AN EMPIRICAL STUDY RELATING OBJECT-ORIENTED METHOD TO SOFTWARE PRODUCTIVITY IN MILITARY ORGANIZATIONS

By Wilson Rosa

B.S. in Mechanical Engineering, June 1995, University of Puerto Rico

Master of Engineering Management, May 2001, George Washington University

A Dissertation submitted to:

the Faculty of
The School of Engineering and Applied Science
of The George Washington University
in partial satisfaction of the requirements
for the Degree of Doctor of Philosophy

January 31, 2010

Dissertation Directed by:

Theresa I. Jefferson
Associate Professor of Engineering Management and Systems Engineering

The School of Engineering Applied Science of The George Washington University certifies that Wilson Rosa has passed the Final Examination for the degree of Doctor of Philosophy as of October 16, 2009. This is the final and approved form of the dissertation.

AN EMPIRICAL STUDY RELATING OBJECT-ORIENTED METHOD TO SOFTWARE PRODUCTIVITY IN MILITARY ORGANIZATIONS

By Wilson Rosa

Dissertation Research Committee:

Theresa L. Jefferson, Associate Professor of Engineering Management and Systems Engineering, Dissertation Director

Howard Eisner, Professor of Engineering Management & Distinguished Research Professor, Committee Member

Michael R. Duffey, Associate Professor of Engineering Management, Committee Member

John R. Harrald, Professor Emeritus of Engineering Management, Committee Member

Bradford Clark, Senior Consultant for Software Metrics, Inc., Committee Member

© **Copyright 2009** by Wilson Rosa
All rights reserved

Dedication

This paper is dedicated to my beloved wife, **Maria Teresa Rivera-Torres,** adorable son, **Wilson Gabriel,** and my blessed unborn daughter, **Gianna Alexandra**. Your love and charm inspired me to finish my dissertation. God bless!

Acknowledgements

I want like to thank the ten sources that provided data for this research. Unfortunately, I am unable disclose their names because of confidential information agreements. I also thank my academic advisor, Dr. Theresa I. Jefferson, for her mentorship and for being patient throughout these seven years.

In addition, I thank my father, **Roberto Rosa,** and mother, **Concepcion Cuevas**, for their encouraging words and prayers throughout these seven years.

Abstract of Dissertation

An Empirical Study Relating Object Oriented Method to Software Productivity in Military Organizations

The purpose of this study was to determine whether military organizations are achieving productivity gains by using the object-oriented method. Twenty projects from ten military organizations were used in the study. Each organization provided data on two homogeneous software projects—one implemented in object-oriented methods and one implemented by using traditional methods. Each pair was similar in size, operating environment, application domain, complexity, personnel capability, and reliability. These control variables were chosen on the basis of their impact on productivity. Results showed that all ten organizations achieved a productivity gain with object-oriented methods and that the effect of the object-oriented method on software productivity is statistically significant.

Table of Contents

Dedication iv

Aknowledgements v

Abstract of Dissertation vi

Table of Contents vii

List of Figures viii

List of Tables ix

List of Symbols xi

List of Equations xvii

1 Introduction 1

2 Literature Review: Object-Oriented Methods 9

3 Literature Review: Software Productivity 18

4 Literature Review: Control Variables 23

5 Literature Review: Impact of Object-Oriented Methods on Productivity 29

6 Significance of the Proposed Study 37

7 Research Method 39

8 Data Analysis 62

9 Conclusion 80

10 References 85

11 Appendices 93

List of Figures

Figure 1-1 Use of Object-Oriented Methods in the Department of Defense 2

Figure 2-1 Studies of Object-Oriented Benefits on Software Productivity 13

Figure 2-2DoD software project comparison by object-oriented method..... 17

Figure 8-1 Unadjusted productivity comparison by object-oriented method 63

Figure 8-2 Adjusted productivity comparison by object-oriented method..... 64

Figure 8-3 Productivity averages by programming language 65

Figure 8-4 Productivity by application domain 66

Figure 8-5 Defect containment by object-oriented method by organization 67

Figure 8-6 Histogram of the residuals for Hypothesis 1 (untransformed) 68

Figure 8-7 Normal residual plot for Hypothesis 1 (untransformed) 69

Figure 8-8 Histogram of the residual for Hypothesis 2 (untransformed) 70

Figure 8-9 Histogram of the residuals for Hypothesis 2(transformed)..... 71

Figure 8-10 Histogram of the residuals (untransformed) 72

Figure 8-11 Normal plot of the residuals (untransformed)..... 73

List of Tables

Table 2-1 Studies of Object-Oriented Benefits on Software Productivity..... 13

Table 2-2Studies of Benefits of Object Oriented on Software Quality..... 14

Table 2-3: Object-oriented applications..... 15

Table 7-1 Elements of object-oriented method..... 40

Table 7-2 Object-oriented method rating scale definitions..... 41

Table 7-3 Object-oriented method rating scale values (0-1)..... 41

Table 7-4 Taxonomy operating environment..... 42

Table 7-5 Taxonomy application domain 43

Table 7-6 Ratings for Complexity Factor..... 45

Table 7-7 Ratings for personnel capability factor..... 46

Table 7-8 Ratings for reliability factor 46

Table 7-9 Ratings for QUAL Factor..... 47

Table 7-10 Ratings for platform volatility factor..... 47

Table 7-11 Project's Context Information..... 49

Table 7-12 Project's Raw Size, Productivity, and Productivity Drivers..... 50

Table 7-13 ESLOC→UFP Conversion Ratios..... 57

Table 7-14 Design matrix for Hypothesis 1..... 60

Table 7-15 Design matrix for Hypothesis 2..... 61

Table 7-16 Design matrix for Hypothesis 3..... 62

Table 8-1 Correlation coefficient..... 75

Table 8-2 Correlation: *P* Values 76

Table 8-3 ANOVA: productivity versus object-oriented method..... 77

Table 8-4 GLIM: productivity versus object-oriented method and complexity..... 78

Table 8-5 Two-way ANOVA: productivity versus OOM and application domain . 79

List of Symbols

Acronym	Definition
ACAP	“analyst capability” [8]
AD	application domain
Adj.	adjusted
ANOVA	analysis of variance
APEX	“application experience”[8]
ASLOC	“adapted source lines of code” [8]
AV	avionics
C2	command and control
C3	communications
CD	controls and displays
CLOS	common lisp object system
CM	“code modified” [8]
CMMI	capability maturity model integration
COCOMO	“COnstructive COst Model” [8]
CPLX	“Product complexity” [8]
DATA	“database size” [8]

Acronym	Definition
DB	“database” [8]
DF	degrees of freedom
DM	“design modified” [8]
DOCU	“documentation” [8]
ESA	European Space Agency
ESLOC	“equivalent source lines of code” [8]
EW	electronics warfare
EXE	executive
FLEX	Development flexibility
GLIM	general linear model
GUI	graphics user interface
H	high
HUD	Heads-Up Display
IEEE	“Institute of Electrical and Electronics Engineers” [1]
ILS	logistics
IM	“integration modified” [8]
ISBSG	International Software Benchmarking Standards Group
KSLOC	“thousand source lines of code” [8]

Acronym	Definition
L	“low” [8]
LTEX	“language and tool experience” [8]
MG	manned ground
MIS	management information systems
MM	mission management
MMOG	military mobile
MODP	modern development practice
MP	mission planning
MS	manned space
MS	“Mean square”
MSLOC	million source lines of code
MUA	missile and unmanned airborne
N	“nominal” [8]
N+	“nominal+” [30]
NSLOC	new lines of code
OE	operating environment
OO	object-oriented
OOM	object-oriented method

Acronym	Definition
PC	personnel computer
PCAP	“programmer capability” [8]
PCON	“personnel continuity” [8]
PERS	“Personnel capability” [8]
PLAT	“platform” [8]
PLEX	“platform experience” [8]
PMAT	“Process maturity” [8]
PP	“product's productivity”
PREC	“precedenteness” [8]
PROC	process control
PVOL	“platform volatility” [8]
PY	payload
QA	quality assurance
QUAL	software quality
RAD	“rapid application development”
RELY	“reliability” [8]
RESL	“Architecture / risk resolution” [8]
RUSE	“developed for reusability” [8]

Acronym	Definition
SA	situation awareness
SAM	software analysis model
SB	shipboard
SCED	“required development schedule” [8]
SDM	software design model
SIM	simulations and modeling
SITE	multisite development
SIZE	product size
SLIM	software lifecycle model
SLOC	“source lines of code” [8]
SM	staff-months
SNR	sonar
SPM	software programming method
SS	“Sum of squares”
TEAM	“team cohesion” [8]
TEL	telecommunications
TRNG	training
UFP	unadjusted function point

Acronym	Definition
UG	unmanned ground
US	unmanned space
VH	“very high” [8]
VH+	“very high+” [8]
VIRT	“virtual machine constraint” [6]
VL	“very low” [8]
WD	weapons delivery
XH	“extra high” [8]
XL	“extra low” [8]

List of Equations

EQUATION 1.1 Equivalent Source Lines of Code (ESLOC) 56

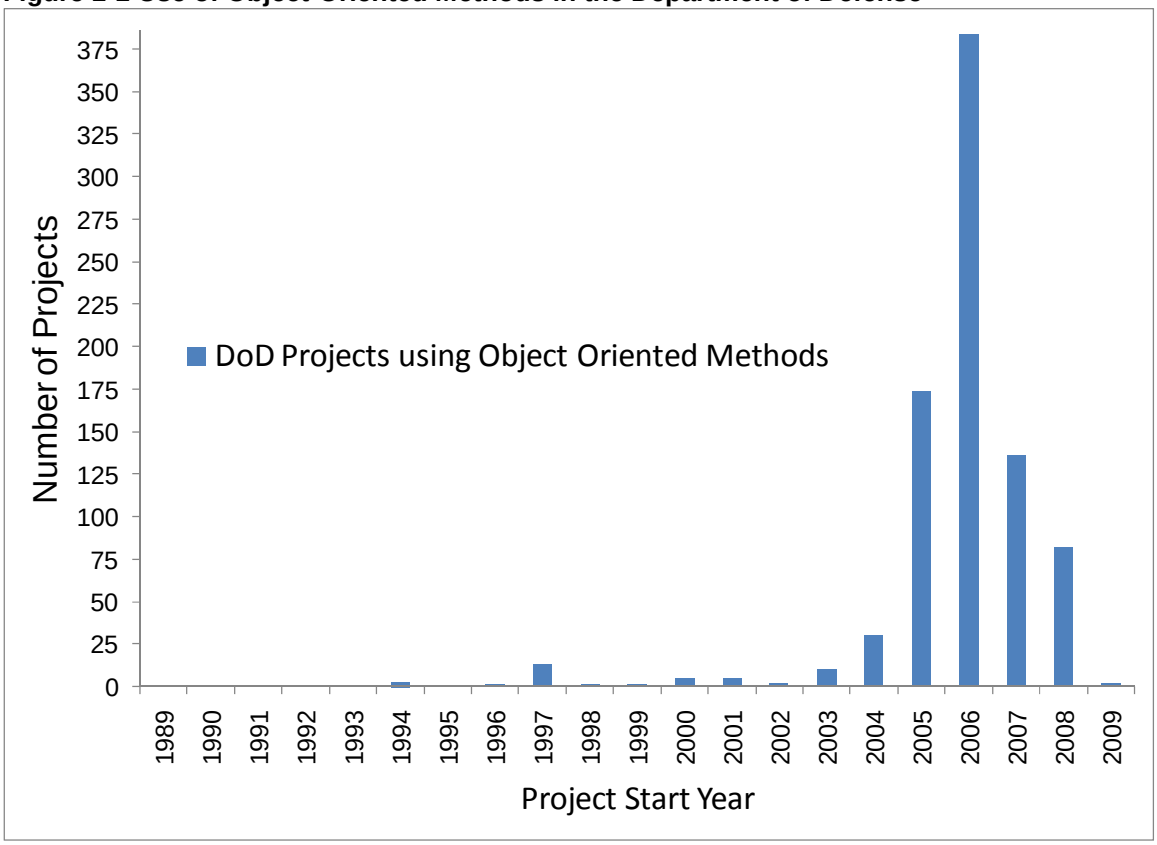
EQUATION 1.2 Product's Productivity (PP)..... 58

1 Introduction

1.1 Problem Statement

Modern software development methods are changing the way military organizations implement software products today. Consequently, these organizations are beginning to realize the benefits of object-oriented methods (Figure 1-1). Emerging military capabilities, such as the United States Army's Future Combat System and the United States Air Force's Single Integrated Air Picture, suggest the need for improved capabilities to accurately estimate software effort and schedule.

Figure 1-1 Use of Object-Oriented Methods in the Department of Defense



During the past seven years, military organizations have underestimated software costs by an average of 37 percent [75]. The lack of recent empirical research and high dependence on proprietary software costing tools drives this estimating error. Although some of the existing tools are addressing this issue, new software development processes and methods have not been strongly matched to enhanced costing models. Modern military weapon systems cannot be predicted by using available software costing tools because their datasets are based on third-generation projects (written in third-generation programming language like Ada 83 and “Jovial”) captured during the late 1980s and mid-1990s.

1.2 Purpose Statement

The purpose of this empirical study was to determine whether military organizations are achieving productivity gains with object-oriented methods. This study examined ten military organizations that have developed military weapon systems for the U.S. Department of Defense. Each organization provided data on two homogeneous software projects—one implemented in object-oriented methods and one implemented in traditional methods.

In this study, the independent variable is the object-oriented method. The dependent variable is productivity, defined as the unadjusted function point (UFP) (a derived size measure) divided by the staff-month. The control variables in this study are the factors that account for the most variance in software productivity—size, operating environment, application domain, complexity, personnel capability, and reliability.

1.3 Deficiencies in Past Studies

Despite increased interest in object-oriented method, few empirical research studies have been conducted relating object-oriented methods to productivity. Of the seven studies relating object-oriented methods to software productivity, only five have been based on empirical data. These five studies, however, had limitations in the sampling procedure, intervening variables, and logic for measuring and comparing software development productivities.

1.4 Significance of Proposed Study

This study will remedy these limitations in three distinct ways:

1. Collect pairs of projects across multiple organizations and not from a single organization.
2. Control for the impact of those variables that account for the most variance in software productivity by asking each organization to provide data on two homogenous projects.
3. Calculate productivity according to best practices set forth by the IEEE Standard (IEEE, 1993, IEEE Std 1045-1992). This includes adjusting reused code to its equivalent in new code and then into function points in order to make cross-language productivity comparisons.

This study is important for several reasons. First, such a study can help reveal whether object-oriented method is appropriate for military weapon systems since these are more constrained (higher quality assurance levels, rigid policies) than commercial software. Second, understanding the relationships between objected-oriented methods, software productivity, and other cost drivers can encourage researchers to explore new strategies for controlling software productivity and quality. Third, the collected data can be used for future software cost estimating research.

1.5 Hypotheses

This section is split into two. The first identifies the original hypotheses included in the dissertation proposal. The second section provides the rationale for changing the original hypotheses. The third section shows the final hypotheses used in this study.

1.5.1 Original Hypotheses

The following 11 hypotheses were identified in the dissertation proposal:

Hypothesis 1A: Do object-oriented methods relate to software productivity gain?

Hypothesis 2A: Do object-oriented methods and COCOMO (COnstructive COst Model) II scale factors relate to software productivity gain?

Hypothesis 3A: Do object-oriented methods and COCOMO II product factors relate to software productivity gain?

Hypothesis 4A: Do object-oriented methods and COCOMO II platform factors relate to software productivity gain?

Hypothesis 5A: Do object-oriented methods and COCOMO II personnel factors relate to software productivity gain?

Hypothesis 6A: Do object-oriented methods and COCOMO II project factors relate to software productivity gain?

Hypothesis 7A: Do object-oriented methods relate to COCOMO II scale factors?

Hypothesis 8A: Do object-oriented methods relate to COCOMO II product factors?

Hypothesis 9A: Do object-oriented methods relate to COCOMO II platform factors?

Hypothesis 10A: Do object-oriented methods relate to COCOMO II personnel factors?

Hypothesis 11A: Do object-oriented methods relate to COCOMO II project factors?

1.5.2 Rationale for changing the Hypotheses

Hypotheses 2A and 4A through 6A were deleted because at least one factor did not meet the terms for analysis of variance (ANOVA). That is, the number of data points from each treatment was not balanced.

Of the four COCOMO Product Factors, only two (complexity and reliability) account for a significant variance in productivity. However, the factor "reliability" was not appropriate for ANOVA because the number of data points from each treatment was not balanced. Thus, Hypothesis 3A was changed to "Do object-oriented methods and complexity relate to software productivity?"

Hypotheses 7A through 11A were deleted because they are examined by using Pearson product moment correlation coefficient.

Since application domain and quality also relates to software productivity, two new hypotheses were added to the study.

1.5.3 Revised Hypotheses

The revised hypotheses are identified below:

Hypothesis 1: Do object-oriented methods relate to software productivity?

Hypothesis 2: Do object-oriented methods and complexity relate to software productivity?

Hypothesis 3: Do object-oriented methods and application domain relate to software productivity?

1.6 Organization

This dissertation paper is organized into 10 chapters:

- Chapter 1 introduces the problem, deficiencies in past studies, proposed solution, and research questions.

- Chapter 2 summarizes the literature review for the independent variable, object-oriented method. It describes the origin and evolution, basic principles, key elements, perceived benefits, and applications.

- Chapter 3 is a literature review of the dependent variable, software productivity. It defines rules for measuring productivity and addresses the effectiveness of two productivity measures.

- Chapter 4 is a literature review of the control variables. It identifies the cost drivers affecting software productivity and highlights important findings.

- Chapter 5 examines the scholarly literature relating object-oriented methods to software productivity and highlights their strengths and weaknesses.

- Chapter 6 highlights the deficiencies found in past studies and how this study will address them.

- Chapter 7 goes over the research method step by step. It briefly summarizes the variables used in the study, explains the collected data and instrumentation, steps for calculating the independent variable, normality test procedures, research hypotheses, and experimental design.

- Chapter 8 discusses the study results, including project similarities, descriptive statistics, normality tests, statistical analyses, and significant findings.

- Chapter 9 presents the research conclusions on the basis of the hypotheses, software quality, and software productivity drivers. It also highlights the limitations, contributions, and further research to follow up on the results.

- Chapter 10 cites the sources used in the dissertation paper.

2 Literature Review: Object-Oriented Methods

This chapter provides the literature for the independent variable, object-oriented methods. The first section explains the origins. The second section lists the four principles. The third section identifies the key elements. The fourth section explains the primary benefits. The fifth section lists the most common applications. The sixth section identifies the object-oriented constraints in the Department of Defense (DoD). The seventh section displays the object-oriented trends in DoD.

2.1 Origin

Software development methods emerged in the 1970s (Booch 1994, 18). Yourdon's "Structured Analysis and Structured Design" method, developed in 1979, was the most significant and popular of these early methods (Rumbaugh et al. 2005, 4). This method became very popular in the Department of Defense, particularly with unmanned space and airborne applications, due to the "Waterfall" paradigm.

As Booch (2004, 19) points out, most software development methods are either "structured . . ., data-driven . . ., [or] object-oriented." The structured method is based on "algorithmic decomposition" (16). The data-driven method maps user inputs to system outputs (19).

The object-oriented technology initiated with “Simula-67” programming language in 1967. Emerging programming languages like “Smalltalk” and “Objective C” served as the vehicle for the wide acceptance in the 1980s (Rumbaugh et al. 2005, 5).

2.2 Principles

A model to be considered object-oriented must include “abstraction, encapsulation, modularity, [and] hierarchy” (Booch 1994, 27). Abstraction displays the relevant attributes of an object while hiding other that is unimportant (49). Abstraction is a design approach that adds modularity to an application (49).

Encapsulation separates the components of an abstraction (Booch 1994, 50). Encapsulation is realized by hiding the attributes and methods of an object (49). It decomposes a software product into encapsulated units, allowing for ease of design, control, and support (49).

Modularity encloses the abstracted objects into unified and “loosely coupled” units (Booch 1994, 55). This principle can also reduce the software’s design complexity by creating boundaries within the application (54).

As Booch (1994, 59) points out, “Inheritance is the most important ‘is a’ hierarchy”. Inheritance is the principle of sharing the composition and **methods** in one or more classes. It allows for creating new classes from preexisting (59). This principle also allows for software code reuse.

2.3 Elements

The object-oriented method includes three key elements:

Process (Booch 1994, 23)

Notation [programming language] (23)

Tools (23)

An object-oriented process is comprised of analysis and design activities. Object-oriented analysis defines software requirement specifications in terms objects and classes [10]. Object-oriented design uses modularity to describe logical and physical diagrams of the application.

Object-oriented programming is the principle of organizing software programs as groups of objects. According to Booch (2004, 38), a programming language to be considered object-oriented must support the concept of "objects . . ., classes . . ., [and] inheritance." A programming language without inheritance is considered "object-based" (39). A programming language that satisfies none of the above concepts is considered structured.

Object-oriented tools provide the means for supporting analysis, design, and programming activities. According to Booch (1994) there are six types of object-oriented tools:

- "Graphics-based system" used for project planning (Booch 1994, 283).
- "Browser" to easily search for abstractions, classes, and objects (283).

- "Incremental compiler" to collect data "declarations" and comment "statements" (283).
- "Class librarian" to easily find and store information about classes, objects, and modules (284).
- "GUI builder" creates "dialogs and windows" for the users (284).
- "Debugger" to control and monitor system performance (284).

In summary, object-oriented analysis, design, programming, and tools are interconnected. The results of object-oriented analysis serve as the models for design. The results of object-oriented design serve as blueprints for programming [10]. Object-oriented tools allow for analysis, design, and programming efficiencies.

2.4 Benefits

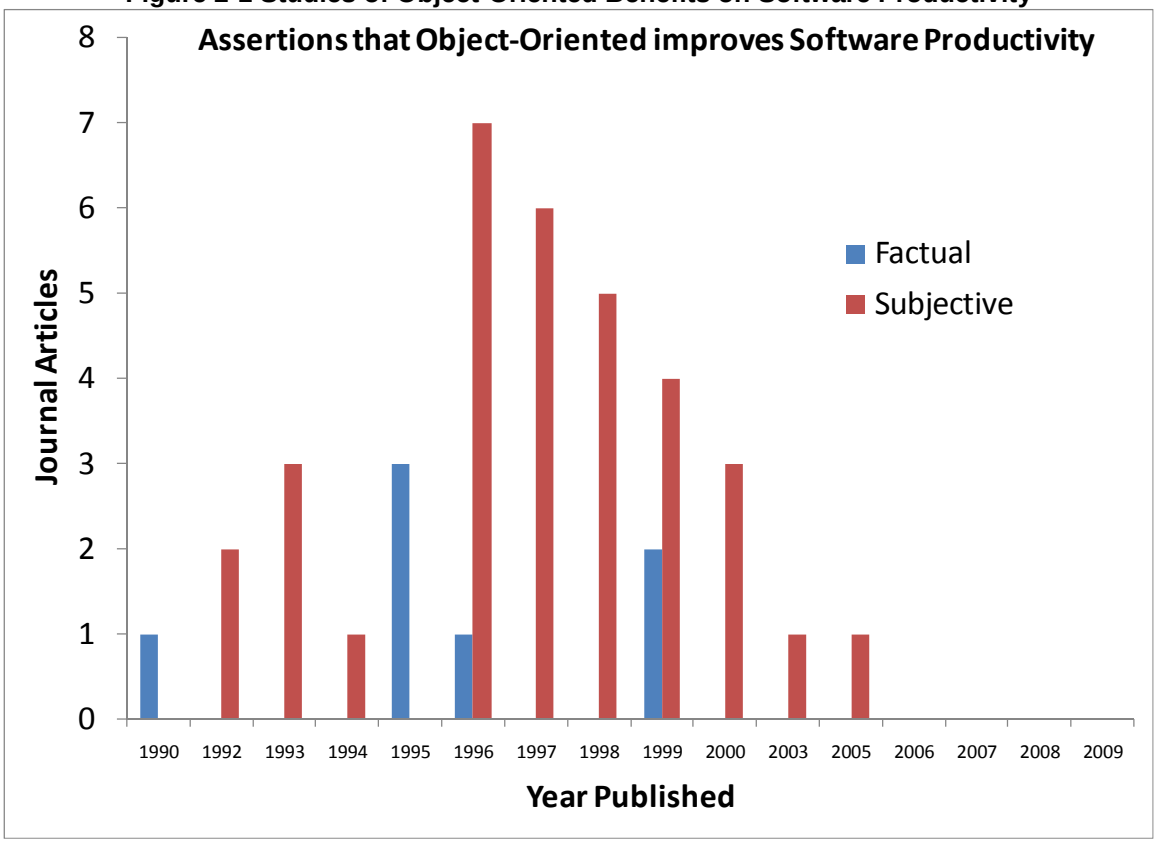
According to Booch (2004), there are two perceived benefits from the object-oriented technology. First, the use of object-oriented methods can help engineers achieve productivity gains during the analysis, design, and programming phase of a product. Second, the use of object-oriented methods can dramatically improve software quality.

Forty studies of object-oriented benefits indicate that software productivity can be improved in any of five ways (Table 2-1). Of these 40 studies, however, only seven had been based on empirical research (Figure 2-1). Despite the increased interest of object-orientation technology in the military sector, researchers have stopped studying its influence on software productivity over the last four years.

Table 2-1 Studies of Object-Oriented Benefits on Software Productivity

Group	Research Finding
1	A small set of studies suggests that object-oriented processes coupled with object-oriented programming increase productivity by reducing coding time [31,52,26,69,34,61,59,86].
2	A larger group of studies finds that object-oriented programming language increases productivity through encapsulation and inheritance [49,10,15,20,57,74,22,33,83].
3	A considerably larger set of studies shows that providing a mechanism to reuse processes from one project to another increases productivity [35,5,24,48,27,84,82,35,67,11,42,12,21,78,77,14,60].
4	A fourth set of studies indicates that object-oriented design can increase productivity by adopting a modular approach to encourage teamwork [44,80,3,70,5].
5	A single study concludes that object-oriented analysis can improve productivity by providing a model for identifying incomplete requirements early in the development phase [28].

Figure 2-1 Studies of Object-Oriented Benefits on Software Productivity



Studies of object-oriented benefits (Table 2-2) indicate an improvement in software quality in any of two ways:

Table 2-2Studies of Benefits of Object Oriented on Software Quality

Group	Research Finding
1	One set of studies provides evidence of an increase in software quality through a reduction in the percentage of lines of code changed for perfective enhancement, defect removal, and adaptive modification [50,43,53].
2	A single study claims software quality can be improved by decreasing the number of defects found during the test phase [5].

2.5 Applications

This section addresses the applications of object-oriented methods using two distinct approaches. The first approach uses peer-reviewed articles collected from three different sources. The second approach examines empirical data provided by a widely known research organization. These two approaches are explained next.

Booch (1994) categorized and grouped 188 journal articles by application and found that the object-oriented method is applicable to 15 application domains. These domains are avionics, command and control, database, engineering and science, graphic, intelligence, management information systems, operating systems, planning and analysis, process control, real-time, simulation, signal processing, software development tools, and communications.

In this comparison study, the applications of object-oriented methods were addressed by examining empirical data provided by The International Software Benchmarking

Standards Group [89] through a license agreement with the author on 14 March 2007. In this study, 4,106 projects were initially examined. Of these projects, only 102 projects were developed by using object-oriented methods. Table 2-3 groups these projects by application type:

Table 2-3: Object-oriented applications

Applications	Software Projects
Management Information Systems	63
Communications	15
Message Switching	10
Multimedia	4
Testing	4
Real-Time	2
Process Control	2
Scientific	1
Software Development Tools	1
Total	**102**

Table 2-3 shows that object-oriented methods are best suited for management information systems (MIS). The results validate two perceptions of object-oriented applications. First, object-oriented methods are more practical for MIS because MIS are smaller in size and simpler algorithms [76]. Second, object-oriented methods are less practical for large and complex systems (e.g. avionics, real-time, communications, process control, multimedia, message switching, scientific) because of execution time constraints, embedded processing, and an inability to integrate legacy algorithms [25,62,63,64,65,67].

2.6 Constraints in the Department of Defense

The delayed acceptance of object-oriented method by the DoD has been constrained by rigid standards, directives, and policies. The following are key elements of these standards, directives, and policies:

- **"DOD-STD-2167A"** was published in 1988 [23,92]. This standard was "*biased toward the Waterfall Model*" and discouraged the use of Computer-Aided Software Engineering tools [92, 40]. This standard disallowed the use of object-oriented methods for developing Major Defense Acquisition Programs from 1988 through 1994.

- **"MIL-STD-498"** was published in 1994 [24, 40]. The intent was to serve as an interim solution until 1998 [93]. Although this standard did not pose any barrier to the use of object-oriented technology, no Major Defense Acquisition Programs used object-oriented methods through this period.

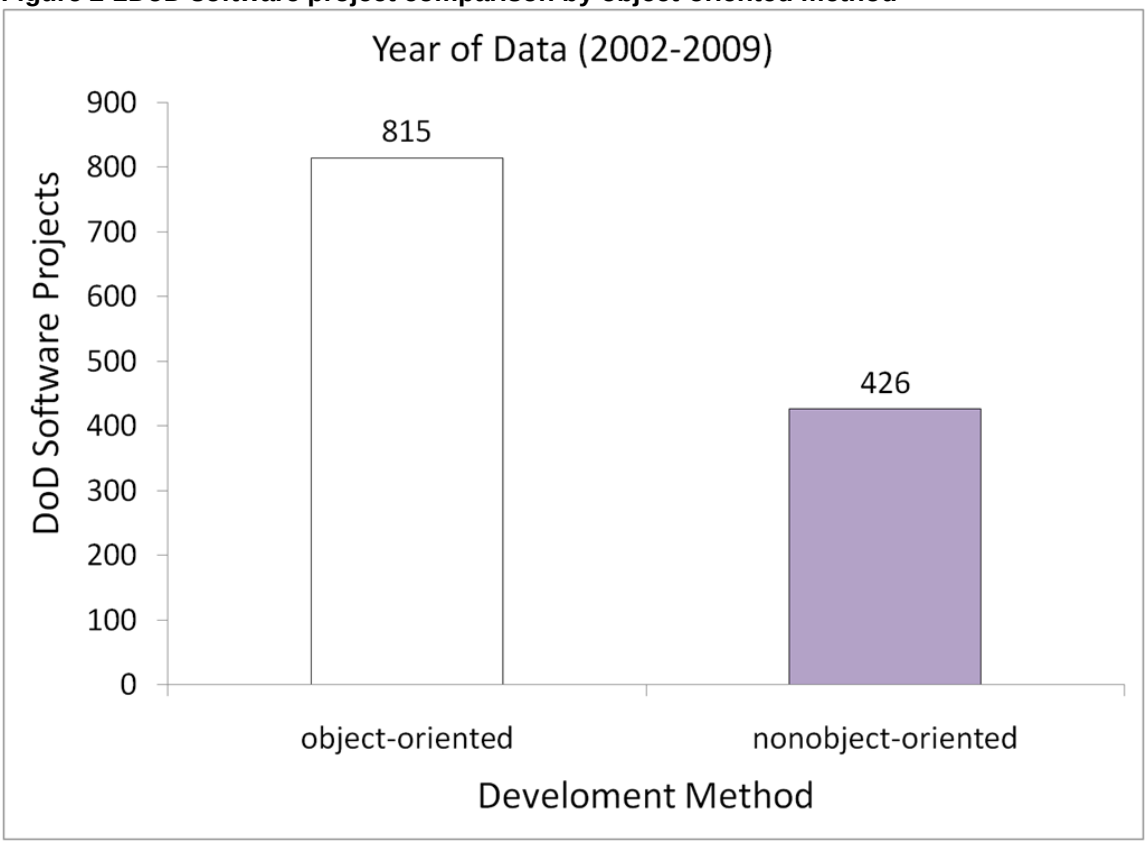
- **"IEEE/EIA 12207.2-1997"** was published in 1998 [2]. This standard encouraged the use object-oriented methods through the "Evolutionary Model" and "Incremental Model".

2.7 Trends in the Department of Defense

An empirical study (Air Force Cost Analysis Agency) reveals that the majority of DoD software projects planned or currently in existence use object-oriented methods. In this comparison study, the use of object-oriented methods was addressed by examining

empirical data from the Defense Acquisition Automated Cost Information System. This repository contains all software data submitted to the Defense Cost Analysis Resource Center, in the form of Software Resource Data Reports (DD Form 2630) dating back to 2002. In this study, 1,241 software projects from Major Defense Acquisition Programs were initially examined. Of these projects, 815 were using object-oriented methods. Figure 1 shows these projects grouped by development methods.

Figure 2-2DoD software project comparison by object-oriented method



3 Literature Review: Software Productivity

This chapter has two sections. The first defines rules for measuring software productivity. The second addresses the two competing software productivity measures.

3.1 Measures

The “IEEE Standard for Software Productivity Metrics” [1] provides guidelines for measuring software productivity. It focuses on how to measure software productivity and what to report. The IEEE standard defines software productivity as the “ratio of the output product to the input effort” [1]. The output product is calculated by using any of three measures: lines of code, function point, and document [1]. The input effort is the expended staff-hour. This standard requires reporting productivity results for lines of code and documents [1]. The use of function points is optional [1]. The rules for measuring staff-hour, lines of code, function points, and software productivity are described next.

3.1.1 Staff-hour

According to the IEEE standard [1], input represents the effort expended to develop the output product. A *staff-hour* is a unit time expended by an individual [1]. Staff-hours shall include both direct and support labor [1]. Direct labor is the effort expended to make the software product including requirements analysis, design, programming, unit testing, and regression testing. Support labor is the effort expended by an individual not working

in the software development site. Examples of these efforts include software management, program support, configuration management, and quality assurance.

3.1.2 Lines of Code

The IEEE standard [1] defines four rules for measuring lines of code:

- Lines of code [1] are either “logical” or “physical”. The “logical” lines of code are measured using the number of instructions, regardless of number of rows [1]. The “physical” lines of code are counted based on the number of “end-of-line” [1] characters.

- Software is classified as either developed or non-developed [1]. Developed software refers to custom or adapted code. Non-developed software refers to reused or deleted statements from preexisting software [1]. Separate counts shall be reported for developed and non-developed.

- Lines of code “in different programming languages shall be counted separately”[1].

3.1.3 Function Points

Allan Albrecht introduced function points in 1979 to measure the functionality from software applications [1]. Function points are derived from software requirement specifications, logical diagrams, physical diagrams, or other design documents. Measuring function points is not required by the IEEE standard; however, organizations

using function points measurements shall also accompany it with the actual lines of code count [1].

3.1.4 Software Productivity

Productivity [1] shall be computed at the completion of a product, that is, when the software is fully tested, accepted and delivered to the customer. According to this standard, when comparing lines of code productivities, the lines shall be in the same programming language [1]. This means that the productivity computed for a software product developed in a third-generation language shall not be directly compared with the productivity of a software product developed in a fourth-generation language.

3.2 Effectiveness

Empirical studies relating object-oriented methods to software productivity tend to compare productivities in any of two ways. A single study compared productivity across projects using function points per staff-hour [69]. A pair of studies compared productivities using lines of code per staff-hour [53,70]. The advantages and shortcomings of these competing measures are described next.

3.2.1 Lines of Code Productivity

The line of code productivity metric is still the most popular measure in the Department of Defense despite the rise of competing measures like use cases, feature points, etc. Collection simplicity is the most significant attribute as the collection of data can be

automated via code counters to capture all source statements (physical and logical), including developed and non-developed code [37].

Measuring lines of code productivity has received considerable criticism. The primary criticism of lines of code is its language dependence [1]. It is not appropriate to compare the lines of code productivities of different programming languages [29,1,51]. Lines of code counts are not known until after regression test is completed [58,85,54]. Further, emerging applications and capabilities can be created with no countable codes [29].

3.2.2 Function Points Productivity

Function points productivity has its advantages over traditional measure, source lines of code. Since function points are software development method and programming language independent, it allows productivity comparisons in different programming languages [54,37,38,29]. Lines of code productivities reported in different programming languages also can be converted into function points by using factors (known as “Backfiring”) developed for particular languages [38,51,8,36]. Since function points are based on software requirements, they can be estimated much earlier than lines of code [85].

Despite these advantages, function points productivity does have its shortcomings. Function points cannot accurately predict productivity for algorithm intensive applications [84]. Most organizations do not track function points metrics because the counting convention is subjective [85]. Function point does not provide the means for software size collection because it requires an additional 2% to 5% effort beyond

implementation costs [37]. In addition, there are no automated counting tools for function points sizing.

4 Literature Review: Control Variables

The chapter summarizes the cost drivers affecting software productivity and highlights important findings from the literature.

4.1 Scholarly Literature

Most software cost studies relate software productivity to product size, operating environment, application domain, and several of the cost drivers addressed in the COCOMO II suite. These cost drivers (Boehm et al. 2000, 33-53) are:

Precedentedness (PREC) (33)

Development Flexibility (FLEX) (33)

Architecture / Risk Resolution (RESL) (34)

Team Cohesion (TEAM) (34)

Process Maturity (PMAT) (34)

Required Software Reliability (RELY) (41)

Database Size (DATA) (41)

Product Complexity (CPLX) (42)

Developed for Reusability (RUSE) (42)

Documentation Match to Life-Cycle Needs (DOCU) (42)

Execution Time Constraint (TIME) (45)

Main Storage Constraint (STOR) (46)

Platform Volatility (PVOL) (46)

Personnel Capability (PCAP) (53)

Personnel Continuity (PCON) (48)

Application Experience (APLEX) (48)

Platform Experience (PLEX) (48)

Language and Tool Experience (LTEX) (48)

Use of Software Tools (TOOL) (49)

Multisite Development (SITE) (49)

Required Development Schedule (SCED) (50)

Maxwell, Wassenhove, and Dutta [55] performed an empirical study to determine whether team size, AD, language, and few COCOMO drivers grouped together would better predict the software productivity variance. Ninety-nine software programs (space and military applications written in high-order programming languages) from the European Space Agency (ESA) were used in the study. Results showed that AD, language, PERS (company), team size, RELY, TIME, STOR, modern development practice (MODP), and TOOL had a significant influence on software productivity. The other two variables, LTEX and virtual machine constraint (VIRT), showed no significant effect.

The authors also performed principal component analysis to measure the variance accounted for by each factor. When PERS, language, and the seven COCOMO drivers are grouped, four variables explain 99% of the productivity variance—PERS, AD, RELY, and TOOL. Despite these findings, the impact of SIZE was omitted in the analysis.

Boehm, Chulani, Fellow, and Steece [20] studied the impact of size, 17 COCOMO II cost drivers, and scale factors on software development productivity. Boehm and colleagues evaluated 161 projects using the Bayesian Theory. The factors accounting for the most variance in software productivity in descending order were SIZE, PERS, CPLX, PREC, RELY, TIME, STOR, DATA, DOCU, PVOL, PCON, TOOL, SITE, and SCED. Three variables, LTEX, RUSE, and APEX, did not show an impact on software productivity. The variable with the most significant influence on software productivity variance was SIZE followed by CPLX and PERS. A second group of factors, TIME, RELY, STOR, and DOCU, also showed a strong influence of productivity.

Boehm and Papaccion [9] studied the interaction between software productivity and quality. They used RELY as an indicator of software quality. Sixty-three development and 25 maintenance projects from the COCOMO database were analyzed. Results showed that the effort to develop high reliable software was significantly greater that for low reliable software, when controlling for the impact of PERS and TOOL.

Putnam and Myers [72] performed an empirical study to determine the impact of SIZE, AD, PERS, defect density, and SCED on software productivity. Sixteen hundred projects from the Quantitative Software Management, Inc. database were used in the study. Results showed SCED, effort, and defect density increased dramatically with larger system size. After SIZE, the second most influencing factor on productivity was AD followed by PERS.

Lian and Noore [47] performed a Monte-Carlo simulation to examine which of the COCOMO II drivers accounted the most variance in productivity. Of the 17 COCOMO II cost drivers, 5 accounted for 57% of the variance in software productivity: CPLX (18%), PERS (21.3%), TIME (6.9%), PCON (5.8%), and RELY (5.1%). The three combined drivers PERS, CPLX, and RELY accounted for roughly 46% of the variance in productivity. Despite these findings, one major limitation was identified in this study: Impact of size was omitted.

Yu [88] conducted a qualitative study to determine the most significant COCOMO II cost drivers affecting software development. This study used a survey among five AT&T Bell laboratories. The factors accounting for the most variance in productivity in descending order were PERS, CPLX, development method, PVOL, and PREC. When combined, PERS and CPLX accounted for 75% of the productivity variance. However, the impact of size was omitted in this study.

Curtis [41] conducted a quantitative study to determine the defect containment effectiveness for military system developers related to Capability Maturity Model Integration (CMMI) levels:

Defect Containment by CMMI Level

CMMI Level	Defect Containment
1	n/a
2	“25.5%”

3	“41.5%”
4	“62.3%”
5	“87.3%”

The analysis showed that CMMI Level bears a direct correlation with defect containment effectiveness.

Capers Jones [39] studied the impact of SIZE, AD, PERS, TOOL, language, RELY, and quality on software productivity. In this study, 1,200 projects from the Software Productivity Research database were used. The variable with the most significant influence on software productivity variance was SIZE. The next variable with the most significant influence was AD followed by PERS. RELY and quality also had a strong influence of productivity.

4.2 Summary

The following conclusions were drawn from the scholarly literature:

1. The most significant factors influencing software productivity are SIZE, AD, CPLX, PERS, and RELY. These factors when combined account for roughly 75% to 90% of total variance in productivity. Thus, it is important to hold these variables constant when evaluating for the impact of object-oriented method.
2. In addition to the top five variables, operating environment also affects software productivity. Thus, it is important to control for its effect when evaluating the impact of object-oriented method.

3. CMMI and RELY level are direct indicators of software quality. Thus, CMMI and RELY can be used as quality indicators in the absence of quality measurements like defect density and defect containment effectiveness.

5 Literature Review: Impact of Object-Oriented Methods on Software Productivity

This chapter contains three sections. The first section addresses the literature that relates object-oriented methods to software productivity. The second section surveys software cost models to determine whether they account for object-oriented methods as an input variable for estimating costs. The third section summarizes the most important studies, notes flaws in the reasoning and logic of previous studies, and suggests why more research relating objected-oriented methods to software productivity is needed.

5.1 Scholarly Literature

This section examines seven studies relating objected-oriented methods to software productivity gain:

Daly, Miller, Brooks, Roper, and Wood [20] examined the perceived benefits of objected-oriented methods with respect to productivity, reuse, and ease of design and maintenance. The method used in the study was based on structured interviews followed by questionnaires. Two hundred and fifty responses were received from experienced practitioners. Of the 250 responses, 200 reported object-oriented as more productive, 25 reported object-oriented as less productive, and 25 provided a response of “don’t know” in regards to object-oriented productivity gains. The authors concluded that most software practitioners believe the objected-oriented design is more advantageous than the

structured design in terms of programmer productivity. However, the finding in this study is inconclusive because the method was not based on empirical data.

Low and Huan [50] examined the impact of objected-oriented methods on software productivity using a multi-site qualitative survey. The first part of the survey gathered general information such as team experience, development method, and product context. The second part of the survey included 18 questions to test the hypothesis. Each question consisted of two scales, one for object-oriented and the other for nonobject-oriented projects. A total of 25 responses were received. Results showed object-oriented methods as a significant factor affecting software productivity in terms of effort spent for enhancements, defect detection, adaptive modifications, and defect removal. However, the findings in this study are subject to criticism because the sampling technique was based on respondents' perceptions rather than empirical observations.

Klepper and Bock [45] examined the impact of fourth-generation language (4GL) on software development productivity. Software productivity was measured using function points per staff-hour. McDonnell Douglas Aerospace Services Forty provided 40 MIS applications, which were used in the study. Klepper and Bock controlled the influence of most COCOMO cost drivers by selecting homogenous projects of the same application, platform, development process, and average team experience. Results showed that projects developed in 4GL achieved greater productivity than projects developed in third-generation language (3GL). However, the finding is subject to criticism because the dataset was from a single organization.

“Hughes Space and Communications Software” [69] conducted an internal study to determine whether they achieved productivity gains after introducing objected-oriented methods. Software productivity was measured using traditional metric (unadjusted function point [UFP] per staff-month) and a new metric called efficiency. Efficiency compares the actual to the initial estimated effort using a parametric cost tool, COCOMO II. Of the four nonhomogenous military projects examined in the study, three were developed in objected-oriented methods and one developed in structured methods. Results showed both, productivity and efficiency gains with the use of object-oriented methods.

Four deficiencies make the findings of this quantitative study subject to criticism: First, it did not address the impact of other intervening variables. Second, projects of varying complexity, platform, and applications were compared. Third, the dataset came from a single organization. Finally, the project’s new, adaptive, and reuse code were aggregated without adjustments.

Potok and Vouk [70] studied the impact of objected-oriented methods on programmer productivity (in lines of code per staff-month). Nineteen software projects developed at IBM Programmer Laboratories were used in the study. All 11 procedural projects were follow-on releases, while 4 of 7 objected-oriented projects were initial releases. Results showed no significant difference in productivities between objected-oriented and structured projects.

The findings in this study, however, had several deficiencies. For example, the study directly compared software productivities in different programming languages. The study also did not address the impact of other intervening variables. Other deficiencies include the comparison of projects of varying complexity, platform, and applications, and the dataset used in the study, which came from a single organization. Finally, the project's new, adaptive, and reuse code were directly compared without adjustments.

Lewis, Henry, Kafura, and Schulman (1992) examined whether objected-oriented methods and code reuse promote greater productivity; using three productivity measures -- "*number of runs*", "*run time errors*", and "*time (in minutes) to fix*" [46]. By studying the same target system, the experiment controlled the effects of some intervening variables. The experimental procedure consisted of two phases. In Phase One, two sets of potentially reusable components were developed for the target system. One set was developed using structured programming language (i.e. Pascal), while the other set was developed in objected-oriented programming language (i.e. C++). Phase Two of the experimental procedure distributed 21 subjects unevenly and randomly across 4 groups numbered 1 through 4:

1. No code reuse using procedural language.
2. No code reuse using object-oriented language.
3. Code reuse using procedural language.
4. Code reuse using object-oriented language.

Results showed that objected-oriented language significantly improves software productivity only when code is being reused and that code reuse improves software productivity regardless of programming language.

In this quantitative study, three deficiencies make the findings subject to other interpretations: Software productivities in different programming languages were directly compared. Participants were graduate students instead of experienced practitioners, and the study did not describe how the project's new, adaptive, and reuse code were adjusted.

Mancl and Havana [53] studied the impact of C++ programming language and objected-oriented process on software productivity. The purpose was to examine the maintenance impact of building an in-house software project using objected-oriented methods. The selection of a target software project written partly in C and partly in C++ programming language controlled the effects of some intervening variables. The study's results showed that the portion of the project written in C++ achieved higher productivity than the portion developed in C: The amount of code reuse doubled from 48,000 to 98,000 lines of code reused, development effort decreased by 19% (through inheritance), and functionalities were added with minor development and interface modification labor. Despite these findings, two deficiencies were identified in this study. First, software productivities in different programming languages were directly compared. Second, the sample used in the study was based on a single observation.

5.2 Software Cost Model Overview

This section surveys four widely used software cost models to determine if they account for objected-oriented methods as an input for predicting software development costs.

5.2.1 SLIM

Lawrence H. Putnam developed Software Lifecycle Model (SLIM) in 1978 [73]. SLIM is a proprietary tool for estimating schedule, effort, and cost [73]. Over 5,300 completed software projects were used to develop the SLIM algorithms. This model measures process productivity as a function of size, effort, and development time. When the process productivity is unknown, SLIM adjusts the default productivity according to the user's responses to 21 assessment questions that are grouped in four areas. This model does not have a direct input for objected-oriented methods. However, the 21 tools and reuse software questions can be calibrated to local project conditions. This calibration reflects the influence of objected-oriented methods on software productivity, although the specific influence of the tools or the reuse software questions cannot be isolated.

5.2.2 SEER-SEM

The SEER-SEM tool (by GALORATH Incorporated) is a proprietary model for estimating effort, size, and schedule [30]. Over 2,000 U.S. Department of Defense projects were used to develop the SEER-SEM algorithms.

SEER-SEM™ has an input variable called development method. Development method describes the methods to be used during the development phase. These methods include rapid application development (RAD), traditional waterfall, objected-oriented, prototype, spiral, or incremental. Changing settings for this parameter also means changing productivity ratio. However, the output is subject to other interpretations because the influence of software development methods (object-oriented and traditional) and development processes (RAD, prototype, spiral, incremental) cannot be segregated. In addition, the statistical test of this variable is proprietary.

5.2.3 PRICE S

PRICE S (by PRICE System L.L.C) is a proprietary tool for estimating software development effort of both commercial and military projects [71]. The key inputs to PRICE S are productivity, software size, program complexity, programming language, operating environment, reuse, test effort, technology maturity, and start-date. The influence of objected-oriented methods on software productivity is not identifiable because it is aggregated in the programming language, technology maturity, and design start-date variables. In addition, the statistical tests of these variables are proprietary.

5.2.4 COCOMO II

Boehm (1981) pioneered and championed the first version of COCOMO. Concerns surrounding next-generation systems led to the implementation of a second version, COCOMO II, in 1997 [8]. COCOMO II (Boehm et al. 2000) is an open-source model for

software cost and schedule estimation. One hundred and sixty-one projects were analyzed. This model has no inputs for programming language level and objected-oriented methods. The effects of these two variables may be aggregated in product and project factors like TOOL and RUSE.

5.3 Summary

A quantitative study of the effect of objected-oriented methods on software productivity is important for several reasons. First, such a study can help reveal an increase in productivity as an underlying benefit of the Object model and hence encourage the U.S. Department of Defense to adapt objected-oriented methods more often. Second, understanding the relationships between objected-oriented methods, software productivity, and COCOMO II cost drivers can encourage researchers to explore new strategies to improve available software cost models. Third, the collected data can be used for future software cost estimating research or cost model improvement initiatives.

6 Significance of the Proposed Study

Despite an increased interest in object-oriented methods, few empirical studies have been conducted relating object-oriented methods to productivity. Of the seven studies relating object-oriented methods to software productivity, only five have been based on empirical data. These five studies, however, had deficiencies in the sampling procedure, intervening variables, and logic for measuring and comparing software development productivities.

The sampling procedure was based either on a single system [53], multiple systems within an organization [70,69,45], or individual students developing the same system [42]. Intervening variables, such as product complexity, application, experience, team cohesion, team capability, and use of software tools, were overlooked. Software productivities were measured without adjusting the project's adapted and reused code [69,53,70,42], and software productivity comparisons were made based on projects written in different programming languages. The last two deficiencies violate the rules for measuring productivity according to the "IEEE Standard for Software Productivity Metrics" [1].

This study will remedy these limitations in three distinct ways:

1. Collect pairs of projects across multiple organizations and not from a single organization.
2. Control for the impact of those variables that account for the most variance in software productivity.

3. Calculate productivity according to best practices set forth by the IEEE Standard for Software Productivity Metrics. This includes adjusting reused code to its equivalent in new code and then into function points in order to make cross-language productivity comparisons.

This study is important for several reasons. First, such a study can help reveal whether object-oriented method is appropriate for military weapon systems since these systems are more constrained (higher quality assurance levels) than commercial software. Second, understanding the relationships between objected-oriented methods, software productivity, and other cost drivers can encourage researchers to explore new strategies for controlling software productivity and quality. Third, the collected data can be used for future software cost estimating research.

7 Research Method

This chapter is comprised of seven sections. The first describes the variables used in the study. The second section describes the participants, rationale for choosing the control variables, and project similarities. The third section identifies the instrumentation used in the experiment and the approach to assigning rating scales. The fourth section summarizes the process for calculating the independent variable from the collected data. The fifth section describes normality test procedures. The sixth section identifies the research questions and hypotheses. The seventh section explains the experimental procedure, including design and experimental manipulation.

7.1 Variables

This chapter describes the dependent variable, independent variable, and control variables that may influence the dependent variable.

7.1.1 Dependent Variable

Software productivity will be the dependent variable in this study. It is measured as the unadjusted function points divided by the staff-months. The process for calculating productivity is described in Section 7.4**.**

7.1.2 Independent Variables

Object-oriented methods will be the independent variable in this study. Object-oriented methods will be measured by collecting data on three elements—analysis method, design method, and programming method. These elements are described in Table 7-1.

Table 7-1 Elements of object-oriented method

Predictor Variable	Symbol	Description
Software Analysis Method	SAM	It can be categorized as one of three kinds—"structured", "data-driven", or "object-oriented"[10].
Software Design Method	SDM	It can be categorized as one of three kinds—"structured", "data-driven", or "object-oriented"[10].
Software Programming Method	SPM	It can be categorized as one of three kinds—"structured", "data-driven", or "object-oriented"[10].

7.1.3 Control Variables

The control variables in this study are size, operating environment, application domain, complexity, personnel capability, and reliability. Many studies have shown these variables are the most significant factors affecting productivity. They account for roughly 75% to 90% of the total variance in productivity.

7.2 Instrumentation

The data collection questionnaire in Appendix A will be used to collect data. With the exception of one new field and a change in the order and titles of some fields, the data collection questionnaire and form were obtained from the COCOMO II [8] text book. The new field is added to pre-existing COCOMO II instrumentation to account for the independent variable, which is object-oriented methods.

The instrumentation [8] allows for the collection of project context, application domain, operating environment, source lines of code, programming language, effort, schedule, object-oriented methods, and COCOMO II cost drivers. Very similar instrumentations have been used in past dissertations [17,18].

Each COCOMO II cost driver can have up to six ratings ranging from "*extra-low (XL)*" to "*extra high (XH)*" [8]. The rating scale descriptions are shown in Appendix A, and the rating scale values are shown in in Appendix B.

7.2.1 Object Oriented Method Questionnaire

The rating scale for object-oriented methods will be similar to use of software tools (TOOL), a particular COCOMO II cost driver. This rating scale was chosen because TOOL is an element of the object-oriented method [10]. The sequence of values decreases from N to VH to assert that productivity increases as more object-oriented methods are used. Table 7-2 describes the rating scales (0-1), and Table 7-3 shows the rating scale values.

Table 7-2 Object-oriented method rating scale definitions

Driver	Nominal	High	Very High
OOM	NonOO methods for analysis, design, and programming activities	1 or 2 OO methods used for analysis, design, and programming activities	Full OO methods used for analysis, design, and programming activities

Abbreviations: OOM = object-oriented method; OO = object-oriented.

Table 7-3 Object-oriented method rating scale values (0-1)

Driver	Nominal	High	Very High
OOM	1.00	0.9	0.78

Abbreviations: OOM = object-oriented method.

7.2.2 Operating Environment Questionnaire

The operating environment describes the primary mission of the software project. The definitions for each operating environment (Table 7-4) were developed by Boehm and colleagues (2009) for the "Software Cost Metrics Manual" sponsored by the Air Force Cost Analysis Agency [91]:

Table 7-4 Taxonomy operating environment

Operating Environment	Description
Avionics	"Software developed for aircraft avionics systems like those found on aircrafts and helicopters." [91]
Business	"Software developed for systems that process business information, such as financial and management information systems." [91]
Manned Space	"Software developed for manned space applications, such as space shuttles or space stations." [91]
Manned Ground	"Software developed for ground-based systems, such as artillery and tactical information centers." [91]
Military Mobile	"Software developed for ground-based mobile systems, such as mobile command stations in trucks or vans or tanks and transports of any kind (e.g., trucks and trains)." [91]
Missile and Unmanned Airborne	"Software developed for missiles or unmanned airborne systems, such as an air-to-air missile or remotely piloted vehicle." [91]
Shipboard	"Software developed for sea-based systems, such as aircraft carriers, destroyers, freighters, submarines and tankers." [91]
Telecommunications	"Software developed for communications systems that transmit voice, data, video, or other information." [91]
Unmanned Ground	"Software developed for unmanned ground-based systems such as reconnaissance vehicles." [91]
Unmanned Space	"Software developed for unmanned space systems used for planetary exploration, imagery, or weather prediction." [91]
Web	"Software developed for applications that run and utilize the Internet." [91]

7.2.3 Application Domain Questionnaire

The application domain describes the primary function of the software. The definitions

for each application domain (Table 7-5) were developed by Boehm and colleagues (2009) for the "Software Cost Metrics Manual" sponsored by the Air Force Cost Analysis Agency [91]:

Table 7-5 Taxonomy application domain

Application Domain	Description
Bus	"Software used to manage infrastructure services (e.g., attitude and articulation control or commanding) for manned or unmanned space applications." [91]
Command and Control	"Software used for network monitoring, network control and switching, sensor control, signal/telemetry processing, message processing, data reduction/analysis, mission control, and command processing in any of the operating environments described above." [91]
Communications	"Software used for transmission and receipt of voice, data, digital, and video information within, to, or from the operating environments described above." [91]
Controls and Displays	"Software that provides the interface between the user and system within any of the operational environments described above. For example, software that controls the Heads-Up Display (HUD) in an aircraft provides the pilot with a virtual environment during engagement." [91]
Database	"Software used to collect, manage, and search information stored as files or databases in any of the operating environments described above."
Electronics Warfare	"Software used to deny the opponent an advantage in the electromagnetic spectrum through its attack, protection, and support components." [91]
Executive	"Systems software used to control the hardware and operating environment within which software applications execute any of the operating environments described above. Executive software is typically specially developed to control the environment because the operating system or standard platform software cannot accommodate specialized run-time requirements." [91]
Logistics	"Software used to manage the flow of goods, information and other resources involved in maintaining and sustaining systems operational in the field." [91]
Maintenance and Diagnostics	"Software used to perform maintenance functions, including diagnostics and diagnosis of problems." [91]
Mission Management	"Software used to support mission management activities, such as the command and control of systems aboard an aircraft, in any of the operating environments described above." [91]

Application Domain	Description
Mission Planning	"Software used to support mission planning activities, including those for scenario generation, feasibility analysis, route planning, and image/map manipulation, in any of the operating environments described above." [91]
Payload	"Software used to manage payload functions (e.g., experiment control or sensor management) for manned or unmanned space applications." [91]
Platform	"Software used to bind, load, schedule, and run applications on the selected computer hardware. Typically, such software includes the operating system, linker/loader, middleware, and utilities operationally required for any of the operating environments described above. In some cases, such platforms provide services like database management and security as part of the platform offerings." [91]
Process Control	"Software that provides closed-loop feedback controls for applications such as attitude control or propulsion that run in real-time in industrial and military environments."
Radar	"Software used to automate radar functions in any of the operating environments described above." [91]
Signal Processing	"Software used to enhance, transform, filter, convert, or compress signal data—typically in real-time—in any of the operating environments described above." [91]
Simulation and Modeling	"Software used to evaluate scenarios and assess empirical relationships that exist between models of physical processes, complex systems, or other phenomena that may have complex relationships in any of the operating environments described above." [91]
Situation Awareness	"Software used to provide decision makers with a perception of their environment within a volume of time and space within any of the complex, dynamic operating environments described above." [91]
Sonar	"Software use to automate sonar functions in any of the operating environments described above." [91]
Test and Evaluation	"Software used to support the testing and evaluation of software within software, hardware, and pseudo-operational environments, such as test labs, within any of the operating environments described above. Activities performed during T&E include test case generation, test data recording, test data reduction/analysis, and test driver/tool development." [91]
Tool and Tool Systems	"Software packages or integrated development environments that are used for analysis, design, construction, and testing of operational software for the operating environments described above." [91]
Training	"Software used to support the education and training of users within any of the operating environments described above. Activities performed during TRNG include computer-based

Application Domain	Description
	training and computer-aided instruction within the operational environment or dedicated training systems." [91]
Weapons Delivery	"Software used to target, control, and command weapons within any of the operating environments described above." [91]

7.2.4 Complexity Questionnaire

This parameter rates (Table 7-6) the software system's inherent difficulty [30].

Table 7-6 Ratings for Complexity Factor

Rating	Symbol	Description
Extra High	XH	"Developed primarily by using micro code"[30].
Very High+	VH+	"Top of the scale for 99% of all defense software"[30].
Very High	VH	"Real-time applications with significant logical code"[30].
High	H	"Applications with significant logical complexity, perhaps requiring changes to the operating system, minor real-time processing or special display hardware"[30].
Nominal+	N+	"Typical command and control programs"[30].
Nominal	N	"New standalone applications developed on firm operating systems" [30] with few interface problems.
Low	L	"Low logical complexity that using straightforward I/O and primarily internal data storage" [30].
Very Low	VL	"Extremely simple application with primarily straightforward code, simple I/O, and internal storage arrays"[30].

7.2.5 Personnel Capability Questionnaire

The personnel capability (Table 7-7) evaluates the analyst/programming team assigned to the software project [30]. This parameter rates the "inherent potential of the individual

team members as well as the team as a whole, independent of experience" [30]. "More experienced personnel are not necessarily more capable, and less capable personnel are not necessarily less experienced"[30].

Table 7-7 Ratings for personnel capability factor

Rating	Symbol	Description
Very High	VH	≥90th percentile from the team
High	H	75th percentile from the team
Nominal	N	55th percentile from the team
Low	L	≥35th percentile from the team
Very Low	VL	≤15th percentile from the team

7.2.6 Reliability Questionnaire

The reliability factor [30] evaluates the rigor of the quality assurance (QA) activities (Table 7-8). The "Quality Assurance level usually is directly related to the impact that a failure in the software would have during its operational phase" [30].

Table 7-8 Ratings for reliability factor

Ratings	Symbol	Description
Very High	VH	A rigorous quality assurance (QA) is required due to risk to human life. Following prescribed plans, procedures, and reporting to ensure the highest reliability. [30]
High	H	Formal QA activities and quality engineering/management are required due to high risk of financial losses. [30]
Nominal	N	Only formal QA is required due to moderate, easily recoverable losses. [30]
Low	L	Only informal QA due to low, easily recoverable losses. [30]
Very Low	VL	No specific QA activities are required due to slight inconveniences. [30]

7.2.7 Software Quality Questionnaire

The "defect containment effectiveness [41]" is the primary measurement (Table 7-9) for software quality in this study. The *defect containment effectiveness* is the cumulative percentage of defects from program insertion up through Final Qualification Test (the last phase before the product is shipped) [41]. The defect density, which is optional, is the alternate measurement in this study.

Table 7-9 Ratings for QUAL Factor

Rating	Symbol	Description
Extra High	XH	Defect containment ≥ 95 %
Very High	VH	Defect containment ≥ 90%
High	H	Defect containment ≥ 87%
Nominal	N	Defect containment ≥ 62%
Low	L	Defect containment ≥41%

7.2.8 Platform Volatility Questionnaire

The platform volatility factor (Table 7-10) "rates the difficulty caused by changes to the development system"[30].

Table 7-10 Ratings for platform volatility factor

Rating	Symbol	Description
Very High	VH	Major: every 2 weeks; Minor: 2 days. [8]
High	H	Major: every 2 months; Minor: 1 week. [8]
Nominal	N	Major: every 6 months; Minor: 2 weeks. [8]
Low	L	Major: every 12 months; Minor: 1 month. [8]

7.3 Data Collection

Data (Tables 7-11 and 7-12) were collected on 20 completed software projects from 10 military organizations that have developed military weapon systems for the U.S. Department of Defense in the last seven years. The questionnaire used to survey each organization specifically asked that the organization provide data on two homogeneous projects—one project that was developed in object-oriented methods and another project developed in traditional methods. Each homogenous pair was similar in size, operating environment, application domain, complexity, personnel capability, and reliability. The rationale for choosing these six variables is based on their significant effect on software productivity. The literature discussed in this study indicates that, when combined, these variables account for roughly 75% to 90% of total variance in software productivity.

Table 7-11 Project's Context Information

Org.	Proj.	Operating Environment	Application Domain	CMM	LANG	OOA	OOD	OOP
A	1	Manned Ground	Mission Planning	5	Ada	N	N	N
	2	Manned Ground	Mission Planning	5	Java	Y	Y	Y
B	3	Manned Ground	Command & Control	4	Pascal	N	N	N
	4	Manned Ground	Command & Control	4	XML	Y	Y	Y
C	5	Manned Ground	Mission Planning	5	Ada	N	N	N
	6	Manned Ground	Mission Planning	5	C++	Y	Y	Y
D	7	Avionics	Radar	3	Ada	N	N	N
	8	Avionics	Radar	3	C++	Y	Y	Y
E	9	Avionics	Mission Management	5	Ada	N	N	N
	10	Avionics	Mission Management	5	C++	Y	Y	Y
F	11	Avionics	Radar	5	C	N	N	N
	12	Avionics	Radar	5	C++	Y	Y	Y
G	13	Shipboard	Communications	3	Ada	N	N	N
	14	Shipboard	Communications	3	C++	Y	Y	Y
H	15	Shipboard	Communications	4	C	N	N	N
	16	Shipboard	Communications	4	C++	Y	Y	Y
I	17	Avionics	Mission Management	5	Ada	N	N	N
	18	Avionics	Mission Management	5	C++	N	Y	Y
J	19	Manned Ground	Command & Control	5	C	N	N	N
	20	Manned Ground	Command & Control	5	C++	Y	Y	Y

Org. = organization; Proj. = project; "CMM = capability Maturity Model"; "LANG. = language"; "OOA = object-oriented analysis"; "OOD = object-oriented design"; "OOP = object-oriented programming"; Y= yes; N = no.

Table 7-12 Project's Raw Size, Productivity, and Productivity Drivers

Org.	Proj.	ESLOC/SM	Total Size Range (KSLOC)	CPLX	PERS	RELY	PVOL
A	1	254	48-83	N	VH	H	N
	2	630	45-78	N	VH	H	N
B	3	382	115-199	N+	H	H	N
	4	439	132-229	N+	H	H	N
C	5	428	221-382	N	N	H	L
	6	768	196-339	N	N	H	L
D	7	52	104-179	VH	N	H	N
	8	70	113-195	VH	N	H	N
E	9	52	434-753	VH	N	H	L
	10	82	550-953	VH	N	H	L
F	11	142	68-117	VH	VH	N	N
	12	153	55-95	VH	VH	N	N
G	13	119	56-98	VH	N	H	L
	14	306	51-88	VH	N	H	L
H	15	66	17-30	H	N	N	L
	16	299	17-31	H	N	N	L
I	17	128	16-27	VH	H	VH	N
	18	160	15-26	VH	H	VH	N
J	19	236	1207-2092	H	H	VH	N
	20	299	1049-1817	H	H	VH	N

Org. = organization; Proj. = project; "EKSLOC = equivalent thousand source lines of code"; "SM = staff-month"; "KSLOC= total thousand source lines of code"; "CPLX = product complexity"; "PERS = personnel capability"; RELY = Reliability; and QUAL = quality ("defect containment"), "N = nominal"; "N+ = nominal+"; "H = high"; "VH = very high".

Organization A: Project 1 and Project 2 are similar in that they are both nonembedded, nonreal-time, manned-ground software. They are provided as mission planning aids to an airborne radar receiver, either display or scenario generation. They are both developed on commercial PCs and servers using nonreal-time systems. The analysts for both projects are of the same skill level.

The similarity of the two project sizes was significant and showed a difference of <7%. The quality of both projects is high on the basis of their defect containment effectiveness (90%) and low defect density (12 defects per KSLOC). The reliability of both projects is high on the basis of their formal quality assurance (QA) level and quality management activities throughout the entire development phase.

Organization B: Project 3 and Project 4 are similar in that they are both nonembedded, nonreal-time, manned-ground software. They are both command and control software with the same core functions—signal processing, data fusion, resource management, and intelligence formation. They are both developed on commercial PCs and servers using nonreal-time systems. The analysts for both projects are of the same skill level.

The similarity of the two project sizes was considerable and showed a difference of <15%. The quality of both projects is nominal on the basis of their defect containment efficiency (64% to 65%) and low defect density (2.8–4.8 defects per KSLOC). The reliability of both projects is high on the basis of their formal QA level and quality management activities throughout the entire development phase.

Organization C: Project 5 and Project 6 are similar in that they are both nonembedded, nonreal-time, manned-ground software. They are both provided as mission aids for the same aircraft with simulation or scenario generation functions. The two projects are built on commercial PCs and servers. The analysts for both projects are of the same skill level.

The similarity of the two project sizes was considerable and showed a difference of <13%. The quality of both projects is very high on the basis of their defect containment efficiency (94% to 97%). The reliability of both projects is high on the basis of their formal QA level and quality management activities throughout the entire development phase.

Organization D: Project 7 and project 8 are very similar in that they are both embedded, real-time, avionics software. They are both airborne radar software built for a fighter aircraft with the same core functions—signal processing, data processing, built-in calibration, and simulation. The analysts for both projects are of the same skill level.

The similarity of the two project sizes was significant and showed a difference of <9%. According to the software project manager, software quality is high for both on the basis of their defect containment effectiveness (88% to 90%). The reliability of both projects is high on the basis of their formal QA level and test activities throughout the entire development phase.

Organization E: Project 9 and project 10 are similar in that they are both embedded, real-time, avionics software. They are both mission management software built for an aircraft mission computer, with the same core functions—target tracking, data processing, and simulation. The analysts for both projects are of the same skill level.

The similarity of the two project sizes was significant and showed a difference of <27%. The quality of both projects is nominal on the basis of their priority level 1 and 2 defect containment effectiveness (62% to 77%). The reliability of both projects is high on the basis of their formal QA level and test activities throughout the entire development phase.

Organization F: Project 11 and project 12 are very similar in that they are both embedded, real-time, avionics software. They are both radar software built for a military helicopter with the same core functions—receiver/exciter processor, and mission communication control. The analysts for both projects are of the same skill level.

The similarity of the two project sizes was significant and showed a difference of <23%. The quality of both projects is nominal on the basis of their defect containment effectiveness (76% to 81%). The reliability of both projects is nominal on the basis of formal QA activities throughout the entire development phase.

Organization G: Project 13 and project 14 are very similar in that they are both embedded, real-time, shipboard software. They are both communication software built for a surface combatant ship with the same core function — jam resistant, encrypted, real time data transfer. The analysts for both projects are of the same skill level.

The similarity of the two project sizes was significant and showed a difference of <11%. The quality of both projects is nominal on the basis of their defect containment

effectiveness (71% to 84%). The reliability of both projects is high on the basis of their formal QA level and test activities throughout the entire development phase.

Organization H: Project 15 and project 16 are very similar in that they are both embedded, real-time (<50%), shipboard software. They are both communication software built for an x-band shipboard radar with the same core function—mission applications processing, navigation mode, and fault detection. The analysts for both projects are of the same skill level.

The similarity of the two project sizes was significant and showed a difference of <3%. The quality of both projects is nominal on the basis of their relatively similar defect containment effectiveness (~67%). The reliability of both projects is nominal on the basis of formal QA activities throughout the entire development phase.

Organization I: Project 17 and project 18 are similar in that they are both embedded, real-time, avionics software. They are both mission management software built for a n aircraft mission computer, with the same core functions—target tracking, and data processing. The analysts for both projects are of the same skill level.

The similarity of the two project sizes was significant and showed a difference of <13%. The quality of both projects is high on the basis of their priority level 1 and 2 defect containment effectiveness (>87%). The reliability of both projects is very high on the

basis of their intense QA level and test activities throughout the entire development phase.

Organization J: Project 19 and Project 20 are similar in that they are both nonembedded, partial real-time, manned ground software. They are both command and control software with the same core functions—data fusion and situational awareness. They are both developed on commercial PCs and servers using non real-time systems. The analysts for both projects are of the same skill level.

The similarity of the two project sizes was significant and showed a difference of <24%. The quality of both projects is nominal on the basis of their defect containment efficiency (75% to 86%). The reliability of both projects is nominal on the basis of their formal QA level and quality management activities throughout the entire development phase.

7.4 Calculating Software Productivity

The method for computing productivity was in accordance with COCOMO II [8] and the "IEEE Standard for Software Productivity Metrics" [1]. The process for measuring software productivity gain is composed of three steps:

1. Adjusting project's raw size.
2. Backfiring.
3. Calculating productivity.

7.4.1 Adjusting the Project's Raw Size

A project's total size [8] consists of new and "adapted" source lines of code (SLOC). New SLOC (NSLOC) is code developed for the new project. Adapted SLOC (ASLOC) is preexisting code either modified or unmodified for the new project.

ASLOC was converted to "equivalent source lines of code (ESLOC)" using the "Adaptation Adjustment Factor (AAF)" captured in COCOMO II (Boehm et al. 2000, eq. **2.4**). This factor captures the additional effort required to redesign, re-implement, and retest the pre-existing code for inclusion in the new project.

The ESLOC formula adjusted from COCOMO II (Boehm et al. 2000, 22-23) is:

$$ESLOC = NSLOC + ASLOC \times AAF \qquad 1.1$$

Where:

ESLOC = total ESLOC
NSLOC = New SLOC
ASLOC = Adapted SLOC

AAF = "(0.4xDM) + (0.3xCM) + (0.3xIM)" (Boehm et al. 2000, eq. 2.4)
DM = percent "design modified" (22)
CM = percent "code modified" (22)
IM = percent "integration modified" (22)

7.4.2 Backfiring

According to the "IEEE Standard for Software Productivity Metrics"[1], the productivity computed for a structured design project developed in a third-generation language (3GL) shall not be directly compared with the productivity of an object-oriented design project developed in a fourth-generation language (4GL). For this reason, the ESLOC will be

converted into “unadjusted function points (UFP)” by using Capers Jones’ method called “Backfiring” [38]. The “Backfiring” conversion ratios (Boehm et al 2000) used in this study are shown in Table 7-13 below.

Table 7-13 ESLOC→UFP Conversion Ratios

Programming Language	ESLOC per UFP
Ada 83	71
Ada 95	49
APL	31
Assembly-Basic	320
Assembly-Macro	213
Basic-ANSI	64
Basic-Compiled	91
Basic-Visual	32
C	128
C++	55
Cobol (ANSI 85)	91
Fortran 77	107
Fortran 95	71
HTML 3.0	15
Java	53
Jovial	107
Pascal	91
PERL	27
Power Builder	16
Visual Basic 5.0	29
Visual C++	34

“ESLOC = equivalent source lines of code”; “UFP = unadjusted function points”.

7.4.3 Calculating Productivity

Once the ESLOC values have been converted into UFP, the final step is to compute the project's productivity. The productivity equation is consistent with the "IEEE Standard for Software Productivity Metrics" [1]. The equation is explained next.

$$PP = UFP \div SM \quad (1.2)$$

Where PP = software productivity achieved by a project
UFP = unadjusted function points produced by a project
SM = staff-months incurred by a project

7.5 Normality Test

This study used histogram and normality plots through MINITAB 14 (Minitab Inc. 2003) to validate the statistical assumption of normality for each hypothesis. A transformation was necessary for Hypothesis 2 because the residuals did not exhibit normality.

7.6 Research Hypotheses

This section identifies the list of hypotheses (from section 1.5.3 Revised Hypotheses), which were addressed in the experiment.

Hypothesis 1: Do object-oriented methods relate to software productivity?

H_{01}: There is no relation between object-oriented method and productivity gain.

H_{11}: There is a relation between object-oriented method and productivity gain.

Hypothesis 2: Do object-oriented methods and complexity relate to software productivity?

H_{02}: Object-oriented methods and complexity are not related to productivity gain.

H_{12}: Object-oriented methods and complexity are related to productivity gain.

Hypothesis 3: Do object-oriented methods and application domain relate to software productivity?

H_{03}: Object-oriented methods and application domain relate to productivity gain.

H_{13}: Object-oriented methods and application domain do not relate to productivity gain.

7.7 Experimental Design

This section has three subsections. The first displays the design matrix for Hypothesis 1. The second presents the design matrix for Hypothesis 2. The third shows the design matrix for Hypothesis 3.

7.7.1 Experimental Design for Hypothesis 1

Hypothesis 1 attempts to compare product's productivity (unadjusted function point divided by staff-month) of object-oriented with nonobject-oriented projects. The factor is object-oriented method. Ten object-oriented and ten nonobject-oriented projects took part in the study. The two levels were coded as object-oriented is A, and nonobject-oriented is B. The dataset is provided in Table 7-14.

Table 7-14 Design matrix for Hypothesis 1

OOM	PP
A	11.5
	14.0
	8.3
	5.4
	5.4
	1.3
	1.5
	2.8
	5.6
	2.9
B	3.6
	6.0
	3.5
	0.5
	1.8
	0.7
	0.7
	1.1
	1.7
	1.8

OOM = object-oriented method; PP = product's productivity

7.7.2 Experimental Design for Hypothesis 2

Hypothesis 2 attempts to compare productivities of object-oriented with nonobject-oriented projects on the basis of four complexity levels. The two factors are—

1. Object-oriented method. Ten object-oriented and ten nonobject-oriented projects took part in the study. The two levels were coded as object-oriented is A, and nonobject-oriented is B.
2. Complexity. The four factor levels were coded as—
    - "Nominal is 1"
    - "Nominal+ is 2"
    - "High is 3"
    - "Very High is 4"

The product's productivity (unadjusted function point divided by staff-month) that each project achieved was recorded as the response variable. The untransformed dataset is provided in Table 7-15.

Table 7-15 Design matrix for Hypothesis 2

	Complexity			
OOM	1	2	3	4
A	11.5	8.3	5.4	1.3
	14.0		5.4	1.5
				2.8
				5.6
				2.9
B	3.6	3.5	0.5	0.7
	3.6		1.8	0.7
				1.1
				1.7
				1.8

OOM = object-oriented method.

7.7.3 Experimental Design for Hypothesis 3

Hypothesis 3 attempts to compare productivities of object-oriented with nonobject-oriented projects on the basis of five application domains. The two factors are—

1. Object-oriented method. Ten object-oriented and ten nonobject-oriented projects took part in the study. The two levels were coded as object-oriented is 0, and nonobject-oriented is 1.
2. Application Domain. The projects performed one of five primary functions. The five levels were coded as—
    - "Mission planning is 1".
    - "Command and control is 2".
    - "Radar is 3".

- “Mission management is 4”.
- “Communications is 5”.

The product’s productivity (unadjusted function point divided by staff-month) that each project achieved was recorded as the response variable. The dataset is provided in Table 7-16.

Table 7-16 Design matrix for Hypothesis 3

	Application Domain				
OOM	1	2	3	4	5
A	3.6	3.5	0.7	0.7	1.7
	6.0	1.9	1.1	1.8	0.5
B	11.5	8.3	1.3	1.5	5.6
	14.0	5.4	2.8	2.9	5.4

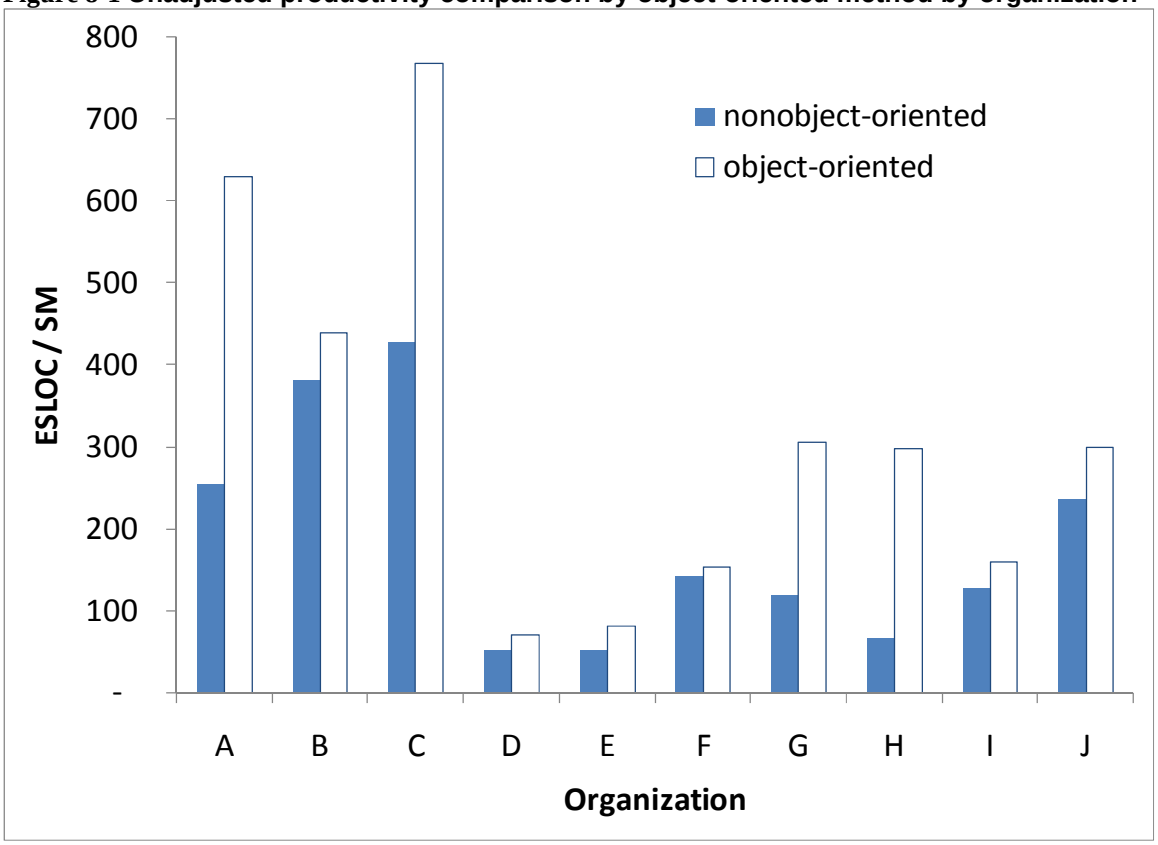
OOM = object-oriented method.

8 Data Analysis

8.1 Descriptive Statistics

Figure 8-1 compares unadjusted productivity by object-oriented method by organization. Unadjusted productivity means that the equivalent size (ESLOC) has not been converted into function points. The result shows that in all ten organizations the object-oriented project’s productivity was higher than the nonobject-oriented project’s productivity.

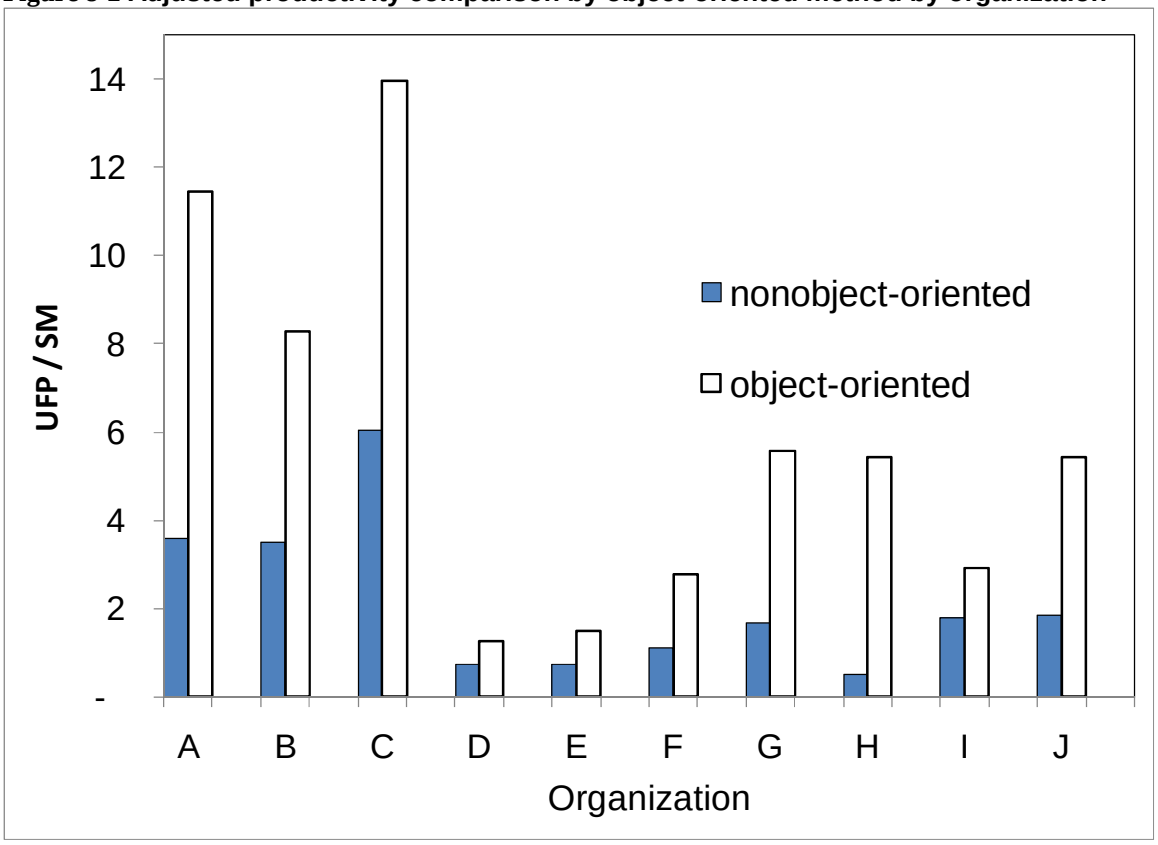
Figure 8-1 Unadjusted productivity comparison by object-oriented method by organization



ESLOC = equivalent source lines of code; SM = staff-months.

Figure 8-2 compares the adjusted productivity by object-oriented method by organization. Adjusted productivity means that the equivalent size (ESLOC) has been converted into function points. Again, the result shows that in all ten organizations the object-oriented project's productivity was higher than the nonobject-oriented project's productivity.

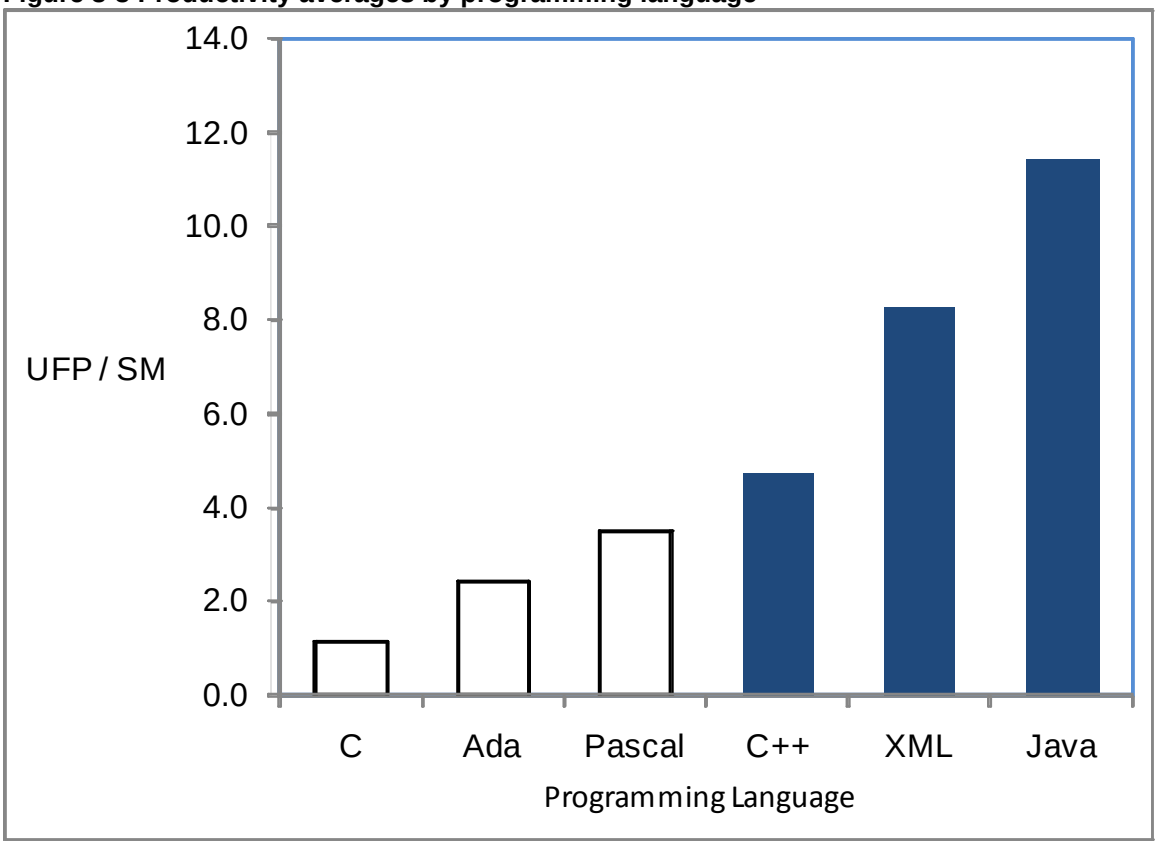
Figure 8-2 Adjusted productivity comparison by object-oriented method by organization



ESLOC = equivalent source lines of code; SM = staff-months.

The projects written in object-oriented programming languages C++, XML, and Java show a higher productivity average than projects written in the third-generation languages (3GLs) Ada, Pascal, and C (Figure 8-3).

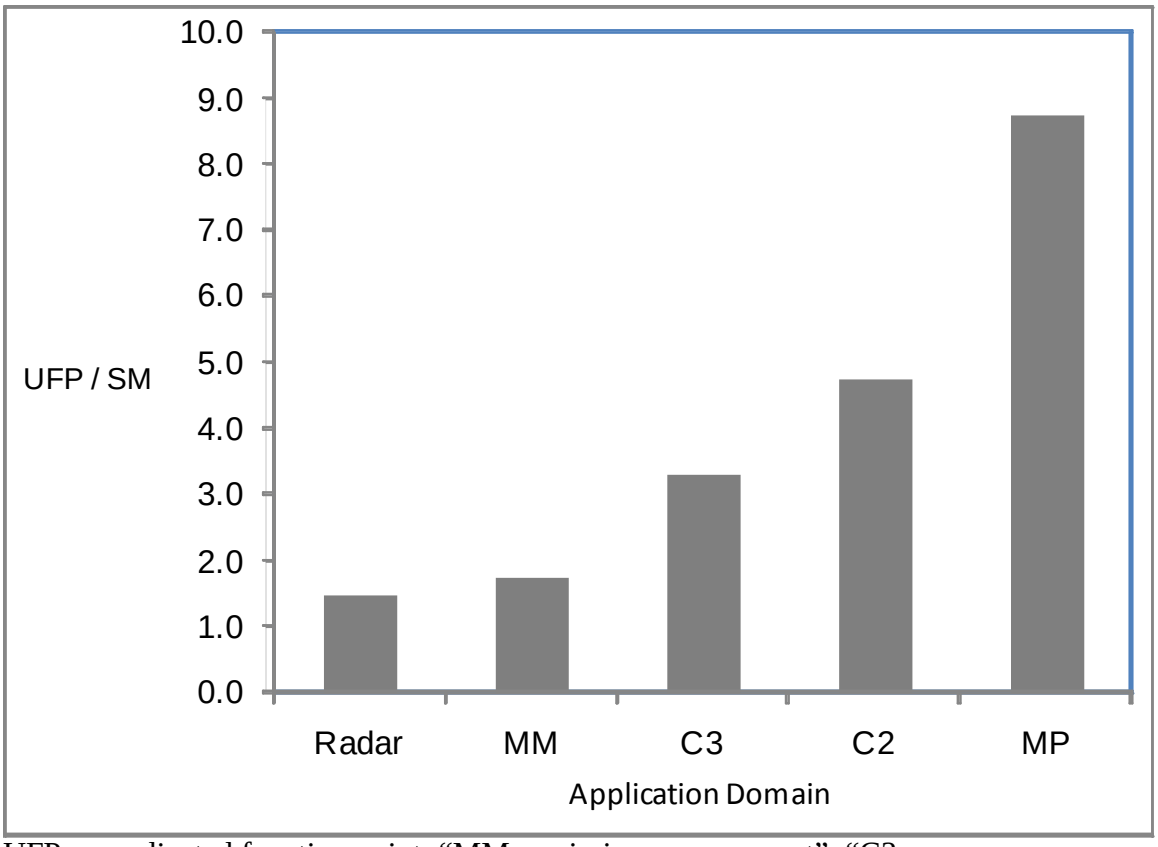
Figure 8-3 Productivity averages by programming language



"UFP = unadjusted function point"; "SM = staff-months".

The most productive application domains in descending order are mission planning (MP), command and control (C2), communication (C3), mission management (MM), and radar (Figure 8-4). The latter three correspond to real-time processing software for either aircraft or shipboard platforms.

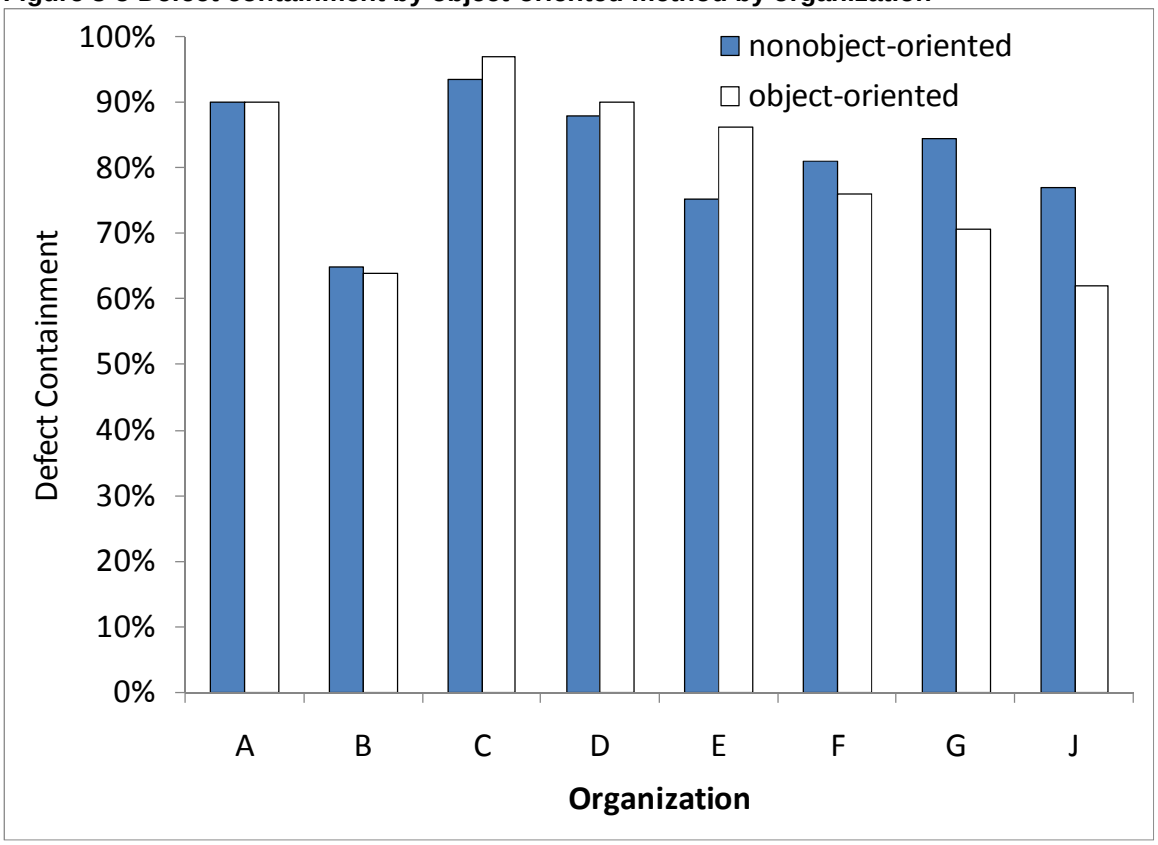
Figure 8-4 Productivity by application domain



UFP = unadjusted function point; “MM = mission management”; “C3 = communications”; C2 = “command and control”; “MP = mission planning”.

Defect containment was compared by object-oriented method (Figure 8-5). Of eight organizations, only three showed their object-oriented projects achieving higher defect containment than their nonobject-oriented one. Thus, it cannot be asserted that object-oriented methods improves software quality.

Figure 8-5 Defect containment by object-oriented method by organization



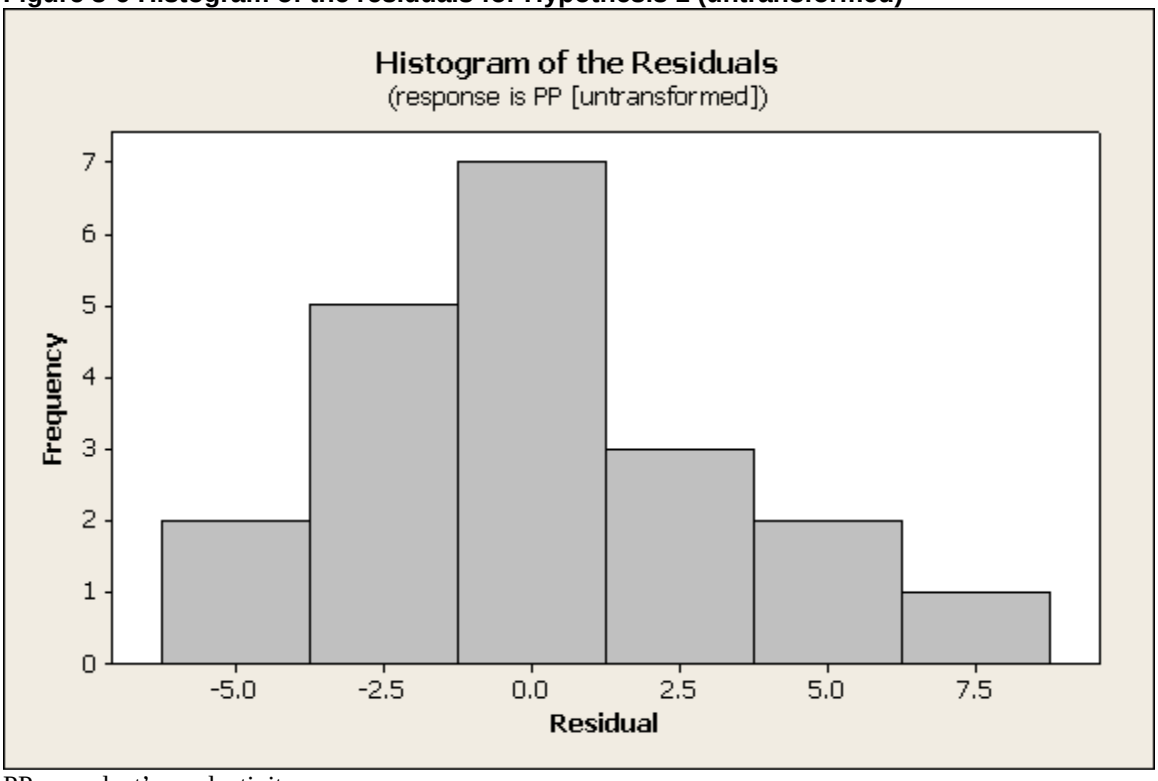
8.2 Normality Test

Many statistical procedures assume that data follow a normal distribution. This section describes the normality test performed on each hypothesis by creating histograms of the residuals and normal residual plot.

8.2.1 Normality Test for Hypothesis 1

The histogram of the residuals shows the distribution for all observations. This plot (Minitab 14) indicates whether the dataset is skewed or whether there are outliers [90]. Based on a histogram of the residuals for Hypothesis 1, no evidence of skewness or outliers exists (Figure 8-6). This suggests that a transformation may not be necessary.

Figure 8-6 Histogram of the residuals for Hypothesis 1 (untransformed)

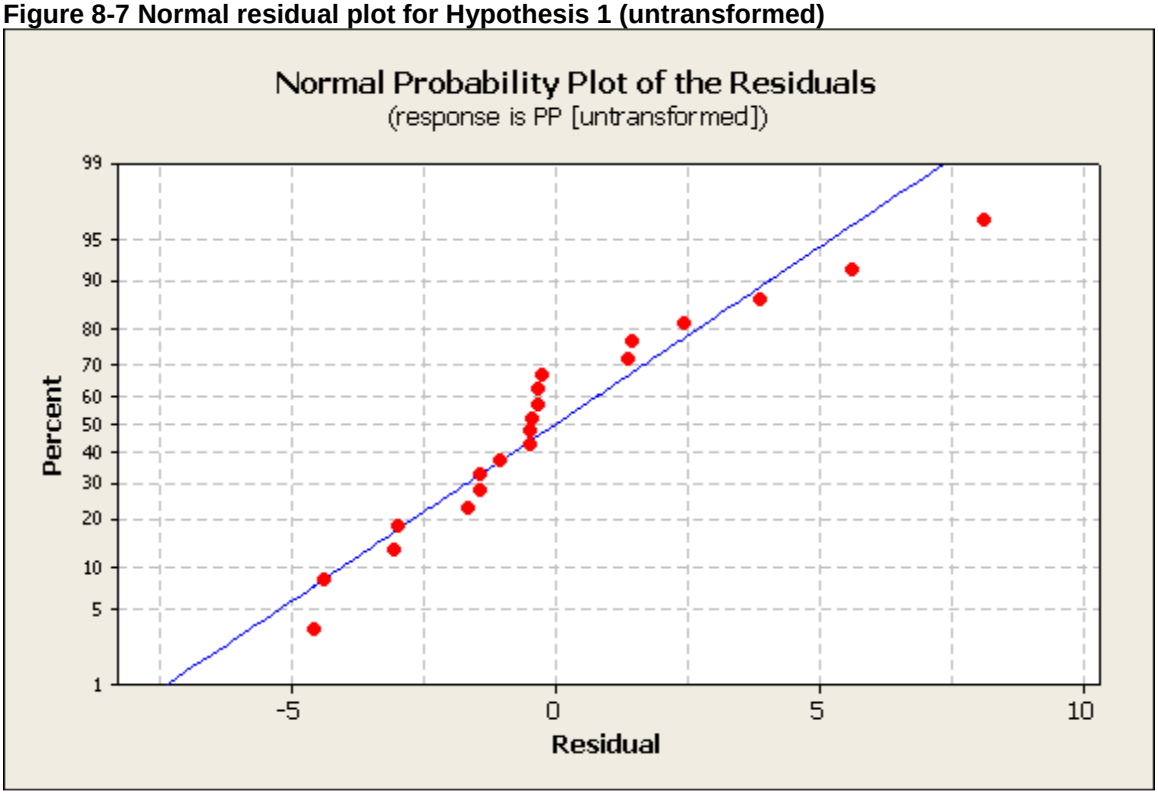


PP = product's productivity.

The normal plot of the residuals (Minitab 14) shows the "residuals" versus estimated values [90]. This plot indicates whether the data follows a normal distribution, or whether there are outliers.

In the normal plot for the software productivity data, the residuals appear to follow a straight line (Figure 8-7). There is no evidence of nonnormality or outliers.

Figure 8-7 Normal residual plot for Hypothesis 1 (untransformed)



PP = product's productivity.

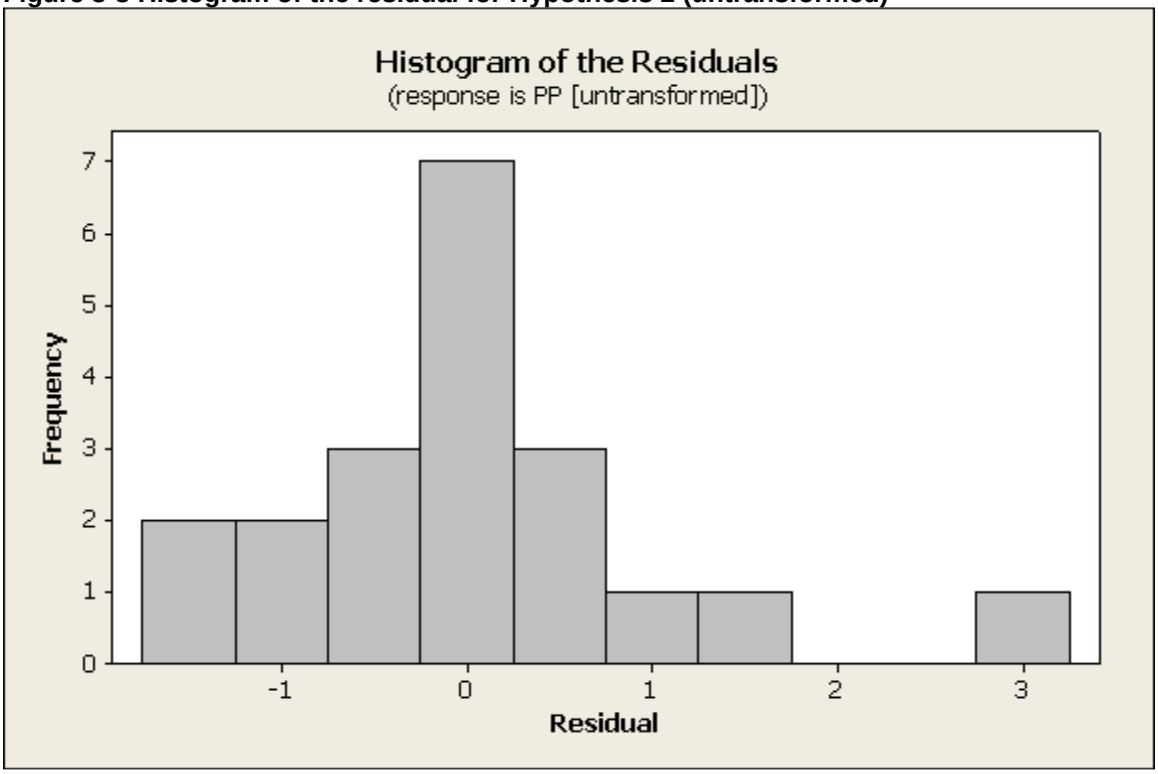
In summary, the two productivity residual plots confirmed no evidence of nonnormality, skewness, or outliers. Thus, ANOVA can be used for testing hypothesis 1.

8.2.2 Normality Test for Hypothesis 2

The General Linear Model (GLIM) through MINITAB 14 (Minitab Inc. 2003) was used for this hypothesis because one of the factors, complexity, did not have the same observations in each treatment (Table 7-15).

Figure 8-8 shows a histogram of the residuals for the product's productivity data using GLIM. This plot shows evidence of an outlier. The outlier suggests that a transformation may be necessary to test Hypothesis 2.

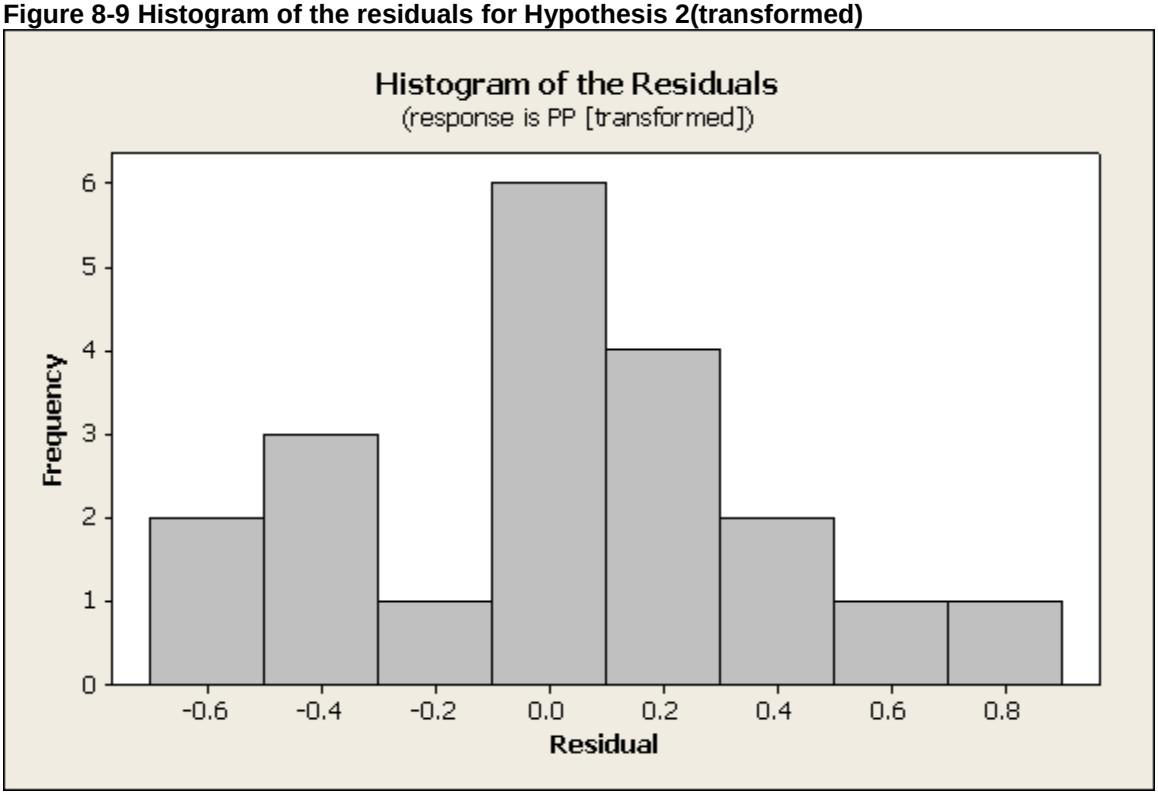
Figure 8-8 Histogram of the residual for Hypothesis 2 (untransformed)



PP = product's productivity.

Figure 8-9 show a histogram of the residuals for the transformed productivity data (lognormal) using GLIM. For the transformed plot, no evidence of skewness or outliers exists.

Figure 8-9 Histogram of the residuals for Hypothesis 2(transformed)



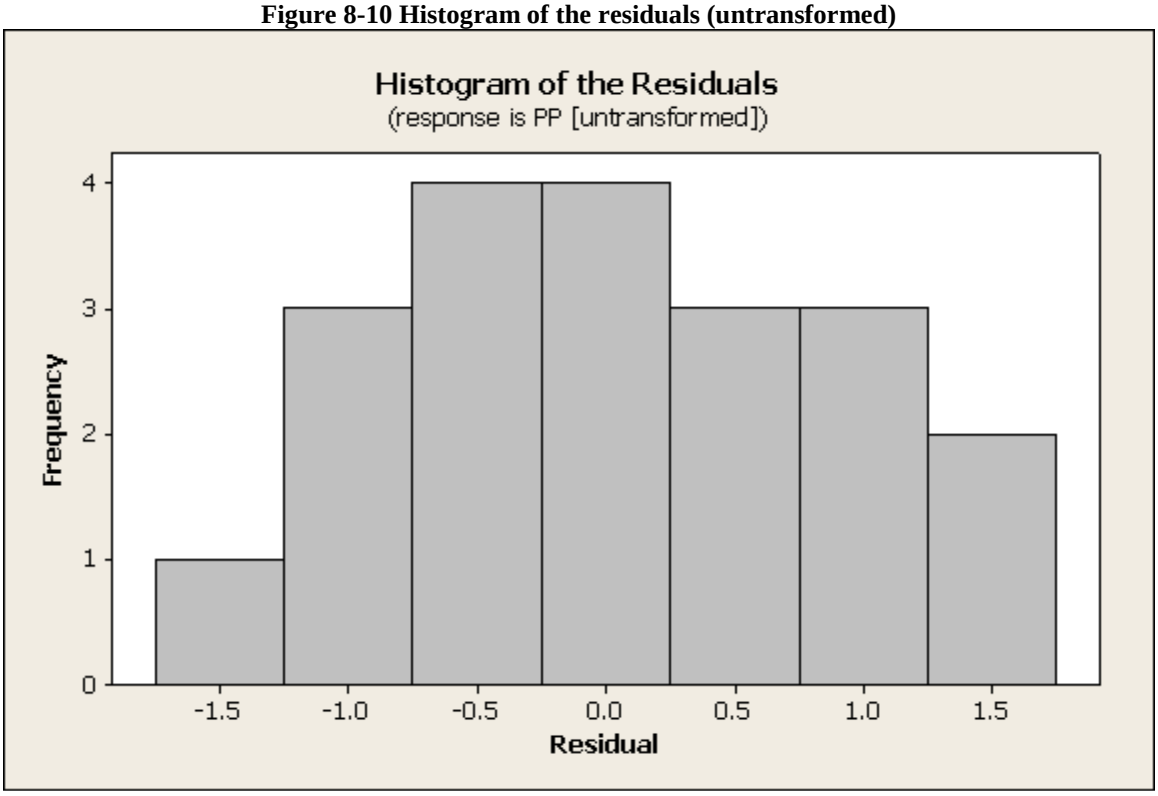
PP = product's productivity

The transformed histogram of the residuals confirmed no evidence of skewness or outliers. Thus, the transformed GLIM can be used for testing hypothesis 2.

8.2.3 Normality Test for Hypothesis 3

Figure 8-10 shows a histogram of the residuals (untransformed) for the productivity data. Based on this plot, no evidence of skewness or outliers exists. This suggests that a transformation may not be necessary.

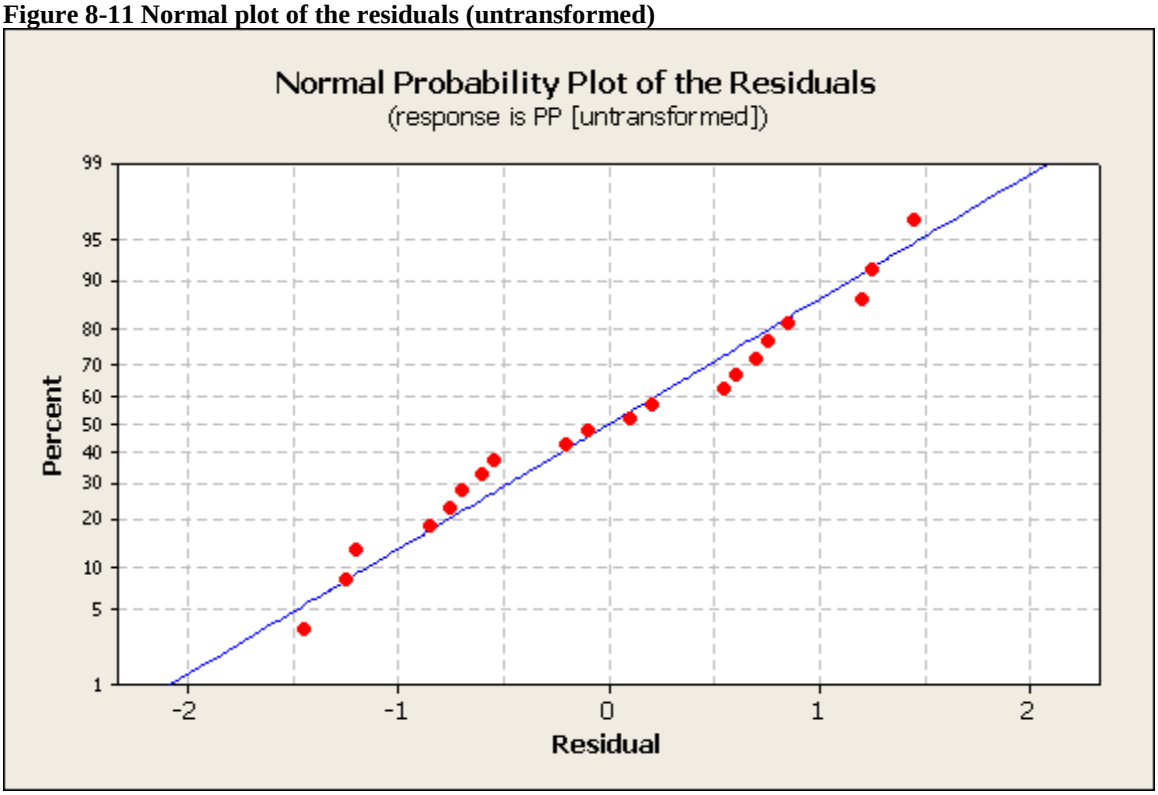
Figure 8-10 Histogram of the residuals (untransformed)



PP = product's productivity

Figure 8-11 shows a normal plot of the residuals for the productivity data. Based on this second plot, the residuals appear to follow a straight line. As with the histogram shown in Figure 8-10, this plot also indicates no evidence of nonnormality or outliers.

Figure 8-11 Normal plot of the residuals (untransformed)



PP = product's productivity

The two untransformed productivity residual plots shown in Figures 8-10 and 8-11 confirmed no evidence of nonnormality, skewness, or outliers. Thus, the two-way ANOVA was used for testing hypothesis 3.

8.3 Co linearity Test

"Pearson" [90] correlation test was necessary to determine whether there is a validity issue with the study given that there may be co linearity between the independent and control variables.

This section has two subsections. The first subsection uses correlation coefficients to measure the extent to which object-oriented method and control variables are related. The second subsection uses *P* value to test the relationship between object-oriented method and control variables.

8.3.1 Correlation Coefficient

The correlation coefficient ranges from negative (≥-1) to positive (≤+1) values [90]. The larger the absolute value, the stronger the correlation. Table 8-1 shows the correlation coefficients between object-oriented method, control variables, and productivity. The criterion for acceptance is 0.45. The results indicate the following:

- The correlation coefficient for the relationship between object-oriented method and productivity (0.50) is strong, indicating that when object-oriented method is chosen over traditional, the productivity tends to increase.

- The correlation coefficient for the relationship between quality and productivity (0.53) is strong, indicating that when the defect containment increases, the productivity tends to increase.

- In contrast, the correlation between complexity and productivity (-0.67) and between application domain (-0.62) and productivity are both strong and negative. This means that as real-time processing or application complexity increase, productivity tends to decrease.

- The quality factor bears a direct correlation (0.45) with capability maturity model integration.

Table 8-1 Correlation coefficient

	PP	*SIZE*	*AD*	*OOM*	*CPLX*	*PERS*	*RELY*	*QUAL*	*PVOL*	*CMMI*
PP	1.00									
SIZE	0.00	1.00								
AD	-0.62	-0.19	1.00							
OOM	0.50	0.22	0.01	1.00						
CPLX	-0.69	-0.01	0.82	-0.06	1.00					
PERS	0.00	0.10	-0.01	0.06	-0.06	1.00				
RELY	0.00	0.47	-0.25	0.01	0.05	0.43	1.00			
QUAL	0.57	-0.08	-0.42	0.05	-0.51	0.02	0.22	1.00		
PVOL	-0.10	0.01	0.23	0.08	-0.03	0.75	0.36	-0.11	1.00	
CMMI	0.04	0.33	-0.04	0.11	-0.02	-0.51	0.26	0.45	0.15	1.00

PP= product's productivity; SIZE = product size; AD = application domain; OOM = object-oriented method; CPLX = complexity; PERS = personnel capability; RELY = reliability; QUAL = quality; PVOL = platform volatility; CMMI = capability maturity model integration.

8.3.2 P Values

The *P* value shows whether the correlation coefficient is different from 0. A coefficient of 0 indicates no linear relationship between independent and control variables. If the *P* value is less or equal than α-level, then a strong correlation exist between object-oriented method and any of the control variables [90]. Table 8-2 shows the linear relationship between object-oriented method, control variables, and productivity. Using α-level of 0.05, the results indicate the following:

- The linear correlations between object-oriented method and quality (p = 0.974) is greater than 0.05. Thus, there is no relation between object-oriented method and quality.

- The linear correlations between reliability and productivity (p = 0.986) is greater than 0.05. Thus, that there is a relation between reliability and productivity.

- The linear correlations between personnel capability and productivity (p=0.986) is greater than 0.05. Thus, there is no relation between personnel capability and productivity.

- The linear correlations between platform volatility and productivity (p=0.986) is greater than 0.05. Thus, there is no relation between platform volatility and productivity.

Table 8-2 Correlation: *P* Values

	PP	*SIZE*	*AD*	*OOM*	*CPLX*	*PERS*	*RELY*	*QUAL*	*PVOL*	*CMMI*
PP										
SIZE	0.994									
AD	0.012	0.372								
OOM	0.020	0.445	1.000							
CPLX	0.001	0.961	0.000	1.000						
PERS	0.986	0.663	0.131	1.000	0.814					
RELY	0.986	0.035	0.475	1.000	0.843	0.061				
QUAL	0.008	0.744	0.007	0.974	0.023	0.945	0.342			
PVOL	0.689	0.965	0.057	1.000	0.914	0.000	0.117	0.645		
CMMI	0.879	0.162	0.126	1.000	0.936	0.021	0.273	0.049	0.519	

PP= product's productivity; SIZE = product size; AD = application domain; OOM = "object-oriented method"; CPLX = "complexity"; PERS = "personnel capability"; RELY = "reliability"; QUAL = quality;

PVOL = platform volatility; CMMI = capability maturity model integration.

8.3.3 Conclusion

The results indicate that the correlation coefficients for the relationship between object-oriented method and any of the control variables were less than 0.11, indicating that co linearity will not pose a threat to validity.

8.4 Statistical Analysis and Results

This section has three subsections. The first shows the statistical analysis results for Hypothesis 1. The second shows results for Hypothesis 2, and third shows results for Hypothesis 3.

8.4.1 Results for Hypothesis 1

"Analysis of variance (ANOVA)" through MINITAB 14 (Minitab Inc. 2003) was used to test whether object-oriented methods relate to productivity. This method was chosen because the dataset followed a normal distribution (see Section 8.2.1).

The results (Table 8-3) indicate that the effect of object-oriented method (*P* value is 0.019) on productivity is significant. Thus, projects developed with object-oriented method experienced higher productivity than projects developed with traditional methods.

Table 8-3 ANOVA: productivity versus object-oriented method

Factor	"DF	SS	MS	F-STAT	*P* value"
Object-Oriented Method	1	69.6	69.6	6.600	0.019
Error	18	189.7	10.5		
Total	19	259.2			

"S = 3.237", "R2 = 26.69%, " R2 (adjusted) = 22.61%"
DF = degrees of freedom; SS = sum of squares; MS = mean square.

8.4.2 Results for Hypothesis 2

The general linear model (GLIM) through MINITAB 14 (Minitab Inc. 2003) was used to perform ANOVA to test whether object-oriented method and complexity relate to software productivity. GLM rather than two-way ANOVA was chosen because the data for one factor, complexity, was unbalanced; it did not contain the same number of data points for each factor level (Table 7-15). A transformation (lognormal) also was necessary because the dataset exhibited nonormality.

The results (Table 8-4) indicate the following:

1. The interaction between object-oriented method and complexity (*P* value is 0.458) is not significant. Thus, the individual effects of either factor can be interpreted alone.

2. The effect of object-oriented method (*P* value is 0.002) on productivity is significant. This indicates that projects developed with object-oriented method experienced higher productivity than those developed with traditional methods.

3. The effect of complexity (p-value is 0.001) on productivity is also significant. This indicates that as the amount of real-time processing increase, more effort is expended to develop a line of code or a function point**.**

Table 8-4 GLIM: productivity (lognormal) versus object-oriented method and complexity

Factor	"DF	SS	SS-Adj.	MS	F-STAT	*P* value
Object-Oriented Method	1	5.00	4.1245	4.1245	15.7	0.002
Complexity	3	7.80	7.8030	2.6010	9.9	0.001
Interaction	3	0.73	0.7290	0.2430	0.9	0.458

Error	12	3.15	3.1460	0.2622
Total	19			

"S = 0.512022", "R2 = 81.14%", " R2(adjusted) = 70.13%""

DF = degrees of freedom; SS = sum of squares; MS = mean square, "Adj. =adjusted.

8.4.3 Results for Hypothesis 3

The results (Table 8-5) indicate the following:

1. The interaction between object-oriented method and application domain (*P* value is 0.013) is statistically significant. Thus, the individual effects of object-oriented method should not be interpreted alone.

2. The effect of object-oriented method (*P* value is 0.000) on productivity is significant. This indicates that projects developed with object-oriented method experienced higher productivity than those developed with traditional methods.

3. The effect of application domain (p-value is 0.000) is also significant. This indicates that one line of code (or function point) of radar software requires more time and effort to develop than a line of code for a mission planning software**.**

Table 8-5 Two-way ANOVA: productivity versus OOM and application domain

Factor	"DF	SS	MS	F-STAT	*P* value"
Object-Oriented Method	1	68.6	68.6	45.8	0.000
Application Domain	4	140.8	35.2	23.5	0.000
Interaction	4	32.8	8.2	5.5	0.013
Error	10	15.0	1.5		
Total	19	257.2			

"S = 1.225", "R2 = 94.17%", "R2 (adjusted) = 88.92%"
OOM = object-oriented method; DF = degrees of freedom; SS = sum of squares; MS = mean square.

9 Conclusion

9.1 Primary Findings

1. For the 20 military software projects in this study, object-oriented method was a significant factor affecting productivity (*P* value is 0.019) (Table 8-3). This significant effect is because projects developed in object-oriented methods reused more software and incurred less regression test labor than those developed in nonobject-oriented methods. Thus, it can be asserted that software reuse also improves productivity.

2. All ten military organizations showed a productivity gain with the use of object-oriented methods over traditional methods, such as structured analysis and design methods and object-based methods. This finding (Figures 8-1 and 8-2) is consistent with the perception that object-oriented method improves analysts' and programmers' productivity.

3. Object-oriented method (Table 8-3) explains 27% of the variation in software productivity. The variation in productivity becomes more significant (94%) when dataset is grouped by application domain (Table 8-5). Thus, the effect of object-oriented method on software productivity shall be interpreted along with application domain.

4. The productivity of a software product developed in a third-generation language was lower than of a software product developed in a fourth- or fifth-generation language (Figure 8-3). Most object-oriented projects are written in a fourth- or fifth-generation language.

9.2 Additional Findings (Software Quality)

1. Object-oriented method has no effect on software quality (defect containment), primarily because defect containment is an indicator of a team's capability to manage risks regardless of process, tool, or language being used. This result invalidates the paradigm that object-oriented methods can dramatically improve software quality (Tables 8-1 and 8-2).

2. The U.S. Department of Defense should adopt defect containment as the standard software quality measurement for two reasons. First, defect containment is the only measure (contrasted with reliability and defect density) to bear a linear relationship (Table 8-2) with software productivity (*P* value is 0.008). Second, comparisons across programming languages, varying product size units, and organizations are possible because defect containment is a relative percentage measure.

3. Defect containment bears a direct relationship with the capability maturity model integration (CMMI) level (Tables 8-1 and 8-2). This finding validates literature

[41] that substantiates that CMMI is a direct indicator of the capability of a software development process to remove defects before the software is shipped.

9.3 Additional Findings (Productivity Drivers)

1. When controlling for the effect of size, the most significant factors influencing software productivity in descending order are complexity, application domain, software quality, and object-oriented method.

2. This study proved that defect containment accounts for a significant variance in productivity (Tables 8-1 and 8-2). This finding is consistent with literature [41] that substantiates the hypothesis that effective front-end defect removal leads to improved quality of the end product.

3. Of the six COnstructive COst MOdel (COCOMO) variables examined in this study—complexity, reuse, personnel capability, reliability, platform volatility, and process maturity—only the complexity variable was the only to show a significant impact on productivity (Section 8.3). Further investigation is necessary because some of these variables did not contain the same number of observations for each factor level.

4. This study also revealed that meaningful software productivity comparisons can be made when projects are grouped by operating environment, application domain, size, and quality.

9.4 Limitations

Selecting the most appropriate productivity equation and controlling for the effect of significant influencing factors, such as size, operating environment, application domain, complexity, personnel capability, and platform volatility, reduced threats to validity.

Although this study mitigated those limitations, it still has two limitations:

1. A larger dataset (>20 projects) is required to increase validity and to examine collinearity between object-oriented method and control variables.

2. The intervening variables were controlled only within organizations. A future investigation should attempt to control for the impact of intervening variables across organizations. This can be achieved by asking each organization to provide data by specific operating environment, application domain, size range, and complexity.

9.5 Contributions

This study resulted in two major contributions:

1. This study revealed object-oriented method is appropriate for military weapon systems regardless of complexity (real-time, embedded), size, and design constraints (quality assurance levels).

2. Object-oriented methods and defect containment account for a significant variance in software productivity. These variables should be in all software cost models such as COCOMO II, SEER-SEM, PRICE S, and SLIM.

9.6 Future Research

The following topics may be considered for future research:

1. Examine the effect of model-driven architecture on software productivity by using a similar experimental design to compare it with object-oriented methods with a larger sample size.

2. Study the effect of object-oriented methods on software quality by using a unit independent measurement, such as defect containment, similar experimental design, and larger dataset.

3. Pioneer a simplified software cost estimating model by investigating the relationship between object-oriented methods, defect containment, complexity, operating environment, application domain, size, and staff-months.

10 References

[1] IEEE, 1993, "IEEE Standard for Software Productivity Metrics," *IEEE Std 1045-1992*.

[2] IEEE, 1998, "*Industry Implementation of International Standard ISO/IEC 12207: 1995. (ISO/IEC 12207 Standard for Information Technology - Software Life Cycle Processes - Implementation Considerations," IEEE/EIA std 12207.2-1997*

[3] Altmann, J., and Pomberger, G., 1999, "Cooperative Software Development: Concepts, Model and Tools," *Proc. Technology of Object-Oriented Languages and Systems (TOOLS 30)*, pp. 194–207.

[4] Arango, G., Shoen, E., and Pettengill, R., 1993, "A Process for Consolidating and Reusing Design Knowledge,"

[5] Basili, V. R., Briand, L. C., and Melo, L., 1996, "How Reuse Influences Productivity in Object-Oriented Systems," *J Commun. ACM*, 39(10), pp. 104–116.

[6] Boehm, B., 1981, *Software Engineering Economics*, Prentice-Hall, Inc., Englewood Cliffs, N.J., pp. 1–730.

[7] Boehm, B., 2000, "Safe and simple software cost analysis," *Software, IEEE*, 17(5), pp. 14–17.

[8] Boehm, B., et al., 2000, *Software Cost Estimation with COCOMO II*, Prentice-Hall, Inc., Upper Saddle River, NJ, pp. 1–493.

[9] Boehm, B., and Papaccio, P., 1988, "Understanding and Controlling Software Costs," *Software Engineering, IEEE Transactions on*, 14(10), pp. 1462–1477.

[10] Booch, G., 1994, *OBJECT-ORIENTED ANALYSIS AND DESIGN with Applications*, 2nd edition, Addison-Wesley, Santa Clara, CA, pp. 1–575.

[11] Bredenfeld, A., 1998, "APICES-rapid Application Development with Graph Pattern," *Proc. 9th Int'l Workshop Rapid System Prototyping (RSP '98)*, pp. 25–30.

[12] Lee, B-J., Moon, B-R., and Wu, C-S., 1998, "Optimization of Multi-way Clustering and Retrieval Using Genetic Algorithms in Reusable Class Library," *Proc. 5th Asia-Pacific Software Eng. Conf. (APSEC '98)*, pp. 4–11.

[13] Cardelli, L., and Wegner, P., 1985, “On understanding types, data abstraction, and polymorphism,” *ACM Comput. Surv*., 17(4), pp. 471–523.

[14] Channarukul, S., Charoenvikrom, S., and Daengdej, J., 2005, “Case-based Reasoning for Software Design Reuse,” *IEEE Aerospace Conf*., pp. 4296–4305.

[15] Childs, B., and J. Sametinger, 1996, “Literate Programming and Documentation Reuse,”

[16] Chulani, S., Boehm, B., and Steece, B., 1999, “Bayesian Analysis of Empirical Software Engineering Cost Models,” *Software Engineering, IEEE Transactions on*, 25(4), pp. 573–583.

[17] Clark, B., Devnani-Chulani, S., and Boehm, B., 1998, “Calibrating the COCOMO II Post-Architecture model,” *Proc. Int’l Conf. Software Eng. (ICSE ’98)*, pp. 477–480.

[18] Clark, B. K., 1997, “The Effects of Software Process Maturity on Software Development Effort,” Technical Report No. AAT 9816016, University of Southern California, CA.

[19] Coad, P., and Yourdon, E., 1998, *OBJECT-ORIENTED DESIGN*, Prentice-Hall, Inc., Upper Saddle River, NJ, 1–190.

[20] Daly, J., Miller, J., Brooks, A., Roper, M., and Wood, M., 1995, “A Survey of Experiences Amongst Object-Oriented Practitioners,” *Proc. Asia-Pacific Software Eng. Conf. (APSEC ’95)*, pp. 137–146.

[21] Kim, D-K., Jung, K-T., and Kim, C-K., 1998, “Techniques for Systematically Generating Framework Diagram Based on UML,” *Proc. Asia Pacific Software Eng. Conf. (APSEC ’98)*, pp. 203–210.

[22] Lee, D-G., Lee, J-K., Lee, B-S., and Lee, H-H., 1998, “A Translation Approach for Object Oriented CHILL Program in Development of Telecommunication Software," *Proc. IEEE Global Telecommunications Conf. (GLOBECOM ’98)*, pp. 1585–1589.

[23] Department of Defense, 1989, *DOD-STD-2167A, Defense System Software Development*, ceng.metu.edu.tr/~gtumuklu/web/SE548/Reading%20Material/2167/dod2167a.pdf, Site accessed on 16 November 2009.

[24] Department of Defense, 1994, *MIL-STD-498, Software Development and Documentation*, www2.umassd.edu/swpi/DOD/MIL-STD-498/498-STD.PDF Site accessed on 16 November 2009.

[25] Dubois, T. A., and R. Hein.,2000, “The Feasibility of Applying Object-Oriented Technologies to Operational Flight Software,”

[26] Ellis, M., 1993,“Object oriented CASE: The TPS Life Cycle Evolution”,

[27] Etzkorn, L. H., Davis, C. G., Bowen, L. L., Etzkorn, D. B., Lewis, L. W., Vinz, B. L., and Wolf, J. C., 1996, “A Knowledge-Based Approach to Object-Oriented Legacy Code Reuse,” *Proc. IEEE 2nd Int’l Conf. Eng. of Complex Computer Systems (ICECCS’96)*, pp. 493–496.

[28] Faulk, S., et al., 1992, “The Core Method For Real-Time Requirements,” *Software IEEE*, 9(5): p. 22–33.

[29] Furey, S., 1997, “Why We Should Use Function Points [software metrics],” *Software, IEEE*, 14(2), pp. 28–30.

[30] Galorath, 2001, *SEER-SEM™ User's Manual*, Galorath, Inc., El Segundo, CA, Chaps.1-4, 6-6, 15-6.

[31] Ghiassi, M., Ketabchi, M. A., and Sadeghi, K. J., 1992, “An Integrated Software Testing System Based on an Object-Oriented DBMS,” *Proc. 25th Hawaii Int’l Conf. System Sciences (HICSS-25)*, pp. 101–109.

[32] Guenther, W., and Wackerbarth, G., 1993, “Object-Oriented Design of ISDN Call-Processing Software,” *Communications Magazine, IEEE*, 31(4), pp. 40–45.

[33] Shen, H-H., Liu, C., Shen, M-M., and Zheng, W-M., 1999, “An Algorithm for Describing Object-Oriented Software Architecture Using Graph,” *Proc. Technology of Object-Oriented Languages and Systems (TOOLS 31)*, pp. 225–231.

[34] Henderson-Sellers, B., 2000, “The OPEN Framework For Enhancing Productivity,” *Software, IEEE*, 17(2), pp. 53–58.

[35] Liao, H-C., Chen, M-F., Wang, F-J., and Dai, J-C., 1997, “Using a Hierarchical Thesaurus for Classifying and Searching Software Libraries,” *Proc. 21st Ann. Int’l Computer Software and Applications Conf. (COMPSAC '97)*, pp. 210–216.

[36] Jeffery, D. R., and Low, G., 1990, “Calibrating Estimation Tools for Software Development,” *Software Engineering Journal*, 5(4), pp. 215–221.

[37] Johnson, R. L., 1991, “An Assessment of Function Point Analysis as an Emerging Software Productivity Technique,” Ph.D thesis, The University of Texas at Arlington: TX.

[38] Jones, C., 1995, “Backfiring: Converting Lines of Code to Function Points,” *Computer*, 28(11), pp. 87–88.

[39] Jones, C., 2003, “Variations in Software Development Practices,” *Software IEEE*, pp. 22–27.

[40] Jordan A., and Frame, C. 1996, *Object-Oriented Development Process for Department of Defense Information Systems*, Institute for Defense Analyses.

[41] Kan, S., 2002, *Metrics and Models in Software Engineering*, Second Edition, Addison-Wesley, Boston, MA, pp. 1–508.

[42] Kang, K. C., 1998, “Feature-Oriented Development of Applications for a Domain,” *Proc. 5th Int’l Conf. Software Reuse*, pp. 354–355.

[43] Karahasanovic, A., and Sjoberg, D. I. K., 2001, “Visualizing Impacts of Database Schema Changes–A Controlled Experiment,” *Proc. IEEE Symp. Human-Centric Computing Languages and Environments (HCC ’01)*, pp. 358–365.

[44] Katic, N., et al., 1996, “Bridging the Gap between Structured Requirements and Object-Oriented Analysis and Design,”

[45] Klepper, R., and Bock, D., 1995, “Third and Fourth Generation Language Productivity Differences,” *Commun. ACM*, 38(9), pp. 69–79.

[46] Lewis, J. A., Henry, S. M., Kafura, D. G., and Schulman, R. S., 1992, “An Empirical Study of the Object-Oriented Paradigm and Software Reuse,” *Proc. Conf. Objected-Oriented Programming Systems, Languages, and Applications (OOPSLA ’91)*, pp. 184–196.

[47] Liang, T., and Noore, A., 2003, “Multistage Software Estimation,” *Proc. 35th Southeastern Symp. System Theory*, pp. 232–236.

[48] Llorens, J., Amescua, A., and Velasco, M., 1996, “Software Thesaurus: A Tool for Reusing Software Objects,” *Proc. 4th Int’l Symp. Assessment of Software Tools (SAST '96)*, pp. 99–103.

[49] Love, T., 1998, “The Economics of Reuse [of Software],”

[50] Low, G., and Huan, S., 1999, “Impact of Object Oriented Development on Software Quality,” *Proc. Software Technology and Engineering Practice (STEP ’99)*, pp. 3–11.

[51] Low, G. C., and Jeffery, D. R., 1990, “Function Points in the Estimation and Evaluation of the Software Process,” *Software Engineering, IEEE Transactions*, 16(1), pp. 64–71.

[52] Luckey, P. H., Pittman, R. M., and Saxton, R. D., 1992, “Integrated CASE: Our Experience with Teamwork and Rational,” *Proc. 2nd Symp. Assessment of Quality Software Development Tools (QSDT ’92)*, pp. 271–284.

[53] Mancl, D., and Havanas, W., 1990, “A Study of the Impact of C++ on Software Maintenance,” *Proc. Conf. Software Maintenance*, pp. 63–69.

[54] Matson, J. E., Barrett, B. E., and Mellichamp, J. M., 1994, “Software Development Cost Estimation Using Function Points,” *Software Engineering, IEEE Transactions*, 20(4), pp. 275–287.

[55] Maxwell, K. D., Van Wassenhove, L., and Dutta, S., 1996, “Software Development Productivity of European Space, Military, and Industrial Applications,” *Software Engineering, IEEE Transactions*, 22(10), pp. 706–718.

[56] Miyazaki, Y., and Mori, K., 1985, “COCOMO Evaluation and Tailoring,” *Proceedings of the 8th international conference on Software engineering*, England IEEE Computer Society Press, London: UK..

[57] Moser, S., and Nierstrasz, O., 1996, “The Effect of Object-Oriented Frameworks on Developer Productivity,” *Computer*, 29(9), pp. 45–51.

[58] Mukhopadhyay, T., and Kekre, S., 1992, “Software Effort Models for Early Estimation of Process Control Applications,” *Software Engineering, IEEE Transactions*, 18(10), pp. 915–924.

[59] Nakajima, S., Tomobe, M., Miki, M., and Hayashi, H., 1996, “Object-Oriented Development Methodology for Telecommunication Network Management Software,” *Proc. Global Telecommunications Conf. (GLOBECOM ’96)*, pp. 665–669.

[60] Nebel, W., and Schumacher, G., 1996, “Object-Oriented Hardware Modelling—Where to Apply and What Are the Objects?” *Proc. Conf. European Design Automation (EURO-DAC '96/EURO-VHDL '96)*, pp. 428–433.

[61] Nishiyama, T., 1999, “Using a ‘Process Warehouse’ Concept: A Practical Method for Successful Technology Transfer,” *Proc. 2nd IEEE Int’l Symp. Object-Oriented Real-Time Distributed Computing (ISORC '99)*, pp. 117–120.

[62] Pao-Ann, H., 1998, “RTFrame: an Object-Oriented Application Framework for Real-Time Applications,”

[63] Pao-Ann, H., et al., 2001, “Verifiable Embedded Real-Time Application Framework,”

[64] Pao-Ann, H., et al., 2004, “VERTAF: An Application Framework for the Design and Verification of Embedded Real-Time Software,” *Software Engineering, IEEE Transactions*, 30(10), pp. 656–674.

[65] Pao-Ann, H., et al., 2002, “VERTAF: an Object-Oriented Application Framework for Embedded Real-Time Systems,”

[66] Pereira, C.E., 1994, “Enhancing Real-Time Object-Oriented Analysis Methods,”

[67] Perez, M., and G. Herrera., 1997, “Metrics of Methodology: FUSION Case,”

[68] Petillo, J., et al., 2002, “The MICHELLE Three-Dimensional Electron Gun and Collector Modeling Tool: Theory and Design,” *Plasma Science IEEE Transactions on*, 30(3), pp. 1238–1264.

[69] Port, D., and McArthur, M., 1999, “A Study of Productivity and Efficiency for Object-Oriented Methods and Languages,” *Proc. 6th Asia-Pacific Software Eng. Conf. (APSEC ’99)*, pp. 128–135.

[70] Potok, T., and Vouk, M., 1995, “Development Productivity for Commercial Software Using Object-Oriented Methods,” *Proceedings of the 1995 conference of the Centre for Advanced Studies on Collaborative Research*, IBM Press, Toronto, Canada.

[71] PRICE, 1998, *Your Guide to PRICE S: Estimating Cost and Schedule of Software Development and Support*, PRICE Systems, L.L.C, Mount Laurel: NJ, pp. 1–452.

[72] Putnam, L., and Myers, W., 1991, *Measures for Excellence: Reliable Software on Time*, Within Budget, Prentice Hall PTR, Upper Saddle River, NJ, pp. 1–356.

[73] QSM, Inc., 2001, *User's Guide to SLIM-Estimate 5.0*, 2nd Edition, QSM, Inc., McLean,VA, pp. 1–360.

[74] Rice, G. F., Corcoran, S., Leiberman, D., and Powers, R., 1997, “Ada 95 and Software Reuse,” *16th Digital Avionics Systems Conf. (DASC ’97)*, pp. 11–18.

[75] Rosa, W., 2008, “Next-Generation, Systems and Software Cost Estimation”, *Proceedings of the 23rd International COCOMO Forum*.

[76] Rumbaugh, J., Jacobson, I., and Booch, G., 2005, *The Unified Modeling Language Reference Manual*, 2nd edition, Addison-Wesley, Boston, MA, pp. 1–707.

[77] Sangdon, L., Choi, H. S., Yang, Y., and Lee, S., 2000, “XML-Based Retrieval of Object-Oriented Frameworks,” *IEEE Int’l Conf. Systems, Man, and Cybernetics (SMC ’00)*, pp. 2953–2958.

[78] Sangdon, L., et al., 1999, “Storage and Management of Object-Oriented Frameworks,” *IEEE Int’l Conf. Systems, Man, and Cybernetics (SMC ’99)*, pp. 762–767.

[79] Wu, S. I. K., 2006, “The Quality of Design Team Factors on Software Effort Estimation,” *IEEE Int’l Conf. Service Operations and Logistics, and Informatics (SOLI ’06)*, pp. 6–11.

[80] Singhal, A., and Chi-Yuan, L. 1995, “Object Oriented Data Modeling for VLSI/CAD,” *Proc. 8th Int’l Conf. VLSI Design*, pp. 25–29.

[81] Smith, R.K., 1998, “An Effort Estimation Model for Component-Based Software Development,” *The University of Alabama: AL*.

[82] Sugawara, M., T. Hagiwara, and G. Numajiri. 1997, “Design Extraction System for Rapid Development of Object-Oriented Switching Software,”

[83] Tyler, D. F., 1999, “Java-Based Automated Test Systems: Management Considerations for an Open Architecture for Test,” *IEEE Systems Readiness Technology Conf. (AUTOTESTCON '99)*, pp. 699–706.

[84] Watanabe, S., 1997, “Professionalism Through OO and Reuse,” *IEEE Software*, p. 26.

[85] Xiangzhu, G., and Lo B., 1994, “An Integrated Software Cost Model Based on COCOMO and Function Point Approaches,” *Proc. Software Education Conf.*, pp. 86–93.

[86] Xu, X. L., Wang, L. Y., and Zhou, H., 2003, “An Object-Oriented Framework for Automatic Test Systems,” *Proc. IEEE Systems Readiness Technology Conf. (AUTOTESTCON ’03)*, pp. 407–410.

[87] Yoda, I., K. Yata, and N. Fujii., 1994, "Object Oriented TMN Based Operations Systems Development Platform,"

[88] Yu, W.D., 1990, "A Modeling Approach to Software Cost Estimation," *Selected IEEE Journal on Areas in Communications*, 8(2), pp. 309–314.

[89] The International Software Benchmarking Group, 2007, *ISBSG Estimating, Benchmarking & Research Release 10*, www.isbsg.org, Site accessed on 14 March 2007.

[90] Minitab Inc., 2003, *MINITAB 14*, www.minitab.com Site accessed in 2005 [student license purchased online]

[91] Rosa, W., Boehm, B., Clark, B., Madachy, R., Reifer, D., 2009, "Software Cost Metrics Manual", *Proceedings of the 42nd Department of Defense Cost Analysis Symposium"*,

[92] Wikipedia, 2009, *DOD-STD-2167A*, en.wikipedia.org/wiki/DOD-STD-2167A Site accessed on 16 November 2009.

[93] Wikipedia, 2009, *DOD-STD-498*, en.wikipedia.org/wiki/MIL-STD-498 Site accessed on 16 November 2009

11 Appendices

APPENDIX A Software Data Collection Questionnaire 94

APPENDIX B Rating Scale for the Productivity Drivers 111

APPENDIX C Raw Project Data 112

APPENDIX A Software Data Collection Questionnaire

APPENDIX A Software Data Collection Questionnaire

Introduction

The Air Force Cost Analysis Agency (AFCAA) in conjunction with the School of Engineering and Applied Science at the George Washington University is conducting research to determine whether defense organizations are achieving productivity gains with the use of Object-Oriented Methods.

A fundamental requirement for such research is to collect data on two software projects. To satisfy this requirement, each organization will be asked to provide data on two homogeneous software projects using the same questionnaire. One project developed in object-oriented methods, while the other project developed in structured methods. These two projects should be similar in terms of application, platform, product size, process maturity, team experience, language experience, tools, and product constraints.

The questionnaire has the same fields addressed in the COCOMO II Data Collection Questionnaire (Version 1.9) plus a new field for Object-Oriented Methods. This questionnaire attempts to address two different levels of data granularity: project level and component level. The project level of granularity is data that is applicable for the whole project. This data includes attributes such as application type, schedule and development activity being reported. Component level data consists of input effort, output product, object-oriented method and intervening variables.

The data you contribute is important to us, and we will follow necessary measures to safeguard your contribution so as not to compromise company proprietary information. Thank you for your willingness to participate in this research.

Sincerely,

Wilson Rosa
Technical Advisor
Information Technology Division
Air Force Cost Analysis Agency
(703) 604-0395 office
(703) 604 6646 fax
wilson.rosa@pentagon.af.mil

Data Submission Address:

Mr. Wilson Rosa
Information Technology Division
Air Force Cost Analysis Agency
201 12th Street, Suite 403
Arlington, VA 22202 (U.S.A)

Project level Information

As described in the introduction section of this questionnaire, project level is applicable for the whole project. This includes aspects such as application type and deve lopment activity being reported.

General Information

Organization Name . Provide the complete name and address of your organization.

_____

Project Name . Provide the complete name of the project.

_____

Date Pr epared . This is the date the data elements were collected for submission.

_____

Application Type . This field captures a broad description of the type of activity this software application is attempting to perform.

Circle One:

Avioni cs	Operating Systems	Signal Processing
Command & Control	Mission Planning	Simulation
Diagnostics	Process Control	Software Dev. Tools
Engineering & Science	Radar	Testing
MIS	Real -Time	Utilities

Other: _____

Development Ty pe. Is the project a new software product or an upgrade of an existing product?

Circle One:

New Product Upgrade

Development Process . This is a description of the software process used to control the software development (e.g. waterfall, spiral, etc.) .

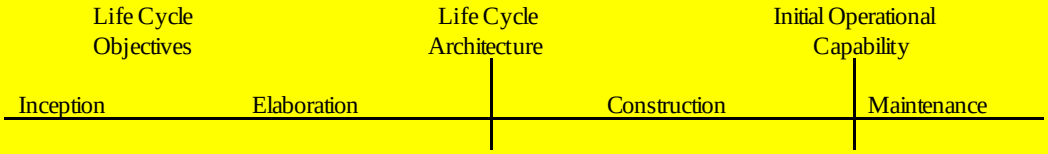
_____

Step in process . This field captures information about the projectÍs position in its development process. The answers depend on the process model being followed.

Requirements Design Code
Unit Test Integration/Test Maintenance
Completed

Other: _____

MBASE Phase . Phase refers to the aggregate of activities between the life cycle anchor points. The four phase s, based on the Rational Unified Process (RUP), and the anchor points are shown on the timeline below. Please circle the most advanced anchor point (milestone) the project has achieved.



The COCOMO II model covers the effort required from the completion of the LCO to IOC. If you are using a waterfall model, the corresponding milestones are the Software Requirements Revi ew, Preliminary Design Review, and Software Acceptance Test.

Development Process Iteration If the product development process is iterative, e.g. spiral or represents a released version, which iteration is it?

_____

If the development process phases are iterative, e.g. spiral, how many iterations have there been?

`

Inception: _____ Elaboration: _____ Construction: _____

Success Rating for Project . This specifies the degree of success for the project.

- Very Successful; did al most everything right
- Successful; did big things right
- OK; stayed out of trouble
- Some Problems; took some effort to keep viable
- Major Problems; would not do this project again

Circle One:

Very Successful Successful OK Some Problems Major Problems

A. Schedule

Year of Development . For reporting of historical data, please provide the year in which the software development was completed. For periodic reporting, put the year of this submission or leave blank.

_____

Schedule Months. For reporting of historical data, provide the number of calendar months from the time the development began through the time of its completion. For periodic reporting, provide the number of months in this development activity.

Circle the life-cycle phases that the schedule covers:

Inception	Elaboration	Construction	Maintenance
Life Cycle Objectives	Life Cycle Architecture	Initial Operational Capability	

B. Distribution of Effort and Schedule by Phase

This subsection has additional metrics that are required to calibrate the distribution of effort and schedule by phase. Please fill this out if the necessary information is available.

Total Effort (Staff-Hours). Divide the total effort required for the project into the amount of effort (in Staff-Hours) required for each of the following three phases: Inception, Elaboration, and Construction

	LCO Inception	LCA Elaboration	IOC Construction
Effort Distribution			

Schedule Months . Divide the total time for development (schedule) required for the project into schedule (in Calendar Months) required for each of the following three phases: Inception, Elaboration, and Construction

	LCO Inception	LCA Elaboration	IOC Construction
Schedule Distribution			

Component Level Info rmation

General Information

If the whole project is being reported as a single component, then skip to the next section.

If the data being submitted is for multiple components that comprise a single project, then it is necessary to identify each compone nt with its project. Please fill out the section for each component and attach all of the component sections to the project sections describing the overall project data.

Component Name . Provide the complete name of the componen t.

_____ _

Component Description . Provide the functional description of the component

Application Type . This field captures a broad description of the type of activity this software application is attempting to perform.

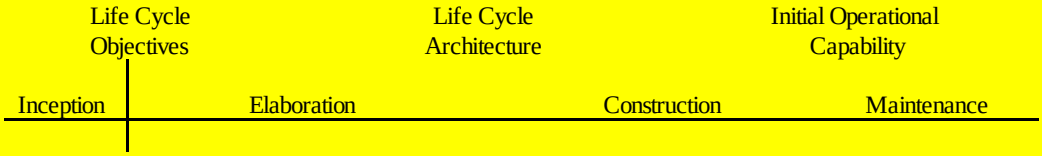
Circle One:

Avionics	Operating Systems	Signal Processing
Command & Control	Mission Planning	Simulation
Diagnostics	Process Control	Software Dev. Tools
Engineering & Science	Radar	Testing
MIS	Real -Time	Utilities

Other: _____

A. Input Effort

Total Effort (Staff-Hours). Circle the life-cycle phases that the effort data covers:



Total Effort _____ _____

Hours / Staff-Month. Indicate the average number of hours per staff-month experienced by your organization.

_____

Labor breakout. Indicate the percentage of labor for different categories, e.g. Managers, S/W Requirement Analysts, Designers, CM/QA Personnel, Programmers, Testers for each phase of software development:

Categories / Phase	Requirements	Design	Code	Test	Management	CM, QA, Documentation
Inception						
Elaboration						
Construction						

A. Project's Size

Percentage of Code Breakage . This is an estimate of how much the requirements have changed over the lifetime of the project. It is the percentage of code thrown away due to requirements volatility. For example, a project that delivers 100,000 instructions but discards the equivalent of an additional 20,000 instructions would have a breakage of 20.

_____

SLOC Count Type . When reporting size in source lines of code, please indicate if the count was for logical SLOC or physical SLOC. If both are available, please submit both types of counts. If neither type of count applies to the way the code was counted, please describe the method.

Circle One:

Logical SLOC	Physical SLOC (carriage returns)
Physical SLOC (semicolons)	Non-commented/Non-Blank SLOC

Programming Language . If you are using the Early Design or Post-architecture model, enter the language name that was used in this component (e.g. Ada 83, C, C++, COBOL, FORTRAN) and the amount of usage if more than one language was used.

Language Used	Percentage Used

New SLOC (NSLOC). This is the number of new source lines of code (SLOC) generated .

_____

Adapted SLOC (ASLOC) . Enter the amounts for the preexisting SLOC (modified plus unmodified)

_____ _____

Design Modified DDMA. Enter percentage of design modified for ASLOC.

_____

Code Modified DCMA. Enter the percentage of code modified for ASLOC.

_____

Integration and Test Modified DIMA. Enter the percentage of the adapted software's (ASLOC) original integration and test effort expended.

A. Object -Oriented Method

The object -oriented method is evaluated based on three elements: analysis method, design method, and programming method.

	Nominal	High	Very High
OOM	Used Non -Object -Oriented Methods for: • Analysis • Design • Programming	Used 1 or 2 Object - Oriented (OO) Methods listed below: • OO Analysis • OO Design • OO Programming	Used Full Object - Oriented (OO) Methods for: • OO Analysis • OO Design • OO Programming

Enter the rating level for object -oriented method used:

Object -Oriented Method (OOM). This cost driver measures the level of object -orientation used in developing the software product .

Nominal	High	VERY High

A. Intervening Variables

These are the 22 cost drivers used in the COCOMO II Post Architecture Model in order to adjust the nominal effort, Staff-Hours, and to reflect the software product under development. They are grouped into five categories: Scale Factors, Product Factors, Platform Factors, Personnel Factors, and Project Factors.

Scale Factors

Driver	Very Low	Low	Nominal	High	VERY High	Extra High
PREC	Thoroughly unprecedented	Largely unprecedented	Somewhat un precedented	Generally familiar	Largely familiar	Thoroughly familiar
FLEX	Rigorous	Occasional relaxation	Some relaxation	General conformity	Some conformity	General goals
RESL*	Little (20%)	Some (40%)	Often (60%)	Generally (75%)	Mostly (90%)	Full (100%)
TEAM	Some difficult interactions	Basically cooperative interactions	Largely cooperative interactions	Highly cooperative	Seamless interactions	

* % significant module interfaces specified, % significant risks eliminated .

Enter the rating level for the fir st four cost drivers.

Precedenteness (PREC). If the product is similar to several previously developed products, then the precedenteness is high. See the model COCOMO II Manual for more details.

Very Low	Low	Nominal	High	VERY High	Extra High	Very Low

Development Flexibility (FLEX) . This cost driver captures the amount of constraints the product has to meet. The more flexible the requirements, schedules, or interfaces, for example, the higher the rating.

Very Low	Low	Nominal	High	VERY High	Extra High	Very Low

Architecture / Risk Resolution (RESL) . This cost driver captures the thoroughness of definition and freedom from risk of the software architecture used for the product.

Very Low	Low	Nominal	High	VERY High	Extra High	Very Low

Team Cohesion (TEAM) . This cost driver accounts for the sources of project turbulence and extra effort due to difficulties in synchronizing the projectÍs stakeholders: users, customers, developers, maintainers, interfaces, and others.

Very L ow	Low	Nominal	High	VERY High	Extra High	Very Low

Process Maturity (PMAT). The procedure for determining PMAT is organized around the Software Engineering InstituteKŞ Capability Maturity Model (CMM). The time period for reporting process maturit y is at the time the project was initiated. We are interested in the capabilities practiced at the project level more than the overall organizationKŞ capabilities. ńOverall Maturity Levelī is a response that captures the result of an organized evaluation based on the CMM. ńNo Responseī means you do not know or will not report the process maturity at the Capability Maturity Model level.

_ No Response

Overall Maturity Level

_ CMM Level 1 (lower half)

_ CMM Level 1 (upper half)

_ CMM Level 2

_ CMM Level 3

_ CMM Level 4

_ CMM Level 5

Basis of Estimate

_ Software Process Assessment (SPA)

_ Software Capability Evaluation (SCE)

_ Interim Process Assessment (IPA)

_ Other: _____

Product Factors

For maintenance projects, identify any differences between the base code and modified code Product Cost Drivers (e.g. complexity).

	Very Low	Low	Nominal	High	Very High	Extra High
RELY	Slight inconvenience	Low, easily recoverable losses	Moderate, easily recoverable losses	High financial loss	Risk to human life	
DATA		DB bytes / Pgm SLOC < 10	10 ? D/P < 1000	100 ? D/P < 1000	D/P ? 1000	
RUSE		None	across project	across program	across product line	across multiple product lines
DOCU	Many life - cycle needs uncovered	Some life - cycle needs uncovered	Right -sized to life -cycle needs	Excessive for life -cycle needs	Very excessive for life -cycle needs	

Enter the rating level for the four cost drivers.

Required Software Reliability (RELY). This is the measure of the extent to which the software must perform its intended function over a period of time.

Very Low	Low	Nominal	High	VERY High	Extra High	Don't Know

Data Size (DATA). This measure attempts to capture the affect large data requirements have on product development (e.g. testing). The rating is determined by calculating D/P, where D is the number of bytes of data in the database at the end of (and necessary for) testing, and P is the number SLOC.

Very Low	Low	Nominal	High	VERY High	Extra High	Don't Know

Develop for Reuse (RUSE). This cost driver accounts for the additional effort needed to construct components intended for reuse on the current or future projects.

Very Low	Low	Nominal	High	VERY High	Extra High	Don't Know

Documentation match to life-cycle needs (DOCU). This captures the suitability of the project's documentation to its life -cycle needs.

Very Low	Low	Nominal	High	VERY High	Extra High	Don't Know

Product Complexity (CPLX):

	Control Operations	Computational Operations	Device-dependent Operations	Data Management Operations	User Interface Management Operations
Very Low	Straight-line code with a few non-nested structured programming operations: DOs, CASEs, IFTHENELSEs. Simple module composition via procedure calls or simple script	Evaluation of simple expressions: e.g., A=B+C*(D-E)	Simple read, write statements with simple formats.	Simple arrays in main memory. Simple COTS-DB queries, updates.	Simple input forms, report generators.
Low	Straightforward nesting of structured programming operators. Mostly simple predicates	Evaluation of moderate-level expressions: e.g., D=SQRT(B**2-4.*A*C)	No cognizance needed of particular processor or I/O device characteristics. I/O done at GET/PUT level.	Single file subsetting with no data structure changes, no edits, no intermediate files. Moderately complex COTS-DB queries, updates.	Use of simple graphic user interface (GUI) builders.
Nominal	Mostly simple nesting. Some inter-module control. Decision tables. Simple callbacks or message passing, including middleware-supported distributed processing	Use of standard math and statistical routines. Basic matrix/vector operations.	I/O processing includes device selection, status checking and error processing.	Multi-file input and single file output. Simple structural changes, simple edits. Complex COTS-DB queries, updates.	Simple use of widget set.
High	Highly nested structured programming operators with many compound predicates. Queue and stack control. Homogenous, distributed processing. Single processor soft real-time control.	Basic numerical analysis: multivariate interpolation, ordinary differential equations. Basic truncation, round-off concerns.	Operations at physical I/O level (physical storage address translations; seeks, reads, etc.). Optimized I/O overlap.	Simple triggers activated by data stream contents. Complex data restructuring.	Widget set development and extension. Simple voice I/O, multimedia.
Very High	Reentrant and recursive coding. Fixed-priority interrupt handling. Task synchronization, complex callbacks, heterogeneous distributed processing. Single processor hard real-time processor.	Difficult but structured numerical analysis: near singular matrix equations, partial differential equations. Simple parallelization.	Routines for interrupt diagnosis, servicing, masking. Communication line handling. Performance-intensive embedded systems.	Distributed database coordination. Complex triggers. Search optimization.	Moderately complex 2D/3D, dynamic graphics, multimedia.
Extra High	Multiple resource scheduling with dynamically changing priorities. Microcode-level control. Distributed hard real-time control.	Difficult and unstructured numerical analysis: highly accurate analysis of noisy, stochastic data. Complex parallelization.	Device timing-dependent coding, micro-programmed operations. Performance-critical embedded systems.	Highly-coupled, dynamic relational and object structures. Natural language data management.	Complex multimedia, virtual reality.

Complexity is divided into five areas: control operations, computational operations, device-dependent operations, data management operations, and user interface management operations. Select the area or combination of areas that characterize the product or a sub-system of the product. The complexity rating is the subjective, weighted average of these areas.

Very Low	Low	Nominal	High	VERY High	Extra High	Don't Know

Platform Factors

The platform refers to the target - machine complex of hardware and infrastructure software.

	Very Low	Low	Nominal	High	Very High	Extra High
TIME			= 50% use of available execution time	70%	85%	95%
STOR			= 50% use of available storage	70%	85%	95%
PVOL		Major change every 12 mo.; minor change every 1 mo.	Major: 6 mo.; minor: 2 wk.	Major: 2 mo.; Minor: 1 wk.	Major: 2 wk.; Minor: 2 days.	

Enter the rating level for the five cost drivers.

Execution Time Constraint (TIME). This is a measure of the execution time constraint imposed upon a software system.

Nominal	High	Very High	Extra High	Don't know

Main Storage Constraint (STOR). This rating represents the degree of main storage constraint imposed on a software system or subsystem.

Nominal	High	Very High	Extra High	Don't know

Platform Volatility (PVOL). In this context, "Platform" refers to the complex of hardware and software (OS, DBMS, etc.) the software product calls on to perform its tasks.

Nominal	High	Very High	Extra High	Don't know

Personnel Factors

	Very Low	Low	Nominal	High	Very High
ACAP	? 15th percentile	35th percentile	55th percentile	75th percentile	? 90th percentile
PCAP	? 15th percentile	35th percentile	55th percentile	75th percentile	? 90th percentile
PCON	? 48% / year	24% / year	12% / year	6% / year	? 3% / year
APEX	? 2 months	6 months	1 year	3 years	? 6 years
PEXP	? 2 months	6 months	1 year	3 years	? 6 years
LTEX	? 2 months	6 months	1 year	3 years	? 6 years

Enter the rating level for the five cost drivers.

Analyst Capability (ACAP). Analysts are personnel that work on requirements, high-level and detailed design.

Very Low	Low	Nominal	High	Very High	Extra High	Don't know

Programmer Capability (PCAP). Evaluation should be based on the capability of the programmers as a team rather than as individuals. Major factors that should be considered in the rating are ability, efficiency and thoroughness, and the ability to communicate and cooperate.

Very Low	Low	Nominal	High	Very High	Extra High	Don't know

Application Experience (APEX). This rating is dependent on the level of applications experience of the project team developing the software system or subsystem. The ratings are defined in terms of the project team's equivalent level of experience with this type of application.

Very Low	Low	Nominal	High	Very High	Extra High	Don't know

Platform Experience (PEXP). The Post-Architecture model broadens the productivity influence of PEXP, recognizing the importance of understanding the use of more powerful platforms, including more graphic user interface, database, networking, and distributed middleware capabilities.

Very Low	Low	Nominal	High	Very High	Extra High	Don't know

Language and Tool Experience (LTEX). This is a measure of the level of programming language and software tool experience of the project team developing the software system or subsystem.

Very Low	Low	Nominal	High	Very High	Extra High	Don't know

Personnel Continuity (PCON). The rating scale for PCON is based on the project's annual personnel turnover.

Very Low	Low	Nominal	High	Very High	Extra High	Don't know

Project Factors

This table gives a summary of the criteria used to select a rating level for project cost drivers.

	Very Low	Low	Nominal	High	Very High	Extra High
TOOL	edit, code, debug	simple, fron tend, backend CASE, little integration	basic life - cycle tools, moderately integrated	strong, mature lifecycle tools, moderately integrated	Strong, mature, proactive lifecycle tools, with well integrated with processes, methods, reuse	
SITE: Coll ocation	International	Multi -city and Multi - company	Multi -city or Multi - company	Same city or metro area	Same building or complex	Fully collocated
SITE: C ommunications	Some phone, mail	Individual phone, FAX	Narrowband email	Wideband electronic communications	Wideband electronic communications, occasional video conferencing.	Interactive multimedia
SCED	75% of nominal	85% of nominal	100% of nominal	130% of nominal	160% of nominal	

Use of Software Tools (TOOL). See the Model Definition Manual

Very Low	Low	Nominal	High	Very High	Extra High	DonK know

Multi - site development (SITE). Given the increasing frequency of multi -site developments and indications that multi -site development effects are significant, the SITE cost driver has been added in COCOMO II. De termining its cost driver rating involves the assessment and averaging of two factors: site collocation (from fully collocated to international distribution) and communication support (from surface mail and some phone access to full interactive multimedia) .

Very Low	Low	Nominal	High	Very High	Extra High	DonK know

Required Development Schedule (SCED) . This rating measures the schedule constraint imposed on the project team developing the software. The ratings are defined in terms of the percenta ge of schedule stretch -out or acceleration with respect to nominal schedule for a project requiring a given amount of effort.

Very Low	Low	Nominal	High	Very High	Extra High	DonK know

APPENDIX B Rating Scale for the Productivity Drivers

Rating / Variable	XL	VL	L	N	H	VH	XH
OOM				1.00	0.90	0.78	
PREC		6.2	4.96	3.72	2.48	1.24	0.00
FLEX		5.07	4.05	3.04	2.03	1.01	0.00
RESL		7.07	5.65	4.24	2.83	1.41	0.00
TEAM		5.48	4.38	3.29	2.19	1.10	0.00
PMAT		7.80	6.24	4.68	3.12	1.56	0.00
RELY		0.82	0.92	1.00	1.10	1.26	
DATA			0.90	1.00	1.14	1.28	
CPLX		0.73	0.87	1.00	1.17	1.34	1.74
RUSE			0.95	1.00	1.07	1.15	1.24
DOCU		0.81	0.91	1.00	1.11	1.23	
TIME				1.00	1.11	1.29	1.63
STOR				1.00	1.05	1.17	1.46
PVOL			0.87	1.00	1.15	1.30	
ACAP		1.42	1.19	1.00	0.85	0.71	
PCAP		1.34	1.15	1.00	0.88	0.76	
PCON		1.29	1.12	1.00	0.90	0.81	
APEX		1.22	1.10	1.00	0.88	0.81	
PLEX		1.19	1.09	1.00	0.91	0.85	
LTEX		1.20	1.09	1.00	0.91	0.84	
TOOL		1.117	1.09	1.00	0.90	0.78	
SITE		1.22	1.09	1.00	0.93	0.86	0.8
SCED		1.43	1.14	1.00	1.00	1.00	

XL = extra low; VL = very low; L= low; N = nominal; H = high; VH = very high; XH = extra high;
OOM = object-oriented method; PREC = precedenteness; FLEX = ; RESL = ; TEAM = ; PMAT = ;
RELY = reliability; DATA = database size; CPLX = complexity; RUSE = developed for reusability;
DOCU = documentation; TIME = execution time constraint; STOR = main storage constraint;
PVOL = platform volatility; ACAP = ; PCAP = personnel capability; PCON = personnel continuity;
APEX = applicaton experience; PLEX = platform experience; LTEX = language and tool experience;
TOOL = use of software tools; SITE = multisite development; SCED = required development schedule.

APPENDIX C Raw Project Data

Context Information		Operating Environment	Application	Language	Object Oriented Method		
Organization	Project				OOA	OOD	OOP
	1	Manned Ground	Mission Planning	Ada	N	N	N
A	2	Manned Ground	Mission Planning	Java	Y	Y	Y
	3	Manned Ground	Command & Control	Pascal	N	N	N
B	4	Manned Ground	Command & Control	XML	Y	Y	Y
	5	Manned Ground	Mission Planning	Ada	N	N	N
C	6	Manned Ground	Mission Planning	C++	Y	Y	Y
	7	Avionics	Radar	Ada	N	N	N
D	8	Avionics	Radar	C++	Y	Y	Y
	9	Avionics	Mission Management	Ada	N	N	N
E	10	Avionics	Mission Management	C++	Y	Y	Y
	11	Avionics	Radar	C	N	N	N
F	12	Avionics	Radar	C++	Y	Y	Y
	13	Shipboard	Communications	Ada	N	N	N
G	14	Shipboard	Communications	C++	Y	Y	Y
	15	Shipboard	Communications	C	N	N	N
H	16	Shipboard	Communications	C++	Y	Y	Y
	17	Avionics	Mission Management	Ada	N	N	N
I	18	Avionics	Mission Management	C++	N	Y	Y
	19	Manned Ground	Command & Control	C	N	N	N
J	20	Manned Ground	Command & Control	C++	Y	Y	Y

OOA = object-oriented analysis ; OOD = object-oriented design ; OOP = object-oriented programming ; Y = yes ; N = no.

Context Information		Productivity			Product Size	
Organization	Project	ESLOC/SM	Backfiring	UFP/SM	Equivalent Size Range(EKSLOC)	Total Size Range (KSLOC)
A	1	254	71	3.58	58-74	48-83
	2	630	55	11.46	54-69	45-78
B	3	382	109	3.50	137-175	115-199
	4	439	55	8.28	158-203	132-229
C	5	428	71	6.03	.5-1	221-382
	6	768	55	13.96	8-11	196-339
D	7	52	71	0.73	155-198	104-179
	8	70	55	1.28	146-186	113-195
E	9	52	71	0.74	34-44	434-753
	10	82	55	1.49	30-39	550-953
F	11	142	128	1.11	15-19	68-117
	12	153	55	2.79	43-55	55-95
G	13	119	71	1.68	68-87	56-98
	14	306	55	5.56	61-78	51-88
H	15	66	128	0.52	20-26	17-30
	16	299	55	5.43	20-25	17-31
I	17	128	71	1.81	16-21	16-27
	18	160	55	2.91	18-23	15-26
J	19	236	128	1.85	32-40	1207-2092
	20	299	55	5.44	1170-1495	1049-1817

ESLOC = equivalent source lines of code ; UFP = unadjusted function points ; EKSLOC = thousand ESLOC ; SM = staff-month ; KSLOC = thousand source lines of code.