRAY val));
CLIPC_STATUS clipc_array_genadd A((CLIPC_ARRAY array, ...));
CLIPC_STATUS clipc_array_vgenadd A((CLIPC_ARRAY array, va_list pvar));

int clipc_array_size A((CLIPC_ARRAY array));
CLIPC_VALUE clipc_array_getval A((CLIPC_ARRAY array, int elem, CLIPC_STATUS *sp));
char *clipc_array_getstring A((CLIPC_ARRAY array, int elem, CLIPC_STATUS *sp));
char *clipc_array_getmem A((CLIPC_ARRAY array, int elem, int *lenp, CLIPC_STATUS *sp));
char clipc_array_getchar A((CLIPC_ARRAY array, int elem, CLIPC_STATUS *sp));
long clipc_array_getint A((CLIPC_ARRAY array, int elem, CLIPC_STATUS *sp));
double clipc_array_getfloat A((CLIPC_ARRAY array, int elem, CLIPC_STATUS *sp));
CLIPC_DICT clipc_array_getdict A((CLIPC_ARRAY array, int elem, CLIPC_STATUS *sp));
CLIPC_ARRAY clipc_array_getarray A((CLIPC_ARRAY array, int elem, CLIPC_STATUS *sp));

/*
 * Values
 */
CLIPC_VALUE clipc_new_string A((CLIPC_MSG msg, char *str));
CLIPC_VALUE clipc_new_mem A((CLIPC_MSG msg, int len, char *mem));
CLIPC_VALUE clipc_new_char A((CLIPC_MSG msg, char num));
CLIPC_VALUE clipc_new_int A((CLIPC_MSG msg, int num));
CLIPC_VALUE clipc_new_float A((CLIPC_MSG msg, double num));
CLIPC_VALUE clipc_new_array A((CLIPC_MSG msg, CLIPC_ARRAY array));
CLIPC_VALUE clipc_new_dict A((CLIPC_MSG msg, CLIPC_DICT dict));
CLIPC_STATUS clipc_value_free A((CLIPC_VALUE value));
CLIPC_VALUE clipc_value_dup A((CLIPC_MSG msg, CLIPC_VALUE value));
CLIPC_VALUE clipc_value_new A((CLIPC_MSG msg, CLIPC_TYPE type));

CLIPC_TYPE clipc_val_gettype A((CLIPC_VALUE value, CLIPC_STATUS *sp));
char *clipc_val_getstring A((CLIPC_VALUE value, CLIPC_STATUS *sp));
char *clipc_val_getmem A((CLIPC_VALUE value, int *lenp, CLIPC_STATUS *sp));
char clipc_val_getchar A((CLIPC_VALUE value, CLIPC_STATUS *sp));
long clipc_val_getint A((CLIPC_VALUE value, CLIPC_STATUS *sp));
double clipc_val_getfloat A((CLIPC_VALUE value, CLIPC_STATUS *sp));
CLIPC_DICT clipc_val_getdict A((CLIPC_VALUE value, CLIPC_STATUS *sp));
CLIPC_ARRAY clipc_val_getarray A((CLIPC_VALUE value, CLIPC_STATUS *sp));

/*
 * Message server
 */

CLIPC_CONN init_clipc_connection A((CLIPC_STATUS *sp));
CLIPC_STATUS close_clipc_connection A((CLIPC_CONN conn));
CLIPC_STATUS clipc_set_connection_type A((CLIPC_CONN conn, CLIPC_TYPE type));
CLIPC_STATUS clipc_listen_all A((CLIPC_CONN conn));
CLIPC_STATUS register_clipc_handler A((CLIPC_CONN conn, char *name));
CLIPC_STATUS register_clipc_listener A((CLIPC_CONN conn, char *name,
int flags));
CLIPC_STATUS register_clipc_synclistener A((CLIPC_CONN conn, char *name));
int clipc_check_for_handler A((CLIPC_CONN conn, char *name, int msg_id, 
   CLIPC_STATUS *sp));
CLIPC_STATUS unregister_clipc_handler A((CLIPC_CONN conn, char *name));
CLIPC_STATUS unregister_clipc_listener A((CLIPC_CONN conn, char *name));
CLIPC_STATUS unregister_clipc_synclistener A((CLIPC_CONN conn, char *name));

/**
 * CLIPC connection
 */
CLIPC_CONN clipc_conn_alloc A((int fd, CLIPC_STATUS *sp));
void clipc_conn_free A((CLIPC_CONN conn, int *free_fd));
void clipc_conn_setfds A((CLIPC_CONN conn, int new_fd));
int clipc_conn_getfd A((CLIPC_CONN conn));
char *clipc_conn_getobuf A((CLIPC_CONN conn));
CLIPC_CONN clipc_conn_findbyfd A((int fd, CLIPC_STATUS *sp));

/**
 * QUEUE support
 */
/**
 * QUEUE system
 */
CLIPC_STATUS clipc_msgqueue_enq A((CLIPC_MSGQUEUE *qp, CLIPC_MSG value, 
   int key));
CLIPC_STATUS clipc_msgqueue_pushq A((CLIPC_MSGQUEUE *qp, CLIPC_MSG value, 
   int key));
CLIPC_MSG clipc_msgqueue_deq A((CLIPC_MSGQUEUE *qp, CLIPC_MSG value, 
   int key));
CLIPC_MSGQUEUE clipc_msgqueue_concat A((CLIPC_MSGQUEUE *qp, 
   CLIPC_MSGQUEUE tail));
CLIPC_MSGQUEUE clipc_msgqueue_copy A((CLIPC_MSGQUEUE qp));
void clipc_msgqueue_free A((CLIPC_MSGQUEUE *qp));
void clipc_msgqueue_rmmag A((CLIPC_MSGQUEUE *qp, CLIPC_MSG value, int key));
CLIPC_MSGQUEUE clipc_msgqueue_next A((CLIPC_MSGQUEUE qp));
CLIPC_MSG clipc_msgqueue_getmsg A((CLIPC_MSGQUEUE qp));

/**
 * misc functions (should be private, but for now, they are useful)
 */
/*
 * int clipc_unimp A((char *str, ...)); */
/*
 * void clipc_error A((char *str, ...)); */
void print_msg A((CLIPC_MSG msg));
void print_val A((CLIPC_VALUE val, char *indent));

#if defined(__cplusplus)
}    /* matches extern "C" ( at start of file */
#endif
#endif}

#endif

#endif

#endif
main()
{
    // Declare variables and functions
    CLIPC_CONN clipc_conn = 0;
    CLIPC_MSG msg, reply_msg;
    clipc_status status;
    CLIPC_ARRAY ret_array;
    int num_of_classes = 0;
    extern void crlf();
    extern CLIPC_CONN init_connection();
    extern void check_status(clipc_status status);
    extern void get_wmc_noc_dic(CLIPC_CONN clipc_conn, CLIPC_ARRAY class_id_array, int array_index,

    FILE *output_file;
    FILE *command_file_ptr;
    FILE *whatis_ptr;
    FILE *lcom_ptr;

    // open up a results file for writing output
    output_file = fopen("results", "w");
    // open command file for writing output
    command_file_ptr = fopen("command_file", "w");
    whatis_ptr = fopen("class", "w");
    lcom_ptr = fopen("last_metric", "w");
    // set up CLIPC connection
    clipc_conn = init_connection();

    // try to get space for a message
    msg = clipc_request_alloc("cxcmd_list_classes", 0, &status);
    check_status(status);
    // call the check routine to print msgs.
    // send out the message
    reply_msg = clipc_request_send(clipc_conn, msg, &status);
    check_status(status);
    // call the check routine to print msgs.

    ret_array = clipc_msg_getarray(reply_msg, "ClassIdList", &status);
    check_status(status);
    // call the check routine to print msgs.

    // find out how many classes are there
    num_of_classes = clipc_array_size(ret_array);
    crlf();
    printf("THIS CLASS LIBRARY CONTAINS %d CLASSES\n", num_of_classes);
    crlf();
}
for (int j = 0; j < num_of_classes; j++)
{
    fprintf(output_file, "\n%6d ", j+1);
    fflush(output_file);
    fprintf(lcom_ptr, "\n%6d ", j+1);
    fflush(lcom_ptr);

    get_wmc_non_dit(clipc_conn, ret_array, j, output_file.command_file_ptr, what);
    crlf();
    crlf();

    printf("Finished metrics for Class Number %6d\n", j+1);
    crlf();
    crlf();
}
    crlf();
    crlf();
    printf("Moose Run Complete\n");
    crlf();
    crlf();
}
//end of main

```c
#include <stdio.h>
#include "clipc.h"
#include "metrics.h"
#include <string.h>

#define LEN 50

int
bigger (int num1, int num2)
{
    if (num1 > num2) return (num1);
    else return(num2);
}

static int count, max;

int
get_parent(CLIPC_CONN clipc_conn, int base_id)
{
    clipc_status status;
    CLIPC_MSG msg2, reply_msg2;
    CLIPC_ARRAY ret_array2;
    extern int bigger(int num1, int num2);
    ++count;
    msg2 = clipc_request_alloc("cxcmd_browse_class", 0, &status);
    // add to message, since browse class requires params
    clipc_msg_addint(msg2, "ClassId", base_id);
    clipc_msg_addint(msg2, "Limit", 0);
    clipc_msg_addint(msg2, "Show", 1);
    reply_msg2 = clipc_request_send(clipc_conn, msg2, &status);
    ret_array2 = clipc_msg_getarray(reply_msg2, "BaseClassList", &status);
    int num_of_parents = 0;
    num_of_parents = clipc_array_size(ret_array2);

    if (num_of_parents != 0) {
        CLIPC_DICT ClassDict = clipc_array_getdict(ret_array2, 0, &status);

        int parent_id = clipc_dict_getint(ClassDict, "ClassId", &status);
        clipc_msg_free(msg2);
        clipc_msg_free(reply_msg2);
        int depth = get_parent(clipc_conn, parent_id);
    }
}
```
void
get_wmc_noc_dic(CLIPC_CONN clipc_conn, CLIPC_ARRAY class_id_array, int array_index, FILE *output_file, FILE *command_file_ptr, FILE *whatis_ptr, FILE *1com_file)
{

  clipc_status status;
  int classid;
  CLIPC_MSG msg2, reply_msg2;
  CLIPC_ARRAY ret_array2;
  FILE *f_func_ptr;
  FILE *f_data_ptr;
  char buf_func[LEN];
  char buf_data[LEN];
  char buffer_tmp[LEN];

  extern void check_status(clipc_status status);
  extern void crif(), fcrlf(FILE *output_file);
  extern int get_parent(CLIPC_CONN clipc_conn, int base_id);

  char *format_data = "%s.data";
  char *format_func = "%s.func";

  // try to get the name of the classes and the class ids

  CLIPC_DICT ClassDict = clipc_array_getdict(class_id_array, array_index, &status);
  check_status(status);

  classid = clipc_dict_getint(ClassDict, "ClassId", &status);
  check_status(status);
if (status == CLIPC_STATUS_OK) printf("\n\r GOT CLASS ID INFO\n");

char * ClassName = clipc_dict_getstring(ClassDict, "ClassName", &status);
check_status(status);

sprintf(buf_data, format_data, ClassName);
sprintf(buf_func, format_func, ClassName);

f_data_ptr = fopen(buf_data, "w");
f_func_ptr = fopen(buf_func, "w");

fwrite(whatie_ptr, strlen(ClassName), 1, f_data_ptr);
flush(whatie_ptr);
fwrite(whatie_ptr, strlen(ClassName), 1, f_func_ptr);
flush(output_file);

// try to get browse class information

msg2 = clipc_request_alloc("cmd_browse_class", 0, &status);
if (status != CLIPC_STATUS_OK) printf("\n\r COULD NOT ALLOCATE SPACE FOR BROWSE\n");
else
{
    // add to message, since browse class requires params

    clipc_msg_addint(msg2, "ClassId", clasic);
    clipc_msg_addint(msg2, "Limit", 0);
    clipc_msg_addint(msg2, "Show", 1);
    reply_msg2 = clipc_request_send(clicp_conn, msg2, &status);
    if (status != CLIPC_STATUS_OK) printf("\n\r COULD NOT SEND BROWSE MESSAGE \n");

    ret_array2 = clipc_msg_getarray(reply_msg2, "MemberFuncList", &status);
    int num_of_funcs = 0;
    num_of_funcs = clipc_array_size ( ret_array2);
    if (num_of_funcs > 0)
    {
        for (int j=0; j < num_of_funcs; j++)
        {
            CLIPC_DICT ClassDict = cl_ipc_array_getdict(ret_array2, j)
            check_status(status);
            char * Member_Func_Name = clipc_dict_getstring(ClassDict
            check_status(status);

            buffer_tmp[0] = '+';
            strcat(buffer_tmp, ClassName);
            buffer_tmp[strlen(ClassName)] = '0';
            if (strcmp(Member_Func_Name, "cmd") == 0)
                strcpy(Member_Func_Name, "clasic".Name);
            if (strcmp(Member_Func_Name, "clanic") == 0)
                strcpy(Member_Func_Name, buffer_tmp);

            printf(command_file_ptr, "sh echo -s:|s >> Or source\n", ClassName, Member_Func_Name. Name);
fprintf(command_file_ptr, "%s::%s() >> OC_source\n", ClassName, Member_Func_Name);
fflush(command_file_ptr);

if (Member_Func_Name[0] == '-') fprintf(f_func_ptr, "\%\-%30s\n", Member_Func_Name);
else fprintf(f_func_ptr, "%30s\n", Member_Func_Name);

} // Now take care of classes with zero member functions

if (num_of_funcs == 0)
{

fprintf(command_file_ptr, "sh echo -%s::dummy >> OC_source\n", ClassName);
fprintf(command_file_ptr, "%s::%s() >> OC_source\n", ClassName);
fflush(command_file_ptr);

}

// Now for LCOM stuff

ret_array2 = clipc_msg_getarray(reply_msg2, "DataMemberList", &status);
int num_of_mems = 0;
num_of_mems = clipc_array_size (ret_array2);
fprintf(lcom_file, "%s::%s\n", ClassName);
fprintf(lcom_file, "%s::%s\n", ClassName);
fflush(lcom_file);

for (int k=0; k< num_of_mems; k++)
{

CLIPC_DICT ClassDict = clipc_array_getdict(ret_array2, k
check_status(status);
char *Data_Member_Name = clipc_dict_getstring(ClassDict
check_status(status);

//
//
//
//
fflush(lcom_file);
fflush(f_data_ptr, "%d\n", Data_Member_Name);
fflush(f_data_ptr);

}

// End of LCOM stuff

ret_array2 = clipc_msg_getarray(reply_msg2, "DerivedClassList", &status);

int num_of_children = 0;
num_of_children = clipc_array_size (ret_array2);
crlf();
fprintf(output_file, "%d", num_of_children);
fflush(output_file);
ret_array2 = clipc_msg_getarray(reply_msg2, "BaseClassList", &status);
int num_of_parents = 0;
num_of_parents = clipc_array_size(ret_array2);
//
// fprintf(output_file,"NOP=%d\n", num_of_parents);
//
// fflush(output_file);
max = 0;
// reset static variables
count = 0;
if (num_of_parents == 0) fprintf(output_file," 0\n",);
for (int i = 0; i < num_of_parents; i++)
{
    CLIPC_DICT ClassDict = clipc_array_getdict(ret_array2, i, &status);
    check_status(status);
    int base_id = clipc_dict_getint(ClassDict, "ClassId", &status);
    check_status(status);
    //
    // fprintf(output_file,"parent Class ID is: %d\n", base_id);
    //
    // fflush(output_file);
    int depth = 0;
    depth = get_parent(clipc_conn, base_id);
    fprintf(output_file," %d\n", depth);
    fflush(output_file);
}
}
fclose(f_func_ptr);
fclose(f_data_ptr);
// end of get_wmc_noc_dict()
#include <stdio.h>
#include "clipc.h"
#include "metrics.h"

CLIPC_CONN
init_connection()
{
    // start of init_connection

    CLIPC_CONN clipc_conn;
    extern void crlf();

    crlf();
    printf("hello world\n");
    crlf();

    // Init the connection to the CLIPC SERVER
    clipc_conn = init_clipc_connection(0);

    if ( clipc_conn == 0)
    {
        printf(" CLIPC CONNECTION FAILED\n");
    }
    else
    {
        printf("CLIPC CONNECTED !!!\n");
    }

    return(clipc_conn);

} // end of init_connection

void
check_status(clipc_status status)
{

    extern void crlf();

    if (status != CLIPC_STATUS_OK)
    {
        crlf();
        printf ("STATUS VOILATION\n");
        printf("Status is %d\n", status);
        crlf();
    }
    else printf("\n\rSTATUS IS OK\n");

} //end of check_status
#include <stdio.h>
#include <errno.h>
#include <string.h>

#define MAXLINE 150
#define BLANK ' '

int
is_same_class(char *Name, char *Basis)
{
    int len;
    char temp[MAXLINE+2];

    // set the first char to - to compare properly
temp[0] = '-';
temp[1] = '\0'; // null terminate the string
strcat(temp, Name);

    len = strlen (Basis); // find the length of the string

    if ((strcmp(temp, Basis, len-1) == 0) &&
        (temp[len] == ':')) return (0);
    else return (-1);
}

main()
{

    FILE *in_file, *out_file;
    char *ClassName;
    char buffer[MAXLINE+1];
    char name_buffer[MAXLINE+1];
    int count, count2;
    extern int is_same_class (char *Name, char *Basis);

    // open files for reading and writing
    if ((in_file = fopen("results.out", "r")) == NULL)
        perror("fopen");
    out_file = fopen("metrics_out", "w");

    // clear the buffers
    for (int j = 0; j < MAXLINE + 1; j++)
    {
        buffer[j] = '\0';
        name_buffer[j] = '\0';
    }

    // set pointer to array
    ClassName = name_buffer;

    fgets(ClassName, MAXLINE, in_file);

    // set the metrics counts to zero
    int class_count = 1;
    count2 = 0;
    count = 0;
    // read till end of file
while ( fgets(buffer, MAXLINE, in_file))
{

    // if NOT a class declaration
    if ( buffer[0] == ' -')
    {
        count++;// add to RFC count
        // add to CBO count only for outside classes
        if ( is_same_class(buffer, ClassName) != 0) count2++;
    }

    // if its a new class line, reset counts and print
    if ( buffer[0] == ' -'&& strcmp(buffer, ClassName) != 0 )
    {
        ClassName[0] = BLANK;
        ClassName[strlen(ClassName)-1] = '\0';
        fprintf(out_file,"%-5d Class: %-30s RFC = %d CBO = %d
", class_count++,
                count = 0;
                count2 = 0;
                strcpy(ClassName, buffer);// keep latest class name
    }

    // print out the metrics for the last class
    ClassName[0] = BLANK;
    ClassName[strlen(ClassName)-1] = '\0';
    fprintf(out_file,"%-5d Class: %-30s RFC = %d CBO = %d\n", class_count,
            count = 0;
            count2 = 0;
            strcpy(ClassName, buffer); // keep latest class name
}

}
main()
{
    FILE *file1_ptr;
    FILE *file2_ptr;
    FILE *file3_ptr;
    FILE *joint_file_ptr;

    int ObsNum, ObsNum2, m1, m2, m3, m4, m5, m6, f_m6;
    char ClassName[LEN], ClassName2[LEN], ClassName3[LEN];
    char junkname1[LEN], junkname2[LEN], junkstr[LEN];
    char j_char[3], j_char2[3];
    int flag;

    file1_ptr = fopen("iv_results", "r");
    file2_ptr = fopen("iv cbo_results", "r");
    file3_ptr = fopen("iv lcm_results", "r");
    joint_file_ptr = fopen("iv six_results", "w");

    printf("Files have been opened...
");

    flag = 0;

    while ((fscanf(file1_ptr, "%d%3d%3d%3d
",
        &ObsNum, ClassName, &m1, &m2, &m3)) != EOF)
    {
        printf("looping...
");

        printf("%5d%30s%5d%5d%5d
",
            ObsNum, ClassName, m1, m2, m3);

        fscanf(file2_ptr, "%d%3s%d%3s%d%3d
",
            &ObsNum2, junkstr, ClassName2, j_char1, j_char2, &m4, junkname2, j_char2);

        printf("%5d%30s%5d%5d%5d
",
            ObsNum, ClassName2, m4, m5);

        if (flag == 0) fscanf(file3_ptr, "%d
", ClassName3, &m6);
        if (strcmp(ClassName2, ClassName3) != 0)
        {
            flag = 1;
            f_m6 = 0;
        }
        else
        {
            flag = 0;
            f_m6 = m6;
        }

        if ((ObsNum == ObsNum2) &&
            ((strcmp (ClassName, ClassName2) == 0)))
            fprintf(joint_file_ptr,
                "%5d%30s%5d%5d%5d%5d%5d%5d%5d%5d%5d%5d%5d%5d%5d%5d%5d%5d%5d%5d%5d%5d
",
                ObsNum, ClassName, m1, m2, m3,
                m4 + m1, m5, f_m6);
        else printf("ERROR - Big time, dude!
");
    }
}
#!/bin/csh -f
foreach class ( 'cat class' )
   #echo $class
   foreach func ( 'cat $class.func' )
      #echo $class $func
      foreach data ( 'cat $class.data' )
         rm -rf junk.out
touch junk.out

   $a.out $class $func $data *.c
endif ($status == 0) then
   # sed -e "/$class\:\:\:\:func\.*(/,\^)/\/{
   # sed -e "$/$class\:\:\:func\.*(/,\^)/\{/{
   #}*$data/w junk.out\n   }" *.c > /dev/null
endif $class.c > /dev/null
   echo -n $class $func $data " 
   wc -l junk.out
rm -rf junk.out
end
struct data_list {
    char* data;
    struct data_list* next;
};

struct mem_list {
    char* mem;
    struct mem_list* next;
    struct data_list* dl;
};

#if 0
class mem data num junk
cl  f1  d1  1  junk.out
cl  f1  d2  2  junk.out
cl  f2  d1  0  junk.out
cl  f2  d2  0  junk.out
cl  f3  d1  1  junk.out
cl  f3  d2  0  junk.out

while ( cl ) {
    do something here.
    struct mem_list* mp = malloc(sizeof (struct mem_list *));
    while ( f1 ) {
        mp->mem = malloc(strlen(mem)+1);
        strcpy(mp->mem, mem);

        struct data_list* dp = malloc(sizeof(struct data_list*));
        dp->data = malloc(strlen(data)+1);
        strcpy(dp->data, data);
        dp->next=0;

        mp->dl = dp;

        continue reading next record as long as "f1"
        struct data_list* dp = malloc(sizeof(struct data_list*));
        dp->data = malloc(strlen(data)+1);
        strcpy(dp->data, data);
        dp->next=0;

    #endif
}
struct mem_list* create_list(struct mem_list* major, char* mem, char* data)
int process(struct mem_list* major, char* class_name);
void get_lcom();

void get_lcom()
{
    FILE* fp = fopen("sample.in", "rw");
    char class[80], mem[80], data[80], junk[112];
    int num;
    static char current_class[80] = "c1000";
    struct mem_list* current = 0;
    struct mem_list* major = current;
    while (fscanf(fp, "%s %s %s %d
", class, mem, data, &num, junk) != EOF)
        if (strcmp(current_class, class) == 0)
            if (num == 0)
                current = create_list(current, mem, data);
        else
            process(major, current_class);
            strcpy(current_class, class);
            current = 0;
            if (num == 0)
                current = create_list(current, mem, data);
            else
                current = (struct mem_list*) malloc(sizeof(struct mem_list*);
                current->next = 0;
                current->dl = 0;
                current->mem = (char*) malloc(strlen(mem)+1);
                strcpy(current->mem, mem);
        major = current;

    if (current == 0)
        process(major, class);

    struct mem_list* create_list(struct mem_list* current, char* mem, char* data)
    {
        struct mem_list* mp;
        struct data_list* dp;
        int junk = 0;
        if (current == 0 || strcmp(current->mem, mem) != 0)
            mp = (struct mem_list*) malloc(sizeof(struct mem_list*);
            mp->next = 0;
            mp->dl = 0;
            mp->mem = (char*) malloc(strlen(mem)+1);
            strcpy(mp->mem, mem);

            dp = (struct data_list*) malloc(sizeof(struct data_list*);
            dp->data = (char*) malloc(strlen(data)+1);
            strcpy(dp->data, data);
            dp->next = 0;
            mp->dl = dp;
            if (current)
                current->next = mp;
            current = mp;
        else
            mp = current;
            dp = mp->dl;
            if (dp == 0) /* special case for first element */
struct mem_list* create_list(struct mem_list* major, char* mem, char* data);
int process(struct mem_list* major, char* class_name);
void get_lcom();

void get_lcom()
{
    FILE* fp = fopen("sample.in", "rw");
    char class[80], mem[80], data[80], junk[132];
    int num;
    static char current_class[80] = "c1000";
    struct mem_list* current = 0;
    struct mem_list* major = current;
    while (fscanf(fp, "%s %s %s %s\n", class, mem, data, &num, junk) != EOF)
    {
        if (strcmp(current_class, class) == 0)
        {
            if (num != 0)
            {
                current = create_list(current, mem, data);
            }
        }
    }
    process(major, current_class);
    strcpy(current_class, class);
    current = 0;
    if (num != 0)
    {
        current = create_list(current, mem, data);
    }
    else
    {
        current = (struct mem_list*) malloc(sizeof(struct mem_list));
        current->next = 0;
        current->dl = 0;
        current->mem = (char*) malloc(strlen(mem) + 1);
        strcpy(current->mem, mem);
    }
    major = current;
}

if (current != 0)
    process(major, class);

struct mem_list* create_list(struct mem_list* current, char* mem, char* data)
{
    struct mem_list* mp;
    struct data_list* dp;
    int junk = 0;
    if (current == 0 || strcmp(current->mem, mem) != 0)
    {
        mp = (struct mem_list*) malloc(sizeof(struct mem_list));
        mp->next = 0;
        mp->dl = 0;
        mp->mem = (char*) malloc(strlen(mem) + 1);
        strcpy(mp->mem, mem);
        dp = (struct data_list*) malloc(sizeof(struct data_list));
        dp->data = (char*) malloc(strlen(data) + 1);
        strcpy(dp->data, data);
        dp->next = 0;
        mp->dl = dp;
        if (current)
            current->next = mp;
        current = mp;
    }
    else
    {
        mp = current;
        dp = mp->dl;
        if (dp == 0) /* special case for first element */

199
junk = 1;
while(dp)
    dp = dp->next;
    dp->next = malloc(sizeof(struct data_list));
    dp->next->data = (char*) malloc(strlen(data)+1);
    strcpy(dp->next->data, data);
    dp->next = 0;
if ( jkun )
    mp->d1 = dp;
*/
while(dp & & dp->next)
    dp = dp->next;
    dp->next = malloc(sizeof(struct data_list));
    dp->next->data = malloc(strlen(data)+1);
    strcpy(dp->next->data, data);
    dp->next = 0;
*/
return current ;
}
int process(struct mem_list* major, char * class_name ){
    struct data_list *dll ,*d12 ;
    struct mem_list *ml1 ,*ml2 ;
    int num, func_count, n_z_count;

    func_count = 0;
    n_z_count = 0;
    ml1 = major;
    if ( ml1 == 0 || ml1->next == 0 ) {
        if (strcmp(class_name, "c1000") != 0) printf("%s 30s 0\n", class_name);
        return 0;
    }
    while ( ml1 && ml1->next ) {
        ml2 = ml1->next;
        while ( ml2 ) {
            /*
            printf("In class %s function: %s intersection %s is %d
", class_n,
            func_count ++;
            if (num != 0)n_z_count ++;
            ml2 = ml2->next;
            }*/
            ml1 = ml1->next;
        }
        printf("%s-30s %5d\n", class_name, (func_count-n_z_count));
    }
int compare_data(struct mem_list* ml1, struct mem_list *ml2)(
    struct data_list *dll ,*d12 ;
    dll = ml1->dll; 
    d12 = ml2->dll;
    while(dll) {
        while(dll2) {
            if ( strcmp(dll->data, dll2->data) == 0 )
                return 1;
            else
                d12 = d12->next;
        } /*
        d11 = dll1->next;
    }
    return 0;
}

main()
{
FILE * out_file;

out_file = fopen("results.lcom", "rw");

get_lcom();

}
Appendix C: Regression Analysis
Variable Definitions:

SIZEL: Size of class in lines of code

SIZEB: Number of kbytes of the class definition file

PROD: Productivity measured as Size/work days

VERS: Version number of the system

BA_SIZE: Size of the batch of classes designed in the same time period

CODER_2, CODER_3, CODER_4, CODER_5: Dummy variables for the programmer/designer

Data Sets:

There are three data sets for this system (TPM):

Design and Maintenance (all coders work included)

Design of new classes (all coders work included)

Coder (this data set comprises of the both the design and maintenance work done by the most prolific coder)
PEARSON CORRELATION MATRIX

<table>
<thead>
<tr>
<th></th>
<th>SIZEL</th>
<th>PROD</th>
<th>Coder_2</th>
<th>Coder_3</th>
<th>Coder_4</th>
</tr>
</thead>
<tbody>
<tr>
<td>SIZEL</td>
<td>1.00</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>PROD</td>
<td>0.76</td>
<td>1.00</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Coder_2</td>
<td>0.30</td>
<td>0.07</td>
<td>1.00</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Coder_3</td>
<td>0.09</td>
<td>-0.04</td>
<td>-0.06</td>
<td>-0.03</td>
<td>1.00</td>
</tr>
<tr>
<td>Coder_4</td>
<td>-0.12</td>
<td>-0.06</td>
<td>-0.09</td>
<td>-0.03</td>
<td></td>
</tr>
<tr>
<td>Coder_5</td>
<td>0.27</td>
<td>0.55</td>
<td>-0.11</td>
<td>-0.04</td>
<td>-0.06</td>
</tr>
<tr>
<td>Coder_6</td>
<td>-0.09</td>
<td>-0.09</td>
<td>-0.06</td>
<td>-0.02</td>
<td>-0.03</td>
</tr>
<tr>
<td>BA_SIZ</td>
<td>-0.24</td>
<td>-0.20</td>
<td>-0.31</td>
<td>-0.17</td>
<td>-0.24</td>
</tr>
<tr>
<td>LCOM</td>
<td>-0.15</td>
<td>-0.18</td>
<td>-0.24</td>
<td>0.15</td>
<td>0.30</td>
</tr>
<tr>
<td>CBO</td>
<td>0.12</td>
<td>-0.11</td>
<td>0.22</td>
<td>-0.02</td>
<td>0.05</td>
</tr>
<tr>
<td>RFC</td>
<td>0.16</td>
<td>-0.07</td>
<td>0.28</td>
<td>-0.03</td>
<td>0.07</td>
</tr>
<tr>
<td>NOC</td>
<td>-0.16</td>
<td>-0.12</td>
<td>-0.05</td>
<td>-0.06</td>
<td>0.22</td>
</tr>
<tr>
<td>DIT</td>
<td>-0.01</td>
<td>0.06</td>
<td>0.15</td>
<td>-0.03</td>
<td>0.15</td>
</tr>
<tr>
<td>WMC</td>
<td>0.11</td>
<td>-0.08</td>
<td>0.25</td>
<td>-0.04</td>
<td>0.09</td>
</tr>
<tr>
<td>SIZEB</td>
<td>0.89</td>
<td>0.68</td>
<td>0.26</td>
<td>0.06</td>
<td>-0.08</td>
</tr>
<tr>
<td>VERSN</td>
<td>-0.45</td>
<td>-0.34</td>
<td>-0.35</td>
<td>0.17</td>
<td>0.17</td>
</tr>
<tr>
<td>D_MAINT</td>
<td>-0.56</td>
<td>-0.43</td>
<td>-0.17</td>
<td>-0.15</td>
<td>0.10</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th></th>
<th>Coder_5</th>
<th>Coder_6</th>
<th>BA_SIZ</th>
<th>LCOM</th>
<th>CBO</th>
</tr>
</thead>
<tbody>
<tr>
<td>Coder_5</td>
<td>1.00</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Coder_6</td>
<td>-0.04</td>
<td>1.00</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>BA_SIZ</td>
<td>-0.09</td>
<td>-0.17</td>
<td>1.00</td>
<td></td>
<td></td>
</tr>
<tr>
<td>LCOM</td>
<td>-0.15</td>
<td>0.06</td>
<td>0.08</td>
<td>1.00</td>
<td></td>
</tr>
<tr>
<td>CBO</td>
<td>-0.11</td>
<td>-0.02</td>
<td>-0.10</td>
<td>0.03</td>
<td>1.00</td>
</tr>
<tr>
<td>RFC</td>
<td>-0.17</td>
<td>-0.11</td>
<td>-0.06</td>
<td>0.12</td>
<td>0.81</td>
</tr>
<tr>
<td>NOC</td>
<td>-0.11</td>
<td>-0.06</td>
<td>0.12</td>
<td>0.25</td>
<td>0.16</td>
</tr>
<tr>
<td>DIT</td>
<td>-0.05</td>
<td>-0.03</td>
<td>-0.15</td>
<td>0.06</td>
<td>-0.00</td>
</tr>
<tr>
<td>WMC</td>
<td>-0.19</td>
<td>-0.11</td>
<td>-0.01</td>
<td>0.23</td>
<td>0.62</td>
</tr>
<tr>
<td>SIZEB</td>
<td>0.28</td>
<td>-0.09</td>
<td>-0.24</td>
<td>-0.25</td>
<td>0.13</td>
</tr>
<tr>
<td>VERSN</td>
<td>-0.21</td>
<td>0.17</td>
<td>0.27</td>
<td>0.28</td>
<td>-0.04</td>
</tr>
<tr>
<td>D_MAINT</td>
<td>-0.10</td>
<td>0.14</td>
<td>0.30</td>
<td>0.37</td>
<td>0.20</td>
</tr>
<tr>
<td></td>
<td>RFC</td>
<td>NOC</td>
<td>DIT</td>
<td>WMC</td>
<td>SIZEB</td>
</tr>
<tr>
<td>------</td>
<td>-----</td>
<td>-----</td>
<td>-----</td>
<td>-----</td>
<td>-------</td>
</tr>
<tr>
<td>RFC</td>
<td>1.00</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>NOC</td>
<td>0.48</td>
<td>1.00</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>DIT</td>
<td>0.04</td>
<td>0.07</td>
<td>1.00</td>
<td></td>
<td></td>
</tr>
<tr>
<td>WMC</td>
<td>0.94</td>
<td>0.59</td>
<td>0.08</td>
<td>1.00</td>
<td></td>
</tr>
<tr>
<td>SIZEB</td>
<td>0.14</td>
<td>-0.20</td>
<td>-0.05</td>
<td>0.08</td>
<td>1.00</td>
</tr>
<tr>
<td>VERSN</td>
<td>-0.04</td>
<td>0.23</td>
<td>-0.09</td>
<td>0.01</td>
<td>-0.43</td>
</tr>
<tr>
<td>D_MAINT</td>
<td>0.18</td>
<td>0.27</td>
<td>0.03</td>
<td>0.21</td>
<td>-0.64</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th></th>
<th>VERSN</th>
<th>D_MAINT</th>
</tr>
</thead>
<tbody>
<tr>
<td>VERSN</td>
<td>1.00</td>
<td></td>
</tr>
<tr>
<td>D_MAINT</td>
<td>0.40</td>
<td>1.00</td>
</tr>
</tbody>
</table>

BARTLETT CHI-SQUARE STATISTIC: 932.626 DF= 136 PROB= 0.000

MATRIX OF BONFERRONI PROBABILITIES

<table>
<thead>
<tr>
<th></th>
<th>SIZEL</th>
<th>PROD</th>
<th>CODER_2</th>
<th>CODER_3</th>
<th>CODER_4</th>
</tr>
</thead>
<tbody>
<tr>
<td>SIZEL</td>
<td>0.00</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>PROD</td>
<td>0.00</td>
<td>0.00</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>CODER_2</td>
<td>0.46</td>
<td>1.00</td>
<td>0.00</td>
<td></td>
<td></td>
</tr>
<tr>
<td>CODER_3</td>
<td>1.00</td>
<td>1.00</td>
<td>1.00</td>
<td>0.00</td>
<td></td>
</tr>
<tr>
<td>CODER_4</td>
<td>1.00</td>
<td>1.00</td>
<td>1.00</td>
<td>1.00</td>
<td>0.00</td>
</tr>
<tr>
<td>CODER_5</td>
<td>1.00</td>
<td>0.00</td>
<td>1.00</td>
<td>1.00</td>
<td>1.00</td>
</tr>
<tr>
<td>CODER_6</td>
<td>1.00</td>
<td>1.00</td>
<td>1.00</td>
<td>1.00</td>
<td>1.00</td>
</tr>
<tr>
<td>BA_SIZ</td>
<td>1.00</td>
<td>1.00</td>
<td>0.39</td>
<td>1.00</td>
<td>1.00</td>
</tr>
<tr>
<td>LCOM</td>
<td>1.00</td>
<td>1.00</td>
<td>1.00</td>
<td>1.00</td>
<td>0.52</td>
</tr>
<tr>
<td>CBO</td>
<td>1.00</td>
<td>1.00</td>
<td>1.00</td>
<td>1.00</td>
<td>1.00</td>
</tr>
<tr>
<td>RFC</td>
<td>1.00</td>
<td>1.00</td>
<td>0.78</td>
<td>1.00</td>
<td>1.00</td>
</tr>
<tr>
<td>NOC</td>
<td>1.00</td>
<td>1.00</td>
<td>1.00</td>
<td>1.00</td>
<td>1.00</td>
</tr>
<tr>
<td>DIT</td>
<td>1.00</td>
<td>1.00</td>
<td>1.00</td>
<td>1.00</td>
<td>1.00</td>
</tr>
<tr>
<td>WMC</td>
<td>1.00</td>
<td>1.00</td>
<td>1.00</td>
<td>1.00</td>
<td>1.00</td>
</tr>
<tr>
<td>SIZEB</td>
<td>0.00</td>
<td>0.00</td>
<td>1.00</td>
<td>1.00</td>
<td>1.00</td>
</tr>
<tr>
<td>VERSN</td>
<td>0.00</td>
<td>0.13</td>
<td>0.10</td>
<td>1.00</td>
<td>1.00</td>
</tr>
<tr>
<td>D_MAINT</td>
<td>0.00</td>
<td>0.00</td>
<td>1.00</td>
<td>1.00</td>
<td>1.00</td>
</tr>
<tr>
<td>---------</td>
<td>------</td>
<td>------</td>
<td>------</td>
<td>------</td>
<td>------</td>
</tr>
<tr>
<td>CODER_5</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>CODER_6</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>BA_SIZ</td>
<td>1.00</td>
<td>1.00</td>
<td>0.00</td>
<td></td>
<td></td>
</tr>
<tr>
<td>LCOM</td>
<td>1.00</td>
<td>1.00</td>
<td>1.00</td>
<td>0.00</td>
<td></td>
</tr>
<tr>
<td>CBO</td>
<td>1.00</td>
<td>1.00</td>
<td>1.00</td>
<td>1.00</td>
<td>0.00</td>
</tr>
<tr>
<td>RFC</td>
<td>1.00</td>
<td>1.00</td>
<td>1.00</td>
<td>1.00</td>
<td>0.00</td>
</tr>
<tr>
<td>NFC</td>
<td>1.00</td>
<td>1.00</td>
<td>1.00</td>
<td>1.00</td>
<td>1.00</td>
</tr>
<tr>
<td>DIT</td>
<td>1.00</td>
<td>1.00</td>
<td>1.00</td>
<td>1.00</td>
<td>1.00</td>
</tr>
<tr>
<td>WMC</td>
<td>1.00</td>
<td>1.00</td>
<td>1.00</td>
<td>1.00</td>
<td>0.00</td>
</tr>
<tr>
<td>SIEE</td>
<td>0.83</td>
<td>1.00</td>
<td>1.00</td>
<td>1.00</td>
<td>1.00</td>
</tr>
<tr>
<td>VERSN</td>
<td>1.00</td>
<td>1.00</td>
<td>1.00</td>
<td>0.86</td>
<td>1.00</td>
</tr>
<tr>
<td>D_MAINT</td>
<td>1.00</td>
<td>1.00</td>
<td>0.48</td>
<td>0.03</td>
<td>1.00</td>
</tr>
</tbody>
</table>

| RFC     | 0.00 |      |      |      |      |
| NOC     | 0.00 | 0.00 |      |      |      |
| DIT     | 1.00 | 1.00 | 0.00 |      |      |
| WMC     | 0.00 | 0.00 | 1.00 | 0.00 |      |
| SIEE    | 1.00 | 1.00 | 1.00 | 1.00 | 0.00 |
| VERSN   | 1.00 | 1.00 | 1.00 | 1.00 | 0.00 |
| D_MAINT | 1.00 | 1.00 | 1.00 | 1.00 | 0.00 |

| VERSN   |      |      |      |      |      |
| D_MAINT | 0.00 |      |      |      |      |

NUMBER OF OBSERVATIONS: 93
### Regression Analysis

**Variables**

<table>
<thead>
<tr>
<th>VARIABLE</th>
<th>COEFFICIENT</th>
<th>STD ERROR</th>
<th>STD COEF</th>
<th>TOLERANCE</th>
<th>T</th>
<th>P (2 TAIL)</th>
</tr>
</thead>
<tbody>
<tr>
<td>CONSTANT</td>
<td>-0.73</td>
<td>5.54</td>
<td>0.00</td>
<td>.</td>
<td>-0.13</td>
<td>0.90</td>
</tr>
<tr>
<td>BA_SIZ</td>
<td>-2.00</td>
<td>0.00</td>
<td>-0.08</td>
<td>0.66</td>
<td>-1.12</td>
<td>0.27</td>
</tr>
<tr>
<td>VERSN</td>
<td>0.78</td>
<td>0.82</td>
<td>0.07</td>
<td>0.62</td>
<td>0.96</td>
<td>0.34</td>
</tr>
<tr>
<td>CODER_2</td>
<td>-8.84</td>
<td>4.83</td>
<td>-0.12</td>
<td>0.74</td>
<td>-1.83</td>
<td>0.07</td>
</tr>
<tr>
<td>CODER_3</td>
<td>-22.16</td>
<td>10.76</td>
<td>-0.13</td>
<td>0.86</td>
<td>-2.06</td>
<td>0.04</td>
</tr>
<tr>
<td>CODER_4</td>
<td>0.51</td>
<td>7.84</td>
<td>0.00</td>
<td>0.82</td>
<td>0.07</td>
<td>0.95</td>
</tr>
<tr>
<td>CODER_5</td>
<td>36.15</td>
<td>6.36</td>
<td>0.35</td>
<td>0.86</td>
<td>5.68</td>
<td>0.00</td>
</tr>
<tr>
<td>CODER_6</td>
<td>-7.89</td>
<td>10.63</td>
<td>-0.04</td>
<td>0.88</td>
<td>-0.74</td>
<td>0.46</td>
</tr>
<tr>
<td>SIZE1</td>
<td>0.08</td>
<td>0.01</td>
<td>0.72</td>
<td>0.70</td>
<td>10.72</td>
<td>0.00</td>
</tr>
</tbody>
</table>

#### Analysis of Variance

<table>
<thead>
<tr>
<th>Source</th>
<th>Sum-of-Squares</th>
<th>DF</th>
<th>Mean-Square</th>
<th>F-ratio</th>
<th>P</th>
</tr>
</thead>
<tbody>
<tr>
<td>Regression</td>
<td>45063.04</td>
<td>8</td>
<td>5632.88</td>
<td>29.02</td>
<td>0.00</td>
</tr>
<tr>
<td>Residual</td>
<td>16303.17</td>
<td>84</td>
<td>194.09</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

**Data set:** Design and Maintenance

**Model includes all "suspects" for a base model**

**Based on this the base model is developed**
DEP VAR: PROD  N: 93  MULTIPLE R: 0.842  SQUARED MULTIPLE R: 0.709
ADJUSTED SQUARED MULTIPLE R: 0.703  STANDARD ERROR OF ESTIMATE: 14.08

<table>
<thead>
<tr>
<th>VARIABLE</th>
<th>COEFFICIENT</th>
<th>STD ERROR</th>
<th>STD COEF</th>
<th>TOLERANCE</th>
<th>T</th>
<th>P (2 TAIL)</th>
</tr>
</thead>
<tbody>
<tr>
<td>CONSTANT</td>
<td>0.21</td>
<td>1.91</td>
<td>0.00</td>
<td>.</td>
<td>0.11</td>
<td>0.91</td>
</tr>
<tr>
<td>CODER_5</td>
<td>39.11</td>
<td>6.18</td>
<td>0.37</td>
<td>0.93</td>
<td>6.33</td>
<td>0.00</td>
</tr>
<tr>
<td>SIZEL</td>
<td>0.07</td>
<td>0.01</td>
<td>0.66</td>
<td>0.93</td>
<td>11.16</td>
<td>0.00</td>
</tr>
</tbody>
</table>

ANALYSIS OF VARIANCE

<table>
<thead>
<tr>
<th>SOURCE</th>
<th>SUM-OF-SQUARES</th>
<th>DF</th>
<th>MEAN-SQUARE</th>
<th>F-RATIO</th>
<th>P</th>
</tr>
</thead>
<tbody>
<tr>
<td>REGRESSION</td>
<td>43514.61</td>
<td>2</td>
<td>21757.30</td>
<td>109.69</td>
<td>0.00</td>
</tr>
<tr>
<td>RESIDUAL</td>
<td>17851.60</td>
<td>90</td>
<td>198.35</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Data set: Design and Maintenance

This is the base model. It shows that size and who did the work explain considerable variance in productivity.
DEP VAR: PROD  N: 93  MULTIPLE R: 0.856  SQUARED MULTIPLE R: 0.733
ADJUSTED SQUARED MULTIPLE R: 0.724  STANDARD ERROR OF ESTIMATE: 13.57

<table>
<thead>
<tr>
<th>VARIABLE</th>
<th>COEFFICIENT</th>
<th>STD ERROR</th>
<th>STD COEF</th>
<th>TOLERANCE</th>
<th>T</th>
<th>P(2 TAIL)</th>
</tr>
</thead>
<tbody>
<tr>
<td>CONSTANT</td>
<td>3.17</td>
<td>2.11</td>
<td>0.00</td>
<td>.</td>
<td>1.50</td>
<td>0.14</td>
</tr>
<tr>
<td>CBO</td>
<td>-0.56</td>
<td>0.20</td>
<td>-0.16</td>
<td>0.96</td>
<td>-2.82</td>
<td>0.01</td>
</tr>
<tr>
<td>CODER_5</td>
<td>36.58</td>
<td>6.02</td>
<td>0.35</td>
<td>0.91</td>
<td>6.08</td>
<td>0.00</td>
</tr>
<tr>
<td>SIZEL</td>
<td>0.07</td>
<td>0.01</td>
<td>0.69</td>
<td>0.90</td>
<td>11.89</td>
<td>0.00</td>
</tr>
</tbody>
</table>

**ANALYSIS OF VARIANCE**

<table>
<thead>
<tr>
<th>SOURCE</th>
<th>SUM-OF-SQUARES</th>
<th>DF</th>
<th>MEAN-SQUARE</th>
<th>F-RATIO</th>
<th>P</th>
</tr>
</thead>
<tbody>
<tr>
<td>REGRESSION</td>
<td>44980.08</td>
<td>3</td>
<td>14993.36</td>
<td>81.44</td>
<td>0.00</td>
</tr>
<tr>
<td>RESIDUAL</td>
<td>16386.12</td>
<td>89</td>
<td>184.11</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Data set: Design and Maintenance

*Model shows that the CBO metric does predict additional variance in productivity and its effect is greater than that of SIZE.*
PEARSON CORRELATION MATRIX

<table>
<thead>
<tr>
<th></th>
<th>PROD</th>
<th>CBO</th>
<th>CODER_5</th>
<th>SIZEL</th>
</tr>
</thead>
<tbody>
<tr>
<td>PROD</td>
<td>1.00</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>CBO</td>
<td>-0.11</td>
<td>1.00</td>
<td></td>
<td></td>
</tr>
<tr>
<td>CODER_5</td>
<td>0.55</td>
<td>-0.11</td>
<td>1.00</td>
<td></td>
</tr>
<tr>
<td>SIZEL</td>
<td>0.76</td>
<td>0.12</td>
<td>0.27</td>
<td>1.00</td>
</tr>
</tbody>
</table>

BARTLETT CHI-SQUARE STATISTIC: 128.885 DF= 6 PROB= 0.000

MATRIX OF BONFERRONI PROBABILITIES

<table>
<thead>
<tr>
<th></th>
<th>PROD</th>
<th>CBO</th>
<th>CODER_5</th>
<th>SIZEL</th>
</tr>
</thead>
<tbody>
<tr>
<td>PROD</td>
<td>0.00</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>CBO</td>
<td>1.00</td>
<td>0.00</td>
<td></td>
<td></td>
</tr>
<tr>
<td>CODER_5</td>
<td>0.00</td>
<td>1.00</td>
<td>0.00</td>
<td></td>
</tr>
<tr>
<td>SIZEL</td>
<td>0.00</td>
<td>1.00</td>
<td>0.05</td>
<td>0.00</td>
</tr>
</tbody>
</table>

NUMBER OF OBSERVATIONS: 93

Data set: Design and Maintenance

The correlation matrix shows that the predictors are reasonably independent (SIZE and CODER 5 are significantly correlated but less than 0.3)
**EIGENVALUES OF UNIT SCALED X'X**

<table>
<thead>
<tr>
<th></th>
<th>1</th>
<th>2</th>
<th>3</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>1</td>
<td>2.16</td>
<td>0.62</td>
</tr>
</tbody>
</table>

**CONDITION INDICES**

<table>
<thead>
<tr>
<th></th>
<th>1</th>
<th>2</th>
<th>3</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>1.00</td>
<td>1.86</td>
<td>3.14</td>
</tr>
</tbody>
</table>

**VARIANCE PROPORTIONS**

<table>
<thead>
<tr>
<th></th>
<th>1</th>
<th>2</th>
<th>3</th>
</tr>
</thead>
<tbody>
<tr>
<td>CONSTANT</td>
<td>0.06</td>
<td>0.00</td>
<td>0.94</td>
</tr>
<tr>
<td>CBO</td>
<td>0.08</td>
<td>0.50</td>
<td>0.43</td>
</tr>
<tr>
<td>SIZE1</td>
<td>0.07</td>
<td>0.37</td>
<td>0.56</td>
</tr>
</tbody>
</table>

Data set: Design and Maintenance

*Model shows that the CBO metric does predict additional variance in productivity and its effect is greater than that of SIZE.*

*Model:*

**Dependent Variable: PROD**

**Independent Variables: SIZE1, CBO and CODER_S (categorical)**
PEARSON CORRELATION MATRIX *(Design only)*

<table>
<thead>
<tr>
<th></th>
<th>PROD</th>
<th>SIZEL</th>
<th>BA_SIZ</th>
<th>VERS</th>
<th>LCOM</th>
</tr>
</thead>
<tbody>
<tr>
<td>PROD</td>
<td>1.00</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>SIZEL</td>
<td>0.70</td>
<td>1.00</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>BA_SIZ</td>
<td>-0.16</td>
<td>-0.09</td>
<td>1.00</td>
<td></td>
<td></td>
</tr>
<tr>
<td>VERS</td>
<td>-0.20</td>
<td>-0.60</td>
<td>0.05</td>
<td>1.00</td>
<td></td>
</tr>
<tr>
<td>LCOM</td>
<td>-0.05</td>
<td>0.24</td>
<td>0.05</td>
<td>0.05</td>
<td>1.00</td>
</tr>
<tr>
<td>CBO</td>
<td>-0.04</td>
<td>0.45</td>
<td>-0.18</td>
<td>0.05</td>
<td>0.07</td>
</tr>
<tr>
<td>RFC</td>
<td>-0.02</td>
<td>0.52</td>
<td>-0.18</td>
<td>0.04</td>
<td>0.16</td>
</tr>
<tr>
<td>NOC</td>
<td>-0.04</td>
<td>0.03</td>
<td>0.03</td>
<td>0.16</td>
<td>0.15</td>
</tr>
<tr>
<td>DIT</td>
<td>0.07</td>
<td>-0.04</td>
<td>-0.22</td>
<td>-0.10</td>
<td>-0.06</td>
</tr>
<tr>
<td>WMC</td>
<td>-0.00</td>
<td>0.53</td>
<td>-0.17</td>
<td>0.03</td>
<td>0.20</td>
</tr>
<tr>
<td>COD_D2</td>
<td>-0.03</td>
<td>0.23</td>
<td>-0.28</td>
<td>-0.16</td>
<td>-0.21</td>
</tr>
<tr>
<td>COD_D3</td>
<td>-0.13</td>
<td>0.00</td>
<td>-0.30</td>
<td>0.32</td>
<td>0.50</td>
</tr>
<tr>
<td>COD_D4</td>
<td>0.04</td>
<td>-0.06</td>
<td>-0.25</td>
<td>0.23</td>
<td>-0.06</td>
</tr>
<tr>
<td>COD_D5</td>
<td>0.74</td>
<td>0.35</td>
<td>0.06</td>
<td>-0.21</td>
<td>-0.13</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th></th>
<th>CBO</th>
<th>RFC</th>
<th>NOC</th>
<th>DIT</th>
<th>WMC</th>
</tr>
</thead>
<tbody>
<tr>
<td>CBO</td>
<td>1.00</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>RFC</td>
<td>0.95</td>
<td>1.00</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>NOC</td>
<td>0.30</td>
<td>0.41</td>
<td>1.00</td>
<td></td>
<td></td>
</tr>
<tr>
<td>DIT</td>
<td>-0.03</td>
<td>0.00</td>
<td>-0.03</td>
<td>1.00</td>
<td></td>
</tr>
<tr>
<td>WMC</td>
<td>0.88</td>
<td>0.98</td>
<td>0.46</td>
<td>0.02</td>
<td>1.00</td>
</tr>
<tr>
<td>COD_D2</td>
<td>0.32</td>
<td>0.29</td>
<td>-0.10</td>
<td>0.30</td>
<td>0.26</td>
</tr>
<tr>
<td>COD_D3</td>
<td>0.01</td>
<td>-0.00</td>
<td>-0.04</td>
<td>-0.03</td>
<td>-0.01</td>
</tr>
<tr>
<td>COD_D4</td>
<td>-0.01</td>
<td>0.07</td>
<td>-0.03</td>
<td>-0.02</td>
<td>0.11</td>
</tr>
<tr>
<td>COD_D5</td>
<td>-0.20</td>
<td>-0.24</td>
<td>-0.06</td>
<td>-0.05</td>
<td>-0.25</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th></th>
<th>COD_D2</th>
<th>COD_D3</th>
<th>COD_D4</th>
<th>COD_D5</th>
</tr>
</thead>
<tbody>
<tr>
<td>COD_D2</td>
<td>1.00</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>COD_D3</td>
<td>-0.11</td>
<td>1.00</td>
<td></td>
<td></td>
</tr>
<tr>
<td>COD_D4</td>
<td>-0.08</td>
<td>-0.03</td>
<td>1.00</td>
<td></td>
</tr>
<tr>
<td>COD_D5</td>
<td>-0.16</td>
<td>-0.07</td>
<td>-0.05</td>
<td>1.00</td>
</tr>
</tbody>
</table>

BARTLETT CHI-SQUARE STATISTIC: 632.913 DF= 91 PROB= 0.000
## MATRIX OF BONFERRONI PROBABILITIES

<table>
<thead>
<tr>
<th></th>
<th>PROD</th>
<th>SIZEL</th>
<th>BA_SIZ</th>
<th>VERS</th>
<th>LCOM</th>
</tr>
</thead>
<tbody>
<tr>
<td>PROD</td>
<td>0.00</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>SIZEL</td>
<td>0.00</td>
<td>0.00</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>BA_SIZ</td>
<td>1.00</td>
<td>1.00</td>
<td>0.00</td>
<td></td>
<td></td>
</tr>
<tr>
<td>VERS</td>
<td>1.00</td>
<td>1.00</td>
<td>0.00</td>
<td>0.00</td>
<td></td>
</tr>
<tr>
<td>LCOM</td>
<td>1.00</td>
<td>1.00</td>
<td>1.00</td>
<td>1.00</td>
<td>0.00</td>
</tr>
<tr>
<td>CBO</td>
<td>1.00</td>
<td>0.18</td>
<td>1.00</td>
<td>1.00</td>
<td>1.00</td>
</tr>
<tr>
<td>RFC</td>
<td>1.00</td>
<td>0.02</td>
<td>1.00</td>
<td>1.00</td>
<td>1.00</td>
</tr>
<tr>
<td>NOC</td>
<td>1.00</td>
<td>1.00</td>
<td>1.00</td>
<td>1.00</td>
<td>1.00</td>
</tr>
<tr>
<td>DIT</td>
<td>1.00</td>
<td>1.00</td>
<td>1.00</td>
<td>1.00</td>
<td>1.00</td>
</tr>
<tr>
<td>WMC</td>
<td>1.00</td>
<td>0.01</td>
<td>1.00</td>
<td>1.00</td>
<td>1.00</td>
</tr>
<tr>
<td>COD_D2</td>
<td>1.00</td>
<td>1.00</td>
<td>1.00</td>
<td>1.00</td>
<td>1.00</td>
</tr>
<tr>
<td>COD_D3</td>
<td>1.00</td>
<td>1.00</td>
<td>1.00</td>
<td>1.00</td>
<td>1.00</td>
</tr>
<tr>
<td>COD_D4</td>
<td>1.00</td>
<td>1.00</td>
<td>1.00</td>
<td>1.00</td>
<td>1.00</td>
</tr>
<tr>
<td>COD_D5</td>
<td>0.00</td>
<td>1.00</td>
<td>1.00</td>
<td>1.00</td>
<td>1.00</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th></th>
<th>CBO</th>
<th>RFC</th>
<th>NOC</th>
<th>DIT</th>
<th>WMC</th>
</tr>
</thead>
<tbody>
<tr>
<td>CBO</td>
<td>0.00</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>RFC</td>
<td>0.00</td>
<td>0.00</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>NOC</td>
<td>1.00</td>
<td>1.00</td>
<td>0.44</td>
<td></td>
<td></td>
</tr>
<tr>
<td>DIT</td>
<td>1.00</td>
<td>1.00</td>
<td>0.00</td>
<td>0.00</td>
<td></td>
</tr>
<tr>
<td>WMC</td>
<td>0.00</td>
<td>0.00</td>
<td>0.12</td>
<td>1.00</td>
<td>0.00</td>
</tr>
<tr>
<td>COD_D2</td>
<td>1.00</td>
<td>1.00</td>
<td>1.00</td>
<td>1.00</td>
<td>1.00</td>
</tr>
<tr>
<td>COD_D3</td>
<td>1.00</td>
<td>1.00</td>
<td>1.00</td>
<td>1.00</td>
<td>1.00</td>
</tr>
<tr>
<td>COD_D4</td>
<td>1.00</td>
<td>1.00</td>
<td>1.00</td>
<td>1.00</td>
<td>1.00</td>
</tr>
<tr>
<td>COD_D5</td>
<td>1.00</td>
<td>1.00</td>
<td>1.00</td>
<td>1.00</td>
<td>1.00</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th></th>
<th>COD_D2</th>
<th>COD_D3</th>
<th>COD_D4</th>
<th>COD_D5</th>
</tr>
</thead>
<tbody>
<tr>
<td>COD_D2</td>
<td>0.00</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>COD_D3</td>
<td>1.00</td>
<td>0.00</td>
<td></td>
<td></td>
</tr>
<tr>
<td>COD_D4</td>
<td>1.00</td>
<td>1.00</td>
<td>0.00</td>
<td></td>
</tr>
<tr>
<td>COD_D5</td>
<td>1.00</td>
<td>1.00</td>
<td>1.00</td>
<td>0.00</td>
</tr>
</tbody>
</table>

**NOTE** This correlation matrix is for the DESIGN data set (i.e. design of new classes for TPM)
**DEP VAR:** PROD  
**N:** 45  
**MULTIPLE R:** 0.907  
**SQUARED MULTIPLE R:** 0.822  
**ADJUSTED SQUARED MULTIPLE R:** 0.800  
**STANDARD ERROR OF ESTIMATE:** 14.73

<table>
<thead>
<tr>
<th>VARIABLE</th>
<th>COEFFICIENT</th>
<th>STD ERROR</th>
<th>STD COEF</th>
<th>TOLERANCE</th>
<th>T</th>
<th>P(2 TAIL)</th>
</tr>
</thead>
<tbody>
<tr>
<td>CONSTANT</td>
<td>18.22</td>
<td>5.91</td>
<td>0.00</td>
<td>.</td>
<td>3.08</td>
<td>0.00</td>
</tr>
<tr>
<td>BA_SIZ</td>
<td>-0.00</td>
<td>0.00</td>
<td>-0.25</td>
<td>0.81</td>
<td>-3.30</td>
<td>0.00</td>
</tr>
<tr>
<td>COD_D2</td>
<td>-12.92</td>
<td>6.20</td>
<td>-0.16</td>
<td>0.78</td>
<td>-2.08</td>
<td>0.04</td>
</tr>
<tr>
<td>COD_D3</td>
<td>-28.68</td>
<td>11.47</td>
<td>-0.18</td>
<td>0.86</td>
<td>-2.50</td>
<td>0.02</td>
</tr>
<tr>
<td>COD_D5</td>
<td>60.42</td>
<td>8.59</td>
<td>0.53</td>
<td>0.81</td>
<td>7.03</td>
<td>0.00</td>
</tr>
<tr>
<td>SIZEL</td>
<td>0.07</td>
<td>0.01</td>
<td>0.53</td>
<td>0.79</td>
<td>6.90</td>
<td>0.00</td>
</tr>
</tbody>
</table>

**ANALYSIS OF VARIANCE**

<table>
<thead>
<tr>
<th>SOURCE</th>
<th>SUM-OF-SQUARES</th>
<th>DF</th>
<th>MEAN-SQUARE</th>
<th>F-RATIO</th>
<th>P</th>
</tr>
</thead>
<tbody>
<tr>
<td>REGRESSION</td>
<td>39189.04</td>
<td>5</td>
<td>7837.81</td>
<td>36.13</td>
<td>0.00</td>
</tr>
<tr>
<td>RESIDUAL</td>
<td>8459.39</td>
<td>39</td>
<td>216.91</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Data set: Design of new classes for TPM (all coders included)

This is the base model for the DESIGN data set (as before SIZE and CODER are significant)
EIGENVALUES OF UNIT SCALED X'X

<table>
<thead>
<tr>
<th></th>
<th>1</th>
<th>2</th>
<th>3</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>2.53</td>
<td>0.36</td>
<td>0.11</td>
</tr>
</tbody>
</table>

CONDITION INDICES

<table>
<thead>
<tr>
<th></th>
<th>1</th>
<th>2</th>
<th>3</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>1.00</td>
<td>2.66</td>
<td>4.81</td>
</tr>
</tbody>
</table>

VARIANCE PROPORTIONS

<table>
<thead>
<tr>
<th></th>
<th>1</th>
<th>2</th>
<th>3</th>
</tr>
</thead>
<tbody>
<tr>
<td>CONSTANT</td>
<td>0.02</td>
<td>0.01</td>
<td>0.97</td>
</tr>
<tr>
<td>BA_SIZ</td>
<td>0.03</td>
<td>0.29</td>
<td>0.68</td>
</tr>
<tr>
<td>SIZE_L</td>
<td>0.05</td>
<td>0.62</td>
<td>0.33</td>
</tr>
</tbody>
</table>

Data set: Design Only

Model:
Dependent Variable: PROD

Independent Variables: SIZE_L, BA_SIZ
**ANALYSIS OF VARIANCE**

<table>
<thead>
<tr>
<th>SOURCE</th>
<th>SUM-OF-SQUARES</th>
<th>DF</th>
<th>MEAN-SQUARE</th>
<th>F-RATIO</th>
<th>P</th>
</tr>
</thead>
<tbody>
<tr>
<td>REGRESSION</td>
<td>41549.97</td>
<td>6</td>
<td>6925.00</td>
<td>43.15</td>
<td>0.00</td>
</tr>
<tr>
<td>RESIDUAL</td>
<td>6098.46</td>
<td>38</td>
<td>160.49</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Data set: Design of new classes for TPM (all coders included)

Model demonstrates that WMC explains additional variance. The Tolerance statistic for the independent variables shows that there is no multicollinearity problem in this model.
EIGENVALUES OF UNIT SCALED X'X

<table>
<thead>
<tr>
<th></th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>3.08</td>
<td>0.60</td>
<td>0.21</td>
<td>0.11</td>
</tr>
</tbody>
</table>

CONDITION INDICES

<table>
<thead>
<tr>
<th></th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>1.00</td>
<td>2.26</td>
<td>3.85</td>
<td>5.32</td>
</tr>
</tbody>
</table>

VARIANCE PROPORTIONS

<table>
<thead>
<tr>
<th></th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
</tr>
</thead>
<tbody>
<tr>
<td>CONSTANT</td>
<td>0.02</td>
<td>0.02</td>
<td>0.00</td>
<td>0.96</td>
</tr>
<tr>
<td>WMC</td>
<td>0.03</td>
<td>0.34</td>
<td>0.62</td>
<td>0.01</td>
</tr>
<tr>
<td>BA_SIZ</td>
<td>0.02</td>
<td>0.16</td>
<td>0.12</td>
<td>0.70</td>
</tr>
<tr>
<td>SIZEL</td>
<td>0.02</td>
<td>0.03</td>
<td>0.77</td>
<td>0.17</td>
</tr>
</tbody>
</table>

Data set: Design Only

Model:
Dependent Variable: PROD

Independent Variables: SIZEL, BA_SIZ, WMC
### Pearson Correlation Matrix

<table>
<thead>
<tr>
<th></th>
<th>SIZEL</th>
<th>LCOM</th>
<th>CBO</th>
<th>RFC</th>
<th>NOC</th>
</tr>
</thead>
<tbody>
<tr>
<td>SIZEL</td>
<td>1.00</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>LCOM</td>
<td>0.24</td>
<td>1.00</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>CBO</td>
<td>0.45</td>
<td>0.07</td>
<td>1.00</td>
<td></td>
<td></td>
</tr>
<tr>
<td>RFC</td>
<td>0.52</td>
<td>0.16</td>
<td>0.95</td>
<td>1.00</td>
<td></td>
</tr>
<tr>
<td>NOC</td>
<td>0.03</td>
<td>0.15</td>
<td>0.30</td>
<td>0.41</td>
<td>1.00</td>
</tr>
<tr>
<td>DIT</td>
<td>-0.04</td>
<td>-0.06</td>
<td>-0.03</td>
<td>0.00</td>
<td>-0.03</td>
</tr>
<tr>
<td>WMC</td>
<td>0.53</td>
<td>0.20</td>
<td>0.88</td>
<td>0.98</td>
<td>0.46</td>
</tr>
</tbody>
</table>

### Bartlett Chi-Square Statistic:
- Value: 435.874
- DF: 21
- Prob: 0.000

### Matrix of Bonferroni Probabilities

<table>
<thead>
<tr>
<th></th>
<th>SIZEL</th>
<th>LCOM</th>
<th>CBO</th>
<th>RFC</th>
<th>NOC</th>
</tr>
</thead>
<tbody>
<tr>
<td>SIZEL</td>
<td>0.00</td>
<td>0.00</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>LCOM</td>
<td>1.00</td>
<td>0.00</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>CBO</td>
<td>0.04</td>
<td>1.00</td>
<td>0.00</td>
<td></td>
<td></td>
</tr>
<tr>
<td>RFC</td>
<td>0.01</td>
<td>0.00</td>
<td>0.00</td>
<td>0.00</td>
<td></td>
</tr>
<tr>
<td>NOC</td>
<td>1.00</td>
<td>1.00</td>
<td>0.95</td>
<td>0.10</td>
<td>0.00</td>
</tr>
<tr>
<td>DIT</td>
<td>1.00</td>
<td>1.00</td>
<td>1.00</td>
<td>1.00</td>
<td>1.00</td>
</tr>
<tr>
<td>WMC</td>
<td>0.00</td>
<td>1.00</td>
<td>0.00</td>
<td>0.00</td>
<td>0.03</td>
</tr>
</tbody>
</table>

### Number of Observations:
- 45

Data set: Design of new classes for TPM
Size is significantly correlated to CBO, RFC and WMC
DEP VAR: PROD    N: 66    MULTIPLE R: 0.573    SQUARE MULTIPLE R: 0.329
ADJUSTED SQUARE MULTIPLE R: 0.318    STANDARD ERROR OF ESTIMATE: 2.3021

<table>
<thead>
<tr>
<th>VARIABLE</th>
<th>COEFFICIENT</th>
<th>STD ERROR</th>
<th>STD COEF TOLERANCE</th>
<th>T</th>
<th>P (2 TAIL)</th>
</tr>
</thead>
<tbody>
<tr>
<td>CONSTANT</td>
<td>1.6370</td>
<td>0.3932</td>
<td>0.0000</td>
<td>4.1630</td>
<td>0.0001</td>
</tr>
<tr>
<td>SIZEL</td>
<td>0.0102</td>
<td>0.0018</td>
<td>0.5732</td>
<td>5.5955</td>
<td>0.0000</td>
</tr>
</tbody>
</table>

ANALYSIS OF VARIANCE

<table>
<thead>
<tr>
<th>SOURCE</th>
<th>SUM-OF-SQUARES</th>
<th>DF</th>
<th>MEAN-SQUARE</th>
<th>F-RATIO</th>
<th>P</th>
</tr>
</thead>
<tbody>
<tr>
<td>REGRESSION</td>
<td>165.9304</td>
<td>1</td>
<td>165.9304</td>
<td>31.3098</td>
<td>0.0000</td>
</tr>
<tr>
<td>RESIDUAL</td>
<td>339.1764</td>
<td>64</td>
<td>5.2996</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

CODER DATA SET (includes both design & maintenance work)
DEP VAR: PROD_H  N: 66  MULTIPLE R: 0.673  SQUARED MULTIPLE R: 0.453  ADJUSTED SQUARED MULTIPLE R: 0.435  STANDARD ERROR OF ESTIMATE: 2.0946

<table>
<thead>
<tr>
<th>VARIABLE</th>
<th>COEFFICIENT</th>
<th>STD ERROR</th>
<th>STD COEF</th>
<th>TOLERANCE</th>
<th>T</th>
<th>P(2 TAIL)</th>
</tr>
</thead>
<tbody>
<tr>
<td>CONSTANT</td>
<td>2.3544</td>
<td>0.4050</td>
<td>0.0000</td>
<td>5.8138</td>
<td></td>
<td>0.0000</td>
</tr>
<tr>
<td>LCOM</td>
<td>-0.0371</td>
<td>0.0098</td>
<td>-0.3526</td>
<td>0.9995</td>
<td>-3.7821</td>
<td>0.0003</td>
</tr>
<tr>
<td>SIZEL</td>
<td>0.0100</td>
<td>0.0017</td>
<td>0.5652</td>
<td>0.9995</td>
<td>6.0624</td>
<td>0.0000</td>
</tr>
</tbody>
</table>

ANALYSIS OF VARIANCE

<table>
<thead>
<tr>
<th>SOURCE</th>
<th>SUM-OF-SQUARES</th>
<th>DF</th>
<th>MEAN-SQUARE</th>
<th>F-RATIO</th>
<th>P</th>
</tr>
</thead>
<tbody>
<tr>
<td>REGRESSION</td>
<td>228.6921</td>
<td>2</td>
<td>114.3461</td>
<td>26.0616</td>
<td>0.0000</td>
</tr>
<tr>
<td>RESIDUAL</td>
<td>276.4146</td>
<td>63</td>
<td>4.3875</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

CODER DATA SET (includes both design & maintenance work)

Note: LCOM and SIZEL are NOT significantly correlated.
**EIGENVALUES OF UNIT SCALED X'X**

<table>
<thead>
<tr>
<th></th>
<th>1</th>
<th>2</th>
<th>3</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>2.12</td>
<td>0.62</td>
<td>0.26</td>
</tr>
</tbody>
</table>

**CONDITION INDICES**

<table>
<thead>
<tr>
<th></th>
<th>1</th>
<th>2</th>
<th>3</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>1.00</td>
<td>1.85</td>
<td>2.84</td>
</tr>
</tbody>
</table>

**VARIANCE PROPORTIONS**

<table>
<thead>
<tr>
<th></th>
<th>1</th>
<th>2</th>
<th>3</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>0.08</td>
<td>0.01</td>
<td>0.92</td>
</tr>
<tr>
<td>CONSTANT</td>
<td>0.09</td>
<td>0.67</td>
<td>0.24</td>
</tr>
<tr>
<td>LCOM</td>
<td>0.08</td>
<td>0.30</td>
<td>0.62</td>
</tr>
</tbody>
</table>

**CODER DATA SET (includes both design & maintenance work)**

*Note: LCOM and SIZEL are NOT significantly correlated.*
PEARSON CORRELATION MATRIX

<table>
<thead>
<tr>
<th></th>
<th>EFFORT</th>
<th>WMC</th>
<th>DIT</th>
<th>NOC</th>
<th>RFC</th>
</tr>
</thead>
<tbody>
<tr>
<td>EFFORT</td>
<td>1.00</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>WMC</td>
<td>-0.32</td>
<td>1.00</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>DIT</td>
<td>0.48</td>
<td>-0.77</td>
<td>1.00</td>
<td></td>
<td></td>
</tr>
<tr>
<td>NOC</td>
<td>0.23</td>
<td>-0.21</td>
<td>-0.08</td>
<td>1.00</td>
<td></td>
</tr>
<tr>
<td>RFC</td>
<td>0.27</td>
<td>0.25</td>
<td>0.11</td>
<td>-0.05</td>
<td>1.00</td>
</tr>
<tr>
<td>CBO</td>
<td>0.28</td>
<td>-0.10</td>
<td>0.28</td>
<td>0.15</td>
<td>0.80</td>
</tr>
<tr>
<td>LCOM</td>
<td>0.28</td>
<td>0.30</td>
<td>-0.12</td>
<td>-0.06</td>
<td>0.41</td>
</tr>
<tr>
<td>LINES</td>
<td>-0.22</td>
<td>0.53</td>
<td>-0.17</td>
<td>-0.46</td>
<td>0.60</td>
</tr>
<tr>
<td>REUSE1</td>
<td>-0.60</td>
<td>0.44</td>
<td>-0.72</td>
<td>0.02</td>
<td>-0.53</td>
</tr>
<tr>
<td>REUSE2</td>
<td>-0.66</td>
<td>0.23</td>
<td>-0.52</td>
<td>-0.03</td>
<td>-0.50</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th></th>
<th>CBO</th>
<th>LCOM</th>
<th>LINES</th>
<th>REUSE1</th>
<th>REUSE2</th>
</tr>
</thead>
<tbody>
<tr>
<td>CBO</td>
<td>1.00</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>LCOM</td>
<td>0.34</td>
<td>1.00</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>LINES</td>
<td>0.45</td>
<td>0.29</td>
<td>1.00</td>
<td></td>
<td></td>
</tr>
<tr>
<td>REUSE1</td>
<td>-0.53</td>
<td>-0.39</td>
<td>-0.06</td>
<td>1.00</td>
<td></td>
</tr>
<tr>
<td>REUSE2</td>
<td>-0.47</td>
<td>-0.44</td>
<td>-0.08</td>
<td>0.81</td>
<td>1.00</td>
</tr>
</tbody>
</table>

BARTLETT CHI-SQUARE STATISTIC: 193.766 DF= 45 PROB= 0.000

MATRIX OF BONFERRONI PROBABILITIES

<table>
<thead>
<tr>
<th></th>
<th>EFFORT</th>
<th>WMC</th>
<th>DIT</th>
<th>NOC</th>
<th>RFC</th>
</tr>
</thead>
<tbody>
<tr>
<td>EFFORT</td>
<td>0.00</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>WMC</td>
<td>1.00</td>
<td>0.00</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>DIT</td>
<td>0.41</td>
<td>0.00</td>
<td>0.00</td>
<td></td>
<td></td>
</tr>
<tr>
<td>NOC</td>
<td>1.00</td>
<td>1.00</td>
<td>1.00</td>
<td>0.00</td>
<td></td>
</tr>
<tr>
<td>RFC</td>
<td>1.00</td>
<td>1.00</td>
<td>1.00</td>
<td>1.00</td>
<td>0.00</td>
</tr>
<tr>
<td>CBO</td>
<td>1.00</td>
<td>1.00</td>
<td>1.00</td>
<td>1.00</td>
<td>0.00</td>
</tr>
<tr>
<td>LCOM</td>
<td>1.00</td>
<td>1.00</td>
<td>1.00</td>
<td>1.00</td>
<td>1.00</td>
</tr>
</tbody>
</table>
NUMBER OF OBSERVATIONS: 28

Dataset: FIS

Effort: is the amount of time taken to rework the class for use in SLB measured in hours

Reuse1: Expert 1's rating of reusability of the class on a scale of 1 to 9 (max reuse)

Reuse2: Expert 2's rating

Lines: Lines of code used in the definition of the class
**Dependent Variable: Effort**

The amount of time taken to rework the class for use in SLB measured in hours.

**Independent variable: Lines**

The lines of code written for defining the class.

**Model suggests that rework effort is not predicted by lines of code for the class.**
DEP VAR: EFFORT   N:  28  MULTIPLE R: 0.505  SQUARED MULTIPLE R: 0.255
ADJUSTED SQUARED MULTIPLE R: 0.195  STANDARD ERROR OF ESTIMATE: 8.0526

VARIABLE    COEFFICIENT  STD ERROR  STD COEF  TOLERANCE  T   P(2 TAIL)
CONSTANT   -2.6250     8.5682     0.0000     .         -0.3064  0.7619
DIT        11.5859     4.4095     0.4600     0.9727    2.6275  0.0145
LINES      -0.0500     0.0605    -0.1446     0.9727    -0.8258  0.4167

ANALYSIS OF VARIANCE

SOURCE    SUM-OF-SQUARES  DF  MEAN-SQUARE  F-RATIO  P
REGRESSION  553.4848     2   276.7424   4.2678  0.0254
RESIDUAL   1621.0974    25   64.8439

Dataset: FIS

Dependent Variable: EFFORT: is the amount of time taken to rework the class for use in SLB

Model suggests that DIT is a significant predictor of rework while LINES is not.
**DEPENDENT VARIABLE:**  \textit{EFFORT}  \hspace{1cm}  \textit{N:}  28  \hspace{1cm}  \textit{MULTIPLE R:}  0.484  \hspace{1cm}  \textit{SQUARED MULTIPLE R:}  0.234  \hspace{1cm}  \textit{ADJUSTED SQUARED MULTIPLE R:}  0.205  \hspace{1cm}  \textit{STANDARD ERROR OF ESTIMATE:}  8.00

<table>
<thead>
<tr>
<th>VARIABLE</th>
<th>COEFFICIENT</th>
<th>STD ERROR</th>
<th>STD COEF</th>
<th>TOLERANCE</th>
<th>T</th>
<th>P(2 TAIL)</th>
</tr>
</thead>
<tbody>
<tr>
<td>CONSTANT</td>
<td>-8.25</td>
<td>5.17</td>
<td>0.00</td>
<td>.</td>
<td>-1.60</td>
<td>0.12</td>
</tr>
<tr>
<td>DIT</td>
<td>12.19</td>
<td>4.32</td>
<td>0.48</td>
<td>1.00</td>
<td>2.82</td>
<td>0.01</td>
</tr>
</tbody>
</table>

**ANALYSIS OF VARIANCE**

<table>
<thead>
<tr>
<th>SOURCE</th>
<th>SUM-OF-SQUARES</th>
<th>DF</th>
<th>MEAN-SQUARE</th>
<th>F-RATIO</th>
<th>P</th>
</tr>
</thead>
<tbody>
<tr>
<td>REGRESSION</td>
<td>509.26</td>
<td>1</td>
<td>509.26</td>
<td>7.95</td>
<td>0.01</td>
</tr>
<tr>
<td>RESIDUAL</td>
<td>1665.32</td>
<td>26</td>
<td>64.05</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Dataset: FIS

**Dependent Variable:** \textit{Effort} \hspace{1cm} is the amount of time taken to rework the class for use in SLB measured in hours

Model suggests that DIT is a significant predictor of rework. As one would expect, the greater the DIT the more time it would take to understand the inheritance relationships and classes with higher DIT will take longer to rework (to be reused) in another system.
DEP VAR: EFFORT N: 28 MULTIPLE R: 0.641 SQUARED MULTIPLE R: 0.411
ADJUSTED SQUARED MULTIPLE R: 0.337 STANDARD ERROR OF ESTIMATE: 7.3083

<table>
<thead>
<tr>
<th>VARIABLE</th>
<th>COEFFICIENT</th>
<th>STD ERROR</th>
<th>STD COEF</th>
<th>TOLERANCE</th>
<th>T</th>
<th>P(2 TAIL)</th>
</tr>
</thead>
<tbody>
<tr>
<td>CONSTANT</td>
<td>-0.7852</td>
<td>7.8105</td>
<td>0.0000</td>
<td></td>
<td>-0.1005</td>
<td>0.9208</td>
</tr>
<tr>
<td>DIT</td>
<td>12.3638</td>
<td>4.0138</td>
<td>0.4909</td>
<td>0.9669</td>
<td>3.0803</td>
<td>0.0051</td>
</tr>
<tr>
<td>LCOM</td>
<td>0.0416</td>
<td>0.0165</td>
<td>0.4138</td>
<td>0.9111</td>
<td>2.5201</td>
<td>0.0188</td>
</tr>
<tr>
<td>LINES</td>
<td>-0.0895</td>
<td>0.0571</td>
<td>-0.2590</td>
<td>0.8993</td>
<td>-1.5674</td>
<td>0.1301</td>
</tr>
</tbody>
</table>

ANALYSIS OF VARIANCE

<table>
<thead>
<tr>
<th>SOURCE</th>
<th>SUM-OF-SQUARES</th>
<th>DF</th>
<th>MEAN-SQUARE</th>
<th>F-RATIO</th>
<th>P</th>
</tr>
</thead>
<tbody>
<tr>
<td>REGRESSION</td>
<td>892.7044</td>
<td>3</td>
<td>297.5681</td>
<td>5.5712</td>
<td>0.0048</td>
</tr>
<tr>
<td>RESIDUAL</td>
<td>1281.8777</td>
<td>24</td>
<td>53.4116</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Dataset: FIS

Dependent Variable: EFFORT: is the amount of time taken to rework the class for use in SLB

Model suggests that DIT and LCOM are significant predictors of rework while LINES is not.
Dataset: F1S

Dependent Variable: Effort: is the amount of time taken to rework the class for use in SLB measured in hours.

Model suggests that LCOM is also a significant predictor of rework effort. Note that LCOM and DIT are not correlated. Again as expected higher lack of cohesion implies greater amount of rework.

None of the other metrics are significant.
**EIGENVALUES OF UNIT SCALED X'X**

<table>
<thead>
<tr>
<th></th>
<th>1</th>
<th>2</th>
<th>3</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>2.10</td>
<td>0.85</td>
<td>0.04</td>
</tr>
</tbody>
</table>

**CONDITION INDICES**

<table>
<thead>
<tr>
<th></th>
<th>1</th>
<th>2</th>
<th>3</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>1.00</td>
<td>1.57</td>
<td>7.03</td>
</tr>
</tbody>
</table>

**VARIANCE PROPORTIONS**

<table>
<thead>
<tr>
<th></th>
<th>1</th>
<th>2</th>
<th>3</th>
</tr>
</thead>
<tbody>
<tr>
<td>CONSTANT</td>
<td>0.02</td>
<td>0.00</td>
<td>0.98</td>
</tr>
<tr>
<td>DIT</td>
<td>0.02</td>
<td>0.01</td>
<td>0.98</td>
</tr>
<tr>
<td>LCOM</td>
<td>0.05</td>
<td>0.92</td>
<td>0.03</td>
</tr>
</tbody>
</table>

Model:
Dependent Variable: EFFORT
Independent Variables: DIT, LCOM

System: FIS
Dependent variable: Reusaba: is the average of the reusability ratings of two experts. Both experts rated reusability of the classes on a scale of 1 to 9. The variable Reusaba is calculated as \(0.5 \times \text{reusel} + 0.5 \times \text{reuse2}\) where reusel and reuse2 are the two expert ratings (correlated 0.81 at a significance level of 0.001).

Independent variable: Lines of code

Model shows that Reusability is not predicted by lines of code.
**Dependent variable:** Reusaba is the average of the reusability ratings of two experts. Both experts rated reusability of the classes on a scale of 1 to 9. The variable Reusaba is calculated as = 0.5*reuse1 + 0.5*reuse2 where reuse1 and reuse2 are the two expert ratings (correlated 0.81 at a significance level of 0.001)

**Independent variable:** LCOM, RFC, DIT and LINES

Model shows that three independent metrics RFC, DIT and LCOM are significant predictors of reusability, but LINES is not. Note RFC, DIT and LCOM are not significantly correlated.
DEP VAR: REUSABA
N: 28
MULTIPLE R: 0.885
SQUARED MULTIPLE R: 0.783
ADJUSTED SQUARED MULTIPLE R: 0.756
STANDARD ERROR OF ESTIMATE: 1.15

<table>
<thead>
<tr>
<th>VARIABLE</th>
<th>COEFFICIENT</th>
<th>STD ERROR</th>
<th>STD COEFF</th>
<th>T</th>
<th>P(2 TAIL)</th>
</tr>
</thead>
<tbody>
<tr>
<td>CONSTANT</td>
<td>13.02</td>
<td>0.80</td>
<td>0.00</td>
<td>16.28</td>
<td>0.00</td>
</tr>
<tr>
<td>DIT</td>
<td>-4.43</td>
<td>0.64</td>
<td>-0.68</td>
<td>-6.93</td>
<td>0.00</td>
</tr>
<tr>
<td>RFC</td>
<td>-0.03</td>
<td>0.01</td>
<td>-0.31</td>
<td>-2.92</td>
<td>0.01</td>
</tr>
<tr>
<td>LCOM</td>
<td>-0.01</td>
<td>0.00</td>
<td>-0.39</td>
<td>-3.65</td>
<td>0.00</td>
</tr>
</tbody>
</table>

ANALYSIS OF VARIANCE

<table>
<thead>
<tr>
<th>SOURCE</th>
<th>SUM-OF-SQUARES</th>
<th>DF</th>
<th>MEAN-SQUARE</th>
<th>F-RATIO</th>
<th>P</th>
</tr>
</thead>
<tbody>
<tr>
<td>REGRESSION</td>
<td>115.26</td>
<td>3</td>
<td>38.42</td>
<td>28.83</td>
<td>0.00</td>
</tr>
<tr>
<td>RESIDUAL</td>
<td>31.98</td>
<td>24</td>
<td>1.33</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

System: FIS

Dependent variable: Reusaba: is the average of the reusability ratings of two experts. Both experts rated reusability of the classes on a scale of 1 to 9. The variable Reusaba is calculated as = 0.5*reusel + 0.5*reuse2 where reusel and reuse2 are the two expert ratings (correlated 0.81 at a significance level of 0.001)

Independent variable: LCOM, RFC and DIT

Model shows that three independent metrics RFC, DIT and LCOM are significant predictors of reusability. Note RFC, DIT and LCOM are not significantly correlated.
EIGENVALUES OF UNIT SCALED X'X

<table>
<thead>
<tr>
<th></th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>2.95</td>
<td>0.86</td>
<td>0.15</td>
<td>0.04</td>
</tr>
</tbody>
</table>

CONDITION INDICES

<table>
<thead>
<tr>
<th></th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>1.00</td>
<td>1.86</td>
<td>4.44</td>
<td>8.36</td>
</tr>
</tbody>
</table>

VARIANCE PROPORTIONS

<table>
<thead>
<tr>
<th></th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
</tr>
</thead>
<tbody>
<tr>
<td>CONSTANT</td>
<td>0.01</td>
<td>0.00</td>
<td>0.05</td>
<td>0.94</td>
</tr>
<tr>
<td>DIT</td>
<td>0.01</td>
<td>0.01</td>
<td>0.08</td>
<td>0.90</td>
</tr>
<tr>
<td>RFC</td>
<td>0.02</td>
<td>0.00</td>
<td>0.97</td>
<td>0.01</td>
</tr>
<tr>
<td>LCOM</td>
<td>0.02</td>
<td>0.74</td>
<td>0.22</td>
<td>0.01</td>
</tr>
</tbody>
</table>

**Model:**
Dependent Variable: REUSABA
Independent Variables: DIT, RFC, LCOM

**System:** FIS

**Dependent variable:** Reusaba:

*Model shows that three independent metrics RFC, DIT and LCOM are significant predictors of reusability. Note RFC, DIT and LCOM are not significantly correlated.*
### ANALYSIS OF VARIANCE

<table>
<thead>
<tr>
<th>SOURCE</th>
<th>SUM-OF-SQUARES</th>
<th>DF</th>
<th>MEAN-SQUARE</th>
<th>F-RATIO</th>
<th>P</th>
</tr>
</thead>
<tbody>
<tr>
<td>REgression</td>
<td>60.3211</td>
<td>1</td>
<td>60.3211</td>
<td>0.7418</td>
<td>0.3970</td>
</tr>
<tr>
<td>Residual</td>
<td>2114.2610</td>
<td>26</td>
<td>81.3177</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

### VARIABLE COEFFICIENT

<table>
<thead>
<tr>
<th>VARIABLE</th>
<th>COEFFICIENT</th>
<th>STD ERROR</th>
<th>STD COEF</th>
<th>TOLERANCE</th>
<th>T</th>
<th>P(2 TAIL)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Constant</td>
<td>5.6786</td>
<td>1.6360</td>
<td>0.0000</td>
<td>.9960</td>
<td>3.3482</td>
<td>0.0024</td>
</tr>
<tr>
<td>Effort</td>
<td>8.7285</td>
<td>3.9299</td>
<td>0.0000</td>
<td>.9960</td>
<td>2.2211</td>
<td>0.0353</td>
</tr>
</tbody>
</table>

### ANALYSIS OF VARIANCE

<table>
<thead>
<tr>
<th>SOURCE</th>
<th>SUM-OF-SQUARES</th>
<th>DF</th>
<th>MEAN-SQUARE</th>
<th>F-RATIO</th>
<th>P</th>
</tr>
</thead>
<tbody>
<tr>
<td>REgression</td>
<td>538.8379</td>
<td>2</td>
<td>269.4189</td>
<td>4.1177</td>
<td>0.0285</td>
</tr>
<tr>
<td>Residual</td>
<td>1635.7443</td>
<td>25</td>
<td>65.4298</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

### VARIABLE COEFFICIENT

<table>
<thead>
<tr>
<th>VARIABLE</th>
<th>COEFFICIENT</th>
<th>STD ERROR</th>
<th>STD COEF</th>
<th>TOLERANCE</th>
<th>T</th>
<th>P(2 TAIL)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Constant</td>
<td>-5.7500</td>
<td>6.4101</td>
<td>0.0000</td>
<td>.9960</td>
<td>-0.8970</td>
<td>0.3783</td>
</tr>
<tr>
<td>Effort</td>
<td>11.8789</td>
<td>4.3925</td>
<td>0.4717</td>
<td>99.91</td>
<td>2.7043</td>
<td>0.0121</td>
</tr>
</tbody>
</table>

### ANALYSIS OF VARIANCE

<table>
<thead>
<tr>
<th>SOURCE</th>
<th>SUM-OF-SQUARES</th>
<th>DF</th>
<th>MEAN-SQUARE</th>
<th>F-RATIO</th>
<th>P</th>
</tr>
</thead>
<tbody>
<tr>
<td>REgression</td>
<td>883.2049</td>
<td>3</td>
<td>294.4016</td>
<td>5.4714</td>
<td>0.0052</td>
</tr>
<tr>
<td>Residual</td>
<td>1635.7443</td>
<td>25</td>
<td>65.4298</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

### VARIABLE COEFFICIENT

<table>
<thead>
<tr>
<th>VARIABLE</th>
<th>COEFFICIENT</th>
<th>STD ERROR</th>
<th>STD COEF</th>
<th>TOLERANCE</th>
<th>T</th>
<th>P(2 TAIL)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Constant</td>
<td>-5.5714</td>
<td>5.8134</td>
<td>0.0000</td>
<td>.9960</td>
<td>-0.9584</td>
<td>0.3474</td>
</tr>
<tr>
<td>Effort</td>
<td>12.8083</td>
<td>4.0003</td>
<td>0.5086</td>
<td>99.807</td>
<td>3.2019</td>
<td>0.0038</td>
</tr>
<tr>
<td>LCOM</td>
<td>0.0425</td>
<td>0.0168</td>
<td>0.4226</td>
<td>88.868</td>
<td>2.5298</td>
<td>0.0184</td>
</tr>
<tr>
<td>SQ_LINES</td>
<td>-0.0004</td>
<td>0.0003</td>
<td>-0.2508</td>
<td>89.00</td>
<td>-1.5040</td>
<td>0.1456</td>
</tr>
</tbody>
</table>
None of the above models show any possible quadratic relationship between LINES and the dependent variables.
**PEARSON CORRELATION MATRIX**

<table>
<thead>
<tr>
<th></th>
<th>WMC</th>
<th>DIT</th>
<th>NOC</th>
<th>RFC</th>
<th>CBO</th>
</tr>
</thead>
<tbody>
<tr>
<td>WMC</td>
<td>1.00</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>DIT</td>
<td>0.53</td>
<td>1.00</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>NOC</td>
<td>-0.17</td>
<td>-0.29</td>
<td>1.00</td>
<td></td>
<td></td>
</tr>
<tr>
<td>RFC</td>
<td>0.95</td>
<td>0.43</td>
<td>-0.22</td>
<td>1.00</td>
<td></td>
</tr>
<tr>
<td>CBO</td>
<td>0.86</td>
<td>0.38</td>
<td>-0.23</td>
<td>0.95</td>
<td>1.00</td>
</tr>
<tr>
<td>LCOM</td>
<td>0.81</td>
<td>0.26</td>
<td>-0.09</td>
<td>0.75</td>
<td>0.60</td>
</tr>
<tr>
<td>EFFORT</td>
<td>0.32</td>
<td>-0.28</td>
<td>-0.20</td>
<td>0.43</td>
<td>0.42</td>
</tr>
</tbody>
</table>

**LCOM   EFFORT**

<table>
<thead>
<tr>
<th></th>
<th>LCOM</th>
<th>EFFORT</th>
</tr>
</thead>
<tbody>
<tr>
<td>LCOM</td>
<td>1.00</td>
<td></td>
</tr>
<tr>
<td>EFFORT</td>
<td>0.30</td>
<td>1.00</td>
</tr>
</tbody>
</table>

**BARTLETT CHI-SQUARE STATISTIC:** 228.962  **DF:** 28  **PROB:** 0.000

**MATRIX OF BONFERRONI PROBABILITIES**

<table>
<thead>
<tr>
<th></th>
<th>WMC</th>
<th>DIT</th>
<th>NOC</th>
<th>RFC</th>
<th>CBO</th>
</tr>
</thead>
<tbody>
<tr>
<td>WMC</td>
<td>0.00</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>DIT</td>
<td>0.12</td>
<td>0.00</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>NOC</td>
<td>1.00</td>
<td>1.00</td>
<td>0.00</td>
<td></td>
<td></td>
</tr>
<tr>
<td>RFC</td>
<td>0.00</td>
<td>0.73</td>
<td>1.00</td>
<td>0.00</td>
<td></td>
</tr>
<tr>
<td>CBO</td>
<td>0.00</td>
<td>1.00</td>
<td>1.00</td>
<td>0.00</td>
<td>0.00</td>
</tr>
<tr>
<td>LCOM</td>
<td>0.00</td>
<td>1.00</td>
<td>1.00</td>
<td>0.00</td>
<td>0.03</td>
</tr>
<tr>
<td>EFFORT</td>
<td>1.00</td>
<td>1.00</td>
<td>1.00</td>
<td>0.74</td>
<td>0.83</td>
</tr>
</tbody>
</table>

**LCOM   EFFORT**

<table>
<thead>
<tr>
<th></th>
<th>LCOM</th>
<th>EFFORT</th>
</tr>
</thead>
<tbody>
<tr>
<td>LCOM</td>
<td>0.00</td>
<td></td>
</tr>
<tr>
<td>EFFORT</td>
<td>1.00</td>
<td>0.00</td>
</tr>
</tbody>
</table>

**NUMBER OF OBSERVATIONS:** 27
### First Regression

**DEP VAR:** EFFORT  
**N:** 27  
**MULTIPLE R:** 0.614  
**SQUARED MULTIPLE R:** 0.378  
**ADJUSTED SQUARED MULTIPLE R:** 0.326  
**STANDARD ERROR OF ESTIMATE:** 9.33

<table>
<thead>
<tr>
<th>VARIABLE</th>
<th>COEFFICIENT</th>
<th>STD ERROR</th>
<th>STD COEF</th>
<th>T</th>
<th>P(2 TAIL)</th>
</tr>
</thead>
<tbody>
<tr>
<td>CONSTANT</td>
<td>12.03</td>
<td>5.00</td>
<td>0.00</td>
<td>2.40</td>
<td>0.02</td>
</tr>
<tr>
<td>WMC</td>
<td>1.95</td>
<td>0.57</td>
<td>0.65</td>
<td>3.41</td>
<td>0.00</td>
</tr>
<tr>
<td>DIT</td>
<td>-9.41</td>
<td>2.88</td>
<td>-0.62</td>
<td>-3.27</td>
<td>0.00</td>
</tr>
</tbody>
</table>

**ANALYSIS OF VARIANCE**

<table>
<thead>
<tr>
<th>SOURCE</th>
<th>SUM-OF-SQUARES</th>
<th>DF</th>
<th>MEAN-SQUARE</th>
<th>F-RATIO</th>
<th>P</th>
</tr>
</thead>
<tbody>
<tr>
<td>REGRESSION</td>
<td>1268.15</td>
<td>2</td>
<td>634.07</td>
<td>7.28</td>
<td>0.00</td>
</tr>
<tr>
<td>RESIDUAL</td>
<td>2090.46</td>
<td>24</td>
<td>87.10</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

*Note: DIT and WMC are not significantly correlated*

### Second Regression

**DEP VAR:** EFFORT  
**N:** 27  
**MULTIPLE R:** 0.662  
**SQUARED MULTIPLE R:** 0.438  
**ADJUSTED SQUARED MULTIPLE R:** 0.391  
**STANDARD ERROR OF ESTIMATE:** 8.87

<table>
<thead>
<tr>
<th>VARIABLE</th>
<th>COEFFICIENT</th>
<th>STD ERROR</th>
<th>STD COEF</th>
<th>T</th>
<th>P(2 TAIL)</th>
</tr>
</thead>
<tbody>
<tr>
<td>CONSTANT</td>
<td>13.92</td>
<td>4.70</td>
<td>0.00</td>
<td>2.96</td>
<td>0.01</td>
</tr>
<tr>
<td>DIT</td>
<td>-8.47</td>
<td>2.56</td>
<td>-0.56</td>
<td>-3.31</td>
<td>0.00</td>
</tr>
<tr>
<td>RFC</td>
<td>0.91</td>
<td>0.23</td>
<td>0.67</td>
<td>3.93</td>
<td>0.00</td>
</tr>
</tbody>
</table>

**ANALYSIS OF VARIANCE**

<table>
<thead>
<tr>
<th>SOURCE</th>
<th>SUM-OF-SQUARES</th>
<th>DF</th>
<th>MEAN-SQUARE</th>
<th>F-RATIO</th>
<th>P</th>
</tr>
</thead>
<tbody>
<tr>
<td>REGRESSION</td>
<td>1471.94</td>
<td>2</td>
<td>735.97</td>
<td>9.36</td>
<td>0.00</td>
</tr>
<tr>
<td>RESIDUAL</td>
<td>1886.66</td>
<td>24</td>
<td>78.61</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

*Note: DIT and RFC are not significantly correlated*
DEP VAR: EFFORT  N: 27  MULTIPLE R: 0.631  SQUARED MULTIPLE R: 0.398
ADJUSTED SQUARED MULTIPLE R: 0.347  STANDARD ERROR OF ESTIMATE: 9.18

<table>
<thead>
<tr>
<th>VARIABLE</th>
<th>COEFFICIENT</th>
<th>STD ERROR</th>
<th>STD COEF</th>
<th>TOLERANCE</th>
<th>T</th>
<th>P(2 TAIL)</th>
</tr>
</thead>
<tbody>
<tr>
<td>CONSTANT</td>
<td>17.09</td>
<td>4.91</td>
<td>0.00</td>
<td>.</td>
<td>3.48</td>
<td>0.00</td>
</tr>
<tr>
<td>DIT</td>
<td>-7.71</td>
<td>2.59</td>
<td>-0.51</td>
<td>0.85</td>
<td>-2.97</td>
<td>0.01</td>
</tr>
<tr>
<td>CBO</td>
<td>3.25</td>
<td>0.91</td>
<td>0.61</td>
<td>0.85</td>
<td>3.58</td>
<td>0.00</td>
</tr>
</tbody>
</table>

**ANALYSIS OF VARIANCE**

<table>
<thead>
<tr>
<th>SOURCE</th>
<th>SUM-OF-SQUARES</th>
<th>DF</th>
<th>MEAN-SQUARE</th>
<th>F-RATIO</th>
<th>P</th>
</tr>
</thead>
<tbody>
<tr>
<td>REGRESSION</td>
<td>1335.21</td>
<td>2</td>
<td>667.60</td>
<td>7.92</td>
<td>0.00</td>
</tr>
<tr>
<td>RESIDUAL</td>
<td>2023.40</td>
<td>24</td>
<td>84.31</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Note: DIT and CBO are not significantly correlated

DEP VAR: EFFORT  N: 27  MULTIPLE R: 0.476  SQUARED MULTIPLE R: 0.227
ADJUSTED SQUARED MULTIPLE R: 0.162  STANDARD ERROR OF ESTIMATE: 10.40

<table>
<thead>
<tr>
<th>VARIABLE</th>
<th>COEFFICIENT</th>
<th>STD ERROR</th>
<th>STD COEF</th>
<th>TOLERANCE</th>
<th>T</th>
<th>P(2 TAIL)</th>
</tr>
</thead>
<tbody>
<tr>
<td>CONSTANT</td>
<td>16.62</td>
<td>5.58</td>
<td>0.00</td>
<td>.</td>
<td>2.98</td>
<td>0.01</td>
</tr>
<tr>
<td>DIT</td>
<td>-5.76</td>
<td>2.81</td>
<td>-0.38</td>
<td>0.93</td>
<td>-2.05</td>
<td>0.05</td>
</tr>
<tr>
<td>LCOM</td>
<td>0.29</td>
<td>0.13</td>
<td>0.40</td>
<td>0.93</td>
<td>2.16</td>
<td>0.04</td>
</tr>
</tbody>
</table>

**ANALYSIS OF VARIANCE**

<table>
<thead>
<tr>
<th>SOURCE</th>
<th>SUM-OF-SQUARES</th>
<th>DF</th>
<th>MEAN-SQUARE</th>
<th>F-RATIO</th>
<th>P</th>
</tr>
</thead>
<tbody>
<tr>
<td>REGRESSION</td>
<td>761.21</td>
<td>2</td>
<td>380.61</td>
<td>3.52</td>
<td>0.05</td>
</tr>
<tr>
<td>RESIDUAL</td>
<td>2597.40</td>
<td>24</td>
<td>108.22</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Note: DIT and LCOM are not significantly correlated
Note: NOC metric is not a significant predictor
Note: Best model is RFC and DIT
Note: In all cases the coefficient of DIT is much larger than the other predictors
### Eigenvalues of Unit Scaled $X'X$

<table>
<thead>
<tr>
<th></th>
<th>1</th>
<th>2</th>
<th>3</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>2.68</td>
<td>0.26</td>
<td>0.06</td>
</tr>
</tbody>
</table>

### Condition Indices

<table>
<thead>
<tr>
<th></th>
<th>1</th>
<th>2</th>
<th>3</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>1.00</td>
<td>3.24</td>
<td>6.46</td>
</tr>
</tbody>
</table>

### Variance Proportions

<table>
<thead>
<tr>
<th></th>
<th>1</th>
<th>2</th>
<th>3</th>
</tr>
</thead>
<tbody>
<tr>
<td>Constant</td>
<td>0.02</td>
<td>0.12</td>
<td>0.86</td>
</tr>
<tr>
<td>DIT</td>
<td>0.01</td>
<td>0.03</td>
<td>0.95</td>
</tr>
<tr>
<td>RPC</td>
<td>0.04</td>
<td>0.88</td>
<td>0.08</td>
</tr>
</tbody>
</table>

*Note: In all other models as well, the condition indices are well below 15*
Consolidated Bibliography


Myers, G. J. (1977) "An extension to the cyclomatic measure of program complexity." *SIGPLAN Notices*, 12, 10, 61-64.


