The Cognitive Complexities Confronting Developers Using Object Technology

by
Pauline H. Mosley

Submitted in partial fulfillment
of the requirements for the degree of
Doctor of Professional Studies in Computing

at

School of Computer Science & Information Systems

Pace University

April 29, 2002

Dissertation Signature (Approval) Page

We hereby certify that this dissertation, submitted by Pauline H. Mosley, satisfies the dissertation requirements for the degree of Doctor of Professional Studies in Computing and has been approved.

_____-_____

Dr. Allen Stix — Date
Chairperson of Dissertation Committee

_____-_____

Dr. Susan Merritt — Date
Dissertation Committee Member

_____-_____

Dr. Mary Courtney — Date
Dissertation Committee Member

School of Computer Science and Information Systems
Pace University 2002

Abstract

The Cognitive Complexities Confronting Developers Using Object Technology

by
Pauline H. Mosley

Submitted in partial fulfillment
of the requirements for the degree of
Doctor of Professional Studies in Computing

April 29, 2002

Many managers are adopting object technology initiatives to develop high-quality products more efficiently. Consequently, software practices are changing and there are repercussions across corporate and academic training. As new practices promise to yield increasing benefits in the software development cycle, the complexity is becoming proportionately greater.

Technology has come so far, so fast, and with such success, that it is easy to forget how difficult computing actually is. Things appear easy, when looked at superficially, because products are so well designed, and cognitive complexities so well hidden, that preschoolers can be computer literate and the public at large can consider themselves computer savvy. Moreover, the toughness of new problems, because of their misleading simplicity, often fools the scientists themselves. When Smalltalk-80 appeared, object technology was touted as intuitively obvious; but experiences over the past 20 years have proven otherwise. Object technology is a cognitive minefield

This research examines the notion that object technologists, like the workers in artificial intelligence, have underestimated the complexities associated with the analysis, design, and coding of software from "virtual things" (i.e. objects). One manifestation is that the information systems community making up the software infrastructure in many organizations is resisting, misapplying, or only very slowly understanding objects. Another manifestation is that the academic community is perplexed by the way that objects are changing the view of software and how, correspondingly, the computer science curriculum must be adapted. Both of these manifestations are documented herein.

Going beyond merely documenting the cognitive complexities of object technology, this research identifies those atomic-level constituents responsible for the shift from simple to complex with respect to analysis and design. This study concludes by proposing a pedagogical model that addresses these constituents and could thereby reduce the learning curve for retraining practitioners and restore clarity to the computer science curriculum. We hope this research contributes to the wider acceptance of object technology.

Acknowledgments

First, I'd like to thank my advisor at Pace University, Dr. Allen Stix, for his assistance, his enthusiasm, and the attention he gave to many of the details of this research. Dean Susan M. Merritt and Dr. Mary F. Courtney deserve special thanks for much important guidance. I am also indebted to Dr. Fred Grossman, Dr. Joseph A. Bergin, and the DPS students and staff for helping me along the way. Kyle Smith, of Smith Editing, made the dissertation conform to style guidelines and copyedited the manuscript. If the text seems to flow nicely, we all owe her our gratitude.

I am grateful for the valuable feedback that I received from my closest YMOL friends and family. My dear sister, Lois Nisbett, provided continuing encouragement, support, and, tangibly, corrections in the early chapters. My parents, my in-laws, my aunts and my uncles were by my side, buoying me up, when things were rough. My deceased grandmother who instilled in me the desire to pursue excellence in all that I do.

My son, Marcus Mosley, deserves a big hug for letting mommy use the computer and for being so understanding. The most loving and greatest thanks of all go to my husband, Paul W. Mosley. Without his unsurpassed generosity I could not have earned a doctorate. He made it come true by solving the tough nitty-gritty problems and, despite the sacrifices it imposed on him, by making me feel good about this whole phase of my life. Most of all, I am thankful to God for giving me the strength to pursue this degree.

Table of Contents

Abstract……iii

List of Tables iix

List of Figures x

1. Introduction 1

1.1 Statement of the Problem 1

1.1.1 Benefits of Object-Oriented Systems 3

1.1.2 Difficulties of Achieving the Benefits 7

1.1.3 Resistance of Difficulty with Use of an Emerging Technology 8

1.1.4 Complexity of Implementing an Emerging Technology 9

1.1.5 Object-Oriented Decomposition of a Large Hierarchy of Classes Is Complex and Difficult to Manipulate 13

1.2 Implications for the Current Study 15

1.2.1 The Paradigm Shift 17

1.2.2 New Approach in System's Development 19

1.2.3 Training Future Software Developers 21

1.3 Research Questions To Be Investigated 24

1.4 External Validity 27

1.5 Addressing Threats to Validity 29

1.5.1 Improve Interviewing Session 29

1.5.2 Increase Sample Population 29

1.5.3 Improve Questionnaire Procedure 30

1.5.4 Develop Different Questionnaires with Same Functionality 30

1.6 Definition of Terms 31

2. Relevance of the Research in the Context of Other Work

2.1 Emergence of Object Technology 35

2.1.1 Object Technology Represents a Radical Change 35

2.1.2 Object Technology Impacts Software Development and Organization 37

2.2 Reusability and Corporate Infrastructures 38

2.3 Object Technology Migration is Hard 40

2.3.1 Object Technology and Practitioners 40

2.3.2 Academia and Object-Oriented Anomalies and Complexities on the Learning Curve 45

2.4 The Contribution This Study Will Make to the Field 48

3. Research Methodology

3.1 Overview of the Research Process 50

3.2 Research Methodology I – Industrial Content Analysis 50

3.3 Research Methodology II – Textbook Analysis 64

3.4 Research Methodology III – Pedagogical Analysis 79

4. Proposed Model for Rationalizing the Approach to Object Technology

4.1 Typology: Orthogonal Sources of Complexity 88

4.2 Proposed OT Cognitive Model 91

4.3 Skill Sets 93

4.4 Summary and Future Directions 96

5. Epilogue

5.1 Challenged by Objects 98

5.1.1 Challenge One 98

5.1.2 Challenge Two 102

5.2 Finding a Focus for Research 102

5.3 A Qualitative Study 107

5.4 My Professional Calling for the Years Ahead 108

References 110

Appendixes

A. Supporting Forms

A.1 Request for Research Participants 118

A.2 Thank you Form Letter for Research Participants 119

A.3 Participant Tracking Log 120

A.4 Preliminary Telephone Interview Questions 121

A.5 Secondary Telephone Interview Questions 122

A.6 Teaching Methodology Survey 123

A.7 Preliminary Face-to-Face Interview Worksheet 124

A.8 Pretest: Student Survey #0 125

A.9 Posttest: Student Survey #1 144

B. Correspondence From Individuals Involved in the Study 178

C. Academia Data

C.1 Teaching Methodology Surveys 194

C.2 Course Outlines 198

C.3 Sample Assignments 205

D. Industrial Data

D.1 Questionnaires 244

D.2 Telephone Interview Schedules 260

E. Additional Tables to Supplement Material in the Text 275

F. Additional Supplements 280

List of Tables

Table 1 – Comparison of SSP and OOP Features 3

Table 2 – Strength of Belief in OOSD Advantages 6

Table 3 – Strength of Belief in OOSD Disadvantages 11

Table 4 – Software Development Change in Methodology and Approach 20

Table 5 – Respondent Industry Profile 56

Table 6 – Respondent Position Profile 56

Table 7 – Percentage of Time the Themes Were Identified in the Responses 59

Table 8 – List of Textbooks 66

Table 9 – The Number of Examples per Programming Construct 69

Table 10 – Design Constructs 74

Table 11 – UML Constructs 76

Table 12 – Software Engineering Constructs 77

Table 13 – Object-Oriented Programming Constructs 78

Table 14 – Participating Universities and Colleges 81

List of Figures

Figure 1 – Elements of a Big-M Methodology 43

Figure 2 – How Problem Size and Methodology Affect Staff Numbers 43

Figure 3 – Java Applets and Programming 70

Figure 4 – The Objectivist Approach 71

Figure 5 – The Procedural Approach 72

Figure 6 – Total Number of Pages Allotted to Programming Constructs 75

Figure 7 – Total Number of Pages Allotted to Design Concepts 75

Figure 8 – The OT Cognitive Model 92

Chapter 1

Introduction

1.1 Statement of the Problem

To survive, software organizations must have the ability to adapt to a dynamically changing technological culture [41]. Literature suggests that survival of the fittest is contingent upon how quickly an organization can respond to technological options and market needs, while at the same time delivering high-quality products and services.

One way organizations' have chosen to deal with these changes is to implement object technology. This methodology facilitates the use of reusable software component libraries thereby producing better systems structures that are adaptable and extensible. Although, the benefits of object-oriented systems are recognized, the cognitive complexities associated with these pure object solutions present an demonstrably steep learning curve, which can lead to longer development cycles and more costly investments. Experts say that this long learning curve prevents companies from using object-oriented programming properly or taking advantage of it, and that its benefits can't really be taught, at least not understood from a book [82,86,73]. In addition, studies have shown that object technologists, similar to the workers in artificial intelligence, have underestimated the complexities associated with "virtual things" (i.e. objects).

One manifestation is that the information systems community making up the software infrastructure in many organizations is resisting, misapplying, or only very slowly coming to proper terms with objects. Another manifestation is that the academic community is perplexed by the way that objects are changing the view of software and how, correspondingly, the computer science curriculum must be adapted.

There is little empirical proof that the complexities associated with object technology are due to the inherent nature of its paradigm or if it is due to improper learning acquisition, which then leads to misapplication of the technology. Mr. Fred Brooks [15], author of *No Silver Bullet*, states that the essence of a software system is the conceptual, almost inspirational, core of logic and design that forms the underpinnings of the system. He goes on to say that because the role of accidental factors is limited and since technology can help developers only with the accidental factors, no development in technology can affect an order of magnitude increase in the productivity of software developers. Therefore, for organizations to capitalize on the benefits of this technology and impact the quality of software, there is a need to improve the usability of objects and object technology to make the concepts amenable to information technologists , but especially programmers. If object-oriented programming is indeed a new paradigm, then all of the old techniques appropriate for other paradigms need to be re-examined and possibly replaced by new techniques. This replacement includes pedagogical techniques no less than it includes programming techniques [12].

The goal of this research is to identify atomic-level properties that cause the knowledge acquisition in object technology to shift from simple to complex with respect to logic and design. In addition, this study seeks to establish a foundation for a

pedagogical model that will reduce the learning curve for educating high quality designers in industry and academia (the retraining of commercial developers and the indoctrination of those new to software development), thereby promoting this technology to gain wider acceptance and proper implementation usage.

1.1.1 Benefits of Object-Oriented Systems

The paradigm shift from algorithmic decomposition (structured procedural programming) to object-oriented programming has impacted not only the software development community, but also it is causing industrial practitioners to reevaluate their software engineering practices because of object technology's numerous benefits. Table 1 contrasts the procedural paradigm to the object-oriented paradigm. The procedural paradigm centers on modules, whereas the object technology is centered on the object.

Table 1

Comparing the module-centric procedural and the object-centric object-oriented programming paradigms.

	Structured-Procedural Programming	**Object-Oriented Programming**
1.	Systems are modularized on the basis of functions.	Systems are modularized on the basis of data structures (objects).
2	In a program module data and procedures are separate.	In a program module objects' state (data types) and behavior (operations) are encapsulated.
3.	To pass parameters, programmers call the active procedures.	To activate their operations, active objects communicate with each other by passing messages.
4.	The programmer must ensure that the procedure will work correctly on the data type it is applied to.	Encapsulation prevents unauthorized access; thus, the message passing ensures that an object's internal state can be accessed only if permitted.

5.	The real world is represented by logical entities and control flow.	The real world is represented by objects mimicking external entities.
6.	Program modules are linked through the parameter-passing mechanism and the operating system.	Because a program is a collection of interacting objects, program modules are integrated parts of the overall program.
7.	Uses procedural abstraction.	Uses class and object abstraction.
8.	Passive and dumb data structures are used by active methods (operations).	Active and intelligent data structures (objects) encapsulate all passive procedures.
9.	The unit of structure is a statement of expression.	The unit of structure is an object treated as a software component.
10.	Uses functional decomposition.	Uses object-oriented decomposition.
11.	Uses procedure-oriented languages such as C or Pascal.	Uses object-oriented languages such as C++, Object Pascal, Smalltalk-80, Java, or Visual Basic.

Note. From Object-Oriented Programming for Structured Procedural Programmers, Khan, IEEE 1995.

Object-orientation is based on four important concepts: objects, abstract data typing (encapsulation and information hiding), inheritance (of attributes and behavior), and polymorphism. Together these concepts provide a framework for producing better structured and more reliable software for complex systems, greater reusability, more extensibility, and easier maintainability.

The advantage of having a methodology centered on the object is that it shifts the responsibility to a more local level. This gives software developers the ability to define elements with specific functionalities thereby promoting object reuse. Establishing various libraries of building parts helps programmers significantly by cutting the amount of code they need to write for new applications. This reusability means shorter, less-costly development cycles. Lastly, according to Fowler, objects can be perceived at the

conceptual level, specification level, or at the implementation level. This flexibility enables software developers to perceive objects at different levels during various phases of the software development life cycle.

Another benefit of this approach is design patterns. Design patterns help with reuse and communication as well as giving a higher perspective on analysis and design. When design patterns are taught properly, they can be used to increase the understanding of basic object-oriented design principles significantly [80]. Because design patterns are time-tested solutions, they make software more modifiable. Another advantage of using design patterns is that they enable designs to be created for complex problems that do not require large inheritance hierarchies.

Although, these benefits remain to be proven, several well-known software development practitioners [43,47,71, 87] are convinced that object-oriented system development delivers distinct advantages.

Richard A. Johnson conducted an Internet survey of 150 seasoned developers from across the U.S. to determine if software developers believe that these benefits were achievable by object technology. Participants were classified as either object-oriented or non-object-oriented developers. Participants rated their belief on a scale of 1-6 with 1 denoting unbelief and 6 denoting a strong belief. The results from his study are indicated in Table 2. According to this study, the number one advantage is that object technology provides an easier modeling process.

Table 2

Strength of Belief in Object-Oriented System Design Advantages

Priority Rank	Compared to conventional systems development, object-oriented system design is characterized by _____.	Median Belief Strength of Developer	
		Object Oriented	Non-Object-Oriented
1	An easier modeling process	6	5
2	Improved modularity of systems	6	5
3	Improved maintainability of systems	6	5
4	Improved quality of systems	6	5
5	More understandable analysis and design models	6	4.5
6	Greater stability of design over time	6	5
7	More flexible/adaptable development	6	5
8	An easier transition between phases	6	5
9	More effective code reuse	6	5
10	Improved communication with developers	5	5
11	Improved productivity of your work	5	4.5
12	More effective analysis and design model reuse	5	5
13	Greater user satisfaction with systems	5	4
14	Improved communication with users	4	4

Note. From The Ups and Downs of Object-Oriented Systems, Johnson, ACM 2001.

Literature suggests that object-oriented programming is considered to be better at modeling the real world than is procedural programming. Object-oriented programming

allows for more complicated and flexible interactions. The shaded portion in Table 2 indicates the top eight reasons object-oriented developers find this approach advantageous. Non object-oriented developers noted that analysis and design models were not more understandable than traditional analysis and design models, hinting that a learning curve may exist. Overall, both groups of developers agree that this approach is definitely advantageous for software development.

1.1.2 Difficulties of Achieving the Benefits

Kenneth Kendall [51] states that the five phases of technological advancement are:

- Technological invention or discovery
- Technological emergence
- Technological acceptance
- Technological sublime
- Technological surplus

He further states that there are chief barriers encountered in the transition from the technological acceptance to the technological sublime phase. They are:

- Resistance of difficulty with use of an emerging technology
- Complexity of implementing an emerging technology

Object technology has reached this point in its technological advancement and is facing these challenges.

1.1.3 Resistance with Using an Emerging Technology

Why might people resist accepting object technology? The most obvious reason is that people are fearful of change. There are other reasons such as apparent lack of knowledge or overly positive expectations on the part of management about object-oriented design and its role in the relevant projects [92].

Unfortunately, there is often a lag between the discovery of a new technology and the practical application of that technology. People want an ideal situation and resist change for the following reasons [92]:

- They're afraid they won't be able to understand object technology.
- They don't want to understand object technology
- They're worried that the plan for adopting object technology has significant problems or flaws.
- They want to adopt object technology, but they want to do it their way.
- They know they won't be able to understand object technology because they've been sloughing for the last ten years and know that this will expose them for what they are.

If management cannot see the return on investment of the complexity's cost, then they are reluctant to embrace this methodology. Many object-oriented books and even some fans of object technology suggest that this technology only shines under fairly ideal conditions [44]. It is a challenge to convince management that object technology has a different area of optimization than other paradigms.

Another reason for resisting the change to embrace object technology may be that little is agreed upon about whether object technology's benefits are mostly limited to large projects, or apply to projects of all sizes [44].

Lastly, resistance with using this emerging technology could be due to poor supportive corporate infrastructures. If organizations do not offer a supportive infrastructure, then practitioners will be reluctant to use new software engineering practices. Adapting improvement ideas, such as object technology, is challenging because commitment and responsiveness to improvement fluctuates depending on the organization's preoccupation with other challenges.

1.1.4 Complexity of Implementing an Emerging Technology

Rentsch predicted, "My guess is that object-oriented programming will be in the 1980s what structured programming was in the 1970s. Everyone will be in favor of it. Every manufacturer will promote his products as supporting it. Every manager will pay lip service to it. Every programmer will practice it (differently). And no one will know just what it is" [74]. Rentsch's predictions still apply to the 2000s. This confusion may reside in the intrinsic complexity of the constructs, the proliferation of abstractions that are seen as objects are applied, or in the barrage of new concepts surrounding object technology (e.g.. Unified Modeling Language; class, responsibility, and collaboration cards; design patterns for best practices; coding patterns; etc.).

One of the biggest challenges practitioners face is failure to apply object-oriented techniques to problem spaces and resorting to classical analysis and design methods. Implementing object-oriented techniques using traditional practices (functional

decomposition) oftentimes leads to the loss of benefits from object-oriented design. This traditional approach also often leads to disillusionment with object-oriented design, which consequently results in a project slip and all that that entails. Poor understanding or inadequate training of the concepts that arise in connection with objects result in misshapen, illogical class hierarchies, and poor mapping between object classes and problem domains and class design [92]. In functional decomposition (process-oriented software design) a direct flow of execution is established. In object-oriented design the flow of execution is far more tedious and difficult than in functional decomposition; object-oriented developers often avoid it [92]. To complicate matters further, there are the issues of inheritance execution and the event-driven environment in which most object-oriented design is implemented.

In an Internet survey of 150 randomly selected systems developers from across the U.S. both object-oriented and non-object-oriented developers indicated that there are complexity issues with object technology (see Table 3) [45]. The 150 systems developers are comprised of U.S. subscribers to Communications (who belong to software engineering, programming interest groups); U.S. subscribers to OOPS Messenger; and registrants at recent OOPSLA conferences. The survey asked developers to rate statements about the possible advantages and disadvantages of object-oriented software design on a seven-point Likert-type scale ranging from 1=Extremely Strong Disagreement to 7=Extremely Strong Agreement. The results of this study from the front-line of software development are a strong indication that complexity does exist. The shaded items in Table 3 indicate where both groups held similar beliefs about object-oriented software design disadvantages.

Table 3
Strength of Belief in Object-Oriented System Design Disadvantages

Priority Rank	Compared to conventional systems development, object-oriented system design is characterized by _____.	Median Belief Strength of Developer	
		Object Oriented	Non-Object-Oriented
1	Decreased system run-time performance	4	5
2	Unavailability of adequate object-oriented database management system	4	5
3	Increased initial development time	4	5
4	Unavailability of object-oriented cast tools	4	4
5	Confusion from too many object-oriented analysis and design methods	4	4.5
6	Inability to try object-oriented software design before committing	4	4
7	Complexity of object-oriented analysis and design methods	4	4
8	Complexity of object-oriented programming languages	4	4
9	Incompatibility of object-oriented software design processes	3	3
10	Inability to demonstrate object-oriented software design benefits	3	4
11	Difficulty learning object-oriented analysis and design methods	3	4
12	A more difficult programming process	3	3
13	Difficulty learning object-oriented programming	3	4
14	Unsuitability of object-oriented software design for projects	2	3

Note. From The Ups and Downs of Object-Oriented Systems, Johnson, ACM 2001.

There are many plausible explanations for these findings:

- Object-oriented developers may not be in an environment that is supportive of object-oriented software design (such as the availability of better training programs), which makes learning easier.

- Programmers who are experienced in using structured tools and techniques find it difficult to model the real world into a set of interacting objects and class hierarchies.

- Inadequate training coupled with lack of corporate support are factors that could contribute to a failed software project.

From this study, Johnson concludes that there are two main lessons for software development management:

- Select the best and most motivated developers for object-technology training.

- Support the developers through quality training, mentoring and tools [45].

Modeling is another complex activity of this paradigm. It entails abstracting information and knowledge from a particular domain to achieve a model containing the essentials from the perspective of the modelers and their given goals [48]. Industrial practitioners may be confused about the latest modeling trend and whether a new approach is useful for them or how to combine it with previous approaches. Literature states that when object-oriented approaches became suddenly popular, the close relations to entity-relationship modeling (from the database community) and to frame systems and semantic networks (from artificial intelligence) were not clear to all computer professionals, even if they had applied one of the other of the approaches previously [48].

Evidence for this claim is demonstrated in disparate views from various experts in the field. Mylopoulos, Chung, and Yu [21] state that the nonfunctional requirements are generally insufficiently studied, and the current methods of object-oriented analysis and design are particularly ignorant in this regard.

Butler, Esposito, and Hebron [48] focus on how modeling in human-computer interaction and objects can be linked together. These authors suggest it is necessary to use human-computer interaction approaches to make a software application more usage-centered. Kaindl and Carroll state that the very idea of object-oriented modeling is much broader than what is covered by the unified modeling language.

Empirical evidence has been provided by van Hillergersberg [88] who investigated the performance and strategies of programmers new to object-oriented techniques. He concluded that object-oriented concepts were not easy to learn and use quickly. Further still, reports from teachers of programming and results from some empirical studies [26] now suggest that the teaching of programming has created significant difficulties for high-school and university students, and has failed to catalyze the development of higher-order thinking skills.

1.1.5 Object-Oriented Decomposition of a Large Hierarchy of Classes Is Complex and Difficult to Manipulate

Object-oriented decomposition provides a method to decompose a complex arrangement by the primary objects apparent in the system. Once the objects are defined and the system functionality is assigned, major components of the software system are developed independently.