Charting the Rocky Shoals of an Object-Oriented Mindshift

By

Deborah J. Armstrong

B.A., California State University, 1986
M.B.A., Avila College, 1996

Submitted to the School of Business and the Faculty of the Graduate School of the University of Kansas in partial fulfillment of the requirements for the degree of Doctor of Philosophy

Dissertation Co-Chairs:

[illegible]	[illegible]	[illegible]
Dr. V. K. Narayanan	Dr. Kay M. Nelson	Dr. Raj Srivastiva
Drexel University	University of Utah	University of Kansas

Committee Members:

[illegible]	[illegible]	[illegible]
Dr. J. James Nelson	Dr. Howard Sypher	Dr. Carson Woo
University of Utah	University of Kansas	University of British Columbia

Date Defended: June 4, 2001

MAY 30 2001

ABSTRACT

This research explored the mindshift currently taking place from procedural software development techniques to object-oriented (OO) techniques. The overarching goal of this research was to answer the questions, "Why is it difficult for procedural experts to learn object-oriented development? And Where in the learning process are developers experiencing difficulty?" Drawing on the schema and skill acquisition theories from the cognitive psychology literature this study uses the concept of proactive interference to address the research questions. Proactive interference occurs when knowledge cannot be integrated with an activated mental model or schema. When proactive interference occurs the previously learned information interferes with the learning of the new information. The result is a more difficult learning process than if there was no previous knowledge. The study was conducted in three phases. In Phase One revealed causal mapping was used to capture the procedural and object-oriented development domain expertise. In Phase Two, the two aggregated maps were quantified into an instrument. In Phase Three the validated instrument administered to a sample of developers with various levels of expertise in both mindsets. Results indicated that the learning curve for procedural software development experts learning object-oriented techniques included learning plateaus. Also, proactive interference was positively associated with those learning plateaus. And, lastly, the plateaus occurred at certain levels of object-oriented experience and clustered around certain object-oriented concepts. Our findings indicate that software developers experience difficulty at several points in the learning process. Proactive interference is the strongest during the processes of understanding object-oriented development within a larger system. An understanding of the learning processes involved in transition from procedural to object-oriented techniques could shorten the learning process, increase software quality, perhaps decrease the frustration level of students during their learning process, and ultimately increase the use of object-oriented techniques. From a theoretical perspective, questions of knowledge transfer and proactive interference are important to our understanding of learning and should continue to be explored.

DEDICATION

There are so many people to thank and so few lines to do it in when you have to double space! Please understand that the acknowledgements may be parsimonious, but the feelings behind them are heartfelt. I would like to thank Kay Nelson, who not only is a fantastic advisor, but also my friend. Thank you for rescuing me from OBlivion and showing me the way. To Jim Nelson, thank you for your genius and calming effect on so many occasions. You Nelsons are some of the most generous (and talented) people I know and I thank God that you were willing to take me on. To V. K. Narayanan, thank you for your knowledge (both causal and evoked), patience, flexibility and most of all your sense of humor during this process. To Raj Srivasitiva, thank you for being just what I needed right when I really needed it. To Carson Woo, thank you for putting up with all the chaos. To Howard Sypher, thank you for giving your time to a not-so-perfect stranger. To Medhi Ghods, thank you for lending me your expertise, enthusiasm and support. To Chris Klomp, thank you for everything I learned from you while interacting with your organization and for your tremendous support during data collection. To Steve Hillmer, thank you for your statistical expertise, and always making time for a chat. To Jim Heintz and Greg Freix, thank you for believing in me and supporting my research. To Bill Hardgrave, Woo-Pig-Suey! (need I say more?). To Mari Buche and Benedict Kemmerer, thank you for your assistance both professional and personal. To Kristy Vetter, the best friend anyone could ask for, thank you (for too many reasons to list). To my friend and colleague Bonnie

O'Neill, my long distance lifeline, thank you for being there whenever I needed a sympathetic ear. To my family, thank you for the love, encouragement and help in the trenches during this process. Lastly, I would like to dedicate this dissertation to my hero, my husband, Ken Armstrong. As Shania says, "My dreams came true because of you."

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

LIST OF APPENDICES

CHAPTER ONE

Introduction

You must unlearn what you have learned.

--Yoda, The Empire Strikes Back, 1980

We have all come to accept that the world is changing at an ever-increasing rate. In many instances these changes are not incremental but quantum shifts in methods, technologies and mindsets. A mindset is a distinctive viewpoint that determines how an individual engages events or views reality (Culbert, 1996); a mental attitude or disposition that predetermines a person's responses to and interpretations of situations (Leonard, 1995). A change in mindset is a change in the thoughts, perceptions and values that form a particular view of reality. When fundamental changes are made to essential or commonly held concepts a revolution occurs. These changes constitute a radical breakaway from the governing mindset (Mey, 1982). For this research a *mindshift* is defined as a revolutionary change in mindset or in the way things are thought about.

We see examples of these revolutionary changes or mindshifts all around us. The introduction of the Internet has caused a multitude of changes. In education, the traditional classroom is being replaced by distributed learning environments. In the business world the traditional model of business is being replaced by the "e" model. The field of Information Systems (IS) is especially sensitive to these revolutions or mindshifts. Examples within IS include the move from hierarchical to relational databases, the shift from mainframe computing to client/server, and the shift from procedural to object-oriented software development techniques.

This research explored the mindshift currently taking place from procedural software development techniques to object-oriented (OO) techniques (Pei & Cutone, 1995; Vessey & Conger, 1994). The term "development" is used to represent the entire systems development life cycle, including analysis, design, programming and maintenance. To address these roles, this study included individuals engaged in a variety of functions within the software development domain, such as: analysts, architects, designers, engineers, developers, programmers and project managers.

While OO techniques hold the promise of shorter development times and easier maintenance, there is a severe shortage of software developers available who can put these techniques into practice (Cassidy, 1997; Eaton & Gatian, 1996; Page-Jones, 1994). There are two likely solutions to this problem, but each of these solutions has its advantages and disadvantages. One solution is to hire object-oriented experts. When they can be found, experts in OO techniques look like the perfect solution. However, successful OO modeling requires business specific domain knowledge (Rosson & Gold, 1989). Understanding the business problem is critical for using object-oriented techniques as problem analysis revolves around modeling "real world" objects. Since external experts do not possess the necessary domain knowledge, hiring OO development experts is not the optimal solution. Conversely, organizations can retrain their procedural development experts in object-oriented methods. The internal developers possess the business specific domain knowledge necessary for successful OO modeling and development.

Software development education researchers have investigated various aspects of transitioning from one development language or mindset to another such as: skill obsolescence (e.g. Fossum, Arvey, Paradise, and Robins, 1986; Gist, Rosen, and Schwoerer, 1988), the benefits of the object-oriented approach (e.g. Guttman & Matthews, 1992), trainee motivation (e.g. Baldwin et al., 1991; Ryan, 1999), and expert versus novice developers (e.g. Liu, Goetze, & Glynn, 1992). A few studies have looked at procedural experts learning object-oriented languages (e.g. Detienne, 1990; Manns & Nelson, 1996). A common assertion found in many of the studies is that it is difficult for an experienced software developer to make the transition to a new language and/or mindset. To date though, few have addressed *why* is it difficult to make the transition, and *where* individuals are experiencing difficulties in the learning process.

To address the difficulties inherent in retraining existing staff, this research sought to understand the difficulties procedural software development experts encounter as they shift to the object-oriented mindset. By identifying the cognitive processes involved in shifting mindsets, we are better able to learn where individuals are experiencing difficulties in the transition. This study addresses those issues and extends the work of Nelson, Irwin, & Monarchi (1997) by focusing on the process and the problems that individuals encounter as they move through the learning process. The overarching goal of this research program is to answer the question, “How can organizations ease the difficulties involved in revolutionary mindshifts?” To accomplish this goal, this study sought to answer the questions, "Why is it difficult for procedural experts to learn object-oriented development?” and “Where in the learning process are developers experiencing difficulty?”

The issues of skill acquisition, transfer and interference in the learning process that were investigated in this study add to our understanding of expert knowledge, software development and cognition. This has implications for both theory and practice. From a theoretical perspective the cognitive models of procedural and object-oriented expertise contribute to our understanding of expert knowledge and cognition. Increasing our understanding of the difficulties that developers are experiencing during a mindshift may help ease the learning process. The practical implication of such theory development and testing offers insight for more effective and more efficient retraining.

Document Overview

Chapter Two of this document discusses the theoretical foundations of IS learning. The learning literature from the fields of Psychology, Educational Psychology and Information Systems are brought together to develop the hypotheses for this dissertation.

Chapter Three discusses the methods used in studying the learning process and the research design for the dissertation.

Chapter Four of this document discusses the results of this research. The results of the analyses performed are reviewed.

Chapter Five begins by discussing the research findings in greater detail. The implications and limitations of the research are then presented. This document concludes with directions for future research suggested by this study.

CHAPTER TWO

A Model Of Learning Software Development

This chapter develops a model of learning to develop software when previous software development expertise is present. In developing this model we sought to:

- Explore and understand the learning process, and
- Articulate the theoretical underpinnings of the software development learning model.

"Computer programming is a complex cognitive task composed of a variety of subtasks and involving several kinds of specialized knowledge" (Pennington, 1987, p. 295). A cognitive process occurs when an individual is involved in any type of information processing, thinking, or learning (Billett, 1994). When an individual develops software he or she solves a problem and engages in a cognitive process. Since learning to develop software is cognitive learning, theories from the cognitive learning literature provide useful tools for identifying and explaining the processes involved in these mindshifts. By tracing the cognitive learning research from its origins we can provide a historical context for this study and demonstrate where cognitive skill learning fits within the learning domain.

The Psychology of Learning

Research in learning originated from the behaviorist tradition popularized by John Watson (1878-1958). Almost all learning theorists of the first half of the twentieth century accepted the behaviorist or associationistic framework. The associationist studies overt behavior in a systematic and objective way. They believe that mental life can be explained in terms of two basic components: ideas and associations (or links) between them (Mayer, 1983). The associationist view assumes that for any stimuli (S) there are associations or links to many possible responses (R1, R2...). The links are assumed to be in the problem solver's head, where they form a family of possible responses associated with a given situation. The responses may vary in strength, with some associations being very strong and others very weak. The responses for any situation may be put into a hierarchy in order of their strength.

In contrast to the associationist view, the cognitive perspective states that behavior is the manifestation of cognition (thinking) and therefore psychological definitions must be tied to the mechanisms that underlie behavior (Mayer, 1983). Cognitive processes can be defined as cognitive activities performed by an individual when engaged in any type of thinking, or learning (Billett, 1994). Hilgard and Bower (1970) summarize the principles emphasized within cognitive theory as:

1. The perceptual features according to which the problem is displayed to the learner are important conditions of learning.
2. The organization of knowledge is essential.
3. Learning with understanding is more permanent and transferable than rote learning.
4. Cognitive feedback confirms correct knowledge and corrects faulty learning.

5. Goal setting by the learner is important as motivation for learning.
6. Divergent thinking (which leads to inventive solutions) is to be nurtured along with convergent thinking.

The cognitive learning theories we know today evolved from these cognitive foundations. Information processing theory is a more recent learning theory based on the cognitive model. According to information processing theory humans are information processors comparable to a computer (Newell & Simon, 1958, 1972). Like a computer, information flows through stages of cognitive processing and storage to provide an output. Three assumptions of the theory are: (1) a control system is comprised of memories, containing information and connected by relationships; (2) information processes operate on the information that is stored in memory; and (3) rules are formulated that combine processes into complete programs (Sahakian, 1970).

Cognitive skills code and interpret incoming sensory information and that information is translated into a skilled response (Colley & Beech, 1989). Individuals use cognitive processes such as learning, thinking and problem solving when engaged in cognitive skills (Billett, 1994). Software developers take incoming information in the form of user requirements, engage in cognitive processes such as problem solving, and translate the information into a solution. When an individual learns to develop software he or she is learning a cognitive skill (Colley & Beech, 1989). Since learning to develop software is cognitive learning, theories from cognitive psychology are appropriate foundations for identifying and explaining the processes involved in these mindshifts.

The information-processing model is a theory from the field of cognitive psychology that provides a method for examining the processes underlying the learning of software development from a cognitive perspective. For this study we used the information processing theory and defined learning from a cognitive perspective as an inferred change in the individual's mental state (Tarpy & Mayer, 1978, p. 3). This change results from experience, and influences in a relatively permanent fashion the individual's potential for subsequent adaptive behavior (Tarpy & Mayer, 1978, p. 3).

Within the information-processing model there are several theories of learning or skill acquisition. Table 1 lists these theories, which can be categorized into three groups: productions systems, mental models and propositional based theories (Villeneuve & Fedorowicz, 1997). The production systems theories typically apply if/then rules to incoming data to determine actions to be taken. The mental model theory asserts that an individual creates a model of a situation before taking action. The propositional based theories posit that incoming knowledge is compared against stored knowledge. The result of this comparison is a new instance of the memory or a refinement of the old memory. Propositional-based theories are often used in research on human expertise. Schema theories are more appropriate than many of the other propositional theories for the study of expertise because they provide mechanisms for learning that others do not (Villeneuve &

Fedorowicz, 1997). The schema theories tend to provide problem-solving context-dependent explanations for human cognitive processes. The current research uses the schema skill acquisition theory originally proposed by Bartlett (1932) because of its emphasis on learning, expertise and the recognition of the importance of context.

Learning a new cognitive skill such as software development has three stages consisting of the accumulation of declarative knowledge, knowledge compilation and the development of procedural knowledge (Anderson, 1982). Declarative knowledge consists of facts, assertions, and concepts. It is accessible and can generally be described. The knowledge compilation stage consists of the transition from the declarative knowledge to the procedural knowledge (Anderson, 1982). Procedural knowledge consists of techniques, skills and the ability to secure goals and is not readily accessible or easily described (Conway & Wilson, 1988).

During the first stage, the learner memorizes general knowledge and rules about the skill and domain. Novices then use general-purpose problem solving techniques with this declarative knowledge to perform the new skill. As the learner practices the skill, the knowledge is transformed from declarative to procedural encoding (knowledge compilation). Initial errors and misconceptions are minimized and the skill is performed more smoothly and automatically. In this stage the domain knowledge is directly incorporated in procedures for performing the skill. In the final stage the learner has transformed the declarative knowledge into procedural knowledge. The learner continually and gradually improves his or her ability in the skill and relies less on memorized rules. In the procedural stage the skill can be more automatically and unconsciously performed.

The Role of Schemas in the Learning Process

During any of the three stages of learning the learner may attempt to map knowledge from familiar domains to the new, unfamiliar domain. The application of knowledge from one situation to another, or from past experience to new learning is known as the adaptation of knowledge schemas (Bartlett, 1932). A schema consists of a set of propositions that are organized by their semantic content (Bartlett, 1932). There are two basic principles of schema theory: that cognitive processing is guided and limited by the application of prior knowledge, and that schemas contain relatively abstract knowledge which is independent of any one event (Ormerod, 1990). Schemas can be thought of as a data structure representing generic concepts stored in memory (Detienne, 1990, 1995). Schemas are active processes that continually evaluate incoming information to discern if it is relevant (Relmann & Chi, 1989).

Individuals must assimilate the new material into the existing concepts or schema. What is stored in memory depends on what was presented and the schema to which it was assimilated. When an unfamiliar event is introduced, the learner activates the schema that is perceived to most closely match the event. The new information is compared against existing knowledge and either refines the existing knowledge or creates a new schema. As an individual's knowledge increases, he or she develops new or revised mental structures for organizing that knowledge

(Kraiger, Ford, and Salas, 1993; Rist, 1989). Thus from a cognitive perspective learning involves the construction or reconstruction of knowledge structures (schemas).

Proactive Interference

Because the learning process is dynamic, the student's previous experience can impact the learning (Tarpy & Mayer, 1978). When a concept is introduced the individual activates the schema that is perceived to most closely match the concept. If the mapping is correct the new information is then integrated with the existing schema. This process is known as positive transfer or making analogies (Manns & Nelson, 1996). This transfer of skill aids the knowledge compilation stage and supports the transformation of declarative knowledge into procedural knowledge (Singley & Anderson, 1989). When a learner makes an incorrect analogy (negative transfer) the existing body of knowledge is said to interfere with the assimilation of new knowledge. The result is a more difficult learning process than if there was no previous knowledge.

For example, there are two groups, an experimental and a control group. The experimental group studies for an accounting exam, then studies for a chemistry exam, and then takes the chemistry exam. The control group performs some irrelevant activity (e.g. watching TV), then studies for a chemistry exam, and then takes the chemistry exam. If the experimental group performs better on the chemistry exam, this is an example of proactive facilitation; prior learning aids the learning of new material (studying accounting aided learning chemistry). If the experimental and control group perform equally, this is an example of zero transfer. The previous learning has no effect on the new learning. If the experimental group performs worse than the control, this is an example of proactive interference; prior learning interferes with the learning of new material (studying accounting interfered with learning chemistry). So proactive interference occurs when new knowledge cannot be integrated with an activated mental model or schema (Manns & Nelson, 1996; Melton & Irwin, 1940; Underwood, 1957).

Proactive interference has been found to impact error rates when a skill is learned and then another antagonistic skill is learned. For example, consider a study of two layouts for numbers on a machine keypad, one similar to an adding machine (789,456,123,0) and the other like a telephone (123, 456, 789, 0) (Conrad & Hull, 1968). The adding machine layout resulted in more keying errors than the telephone layout. The telephone layout was part of an existing schema whereas the adding machine layout was antagonistic to that schema. When the subjects were learning the machine layout the knowledge of the telephone layout was interfering with the learning and producing negative transfer.

Another study compared the performance of experienced and inexperienced pilots under normal and reversed control stick conditions (Hendrick, 1983). Under the normal condition, experienced pilots committed less than half as many errors as the inexperienced pilots. Under the reversed control stick condition, both groups performed much poorer, but the decrease in performance was greater for the experienced pilot group. The experienced pilots encountered more proactive interference than the novices. In a study of tennis students, negative transfer occurred when the subject who first learned the forehand stroke was required to then learn the backhand stroke (Eason, Smith & Plaisance, 1989). In another study, ethnographic methods were used to study the acquisition of English as a second language (Schmidt, 1988). The overlearning of the first language created proactive interference in which the native language interfered with learning English and caused a distinctive accent. For example German /English was spoken with a distinct German accent and French/English was spoken with a distinct French accent. In each of these studies the acquisition of new knowledge was hindered by the previous knowledge.

Osgood (1949) extended the concept of proactive interference or negative transfer with the development of the transfer surface. The surface in Figure 1 is based on two dimensions, the similarity of the stimuli and the similarity of the responses. Osgood predicted the amount and direction for the various combinations. For example if the stimuli are identical and the responses are identical then we should see a correct mapping or positive transfer from one domain to another. If two systems involve learning different responses to the same or similar stimuli, interference between the systems will be the greatest when the systems are close to each other in cognitive space (Bruce, 1933; Gagne & Foster, 1949; Gibson, 1940; Saltz, 1971; Siipola & Israel, 1933). If the stimuli are identical but the responses are antagonistic then we should see an incorrect mapping or negative transfer from one domain to another.

Learning Curves: The Graphical Aspect of Proactive Interference

From a graphical perspective the proactive interference experienced during a mindshift may impact the learning curve. A learning curve has been defined as a way of representing the progress of learning (Travers, 1963); a graphical representation of the learning phenomenon observed in people performing tasks (Thurstone, 1919); and a graphic depiction of changes in performance (knowledge) during a specified time period (McCray & Blakemore, 1989). Learning curves have been used to graphically depict of the effects of proactive interference (Briggs, 1954). In Briggs' (1954) study learning curves were used to provide a graphic description of the frequency of the new response and the old response to the original stimulus. The results indicated that the amount of original learning exerted some interference on the new learning.

One of the most important aspects of a learning curve is its form, which shows the relative influence of experience (McGeoch, 1952) on knowledge. When the learning function for a simple process proceeds undisturbed by external or internal distraction the learning curve usually follows one of two shapes, as seen in Figure 2. The curve may have a positive acceleration (curve A) in which the incremental gains are slight at the beginning but become progressively larger. The other curve (curve B) may follow the law of diminishing returns (Thurstone, 1919). In this case the curve will show negative acceleration by having large incremental gains at the beginning and smaller gains as time passes. When studying the learning of complex processes the learning curve generally takes the form of an S-curve. Figure 3 begins with positive acceleration at the initial stage of the learning, passes through a region where linearity is approached, and then becomes negatively accelerated (McGeoch, 1952).

When dealing with complex processes, occasionally learning curves show marked deviation from the form found in Figure 3. One such deviation is the occurrence of one or more plateaus in the learning curve. A learning plateau has been defined as a period of little or no change in performance, which is preceded and followed by periods of improvement (McGeoch, 1952, p. 29). Figure 4 represents this graphically by a horizontal section (learning plateau) in the learning curve. The first learning plateau was documented in Bryan and Harter's (1897, 1899) studies of individuals learning telegraphy. They found that operators learning the telegraphic language experienced periodic stagnation in their learning.

Software Education Foundations

Historically researchers have used two main approaches to study learning software development: expert-novice differences and the transfer of problem solving skills (Ormerod, 1990). Because this study was focused on an expert in one mindset learning a new mindset and is not concerned with novice developers we based our research on the transfer of problem solving skills approach. A brief description of the existing literature using each approach will provide a context for the study.

Expert-Novice Differences

Within the expert-novice comparison approach, prior research has focused primarily on differences between developers within the same development mindset (e.g. procedural). Studies have found that when comprehending a program, experts tend to form abstract representations containing general knowledge about what a program does whereas novices form a more concrete representation of how a program functions (Adelson, 1981, 1984; Kahney, 1983; McKeithen, Reitman, Rueter, & Hirtle, 1981; Murphy & Wright, 1984; Shneiderman, 1976; Vitalari, 1985; Weiser & Shertz, 1983). As a developer becomes more experienced he or she develops larger and larger chunks of information to represent important functional units or structures. The experienced developers recode the syntactic form in their minds and deal with the problem at a semantic level. The novices were constrained to deal with the syntactic inputs and have greater difficulty with the complex details of the statements.

There have also been a few studies of experts in one development mindset being re-trained in another mindset using the expert-novice comparison approach. Research based on learning theories indicates that prior knowledge of procedural techniques hinders the transition to object-oriented techniques compared to the performance of students who have no prior experience with procedural techniques (Dumas & Parsons, 1995; Nelson et al., 1997; Rosson & Alpert, 1990; Rosson & Carroll, 1990; Vessey & Conger, 1994;). Procedural experts introduced to object-oriented concepts will often fall back on their procedure-oriented knowledge

(Detienne, 1995; Gibson, 1991; Manns & Nelson, 1996; Pennington, Lee, & Rehder, 1995).

While providing an important historical context, there are two reasons why the expert-novice comparison approach is not appropriate for this research. First, the studies using expert-novice comparison approach have focused on the differences between the two groups at static points rather than the dynamic process of learning. They do not investigate the learning process as it develops over time (Nelson et al., 1997). Second, we are not investigating the training of novice developers, but are concerned with the issue of retraining existing expert developers. Therefore, the novice-expert framework is not appropriate for this research.

Transfer of Problem Solving Skills

The second approach used in software education research has been the transfer of problem solving skills. Limitations of knowledge organization, representation and application are major constraints for the problem solver. A way to overcome these constraints is to acquire expertise. One method of gaining expertise is to transfer skills used in one problem domain to another (Ormerod, 1990). The transfer of skills approach is appropriate for this research because we were concerned with the question, "Will students with previous development experience transfer their existing skills and knowledge to the new development mindset? And if so will that previous knowledge aid or interfere with the learning process?"

In a study that looked at learning software development from a transfer perspective within the same mindset Wu & Anderson (1991) found that subjects who knew LISP (a language in the functional mindset) transferred their knowledge to the production of PROLOG (also in the functional mindset) solutions, especially when they had been shown LISP solutions to the problems. Two studies (Scholtz & Wiedenbeck, 1990, 1992) found programmers were able to make use of their previous knowledge base when learning a new language in the same development mindset.

From a transfer of skills perspective, the shift from procedural to object-oriented techniques has been addressed in a small number of studies. One study observed procedurally oriented developers trying to understand Smalltalk code by using a procedurally motivated strategy (Campbell, Brown, and DiBello, 1992). Whereas another study observed that experienced procedural oriented developers unsuccessfully mapped new object-oriented concepts onto their procedurally oriented development knowledge (Nelson et al., 1997). They identified five categories of object-oriented learners (slow and steady, well-rounded, single-paradigm, minimalists and zealots) who adopted different strategies to overcome the obstacles experienced in the learning process. In their study of procedural programmers learning object-oriented concepts both Dué (1993) and Manns and Nelson (1996) found that the biggest training issue with regard to object-oriented techniques is the mindshift from procedural to object-oriented thinking.

Research to date indicates that when learning object-oriented techniques, experienced developers try to understand the new object-oriented concepts from their procedurally oriented perspective. Existing research in the software development education field has determined that making the transition from procedural to object-oriented techniques *is* difficult. But to date, no one has investigated *why* the transition is so difficult. This research sought to understand the learning processes *and* the impact of previous knowledge on procedural expert developers during the shift to object-oriented techniques.

As stated earlier, during the learning process the learner may attempt to map knowledge from familiar domains (procedural software development mindset) to the new domain (object-oriented development mindset). When an unfamiliar event is introduced, the learner activates the schema that is perceived to most closely match the event. With the introduction of object-oriented methods, the learner may activate the procedural software development schema. The new information is compared against existing schema and either refines the existing knowledge or creates a new schema. In the case of learning a new programming language within the same mindset much of the new information is consistent with the schema. Therefore the information refines the existing schema. In the case of learning a new development mindset (such as OO) much of the new information is inconsistent with the active schema. Eventually a new schema will be created, but during the learning process the learner attempts to map the new knowledge onto the old schema. This causes the learner to create an incorrect analogy and experience proactive interference.

When a learner makes an incorrect analogy (negative transfer) the existing body of knowledge is said to interfere with the assimilation of new knowledge. Within the domain of software development an example of proactive interference (negative transfer) would be if a developer is asked about data and behavior within each mindset (procedural and OO). The developer's knowledge of data and behavior within the procedural mindset would not map correctly onto the object-oriented concepts of data and behavior (encapsulation).

From this proactive interference procedural experts learning object-oriented techniques may experience periods in which little progress is made. These periods of stagnation can be seen in the learning curves as learning plateaus. Since learning software development is a complex process (rather than a simple process) its learning curve should more closely resemble Figure 3. But, if the experts experience proactive interference then the learning curve would more closely resemble the curve containing plateaus as in Figure 4. Which leads to our hypotheses:

> *H1: The learning curve for procedural software development experts learning object-oriented techniques will include learning plateau(s).*
>
> *H2: Proactive interference is positively associated with learning plateau(s).*

H3: Learning plateau(s) will occur at certain levels of object-oriented experience.

H4: Learning plateau(s) will cluster around certain object-oriented concepts.

Given the above hypotheses, how do we identify the interference? As there was no existing measurement instrument we developed and validated a measurement instrument as part of this study.

Summary of the Software Development Learning Model

The goal of this chapter was to investigate the learning process involved as IS experts shift from an existing mindset to a new one, and how previous knowledge interferes with the process. Elaborating on previous models of learning, a new theoretical model of learning within the software development domain was produced. During the learning process the learner may attempt to map knowledge or transfer skills from familiar domains to the new, unfamiliar domain. When a learner makes an incorrect analogy the existing body of knowledge is said to interfere with the assimilation of new knowledge. The result is a more difficult learning process than if there was no previous knowledge.

From this proactive interference procedural experts learning object-oriented techniques may experience periods in which little progress is made. These periods of stagnation can be seen in the learning curves as learning plateaus. These plateaus in the learning curves will occur at certain levels of object-oriented experience and will cluster around certain object-oriented concepts.

Using the previous theoretical arguments as a framework, the next chapter discusses the research method that was employed to empirically test the hypotheses. A field study was conducted in order to provide a more comprehensive examination of the phenomena from an organizational perspective (Babbie, 1995).

CHAPTER THREE

Research Design And Method

In the previous chapter we explored the development of learning theory, schema theory and proactive interference through the field of psychology. We then examined the software education literature to aid model development. Lastly, we developed a new theoretical model of learning within the software development domain from the intersection of the learning and software development education research.

This chapter discusses the research design and method used in this study. The first section details the evocative nature of this study and the need for a multiple method research design. Since we were studying the process of learning, the second section discusses the issue of the data collection frame with regard to the learning process. The next section lays out the research design for the study. This research was carried out in three phases. In Phase I we elicited domain specific knowledge from software development experts. In Phase II we created and validated a measurement instrument from the Phase I data. In Phase III we administered the Software Development and Maintenance Approach instrument to a large sample of software developers. The final section of this chapter summarizes the research design.

Evocative Research

A systematic identification of the major constructs for procedural and object-oriented software development expertise has yet to receive significant attention.

General theoretical frameworks have been developed in the software development expertise field (e.g. Lee & Pennington, 1994; Manns & Nelson, 1996; Shneiderman, 1976). But the frameworks that have been developed were from primarily exploratory methods (Detienne, 1995; Nelson et al., 1997; Soloway & Ehrlich, 1984; Spohrer & Soloway, 1986;). To date, no one has attempted to operationalize the constructs of object-oriented and procedural software development expertise so they may be empirically tested. To move this area of research forward we need to progress from the general, qualitative based, theories toward quantitative verification. To accomplish that goal we needed techniques that can bridge the gap from qualitative identification to quantitative verification.

"Evocative" studies address a class of theoretical problems between general domains with undeveloped theories and specific domains with clearly formulated theories. These mid-range theories can be evoked through qualitative research methods using experts in a particular domain as respondents. The theories are then interpreted through the knowledge found in generalized theories, in this case, theories of learning and expertise.

It has been suggested that evocative methodologies are appropriate in examining issues in IS such as software development expertise (Nelson, Nadkarni, Narayanan, and Ghods, 2000). The goal of this study was to operationalize and test the constructs of object-oriented and procedural software development expertise. Therefore an evocative approach was appropriate to study this phenomenon. We used an evocative design in Phase I to elicit the constructs of software development learning theory and develop an instrument to verify the evoked theory.

Longitudinal Approximation

In contrast to prior research, which has examined learning from a static point, this study examined learning from a dynamic view. To better understand the long-term process of learning, this study examined the *transition* from the procedural mindset to the object-oriented mindset. There are two methods that can be used to study the long-term process of learning: a longitudinal study or a cross-sectional field study. A longitudinal study follows a single group throughout the entire learning process. The limitations of a longitudinal study include participant attrition, cost, and the length of time required to detect changes (Simon & Burstein, 1985). Due to the impossibility of following any one group of software developers for an extended period of time given current IS job market volatility a cross-sectional design was used to approximate a longitudinal study (Gerencher, 1999; Goodner, 2000). A cross-sectional study infers the process from a much larger group. Cross-sectional data that captures responses at different points in the process can be used to approximate longitudinal data (Babbie, 1973; Simon & Burstein, 1969). It is possible to draw approximate conclusions about processes that take place over time even when only cross-sectional data are available (Babbie, 1973).

Research Design

This section describes the design and data collection procedures used for this study. This dissertation was conducted in three phases. In Phase I, domain knowledge was elicited from expert procedural and object-oriented software developers. Cognitive modeling techniques were used to evoke the domain knowledge of the experts. In Phase II, this domain knowledge was used to create a measurement instrument. The Software Development and Maintenance Approach instrument was designed to measure an individual's object-oriented mindset and if any proactive interference was present. After the instrument was created, it was then formally validated. In Phase III the validated Software Development and Maintenance Approach instrument was administered to a large sample of software developers.

Phase One: Knowledge Elicitation

The major task of Phase I was to elicit the relevant knowledge of expert procedural software developers and expert object-oriented software developers so as to understand their cognitive structures. To understand cognitive structures we need to study the cognitive representations of the individual (Pennington, 1987).

Cognitive structures have been explored through a number of data collection methods: structured and unstructured interviews; task performance; and verbal protocols to name a few (Gordon, 1992). Structured and unstructured interviews require the interviewer to ask questions of the expert regarding the relevant domain. With structured interviews the questions are established a priori and follow a predetermined pattern. Unstructured interviews can be more flexible with the interviewer asking questions that seem relevant and likely to elicit new information. With task performance, rules are developed by observing multiple examples of actual task performance under a variety of circumstances. The data collected from the observation are then used to infer the relationships between informational input and behavioral/verbal output.

Protocol analysis has been the prominent qualitative analysis method used to elicit cognitive representations (Ericcson & Simon, 1980; Koubek, Salvendy, Dunsmore, and LeBold, 1989). In protocol analysis, subjects perform a task and verbalize their thoughts as they solve the problem. With both task performance and protocol analysis, because the individual is working at the task level, the data gathered are low level abstractions that are intertwined with the programming language used by the individual (Gibson, 1991). The emphasis of this research study was on the conceptual or high level abstractions that define each mindset. This information is language independent and needed to be gathered using a language independent method. Since the expertise gathered was at the conceptual level and not the task level, task performance and protocol analysis were deemed inappropriate methods.

An alternative to the previously mentioned methods of eliciting expertise is causal mapping. Causal mapping is a collection of techniques used to explicate and assess the structure and content of mental models (Axelrod, 1976; Fiol & Huff, 1992). Causal mapping provides a method to structure and simplify thoughts, to make sense of them, and to communicate information about them (Fiol & Huff, 1992). Causal mapping can be used at the conceptual level and is thus language independent. Therefore, casual mapping was the better choice to gather the language independent concepts and higher-level abstractions of each software development mindset.

Another benefit of the causal mapping technique is its expandability. Causal maps can be used at both the individual and the group (aggregate) level (Axelrod, 1976; Huff, 1990). Aggregate maps have been successfully used to elicit group level cognition (Bougon, Weick, and Binkhorst, 1977; Eden, Jones, Sims, and Smithin, 1981; Narayanan & Fahey, 1990; Fiol & Huff, 1992). Since this study explored software development expertise at the group level (object-oriented versus procedural mindsets) it was important to use a method that adapted to the aggregate level.

Revealed Causal Mapping.

There are several forms of causal maps that allow researchers to create theoretical representations of a phenomenon (Boland et al., 1994; Bougon et al., 1977; Eden, Ackerman, & Cropper, 1992; Ford & Hegarty, 1984; Narayanan & Fahey, 1990; Zmud et al., 1993). To understand the cognitive representations of experts we used a form of causal mapping called revealed causal mapping (RCM) as our foundation. The maps provide a frame of reference for what the expert knows and exhibits and the reasoning behind the expert's actions.

There are two approaches to using revealed causal mapping, confirmatory and exploratory. The confirmatory approach is most appropriate when well-established theories exist for a given research domain. The exploratory approach is appropriate when few theories have been developed for the research domain and is consistent with the evocative nature of this study. The exploratory approach consistent with the revealed causal mapping procedure used by Nelson, Nadkarni, Narayanan, and Ghods (2000) and Narayanan and Fahey (1990) was used for this study.

Qualitative Identification: Step 1.

In the first step, the data source (domain experts) is selected and narratives are gathered. This is accomplished through open-ended interviews, which are discussed later in this section. The researcher's goal is to gather their knowledge and cast it into available theoretical frameworks to construct domain specific theories. The task then is to access relevant experts and assist them in articulating their mostly tacit knowledge. To accomplish this task expert object-oriented and expert procedural software developers were identified using a snowball technique (Shanteau, 1987, 1992) and convenience sample (Stone, 1978).

Sampling Method.

Snowball sampling is a method used when members of a domain cannot easily be located by random sampling or by screening, and where the members of a domain know other members of the domain (Simon & Burstein, 1985). One application of the snowball sampling technique is in the surveying of rare populations (Simon & Burstein, 1985). The snowball method was appropriate for this research because we were dealing with a rare population (*expert* software developers). The snowball technique asserts that those individuals closest to a domain are appropriate to define the experts of that domain (Shanteau, 1987, 1992).

The initial respondent is chosen and additional respondents are obtained from information provided by the initial respondent. One expert identifies another and that expert identifies another, and so on. Once identified, each expert software developer was interviewed (Axelrod, 1976; Huff, 1990).

The snowball sampling method is a nonprobability method and consequently there is a potential for sample bias. The bias results from the person who is known to more people having a higher probability of being mentioned than the person known only to a few others (Sudman, 1976). This bias was minimized for this project because we asked software development professionals to identify others in their area with expertise. Professionals in a field are competent to identify a consistent set of attributes they associate with expertise (Abdolmohammadi & Shanteau, 1992). Thus selection was based on expertise and not familiarity.

Sample.

The participants in Phase I were expert procedural and object-oriented software developers, as acknowledged by their peers using the snowball sampling technique detailed above. Organizations were selected based on their identification of available "expert software developers" and their willingness to participate. As in any organizational field study, the organization's willingness to participate was a determining factor in their presence in our sample. Over fifteen organizations of various sizes (15-10,000 employees) and industries (e.g., telecommunications, manufacturing, consulting, and services) provided access to their software developers. Table 2 describes the Phase I participants.

Interview Method.

The interview process consisted of open-ended interviews with probes (Rossi, Wright, and Anderson, 1983). An interview guide was adapted from Nelson, Armstrong, and Ghods (in press) by the primary researcher to facilitate the interview process. The guide was then validated by two researchers, one with extensive RCM experience and the other a software development expert. See Appendix A for the interview guide. During the interviews respondents were asked questions regarding how they *think* about software development. For example:

> Think of a time when you have been given a requirements document (for example, to develop an accounting system) and asked to produce an object-oriented (procedural) solution. What was the first thing you thought about?

Based on the respondent's answer to this question, follow up probes were asked to elicit further details regarding their software development thought process (e.g. "What did you think about next?"). The interviewer did not constrain the responses to the questions, but allowed the participant to expound. Each interview lasted from 30 to 90 minutes. The interviews were transcribed into a document format ranging from 4 to 14 pages.

Point of Redundancy.

Within the RCM method, the researcher should interview to the point of redundancy, which calculates the adequacy of the sample size (Axelrod, 1976). In causal mapping research the point of redundancy, or saturation, among the subjects represents the point at which further data collection would not lead to the identification of additional concepts. This point serves as a way of establishing the adequacy of the sample. Prior to commencement, we estimated the number of interviews necessary to reach redundancy or saturation of concepts at 60 (30 procedural and 30 object-oriented). The point of redundancy is operationalized by aggregating the concepts mentioned by each participant. The difficulty is that the point of redundancy is not calculated until after the interviews have been completed and the classification scheme has been developed. If redundancy is not reached, additional interviews are conducted. For this project, redundancy on the procedural concepts was reached at 7 participants and at 20 participants for the object-oriented and is demonstrated in Figure 5. The point of redundancy suggested that the achieved sample of 55 respondents (35 object-oriented and 20 procedural software development experts) was more than sufficient to capture all of the relevant concepts.

Qualitative Identification: Step 2.

The two main considerations within this step are the identification of the causal and evoked statements and the establishment of the reliability of the identification procedure (Axelrod, 1976). The first task is to identify the causal statements from the interview transcripts. Causal statements are statements that imply a cause-effect relationship. Some of the key words used in identifying explicit causal statements are "if-then," "because," "so," and so forth. Due to the cognitive nature of this study it was determined that in addition to explicit causal statements, implicit causal statements should also be recorded. The key words used in identifying implicit causal statements are "think," "know," "use," and "believe". The two sets of causal statements (explicit and implicit) were kept segregated throughout the research project. Consistent with Narayanan and Fahey (1990), all the statements in the form of concepts and cause-effect relationships are captured in the language of the experts. Examples of these statements are listed in Table 3.

In addition to causal statements, evoked statements were also elicited from the transcripts. An evoked statement is a declarative statement made by an expert regarding a specific domain. It is a statement that is a reflective indicator of the underlying concept. In this study the evoked statements concerned software development concepts. The statements expressed the components of object-oriented and procedural software development as seen through the expert software developer's eyes. The evoked statements were elicited in the same manner as the causal statements. Each transcript was reviewed and the evoked statements were identified. A statement was classified as evoked if it contained a definitional reference to either mindset. Examples of evoked statements are presented in Table 4.

To establish the reliability of the identification procedure, each interview text was coded by the primary researcher and one of the three raters. The raters were deemed appropriate to identify both causal and evoked statements because of their familiarity with the technique and the domain under study. A glossary of software development terms was provided to the researchers for identification purposes. See Appendix B for the glossary of terms. The glossary was created from the interviews and existing software development textbooks. There were two rounds of coding that covered 12 interviews. Six object-oriented and six procedural interview texts were chosen at random from the object-oriented and procedural interviews. Comparisons were made for agreement and disagreement between the researchers. Where disagreement occurred the discrepancies were resolved through discussion.

The reliability between the researchers was calculated by measuring the level of agreement on terms, linkages, and whether a statement was evoked, causal, both or neither. The level of agreement between the researchers was measured using the Fisher exact test and the Chi-Square (^{2}) test. For the Fisher test, the two-sided p value for causal statements (.840) and evoked statements (.643) indicates an acceptable level of agreement among the researchers. For the ^{2} test the data confirm an acceptable level of agreement among the researchers for causal statements (^{2} = .135, df =1, p=ns) and evoked statements (^{2} = .219, df = 1, p=ns).

The Kruskal-Wallis significance test was performed to compare the two groups of experts (procedural and object-oriented). The two groups were determined equivalent in terms of the causal statements (b= .011, df =1, p= ns) and the evoked statements (b= 2.557, df=1, p= ns) they produced. Thus the two groups could be treated as equals for coding purposes.

<u>Qualitative Identification: Step 3.</u>

In the next step, the relevant concepts are identified from the statements (Narayanan & Fahey, 1990). The coding process begins with grouping frequently mentioned words in the statements. The categories are identified as the narrative texts are reviewed. The statements containing the same concept were grouped together. A word or word group was created that captured the essence of the statement. For example, the sentence "You group the requirements document items based on functions" was labeled "functions". A second researcher who also is a software development expert reviewed the statements and independently placed them into categories. Comparisons were made for agreement and disagreement. Where disagreement occurred the discrepancies were resolved through discussion. Once the conceptual level scheme was developed, the statements were placed into the appropriate categories. If a statement fit more than one category the sentence was placed in both categories.

A second round of concept identification was accomplished by four additional individuals working independently. Three object-oriented and one procedural expert validated the concept level scheme. The interrater reliability between the researchers was measured using the Spearman's Rank Correlation Coefficient and the Chi-Square (^{2}) test. The value for the Spearman's Rank Correlation test (.629) indicates an acceptable level of agreement among the researchers. The value for the ^{2} test (^{2}= 11.071, df=5, p=ns) confirms an acceptable level of agreement among the researchers. The participants then worked together to resolve any discrepancies in the identification of the concepts.

There were a total of 78 concepts identified (53 object-oriented and 25 procedural). After further analysis of the classification scheme a few of the concepts were determined to be conceptually equivalent. From the object-oriented list it was found that the statements included in the "behavior," "method" and "responsibility" categories were not conceptually distinct. The statements under these concepts were combined into the "method" concept. For the procedural list "functional decomposition" and "top down design" were determined to be conceptually equivalent and combined into the "functional decomposition" concept. The positively and negative worded statements for the "object-oriented development" concept were combined. This took the total number of concepts to 74 (50 object-oriented and 24 procedural).

Qualitative Identification: Step 4.

In the next step a construct and meta-construct level classification scheme was developed. There were four iterations of the construct and meta-construct level classification scheme. The first cut of the construct level classification scheme was accomplished by the principal researcher. Three categories were initially created: "Definitions," "Benefits" and "Techniques". In the second iteration, the "Definitions" category was further subdivided into 4 categories: "Definitions," "Actions," "Interactions," and "Elements". At that point the construct level classification scheme was reviewed by two object-oriented and one procedural software development expert.

In the third iteration the “Definitions” category was relabeled “Application Design”. The “Interactions” and “Actions” categories were reformulated into Design and Runtime respectively. These meta-constructs were further broken down into “Design Characteristics,” “Design Relationships,” “Execution Relationships” and “Execution Interaction”. The “Elements” category was relabeled “Object (Procedural) Characteristics” and grouped with “Techniques” and “Benefits” under the “Conceptual” meta-construct label. In the final iteration, the “Conceptual” meta-construct was used only for the “Object (Procedural) Characteristics” construct. The “Techniques” category was renamed “Analysis” at the meta-construct level and subdivided into “Analysis Tools” and “Analysis Techniques” constructs. The conceptual scheme was finalized with 6 meta-constructs: Application, Design, Runtime, Conceptual, Analysis and Benefits. There were 9 constructs: Application Design; Design Characteristics, Design Relationships; Execution Characteristics, Execution Interaction; Object (Procedural) Characteristics; Analysis Techniques, Analysis Tools; and Benefits. See Table 5 for the Expert Classification Scheme.

Qualitative Identification: Step 5.

Once the classification scheme was completed, the causal and evoked statements were placed into the appropriate categories. A total of 366 causal statements and 899 evoked statements were elicited from the transcripts.

Phase Two: Instrument Development

The Software Development and Maintenance Approach instrument was developed from the statements elicited in the Qualitative Identification Phase of the project. The 1265 statements (366 causal + 899 evoked) that were categorized based on the classification scheme were evaluated to determine their appropriateness for inclusion in the instrument. The criteria for inclusion in the instrument were the statement's content (was the statement definitional or merely commentary), accuracy, and parsimony. As a starting point, five statements per concept were selected for inclusion in the instrument. A statement was selected based on its content and clarity. The first draft of the Software Development and Maintenance Approach instrument consisted of 370 statements (74 concepts x 5 statements per concept).

We then reviewed the instrument to determine an appropriate length. The criteria for questionnaire length should include cost, response rate, and the limits of respondent willingness to answer questions (Fowler, 1993). Because the instrument was being administered online, its length did not impact the costs incurred. But the length of an instrument is seen to have a negative impact on response rates in that the longer the instrument, the more likely it is that the response rate will be lower (Bolton, Chapman, and Zych, 1990; Herberlien & Baumgartner, 1978; Steele, Schwendig, and Kilpatrick, 1992; Yammarino, Skinner, and Childers, 1991). One study discovered that participants in business-oriented studies were more cognizant of survey length than consumers (Jobber & Saunders, 1993), while another found survey length to be one of the main reasons for a business persons' non-response (Tomasokovic-Devey et al., 1994). Written questionnaires should not exceed 30-45 minutes in length if you want to obtain reasonable response rates (Sudman & Bradburn, 1982). It took the primary researcher two hours to complete a written version of the 370-item instrument. Therefore, based on past findings, 370 items were deemed too many to realistically answer.

The Expert Classification Scheme was scrutinized to determine if any concepts, constructs or meta-constructs could be combined or eliminated. Upon further analysis of the classification scheme two constructs, "Benefits" and "Analysis Tools" were eliminated from the instrument development process. These concepts, while part of object-oriented development were determined to be outcomes of using the mindset not components of the mindset. The "Benefits" construct consisted of concepts such as "change," "extensibility," "natural," and "reuse". Since we were trying to capture the cognitive components of each mindset not the outcomes of using the mindset, the "Benefits" construct was deleted from the instrument. This eliminated 17 concepts (14 object-oriented and 3 procedural).

The "Analysis Tools" construct was also deleted from the instrument development. This construct was eliminated from the instrument because the concepts in this construct were tools used by the developers not components of the development mindset. The "Analysis Tools" construct consisted of 6 concepts: "class diagram," "CRC cards," "design patterns," "sequence diagram," "UML," and “use cases”. This eliminated 8 concepts (6 object-oriented and 2 procedural), making a total of 49 concepts (30 object-oriented and 19 procedural).

The texts were then reviewed for outliers, concepts that were mentioned by only 1 individual. For example only one individual mentioned the concept “thread,” and he mentioned it four times. In contrast, the average mention rate was 19.38 for an object-oriented concept and 10.00 for a procedural concept. Therefore, the “thread” concept was determined to be an outlier and deleted from the instrument. Other concepts that were deleted based on their outlier position were “aggregation” and “glossary” from the object-oriented mindset, and “utilities” from the procedural mindset.

Three concepts (“iterative,” “business knowledge,” and “testing”) were eliminated from the instrument because they did not provide any insight regarding the transitional learning process. “Iterative” (as in an iterative method of developing software), “business knowledge” and “testing” were identified as important aspects of software development regardless of the development mindset (procedural or object-oriented). Since there was no discernment between respondents these concepts were eliminated. “Process model” was eliminated because it was confusing and deemed not applicable to the transition from procedural to object-oriented software development. This took the total to 41 concepts (25 object-oriented and 16 procedural). Ten statements were deleted for clarity, irrelevance or confusion, making a total of 195 statements for the pretest.

Pretest Sort.

A pretest sort was used to address the internal and construct validity of the preliminary Software Development and Maintenance Approach instrument. A convenience sample of five IS experts (three object-oriented and two procedural) participated in the pretest sort. The criterion for selection was the subject's willingness to participate. A list of 195 statements (approximately 5 statements per construct) was presented to the subjects along with a separate list of 41 constructs. The facilitator had the subjects sort statements by construct (Anderson & Gerbing, 1991). Based on the results of the sort the statements were re-worded or deleted as needed. The pilot instrument was developed from the remaining list of 190 statements.

Pilot Instrument.

A pilot test of 20-50 subjects is usually sufficient to discover the major flaws in a survey and perform statistical analysis (Sudman, 1976). The Software Development and Maintenance Approach instrument was given to individuals in a group setting with group sizes ranging from 1 to 20. Eight subjects (from the group of 20) had no previous software development experience and were eliminated from the pilot test. The final sample consisted of 31 respondents. The participants represented a cross section of procedural experts, object-oriented experts and individuals at various points in the learning process. Table 6 describes the Phase II participants.

The instrument had two sections: Section 1 contained questions regarding how developers think about software development and Section 2 contained demographic questions. The questions in Section 1 covered both the procedural and object-oriented development mindsets. Each question was rated on a five-point Likert scale (Babbie, 1979, p. 410; Nachmias & Nachmias, 1981, p. 278-280) using the labels "strongly disagree", "disagree", "neither agree nor disagree", "agree", and "strongly agree" with the addition of a "don't know" option. A 5-point scale was used because we were looking at a person's mindset. As previously stated, a mindset is a distinctive viewpoint that determines how an individual views reality (Culbert, 1996). The participant (experienced software developer) thinks about software development from one mindset or the other. It was expected that the participant would respond from a binary perspective (the statement is consistent or inconsistent with the participant's mindset). Therefore a 5-point scale was sufficient to discriminate between participant responses.

The response choices were also reviewed. The selection of the wording of the middle response (Sudman & Bradburn, 1982, p.141) choice "neither agree nor disagree" was scrutinized. Several other options including "undecided" and "neutral" were discussed but discarded. The selection of the "neither agree nor disagree" wording was based on the respondent pool and the study domain. The researchers felt that a software developer would not be neutral or undecided about a software development mindset. The "neither agree nor disagree" option more closely represented the cognitive position of the subjects.

Because the instrument was anchored in the object-oriented perspective the "don't know" response choice was included. It was felt that respondents who were not anchored in the object-oriented perspective needed an option to reflect their lack of knowledge (Schuman & Presser, 1981, p. 170). The wording of the "don't know" option was also closely scrutinized. Several choices were reviewed such as "uncertain" and "no opinion". The "uncertain" option was deemed too similar to the "neither agree nor disagree" option and the "no opinion" option reflected a different connotation so both were discarded. The "don't know" option was felt to most accurately reflect the cognitive state of the respondent. The phrase "no knowledge or experience in this area" was added to the "don't know" option. The phrase was added to further clarify and distinguish the "don't know" option from the "neither agree nor disagree" option.

After completing the instrument each respondent was debriefed to determine if questions were confusing or if the terminology used related in a meaningful way to the concepts they were intended to measure. The in-depth discussions of each question checked for content, clarity and meaning. The criterion for keeping the question was its clarity, meaningfulness, ability to measure the construct, and understandability. Based on the evaluations of the participants the questions were further refined and/or deleted. Each version of the instrument incorporated the improvements suggested by the participants debriefed up to that point. The instrument was continuously re-edited until a form of consensus was reached. Through the pilot study phase, the number of questions was gradually reduced from 190 to 175.

<u>Pilot Study Data Analysis.</u>

Statistical analysis was performed on the pilot study data using SPSS version 10.0. Thirty-one subjects were used in the initial analysis. Histograms were created and reviewed for each of the 175 questions. At a question level the distinction of responses was observed. Individuals with greater object-oriented experience answered "strongly agree," to the object-oriented questions and "strongly disagree" to the procedural questions. The individuals with only procedural experience answered "strongly disagree," to the object-oriented questions and "strongly agree" to the procedural questions. The individuals in transition were scattered about the scale. These results indicated that the items were detecting differences in the individual's mindsets.

Scale reliabilities were used to determine the three questions to retain for the final instrument (Huck, 2000). Scale internal consistency reliability was assessed using Cronbach's alpha (Huck, 2000, p. 91). The 'interface' and 'class hierarchy' concepts were deleted because their scale reliabilities were well below the .60 recommendation for newly developed scales (Nunnally, 1967). Only 5 concepts were between .60 and .70 scale reliability, all others were above .70. See Appendix C for scale reliabilities. Correlations between variables were analyzed, and questions were again deleted or re-worded for clarity.

The focus of this study was on understanding the shift to object-oriented techniques and how procedural knowledge interferes with the process. We were not concerned with procedural knowledge per se, but only in the procedural knowledge as it interfered with the object-oriented learning. Therefore, procedural variables were deleted if they did not correspond to object-oriented variables. For example, the procedural concept "data model" corresponds with the "object model" concept, so "data model" was included in the instrument. Those procedural concepts that did not correspond to an object-oriented concept and were deleted were 'control', and 'systems development life cycle'.

After completing the group level analysis, the sample was subdivided and the responses were analyzed by subgroup. The 31 subjects were subdivided into procedural experts (8), object-oriented experts (6) and transitioners (17). The divisions were made based on the individual's years of procedural and object-oriented software development experience, and the number of projects he or she participated in with both mindsets. The "transitioners" were procedural software developers who had some exposure to the object-oriented mindset. The subgroup responses were analyzed using histograms, and the data was appropriate for each group. The expert histograms were highly skewed to the scale extremes and the transitioning group histograms were fairly normally distributed.

The transitioning group of 17 participants was analyzed in more detail and was subdivided into three groups based on their years of procedural and object-oriented software development experience, and the number of projects participated in with both mindsets. Subgroup 1 consisted of seven participants who had no object-oriented experience, but had participated in one object-oriented project. The histograms for subgroup 1 were skewed toward procedural responses. Subgroup 2 consisted of six participants, who had 1-3 years of object-oriented experience, and participated in one object-oriented project. The histograms for subgroup 2 were also skewed toward procedural responses, but not as strongly as group 1. Subgroup 3 consisted of four participants, who had 2-3 years of object-oriented experience, and participated in two object-oriented projects. The histograms for subgroup 3 were slightly skewed toward object-oriented responses. These results confirmed that the instrument was detecting differences in the developers' mindsets. At that point it was determined that further subgrouping was not feasible or needed.

Once the data analysis was complete, two content experts and one survey construction expert scrutinized each question of the instrument. Question wording and grammatical changes were made. The final instrument consisted of 90 questions covering 34 concepts (22 object-oriented and 12 procedural). The instrument was formatted into its online appearance. The result of Phase II was a comprehensive online instrument designed to measure the extent of an individual's object-oriented software development mindset.

Phase Three: Quantitative Verification

The major task of Phase III was to distribute the online instrument to a large sample population of software developers so as to understand their cognitive processes with regard to software development.

Sample.

In Phase III the validated instrument was administered to a large sample of software developers. Study respondents were chosen based on a key-informant method (Bagozzi, Yi, and Phillips, 1991). The instrument was given to procedural developers with various levels of expertise in the object-oriented development mindset and object-oriented developers with no previous procedural experience. Participants in this phase of the study were different than the respondents used in either of the previous phases of the project. Organizations were selected based on their identification of available "software developers" and their willingness to participate. As in any organizational field study, the organization's willingness to participate was a determining factor in their presence in our sample. Over 32 organizations of various sizes (5-100,000 employees) and industries (e.g., telecommunications, manufacturing, consulting, and services) provided access to their software developers. The number of organizations represents a broad array of organization and industry types. Table 7 describes the Phase III participants.

Data Collection.

The research instrument used for the collection of software development mindset data was the software development and support questionnaire. The questionnaire asked questions regarding how an individual thinks about software development. This instrument contained multiple questions for each of the 34 variables, all of which were tested on the pilot instrument. This instrument used a five point Likert Scale numbered from 1 to 5 (Strongly Disagree, Disagree, Neither Agree Nor Disagree, Agree, Strongly Agree) with a "Don't know, no knowledge or experience in this area" option numbered 0. The response choices were listed horizontally across the screen with the "Don't know, no knowledge or experience in this area" as the first option at the left edge of the screen. The most socially undesirable response choice was listed first (Sudman & Bradburn, 1982, p. 156). If the socially desirable choice(s) are placed first, the respondent might choose without reading through the entire list. By placing the socially desirable choices at the end (to the right reading from left to right), this increases the likelihood that the respondent would review all of the response choices before making a selection.

Data collection was accomplished via online instrument. The instrument was developed using SPSS Data Entry Builder version 2.0. The data was collected directly into an SPSS file for data analysis. The survey was created and uploaded to the University of Kansas SPSS data server (swift). The results were then captured by the server and transferred to the researcher's data collection file. Each participating organization provided a liaison for the researcher to work with. An email message was sent to the contact person at each organization. He or she would then forward the email (which included the instrument URL) to their software development personnel. Although some selection bias may have resulted from this technique, this manner of selection greatly aided the distribution of the instrument and showed internal support for the study by the organization. The URL for the survey was:

http://swift.cc.ukans.edu/darmstrong/djasurvey/webfiles/index.htm.

When the URL was activated from the respondent's browser, the first screen to be seen was the "form loading" screen. See Appendix D for the "Form Loading" screen. A limitation of the SPSS software is the need for a Java script compatible browser. Respondents needed a browser that was compatible with Java scripts, such as Internet Explorer version 4.0 or higher or Netscape version 4.7 or higher. The message on the browser informed the respondent that he or she needed a Java script compatible browser. The next screen was the software development instrument. See Appendix E for the Software Development and Maintenance Approach instrument format. A brief description of the study was provided prior to the Section I questions. At the completion of the instrument there was a "submit" button. When the respondent clicked on that button the data was transmitted to the SPSS server and a "thank you" screen appeared. See Appendix F for the "Thank You" screen. This screen thanked the participant and asked them to close their browser. This statement was inserted because if the respondent used the back feature on their browser they could end up submitting their data several times. Data collection began on March 5, 2001 and was concluded on March 30, 2001. The initial email was sent on March 5, 2001. Follow up emails were sent on March 13, 2001 and March 20, 2001. In addition, telephone contact was made with three organizations on March 20, 2001. Due to technical incompatibilities with one organization's Internet browser, the instrument was sent as an attachment via email. The participants of this organization responded directly to the primary researcher via email. Data collection was concluded on April 5, 2001.

Summary of Chapter Three

Chapter Three explicated the research design and method for this study. The research design, sampling strategy and procedure for the three phases of the study were presented. In Phase I domain specific knowledge was elicited from software development experts. In Phase II a measurement instrument was created from the Phase I data and validated. In Phase III the instrument was given to a large sample of software de''elopers. The next chapter discusses the results of the data analysis for the Software Development and Maintenance Approach instrument.

CHAPTER FOUR

Data Analysis And Results

The research questions, "Why is it difficult for procedural experts to learn object-oriented development?" and "Where in the learning process are developers experiencing difficulty?" drove the method of data analysis. We gathered data about an individual's knowledge of object-oriented and procedural software development. We expected to categorize respondents based on common threads across the learning process. Does everyone have trouble with the concept of polymorphism, or only under certain conditions? We were looking for relationships between knowledge and experience within the data. One method for understanding relationships is cluster analysis (Gordon, 1999). In this study we clustered the responses of software developers. This chapter discusses the data analysis procedure used and the findings of this study. The first section details the two primary data analysis methods used: cluster analysis and factor analysis. The second section addresses the hypotheses and reports the findings from the study. The final section of this chapter provides a summary of the findings.

Cluster Analysis

Cluster analysis refers to techniques used in the classification of similar objects into groups (Kaufman & Rousseeuw, 1990). Other names given to cluster analysis are "numerical taxonomy," "pattern analysis," and "typing" (Lorr, 1983). One must be careful to distinguish cluster analysis from discriminant analysis. Discriminant analysis is a process undertaken to differentiate between groups formed on an a priori basis. The goal of discriminant analysis is not to discover groups but to identify a set of characteristics that can significantly differentiate between the groups. With cluster analysis, the number and nature of the groups are not known in advance. The clustering process generates a classification scheme for unclassified data (Lorr, 1983). For this study, cluster analysis was deemed the appropriate analysis technique because we were dividing the data into groups based on relationships within the data and without a preconceived idea of the groupings.

Clustering techniques have several goals including: finding a typology or classification, investigation of a conceptual scheme for grouping entities, data exploration and hypothesis generation, and hypothesis testing or classification affirmation (Aldenderfer & Blashfield, 1984, p. 9). Many studies address multiple goals when using clustering techniques. In this study our goal was to develop a classification and a conceptual scheme to explain the classification.

There are five basic steps that characterize all cluster analysis studies: 1) selection of a sample to be clustered; 2) definition of a set of variables that measure the respondents; 3) computation of similarities among the respondents; 4) creation of groups; and 5) validation of the resulting clusters (Aldenderfer & Blashfield, 1984).

Sample Selection

As described in Chapter Three, the sample consisted of software developers with a wide range of experience. Since we were interested in expert procedural developers making the shift to object-oriented techniques, the sample frame population was expert procedural software developers. If the participant had less than 4 years of procedural software development experience they were omitted from the sample. Four years was selected as the cutoff point based on previous work in this area (Manns & Nelson, 1996). A total of 148 responses were originally recorded with 17 being eliminated due to the respondents' lack of software development experience. This left a sample of 131 respondents.

Variable Definition

Statistical analysis began by examining the histograms and scale reliabilities of the relevant variables from the field study. Histograms and scatterplots were used to examine the data visually at the item and concept level. Findings were consistent with pilot study results and indicated that the items were detecting differences in the individual's mindsets. All statistical computations were performed using SPSS v 10.0. Two-tailed tests of significance were used in all data analysis (Siegel, 1956, p. 13; Huck, 2000).

Alpha coefficient ranges in value from 0 to 1 and describe the reliability of factors extracted from scales. The higher the score, the more reliable the generated scale. A generally acceptable reliability coefficient for an established scale is 0.7 (Nunnally & Bernstein, 1994, p. 265), but a lower threshold of 0.6 can be used in the development of a new measurement instrument (Nunnally, 1967). For concept level Cronbach's coefficient alpha see Appendix C. All concept level scale reliabilities were above .60 with the exception of the "object" concept. The reliability for the object scale was = .2730. There were 2 items in the "object" scale. Upon further analysis, one of the two items loaded onto the "attribute" concept. We determined that the object scale was not measuring the "object" concept and was therefore not used in any further data analysis. This left 33 concepts in the instrument.

Existing theory should be used to guide the choice of appropriate variables. As there was no existing theory to draw on, we used the Expert Classification Scheme developed in Phase I to guide variable selection. The 33 concepts on the instrument were measured using one to four items (questions). Each concept variable was scaled using the mean of the items before clustering. It is common for researchers to attempt cluster analysis around too many variables (Aldenderfer & Blashfield, 1984; Everitt, 1980; Lorr, 1983). One procedure for overcoming this difficulty is to perform a principal components factor analysis on the data (Everitt, 1980). This reduces the number of variables to a parsimonious set that can be used in the analysis. The greater the number of positively correlated measures that are combined into a summary score, the more reliable the composite (Cronbach & Gleser, 1953). The construct level of analysis was deemed more appropriate for this aspect of the research and a factor analysis was conducted on the data.

Factor analysis consists of a collection of procedures for analyzing the relations among a set of random variables observed or measured for each individual in a group (Cureton & D'Agostino, 1983). The factors are random variables that cannot be observed or measured directly, but which are presumed to exist in the population and hence in the sample. The random variables of the set to be analyzed may consist of any attributes on which the members of the group differ. Factor analysis is a way of decreasing the number of variables to cluster on.

A principal components factor analysis with varimax rotation was used to determine the construct level variables. Past research has provided guidelines for the minimum sample size needed to conduct factor analysis. Some have suggested the ratio of sample size to number of variables as a criterion: the recommendations range from 2:1 to 20:1. Others have suggested using a minimum sample size as the criterion. For example, Lawley and Maxwell (1971) suggest that there should be 51 more cases than the number of variables. In their 1988 study, Guadagnoli and Velicer found that absolute sample size was more important than functions of sample size in determining stable solutions and recommend 100 to 200 observations. Since our sample of 131 respondents fell within the acceptable range, all 33 concept level variables were included in the factor analysis. Table 8 lists the four constructs revealed in the data.

The first factor (construct) was the labeled the "Basic Level" construct. The title "Basic Level" was chosen because understanding the concept of an object and that everything is an object is fundamental to understanding object-oriented techniques. The "Basic Level" construct is comprised of introductory concepts, the first basic ideas that are introduced to individuals as they learn about object-oriented techniques. The Cronbach's coefficient alpha for this factor was .8949.

The second construct was labeled the "Object Level" construct. The title "Object Level" was chosen because these concepts primarily focus on the development and functioning of an object. The Cronbach's coefficient alpha for this factor was .9534. The third construct was labeled the "System Level" construct. The title "System Level" was chosen because these concepts focus on how objects function within the larger system. The Cronbach's coefficient alpha for this factor was .9252. The final construct was labeled the "Procedural" construct. This construct included all of the procedurally oriented concepts. The Cronbach's coefficient alpha for this factor was .9580.

The conceptual scheme developed from the instrument data was significantly different than the structure developed from the interview data. The classification scheme developed in Phase I was based on data from two distinct groups, expert procedural and expert object-oriented software developers. The conceptual scheme developed in Phase III was based on data from expert procedural software developers learning object-oriented techniques. These individuals were at various stages in the learning process and thus their conceptual scheme reflected the learning process. The structure found in Table 8 was utilized for the analysis because it was consistent with the data in this phase of the study and with structures found in previous research (Nelson et al., 1997).

Similarity Measures

The similarity between respondents can be decomposed into three parts: shape, the pattern of dips and rises across the variables; scatter, the dispersion of the scores around their average; and elevation, the mean score of the case over all of the variables (Aldenderffer & Blashfield, 1984). There are four types of similarity measures: distance, correlation coefficients, association coefficients and probabilistic similarity coefficients. The similarity measures address the three components (shape, scatter and elevation) differently. Correlation and distance measures are the common choices in the social sciences. With a distance measure the two cases are identical if each one is described by variables with the same magnitudes (0 distance). The most popular distance measure is the Euclidean distance or Squared Euclidean distance. The disadvantage of using a distance measure is that the similarity estimation is strongly affected by elevation differences. A correlation coefficient (the most popular is the product-moment correlation coefficient) is used to determine the correlation between respondents. The disadvantage of using a correlation measure is its sensitivity to shape at the expense of the magnitude of differences between the variables. Both the association correlation and probabilistic measures are used with binary data.

If the similarity is assessed using a distance function, all of the information is preserved. If the raw scores are converted to deviation scores then the source of information is lost. If the similarity is assessed using a correlation coefficient, the information regarding the elevation and scatter are lost. Thus the Squared Euclidean distance similarity measure was used to retain as much information as possible.

Clustering Method

There are many clustering methods available (e.g. hierarchical, partitioning, factor analytic, density, and clumping), but the two predominant forms of cluster analysis are partitioning and hierarchical. With the partitioning method a single partition is constructed with *k* clusters. With the hierarchical method the hierarchy is a nested set of non-overlapping clusters in which each level is assigned a rank (Lorr, 1983, p. 19). The hierarchical method is preferred when there is structure in the data, or a developmental sequence. It is also used when the number of groups in the data is unknown. A disadvantage is that hierarchical methods make only one pass through the data, and a poor early partition of the data cannot be modified later. The partitioning method's advantage is its iterative property. Poor initial cluster selection can be overcome through the clustering process. To draw on the strengths (and diminish the weaknesses) of both methods we used both hierarchical and partitioning methods. We used the hierarchical method initially because we did not know the number of groups a priori. From the results, the number of groups and outliers were identified. After the identification and subsequent elimination of outliers, the cluster analysis was rerun using a partitioning method (Aldenderfer & Blashfield, 1984, p. 61; Everitt, 1980, p. 103).

Within the hierarchical methods there are two ways to cluster the cases: agglomerative and divisive. The agglomerative method starts with each respondent as a cluster. Then in each step two clusters are merged until only one is left. The divisive method begins with all participants in one cluster. In each following step the cluster is split. The division continues until each participant is a cluster. We used the agglomerative method because of its wide usage in the behavioral sciences.

Another decision that must be made with regard to cluster analysis is how the agglomeration is accomplished. There are several choices including between-groups linkage, within-groups linkage, single linkage, complete linkage, centroid, median clustering and Ward's method. The four most commonly used in the social sciences are single linkage, complete linkage, average linkage and Ward's method (Aldenderffer & Blashfield, 1984).

Single linkage (nearest neighbor) begins by searching for the two most similar entities in the matrix. It then joins the entities that have the two most similar individual points. Only a "single link" is required between two entities for them to merge. Two common problems in cluster analysis are chaining and reversals. Chaining occurs where single samples join a larger cluster each time. This causes ordination, and no true hierarchical structure appears. Reversals are caused when an entity joins another cluster at a higher level of similarity than was there before. One of the main drawbacks to the single linkage method is its propensity to chain. Complete linkage (furthest neighbor) adds an entity to an existing cluster if that entity has a certain level of similarity to all members of the cluster (Sokal & Michener, 1958).

Average linkage computes an average of the similarity of an entity under consideration with all entities in the existing cluster and joins the entity to the cluster if a given level of similarity is achieved using this average value (Aldenderffer & Blashfield, 1984). Two common variants of the average linkage method are the median and centroid (distance between groups is distance between group centroids) clustering. The drawback of the centroid method is that if you are merging two groups of disparate size, the larger group will have a large impact on the location of the new centroid. The advantage of median clustering is there is no impact if the groups being merged are of unequal size.

Ward's method is based on optimizing the minimum variance within clusters (Ward, 1963). Ward's method is also known as the within-groups sum of squares or error sum of squares (ESS) method. The method works by joining those entities or groups that result in the minimum increase in the ESS.

One way to compare hierarchical clustering methods is to analyze how these methods transform the relationships between the points (Aldenderfer & Blashfield, 1984, p. 44). Space contracting methods affect these relationships by reducing the space between groups in the data. New points tend to be joined to existing groups. With space dilating methods, new points tend to form new groups. Thus smaller, more distinct clusters are formed. This also tends to create clusters of roughly equivalent sizes and shapes. The space dilating methods are seen as superior to space contracting (Aldenderfer & Blashfield, 1984; Williams, Lance, Dale, and Clifford, 1971). Complete linkage and Ward's method are space-dilating methods. For this study we chose the Ward's method (hierarchical) to determine the number of clusters. This selection was made because it is a space dilating method that is widely used in social sciences (Blashfield, 1980).

The primary goal of this step was to determine the optimal number of groups in the data. A common problem to all clustering techniques is the difficulty in deciding the number of clusters present in the data (Everitt, 1980, p. 64). Heuristic procedures were applied to determine the number of clusters present. Using the dendrogram the hierarchical tree is inspected for different levels of agglomeration. See Appendix G or the Dendrogram. In addition, the number of clusters implied by the tree was compared against the agglomeration coefficient. The agglomeration coefficient is the numerical value at which various cases merge to form a cluster. A marked jump in the agglomeration coefficient suggests that no new information is portrayed by the further merger of clusters (Aldenderfer & Blashfield, 1984). See Appendix H for the Agglomeration Schedule. Note the jump in coefficient from three clusters (227.383) to two clusters (313.476). From the hierarchical clustering dendrogram and analysis of the agglomeration schedule, three clusters were observed and confirmed. A *k*-means cluster analysis was then performed using three clusters. The participants were clustered based on the construct level variables. Table 9 presents the means for the three clusters. The cluster centers for each construct are plotted on the Figure 6 graph by construct.

We examined the means for each cluster on each dimension to assess how distinct our three clusters were. Ideally, we would obtain very different means for most, if not all dimensions, used in the analysis. Comparisons of the cluster means were conducted using t-tests. The test was conducted to confirm the uniqueness of each cluster. As seen in Table 10, the clusters differed significantly on all construct level means. Thus each of the three clusters was distinct.

Table 11 presented the ANOVA data for the study. The magnitude of the *F* values from the analysis of variance performed on each dimension is another indication of how well the respective dimension discriminates between clusters.

The cluster analysis provided the location of the respondents on the Y-axis. We then needed to place the respondents on the X-axis. The X-axis was a measure of object-oriented experience. Past research has used several approximations for measuring software development experience. One method was to have the researcher categorize the participant based on language usage (e.g. Detienne, 1995). One problem with this method is the arbitrary nature of group assignment. For example in the Detienne (1995) study, individuals were categorized as beginners and experienced based on how "frequently" they used the software development language. Scholtz and Wiedenbeck (1990) defined experienced programmers as either professional programmers or advanced graduate students. Schenk, Vitalari, and Davis (1998) defined experienced analysts based on supervisor performance ratings. Shneiderman (1976) used a student sample and defined "advanced" developers as graduate students or faculty.

Other studies used the number of years of an individual has developed software as a measure of software development experience (e.g. Lee & Pennington, 1994; Manns & Nelson, 1996). For example in their study, Manns and Nelson (1996) defined a professional programmer as one with "3-10 years of experience." Campbell, Brown, and DiBello (1992) used the term professional programmers to describe a developer with at least five years professional experience.

There are a few studies that have attempted to use multiple factors to determine an individual's level of object-oriented experience. Harrel and McLean (1985) determined a developer's level of experience using a questionnaire that elicited information on the individual's years of formal training, years of experience, and the number of programs written. Liu, Goetze, and Glynn (1992) measured experience in terms of lifetime lines of code, most recent programming experience (months ago), largest program (lines of code), largest number of weeks spent writing a program, and the number of languages known. One limitation of this study was the use of a student sample. Pennington (1987) measured: programming languages known, number of programming courses taken, years as a professional programmer, number of hours spent professionally programming (coding, debugging and maintenance), if taught programming course. Drawing on this work and personal observation, we believed that object-oriented experience would be indicated by multiple factors. We collected information on the following variables:

Age

Gender

Formal Education

Organizational Tenure

Job Tenure

Job Description

Industry

Methods of learning software development

Years of Procedural Experience

Number of Procedural Projects

Years object-oriented Experience

Number of object-oriented Projects

Number of software development languages used in job

Number of professional organizations member

Average length of projects

Average cost of projects

The only variables that had significant correlations with the construct level factors (Basic, Object, System, Procedural) were the years of object-oriented experience (0.538, p= .000) and number of object-oriented projects (.627, p = .000).

The correlation between the years of object-oriented experience and number of object-oriented projects was .790 and is graphically represented in Figure 7. While that correlation was high, the analysis needed to be conducted using both variables. The Basic Level construct was linked primarily to the years of object-oriented experience, the Object Level construct was linked with both years of experience and number of object-oriented projects, and the System Level construct was primarily linked with the number of object-oriented projects. Therefore we conducted the analysis using years of object-oriented experience and number of object-oriented projects separately as the X-axis.

Hypotheses

One way to analyze the learning process is through a graphical representation of the data. We examined the learning curve to see where in the process software developers were getting stuck in the transition from procedural to object-oriented techniques. We wanted to measure object-oriented knowledge at different points in the learning process. To do this we needed to determine the software developer's level of object-oriented knowledge and their level of object-oriented experience. The Software Development and Maintenance Approach instrument measured variables that indicated the individual's level of object-oriented knowledge, and demographic information that indicated the individual's level of object-oriented experience. Therefore, the Y-axis represented the level of object-oriented knowledge and the X-axis the amount of object-oriented experience.

Hypothesis 1

Hypothesis 1 stated, "*The learning curve for procedural software development experts learning object-oriented techniques will include learning plateaus.*" To address this hypothesis we first conducted a visual inspection of the data using scatterplots. Scatterplots highlight the relationship between variables by plotting the actual values along two axes. Scatterplots reveal relationships, such as a curvilinear pattern, that descriptive statistics do not reveal. We created three sets of graphs, one for each object-oriented construct (See Figures 8, 9 and 10). There are two graphs per construct, the top one showing the object-oriented experience time component on the X-axis, and the bottom showing the number of object-oriented projects on the X-axis. SPSS allows you to select an interpolation method for connecting the data points in a scatterplot. Once the method is selected the kernel and bandwidth must be set. The kernel determines the smoothness of the curve. The kernel options (from most smooth to least smooth) are normal, Epanechnikov and uniform. (For more information, see Simonoff, Jeffrey S., *Smoothing Methods in Statistics*, 1996, New York: Springer-Verlag.) The bandwidth multiplier changes the amount of data that is included in each calculation of a small part of the smoother. The multiplier can be adjusted from 0 to 10 to emphasize specific features of the plot that are of interest. The larger the multiplier, the smoother the curve. The default size of the bandwidth is one. The fit line in the graphs below is a smoother line (local linear regression) with a normal kernel (default) and a bandwidth of one (default). The defaults were used because they best represented the data without "over-smoothing".

From the scatterplots in Figure 8, two plateaus can be seen in the top graph, with "years of OO experience" on the X-axis. This indicates that for the Basic Level factor (consisting of two variables) there are two plateaus or slowing points in the learning across time. In the lower graph, with "number of object-oriented projects" on the X-axis, only one plateau is seen. In Figure 9, one rather large plateau occurred for the Object Level construct with both years and number of projects on the X-axis. In Figure 10, two plateaus can be seen with both "years of object-oriented experience" and "number of object-oriented projects" on the X-axis. This indicates that for the System Level factor (consisting of 9 variables) there are two plateaus in the learning.

The shape of the line was then statistically tested using bivarite linear and nonlinear regression. We compared the "goodness of fit" for three regression models, a linear, quadratic and curvilinear model. The comparison is commonly made using the "r," "r^2" and "Se" statistics. The "r" quantifies the degree to which the predicted scores match up with the actual scores (Huck, 2000, p. 576). The "r^2" is the coefficient of determination, which indicates the proportion of variability in the dependent variable that is "explained" by the independent variable (Huck, 2000, p. 576). When measuring the strength of relationship between independent variables the preferred measure is the r^2 rather than r (Lewis-Beck, 1980). The standard error (Se) is an estimate of the standard deviation of the slope estimate (Lewis-Beck, 1980). The models used in the analysis were:

Model	**Equation**
Linear	Y = b0 + b1*x +e
Quadratic	Y = b0 + b1*x2 + b12*x + e
Curvilinear	Y = b0 + b1 * x + b01 * d1 + b11 * d1 * x + b02 * d2 + b12 * d2 * x + e

where b0-b02 were the y-intercept parameters, b1-b12 were the slope parameters, c is the constant, d1 and d2 were variables used to distinguish the clusters, and x was the independent variable (years of OO experience or number of OO projects). The cluster variables (d1 and d2) were assigned as follows: cluster 1 d1 = 0, d2 = 0; cluster 2 d1 = 1, d2 = 0; and cluster 3 d1=0, d2 = 1. For example, using the Basic Level construct as the dependent variable, and "years of object-oriented experience" as the independent variable, the equations would read:

Linear Basic Level =

2.591 + .107 * years of object-oriented experience + .211

Quadratic Basic Level =

2.467 + (-.007)*(years of object-oriented experience)^2 + .183 * years of object-oriented experience + .329

Curvilinear Basic Level =

3.431 + (-.266) * (years of object-oriented experience) + (-2.508) * d1 + (.350) * d1 * (years of object-oriented experience) + .375 * d2 + .277 * d2 * (years of object-oriented experience) + .860

Table 12 presents the "r^2" information and Table 13 presents the standard error (Se) information for the linear, quadratic and curvilinear models.

In the curvilinear regression the r^2 was higher than the quadratic regression r^2 and the linear regression r^2. This indicates that with the curvilinear model, more of the variability in the dependent variable was being "explained" by the independent variable than with the linear or quadratic models. In addition, the amount of error

(Se) was consistently lower in the curvilinear model than the linear or quadratic models. Combining the scatterplot and the regression data indicates that a curvilinear model better fits the data. As a result, Hypothesis 1 is supported.

Hypothesis 2

Hypothesis 2 stated, "*Proactive interference is positively associated with learning plateaus.*" Proactive interference occurs when previously learned information interferes with the assimilation of new knowledge. In the case of expert procedural software developers learning object-oriented techniques, proactive interference occurs when an individual's existing knowledge of a procedural concept interferes with learning the object-oriented concept. This interference can be assessed by comparing antagonistic concepts. For example, in terms of tennis and racquetball the concept of "forehand" is antagonistic. Someone who is thinking from the tennis mindset will conceptualize the forehand as a swing dominated by the shoulder, and from the racquetball mindset will conceptualize the forehand as a swing dominated by the wrist. Applying the theory of proactive interference to the Software Development and Maintenance Approach instrument, if an individual scores high on an object-oriented concept and low on the antagonistic procedural concept, then she is thinking in the object-oriented mindset. If the individual scores similarly on both the object-oriented and procedural concepts then she is experiencing confusion and proactive interference. When we analyzed the sample, there were individuals experiencing "confusion". We observed a pattern to this confusion across the sample. As the level of object-oriented experience (as expressed by years of object-oriented experience and number of object-oriented projects) increased the response pattern changed.

For example, Table 14 shows the concepts of "class" and "subroutine" as antagonistic. As an individual's level of object-oriented experience increases the response to the "class" statements increase (1=strongly disagree to 5=strongly agree). In contrast, as the individual's level of object-oriented experience increases the response to the subroutine statements decreases. Looking at the "class" concept there was a steady increase in the level of response for response choices 1.00 to 3.00. This coincides with the increase in the years of object-oriented experience and number of object-oriented projects. As the response choice moves from 3.00 to 3.33 the object-oriented experience level makes a drastic increase from 2 to 4 years and the number of object-oriented projects jumps from 1 to 3 projects. After that point, the response choices and object-oriented experience steadily increased.

Looking at the "subroutine" concept there was a steady decline for response choices 1.00 to 2.00. This coincides with the increase in the years of object-oriented experience and number of object-oriented projects. As the response choice moves from 2.50 to 4.00 the object-oriented experience level and the number of object-oriented projects hovers around 3. After that point, the response choices and object-oriented experience become steady again.

Similar results were found for the following concepts:

Procedural

Data Model

Interaction

Monolithic

Functional Decomposition

Functions

Object-Oriented

Object Model

Interaction

Layer

Noun-Verb Analysis

Things as Objects

Figures 11 – 16 graphically represent the interference between the respective concepts for both years of object-oriented experience (top graph) and number of object-oriented projects (lower graph). The crossing lines of the graph demonstrate the interference. From the graphs (Figures 11-16), it can be seen that the knowledge of the procedural concepts interferes or stagnates the learning of the object-oriented concepts. As a result, Hypothesis 2 is supported.

Hypothesis 3

Hypothesis three stated, "*Learning plateau(s) will occur at certain levels of object-oriented experience.*" Looking at the graphs we see that the plateaus occur at different locations depending on the construct. Plateaus occur on the Basic Level construct when compared to the years of object-oriented experience and the number of object-oriented projects. Two plateaus can be observed on the Basic

Level construct versus years of object-oriented experience graph (see Figure 8). One longer plateau can be seen on the Basic Level construct versus number of object-oriented projects graph (see Figure 8).

The reason the first plateau occurs on the Basic Level construct versus years of object-oriented experience graph was because of the proactive interference between "function" and "things as objects". See Figure 17A. Based on the detailed top graph it can be seen that this interference occurs between 1.25 and 3.67 years of object-oriented experience. The crossing of the two lines and the proximity of the lines during the time period 1.25 to 3.67 years of object-oriented experience demonstrates the interference. Based on the detailed bottom graph it can be seen that interference occurs between 2 and just over 6 object-oriented projects. The individual moves out of the plateau when the interference was no longer present.

The reason the second plateau occurs on the Basic Level construct was because of the proactive interference of "function" and "converting things into objects". See Figure 17B. Based on the graph it can be seen that this interference occurs between 5.25 and 7.5 years of object-oriented experience.

Three plateaus occur on the System level factor based on the number of object-oriented projects experienced. See Figure 18. A plateau occurs between 2.00 and 3.25 projects, another plateau occurs between 4.25 and 5.50 projects, and the final plateau occurs between 7.25 and 8.00 projects.

The reason the first two plateaus occur on the System construct was because of the proactive interference between the Procedural and System Level concepts.

One of those interference points was between "abstraction" and "data modification." The plateaus correspond to the solid line decreasing while the dashed line was above it. The periods of learning correspond to increases in the solid line. These are periods in which an individual's knowledge of the object-oriented mindset is increasing. Based on the graph it can be seen that this interference occurs between 2.00 and 3.25 object-oriented projects and again from 4.25 to 5.50 projects. The proactive interference also occurs between the "object-oriented interaction" and "input-process-output," and "object model" and "data model" concepts. See Figures 19 and 20.

As the software developer makes the transition to object-oriented techniques he or she experiences periods of learning and periods of stagnation. The periods of learning are represented in the graph as the positively sloped portion of the curve. The periods of stagnation are represented as plateaus. The proactive interference can also be represented as the intersection of antagonistic concepts. As the previously learned procedural information interferes with the assimilation of the new information the plateaus occur. When the individual is no longer confused by the procedural concept the interference ceases. At that time the plateau ends and the individual continues learning as demonstrated by the positive sloping learning curve. As a result, Hypothesis 3 is supported.

Hypothesis 4

Hypothesis four stated, "*Learning plateau(s) will cluster around certain object-oriented concepts.*" When analyzing this hypothesis with regard to the years of object-oriented experience, there were twelve concepts that exhibited multiple plateaus. Of these two were from the Basic Level construct, three from the Object Level, and seven from the System Level construct. See Table 15. The plateaus occurred at the same level of years of experience for all of the constructs with two plateaus. For example, Figure 21 reveals two plateaus for the "converting things into objects" construct when measured against years of object-oriented experience. If you overlaid the graphs of the other eleven multiple plateau concepts the plateaus would occur at approximately the same location on the X-axis (years of OO experience).

When analyzing this hypothesis with regard to the number of object-oriented projects, there were ten concepts that exhibited multiple plateaus. Of these one was from the Basic Level construct, three from the Object Level, and six from the System Level construct. See Table 16. The plateaus for "abstraction" and "converting things into objects" occurred at the same location along the x-axis. For example, Figure 22 shows two plateaus for the "abstraction" construct measured against the number of object-oriented projects. If the graph of the "converting things into objects" concept was overlaid, the plateaus would occur at approximately the same location on the X-axis (number of OO projects).

The plateaus for "inheritance," "interaction," "message passing," "oo development," "relationships," and "components," occurred at the same location. Figure 23 shows two plateaus for the "message passing" construct when measured against the number of object-oriented projects. If you overlaid the graphs of the other five multiple plateau concepts the plateaus would occur at approximately the same location on the X-axis (number of OO projects).

Multiple plateaus on both X-axis (years and projects) occurred for seven concepts: "abstraction," "components," "converting things into objects," "interaction," "layer," "message passing," and "noun-verb analysis." As a result, Hypothesis 4 is supported.

"Don't Know" Analysis

As detailed previously, because the instrument was anchored in the object-oriented perspective, a "don't know" response choice was included. This option was included to give respondents who were not anchored in the object-oriented perspective an option to reflect their lack of knowledge (Schuman & Presser, 1981, p.170). We felt that capturing an admitted lack of knowledge was as important as capturing the level of knowledge. When a respondent selected "Don't Know, no knowledge or experience in this area" he or she was not just stating that they were neutral about the topic, but specifically stating they had no knowledge of the topic. If he or she was neutral about the topic, they would have most likely selected the "neither agree nor disagree" option.

The "don't know" responses were not included in the primary analysis. A separate SPSS file was created from the original data so as to analyze the "don't know" response choice. This file was transformed to reflect the comparison of answering "know" versus "don't know". The "don't know" response was coded 1 and the "know" (any other response choice) was coded a 0. The data was examined with regard to the frequency of the "Don't Know" response. The largest percentage of "Don't Know" responses (over 20%) occurred at the "Object Level" and "System Level" constructs. See Table 17.

It can be seen in Figures 24 through 27 that the "don't know" scatterplots are the reverse image of the full data scatterplots for the constructs. The plateaus occur at the same points on the X-axis in the graph. For example, in the Basic Level versus years of object-oriented experience graph (Figure 24, top graph) there was a plateau between 1.5 and 3.5 years of object-oriented experience. There was a second plateau between 6 and 10 years of experience. Looking at Figure 27, you will see that the plateaus coincide with the Basic Level versus years of object-oriented experience graph for the primary analysis (same as the lower graph of Figure 8). The "don't know" analysis demonstrates proactive interference from another angle. Software developers that selected the "don't know" response were experiencing proactive interference (as demonstrated by plateaus) similar to the developers that selected a "know" response. This affirms the existence and location of the plateaus.

Summary of Chapter Four

This chapter discussed the data analysis procedure used and reported the findings of this study. The first section detailed the two primary data analysis methods used: cluster analysis and factor analysis. Because we wanted to find groups within the data, cluster analysis was the primary method of data analysis. The clusters were formed based on responses to the software development and maintenance approach items. After the respondents were placed into a cluster, then the membership of the cluster was analyzed. We looked for correlations or patterns between the demographics of the groups and how they answered the questions.

This information was then used to describe the group's movement through the learning process. Combining the respondent's object-oriented knowledge with their demographic information allowed us to graphically represent the transition from procedural to object-oriented software development. The level of object-oriented knowledge placed the cluster on the Y-axis and the demographic information determined their location on the X-axis. The second section addressed the hypotheses and reported the findings from the study. Factor analysis and regression were used to support the cluster analysis and the hypotheses. All four of the hypothesized relationships were supported. Thus, the learning curve for procedural software development experts learning object-oriented techniques included learning plateaus. Also, proactive interference was positively associated with those learning plateaus. And, lastly, the plateaus occurred at certain levels of object-oriented experience and clustered around certain object-oriented concepts. See Table 18 for a summary of the findings for each hypothesis. The next chapter discusses these results in greater detail.

CHAPTER FIVE

Discussion

This study asked the questions, "Why is it difficult for procedural experts to learn object-oriented development? And "Where in the learning process are developers experiencing difficulty?" Our findings indicate that proactive interference was one factor contributing to the difficulty in making the transition. An individual's experience in procedural software development interferes with learning object-oriented techniques. Our findings also indicate that software developers experience difficulty at several points in the learning process.

Other interesting findings were uncovered in the data analysis process. One of the most significant findings was the identification of the object-oriented concepts on which learners experience proactive interference. Another finding was the development of two distinct schemes. The data from Phase I support an expert classification scheme that reflects a software developer's thinking once he or she is an expert. In contrast, the data from Phase III support a learning conceptual scheme that reflects the learning process as software developers transition to object-oriented techniques. Another interesting finding was the discovery of three clusters in the data. The clusters could be categorized as novice object-oriented developers, transitional developers and experienced object-oriented developers. This chapter details the research findings, theoretical and managerial implications and limitations of this study. Future directions and a summary are provided at the end of the chapter.

Research Findings

Classification Scheme

The goal of Phase I was to identify the concepts associated with procedural and object-oriented software development from an expert's perspective. Interviews were conducted with expert procedural and object-oriented software developers to gather the relevant concepts. An expert classification scheme was developed from the interviews and provided seven constructs for object-oriented development and six for procedural development. The expert classification scheme identified the conceptual end points of the learning process and reflected the states (design, execution) and the components (characteristics, relationships) of software development.

The Phase III data identified a learning conceptual scheme containing three object-oriented and one procedural construct. The goal of Phase III was to document the learning process as developers transition from procedural to object-oriented techniques. We wanted to elicit the progression of understanding and identify where developers were experiencing difficulty making the transition. The learning conceptual scheme developed in Phase III was based on data from expert procedural software developers learning object-oriented techniques. These individuals were at various stages in the learning process and thus their learning conceptual scheme reflected the process. The Phase III data produced a learning conceptual scheme that was grounded in the progression of understanding.

Each construct in the Phase III learning conceptual scheme reflected a stage of learning. The Basic Level construct can characterized as introductory concepts, the first ideas that are introduced to individuals as they learn object-oriented techniques. They are also the commonly used "buzzwords" that individuals could use without really understanding the underlying concepts. The concepts from the Object Level construct are related to an early stage of software development learning. These concepts primarily focus on the development of an object and are more micro-focused. The concepts from the "System Level" construct are related to a later stage in the software development learning process. These concepts focus on how the objects function within the larger system. Refer to Tables 5 and 8 for the expert classification and learning conceptual schemes.

The learning conceptual scheme reflects the thought processes of software developers as they learn object-oriented techniques. The learning conceptual scheme helps pinpoint what concepts are more difficult for the learners. For example, from this scheme we can see that the individuals who understood the concept of "abstraction" were more experienced object-oriented developers, whereas most individuals understood the concept of "things as objects" regardless of their experience. This conceptual scheme reflects the progression of learning. As a developer is learning object-oriented techniques he or she generally connects the object-oriented concepts starting with the "Basic Level" concepts and progressing through to the "System Level" concepts.

While the learning conceptual scheme is of primary importance to this research project, the expert classification scheme also contributes to our understanding of expertise and Information Systems. We can utilize both of the schemes to paint a more complete picture of the learning process. The expert classification scheme identified the object-oriented concepts essential for a deep level of understanding and proficiency. That information was used to create the Software Development and Maintenance Approach instrument. The data from the instrument produced the learning conceptual scheme. The learning conceptual scheme identified where software developers experience difficulty in the learning process. This information can be used to improve the learning process and increase comprehension. Therefore, both conceptual schemes contribute to our understanding of software development learning and expertise.

Cluster Analysis

As previously stated, three clusters were revealed in the data. Table 9 presented the mean scores on the object-oriented and procedural constructs for each cluster. The respondents in cluster 1 were procedural thinkers. They answered the procedural questions significantly higher than they answered the object-oriented questions. When they develop software they develop from the procedural mindset, thinking procedurally as opposed to object-oriented. Looking at the demographic information associated with cluster 1, it was not surprising in light of their performance. The average years of object-oriented experience was slightly less than 2 years (1.75) and the number of object-oriented projects they had participated in was just under 1 (.952).

The respondents in cluster 2 were in transition. When looking at the mean scores, they answered questions in the middle. When the actual responses are analyzed very few respondents selected "neither agree nor disagree" (choice 3). If the item had two questions, most respondents selected 4 for one item and 2 for the other. This indicates not that they were indifferent to the concept, but confused by the concept and did not answer consistently. The fluctuating scores indicate the confusion. The respondents mean scores ended up close to neutral (3.00) on the Basic and Object Level object-oriented constructs, disagreed with the System Level construct, and slightly agreed with the procedural construct. These individuals are beginning to internalize the Basic and Object Level constructs, but not the System Level constructs. They are in the process of making the transition to object-oriented techniques. When looking at their demographics, the respondents in Group 2 have a few years of object-oriented experience (2.77) and a few projects (2.33) under their belt.

The respondents in cluster 3 were object-oriented thinkers. They answered the object-oriented questions significantly higher than the procedural questions. When the cluster 3 respondents develop software they develop from the object-oriented mindset as opposed to procedurally. Looking at the demographic information associated with cluster 3, the average years of object-oriented experience was slightly over 5 years (5.086) and the number of object-oriented projects they had participated in was about 4.5 (4.484). Cluster 3 participants were experienced in object-oriented software development techniques and use that knowledge to develop software.

Hypotheses

The scatterplot and regression analysis performed supports the theory that the learning curves for procedural software development experts learning object-oriented techniques exhibit plateaus. At the construct level, plateaus were observed for all three constructs. Multiple plateaus were observed on the Basic and System Level constructs. The plateaus on the Basic Level were most pronounced when viewed with respect to the individual's years of object-oriented experience. The plateaus on the Object Level construct were mixed. The plateaus on the System Level were most pronounced when viewed with respect to the individual's quantity of object-oriented projects. This indicates that as an individual moves through the learning process the "time" aspect becomes less predictive than the number of experiences.

On the Basic Level construct two plateaus occurred. One plateau occurred at the beginning of the learning process (1.5-3.5 years of experience) and another toward the end of the process (6.0-8.0 years of experience). This makes sense because as a software developer learns object-oriented techniques he or she becomes comfortable with the surface level concept of an object and can use it in their applications. As the developer deepens his or her understanding of the "object" the slope of the learning curve increases. The learning stalls again as the developer struggles with the complexity of the object as it is used within the larger context.

On the System Level construct three plateaus occurred. These plateaus occurred with regard to the number of object-oriented projects the developer had completed. The first plateau occurred between 2 and 3 projects, the second between 4 and 5 and the third between 7 and 8 projects. The first 2 plateaus coincide with the cluster breakdowns. Cluster 2 has an average number of object-oriented projects of 2.33. Cluster 3 has an average number of object-oriented projects at 4.48. One possible explanation for this finding is that as individuals move to a new phase in the learning they experience a plateau. Once they break the plateau (the interference) they move up the curve and into the next cluster.

Not only do these plateaus occur at specific times, they occur around specific concepts. At the construct level the Basic Level construct exhibited multiple plateaus with regard to the years of object-oriented experience, and a single plateau with regard to number of projects. The Object Level construct exhibited a single plateau with regard to both years of experience and number of projects. The System Level construct exhibited multiple plateaus with regard to the number of projects. Again, the plateaus and thus the learning appear to be linked with the time component early in the learning process and with the practice component later in the learning process.

Another finding is the location of the single and multiple plateaus at the concept level. The concepts that contained multiple and single plateaus are listed in

Table 19. Looking at the table the things as objects, collaboration, method, information hiding, and object model concepts revealed multiple plateaus when using years of OO experience as the X-axis. The inheritance, OO development, and relationship concepts revealed multiple plateaus when using number of OO projects as the X-axis. The converting things into objects, noun-verb analysis, abstraction, components, interaction, layer and message passing concepts revealed multiple plateaus when using either X-axis (years or projects). Of those seven concepts, 5 are members of the System Level construct. Thus when learning object-oriented techniques it is at the System Level where learners experience the largest proportion of interference as demonstrated by multiple plateaus in the learning curve.

When looking at the location of the single plateaus the pattern is reversed. The attribute, class, instantiation, OO development, information hiding, object model and relationship concepts revealed single plateaus when using years of OO experience as the X-axis. The things as objects and polymorphism concepts revealed single plateaus when using the number of OO projects as the X-axis. The encapsulation concept was the only one to reveal a single plateau when using either X-axis (years or projects). Referring to Table 19 the majority of the multiple plateaus occurred on concepts included in the System Level construct, and the majority of the single plateaus occurred on concepts included in the Object Level construct. In fact, all of the concepts that had *only* single plateaus were concepts included in the Object Level construct (attribute, class, encapsulation, instantiation, polymorphism). The other two concepts that had single plateaus had multiple plateaus on the other X-axis. The things as objects concept (from the Basic Level construct) had multiple plateaus when using years of OO experience as the X-axis, and a single plateau when using the number of OO projects as the X-axis. The relationship concept (from the System Level construct) had multiple plateaus when using number of OO projects as the X-axis, and a single plateau when using the years of OO experience as the X-axis. This finding is consistent with our assertion that the plateaus and thus the learning appear to be linked with the time component early in the learning process (Basic Level construct) and with the practice component later in the learning process (System Level construct).

As previously stated, during the learning process the learner may attempt to map knowledge from familiar domains (procedural software development mindset) to the new domain (object-oriented development mindset). When an unfamiliar event is introduced, the learner activates the schema that is perceived to most closely match the event. The new information is compared against existing schema and either refines the existing knowledge or creates a new schema. With the introduction of object-oriented methods, the learner may activate the procedural software development schema. Unfortunately, much of the new information is inconsistent with the active schema. Eventually a new schema will be created, but during the learning process the learner attempts to map the new knowledge onto the old schema. This causes the learner to create an incorrect analogy and experience proactive interference.

Proactive interference is reflected in the scores of the individual on the antagonistic concepts. If an individual scores high on an object-oriented concept and low on the antagonistic procedural concept, then he or she is thinking in the object-oriented mindset. If the individual scores similarly (or erratically) on both the object-oriented and procedural concepts then he or she is experiencing confusion and proactive interference. A graphical analysis was used to discover which concepts were incorrectly mapping to each other. We used a two variable line graph to observe where the lines of the two concepts cross. The crossing points demonstrate the incorrect mapping and proactive interference. See Figure 16 for an example.

In this research five of the concepts contained *only* single plateaus. All of the concepts with only single plateaus were elements of the Object Level construct. For the object-oriented attribute concept the data suggest that respondents were mapping to the procedural function concept. For the class concept the data suggest that respondents were mapping to the subroutine concept. For the encapsulation concept the data suggest that respondents were mapping to the functional decomposition concept. For the instantiation concept the data suggest that respondents were mapping to the function concept. For the polymorphism concept the data suggest that respondents were mapping to the data modification and functional decomposition concepts. Thus the respondents were experiencing proactive interference once during the learning process with regard to the five Object Level concepts listed. The data suggest that within the Object Level construct the two primary causes of interference for these concepts was the function and functional decomposition concepts.

Referring to Table 19, the majority of the concepts that contained multiple plateaus were System Level concepts (eight of the fifteen). It is interesting, but not surprising, that the majority of the proactive interference occurred at the System Level construct. Eight of the nine concepts included in the System Level construct experienced multiple plateaus and proactive interference. That data suggest that respondents were mapping the procedural function concept to both the converting things into objects and the things as objects concepts. Since the Basic Level construct is comprised of only these two concepts we can state that the interference experienced when learning the Basic Level constructs is related to the functional nature of the procedural mindset.

Within the Object Level construct the data also suggest that respondents were thinking more functionally. Respondents mapped the procedural function concept to the collaboration, inheritance, and method concepts. For the noun-verb analysis concept the data suggest that respondents were mapping to the functional decomposition concept. For the OO development concept the data suggest that respondents were mapping to the data model, functional decomposition, and data modification concepts. Again, we can see that the interference experienced when learning the Object Level constructs is related to the functional nature of the procedural mindset.

The interference experienced when learning the System Level constructs is located in a different portion of the procedural mindset. For the abstraction concept the data suggest that respondents were mapping to the data modification concept. Respondents mapped the procedural input-process-output concept to the components, information hiding and relationship concepts. For the object-oriented interaction concept the data suggest that respondents were mapping to the procedural interaction concept. For the layer concept the data suggest that respondents were mapping to the monolithic concept. For the message passing concept the data suggest that respondents were mapping to the linear form and subroutine concepts. For the object model concept the data suggest that respondents were mapping to the data model concept. From these results we can see that the interference experienced when learning the System Level concepts is related more to the movement of data within the system.

The Object Level construct focuses on the development and functioning of an object. Another phrase for that focus is development in the "small," which emphasizes the creation of individual components for an application. The concepts in the Object Level construct address the smallest parts of the system where you create individual classes and methods for an application. From the data we can see that within the Object Level construct the proactive interference primarily arises from the functional nature of the procedural mindset. This is consistent with the Object Level construct's emphasis on development at the object level. So when individuals are learning the Object Level concepts they are experiencing interference from the procedural emphasis on "functions" in conflict with the object-oriented emphasis on "objects".

In contrast to the Object Level construct's emphasis on the individual object, the System Level construct focuses on how objects function within the larger system. Another phrase for that focus is development in the "large," which emphasizes how components are linked together. The concepts in this construct address finding, modifying, and assembling the classes and methods that you need to support an application as well as the interactions with the system. From the data we can see that within the System Level construct the proactive interference primarily arises from the procedural emphasis on movement of the data throughout the system versus the object-oriented emphasis on connections between objects within the system. Thus learners are experiencing proactive interference from different sources within the procedural software development mindset. They experience functionally motivated proactive interference from the procedural mindset during the learning of the development of objects (Object Level concepts). Whereas they experience more interaction oriented proactive interference when learning how the objects fit into the bigger system.

Implications

The purpose of this study was to understand the learning process experienced by expert procedural developers transitioning to object-oriented techniques and how previous knowledge interferes with the process. To date this topic has not been significantly addressed in the IS literature. There has been little or no success in developing learning theory or explaining the difficulties software developers experience as they shift their mindset. One of the strengths of this study was its approach to understanding software expertise and learning. This study has taken an evocative approach to examining learning and mindshifts that is unlike any study previously conducted. This is one of the first studies to develop theories of software development learning and empirically test those theories.

A contribution of this study is the identification of the learning plateaus. This study identified where software developers are experiencing difficulty making the transition to object-oriented techniques. We used the learning plateaus to identify the concepts around which learners experience the most proactive interference. We identified the places (concepts) where we can decrease people's incorrect mappings. Those concepts are: abstraction, collaboration, components, converting things into objects, inheritance, information hiding, interaction, layer, message passing, method, noun-verb analysis, OO development, object model, relationship, things as objects. Now that the concepts have been identified, we can re-focus our training to ease the difficulty with certain object-oriented concepts (e.g. interaction). Learning will not just be a factor of years of object-oriented experience and/or the number of object-oriented projects a developer has, but can be enhanced by targeting the object-oriented concepts identified in this study.

Another contribution of this study is the learning theory that was evoked from the data. Cognitive models of both procedural and object-oriented development expertise were developed. The models identify both the concepts inherent in each software development mindset and the structure of those concepts into constructs. These models of expert software developer cognition will add to our understanding of expertise, software development and cognition.

This study also contributes to the extension of qualitative methods in the IS research environment. The aim of qualitative inquiry is to develop a body of knowledge that describes the individual case (Lincoln & Guba, 1985). Qualitative methods like the method used in this study are especially useful because they are able to deal with the unstructured data sets found in exploratory settings. As the qualitative data is analyzed and interpreted, theories emerge from the data. However, qualitative methods are not appropriate to test emergent theory. Quantitative methods may be employed at this stage to extend the existing body of knowledge in the form of generalizations. To accomplish this the theory must be transformed into testable hypotheses and then operationalized into measurable constructs. Once this transformation is complete, the theory can be tested using quantitative methods. This study used revealed causal mapping successfully to develop theory. Once the theory was developed, cluster analysis and bivariate regression were used to test the theory. Similar techniques may be used in examining other important issues in the IS discipline that require a multimethod approach.

In addition to theoretical implications, several managerial implications can be drawn from this study. The first implication is on the training process. Limitations of knowledge organization, representation and application are major constraints for the software developer. A way to overcome these constraints is to acquire expertise. As this study indicates, under the current training conditions it takes many years and experiences in object-oriented techniques to garner expertise. Unfortunately, organizations cannot wait five years to develop object-oriented experts. They need object-oriented software developers and they need them now! The findings from this research study can aid organizations in meeting their need for immediate object-oriented software developers. By putting these findings into practice we can have an immediate impact on both academia and the software development industry.

From an academic perspective, by incorporating these findings into our software development classes we can send students into the workforce with a greater grasp of the object-oriented mindset. They will be able to function within a professional environment with an understanding not just of development of an object (programming in the small), but also how that object fits into an entire system (programming in the large). From an industry perspective, by modifying training to emphasize the concepts that have proactive interference we can decrease the length of the plateaus. For example, we know that abstraction and components are two of the object-oriented concepts on which individuals experience proactive interference. By breaking down the existing procedural schema and strategically emphasizing the antagonistic object-oriented concepts in the training process we can shorten the learning process. By shortening the learning process we will create software developers that have more expertise in less time.

Another managerial implication is the Software Development and Maintenance Approach (SDMA) instrument. The instrument measured from what mindset an individual is developing and/or maintaining software (procedurally, object-oriented, or a mixture). A graph created from the responses demonstrated the location of individuals along the learning curve. As expected individual learning difficulties clustered around a few concepts with regard to object-oriented techniques. It is quite conceivable that proactive interference might be diminished if proper guidance could be given during the learning process (Travers, 1963:287). This study identified the concepts that individuals experienced the most difficulty in learning. These "more difficult" topics may be better addressed by modifying object-oriented instructional design.

For example, we can provide training exercises to break down the procedural schemas associated with the antagonistic topics and open the learners to new information. Training can be modified to allow more discussion and practice time on the concepts that individuals experience the most proactive interference. Another change might be to create software development assignments that focus on utilizing the concepts identified as having high proactive interference. Larger scale projects could be introduced that would include the utilization of the System Level constructs. By using the information in this study we can make training faster and more productive, shorten the learning process, and ease the learners' frustrations when making the transition to object-oriented techniques. This can lead to a more effective and efficient learning process and more and better-trained object-oriented developers.

Another application of this instrument could be to measure the level and type of development expertise within an organization. Individual as well as departmental expertise could be measured using the instrument. The results of this measurement would identify current development expertise levels and aid organizations in resource allocation and planning.

This research has significant implications for managers because IS personnel are continuously required to make shifts in their mindset. We are using the mindshift from procedural to object-oriented software development as an example. The methods used in this model have the potential to be relevant for other mindshifts. When a shift occurs, experts under the previous mindset will need to learn the new information. An understanding of how previous knowledge interferes during a mindset change will aid instructional design, and facilitate learning.

Limitations

The first limitation deals with external validity. Software development learning theory has not been previously defined within the IS domain. Because of this, the results of this study need to be replicated before claims of generalizability can be made. Threats to external validity were minimized somewhat in this study by the variety of organizations and industries participating. However, we recognize that the ability to generalize these findings to all software developers and organizations may be limited due to self-selection bias, a non-random sample, and the use of an online/email data collection method.

Another limitation is the study design. This study was a cross-sectional study designed to approximate a longitudinal field study. Advances to research would be made and knowledge would be gained from taking a longitudinal approach to examining the learning process over time for specific individuals. While this would contribute to the field, the reality is that conducting a longitudinal study is unworkable. Following any one group of software developers for an extended period of time given current IS job market volatility is virtually impossible (Gerencher, 1999; Goodner, 2000).

One limitation of this study was the selection of cluster analysis as the method of statistical analysis. This method is not supported by extensive body of statistical reasoning (Aldenderfer & Blashfield, 1984, p. 14). This is a primarily exploratory method and thus generalizations should be made with care. One limitation of cluster analysis is that different clustering methods can generate different solutions to the same data set. This risk was minimized by the use of multiple clustering methods. With multiple clustering methods the data were analyzed and clustered from two perspectives (Wards hierarchical and *k*-means).

One alternative explanation could be that the anticipated learning curve (Figure 4) is the natural learning process for object-oriented techniques, and not caused by proactive interference. This explanation could have been controlled for by including individuals with only object-oriented experience (no previous procedural experience). In analyzing their learning curves if there are no plateaus, the plateaus are different, or the plateaus are in different locations along the curve than the rest of the population, then the learning curve in Figure 4 is not the natural learning process for object-oriented techniques. This would confirm the presence of proactive interference and strengthen the study. Unfortunately, with a field study and voluntary respondents, it was not possible at this time to find object-oriented software developers with no procedural experience. In addition, it is highly unlikely that we could find professional developers with only object-oriented experience. Object-oriented techniques are too new to have individuals out in the field with no previous procedural experience. The only respondents available would be those straight out of college, and this population would not meet the study criteria due to their lack of real world experience.

Another alternative explanation is that object-oriented software development is more difficult to learn because it is more complex. It is more complex because there are more concepts to learn with object-oriented techniques than with procedural techniques. If the difficulties in making the transition from procedural to object-oriented software development were a function of the number of concepts only, then we would not see the plateaus in the learning curve. The findings indicated that the learning curve for procedural experts transitioning to object-oriented techniques includes plateaus and that the plateaus are a graphical manifestation of proactive interference. It is the experts' knowledge of the procedural concepts that is interfering with the learning of the object-oriented concepts. It is the interference that makes the transition so difficult, not the complexity of the mindset.

Future Research

One of the first avenues for future research is replication. Due to the limitations listed in the previous section, a replication of this study would be useful. Another avenue for research from the limitations of this study would be to find participants who are object-oriented developers with no procedural experience. An analysis of their learning curves would not only be interesting but also affirm or rebut the finding of this research.

The overarching goal of this research program is to answer the question, "How can organizations ease the difficulties involved in revolutionary mindshifts?" Past research has identified that expert procedural software developers have difficulty making the transition to object-oriented methods. This study focused on understanding the difficulties. We answered the *why* and *where* questions. Now that we understand the problem, the next logical step is to solve the problem. How can organizations ease the difficulties involved when shifting mindsets? Future research could explore modifications to present training methods. Modifications can be made to current instructional design and tested in an experimental setting.

Long-term future research could generalize the principles found in this study to mindshifts in other domains. The principles identified in study may be generalized to other domains such as the shift from mainframe to client/server technology. Shifts in mindset occur not only in the software development domain, but also in IS and throughout organizations.

Summary of Chapter Five

At the beginning of this document was a quote, "You must unlearn what you have learned." As we have seen in this study that is not just a clever quote but also a maxim for making the transition to object-oriented techniques. Past research told us that making the transition to object-oriented techniques was difficult. This study asked the questions, "*Why* is it difficult for procedural experts to learn object-oriented development? And *Where* in the learning process are developers experiencing difficulty?" Our findings indicate that proactive interference is one factor contributing to the difficulty in making the transition. An individual's experience in procedural software development does impinge on the learning of object-oriented techniques. Our findings indicate that software developers experience difficulty at several points in the learning process. Proactive interference is the strongest during the processes of understanding object-oriented development within a larger system.

The successful adoption of a technology is dependent on enough people learning and using the technology successfully (Markus, 1990; Pool, 1997). An understanding of the learning processes involved in transition from procedural to object-oriented techniques could shorten the object-oriented learning process, increase software quality, perhaps decrease the frustration level of students during their learning process, and ultimately increase the use of object-oriented development techniques. Understanding the difficulties involved in learning a new software development mindset will provide principles for more effective and efficient instruction and/or retraining of developers. From a theoretical perspective, questions of knowledge transfer and cognitive interference are important to our understanding of learning and should continue to be explored.

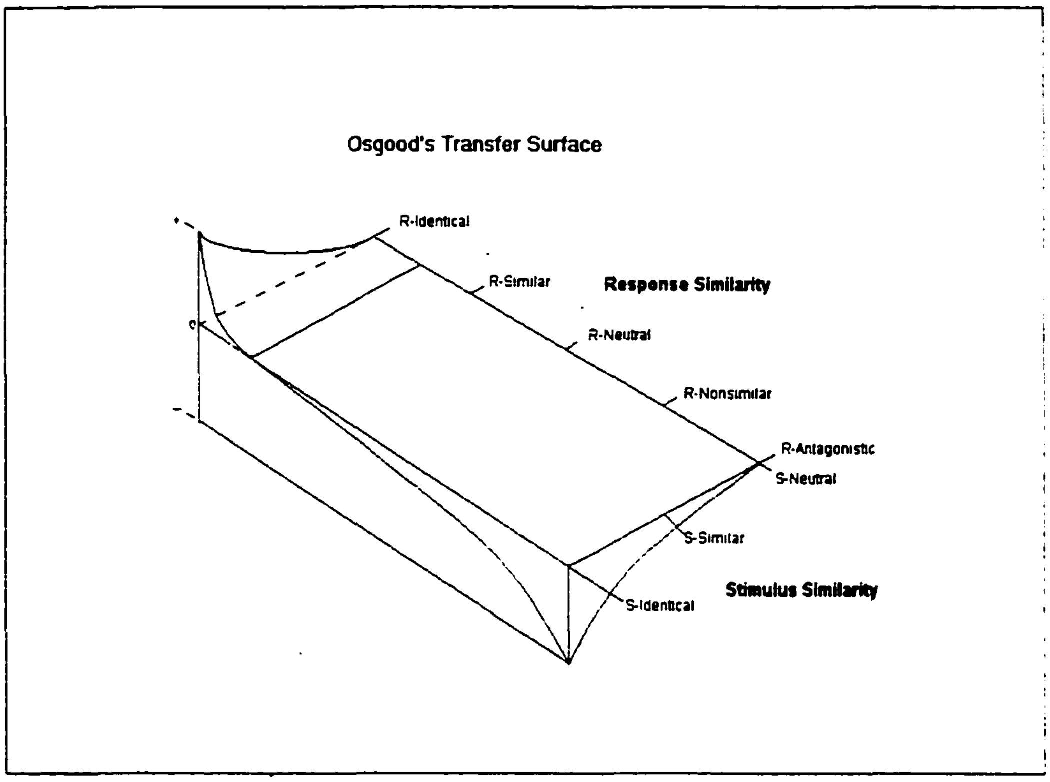


Figure 1. The Osgood Transfer Surface.

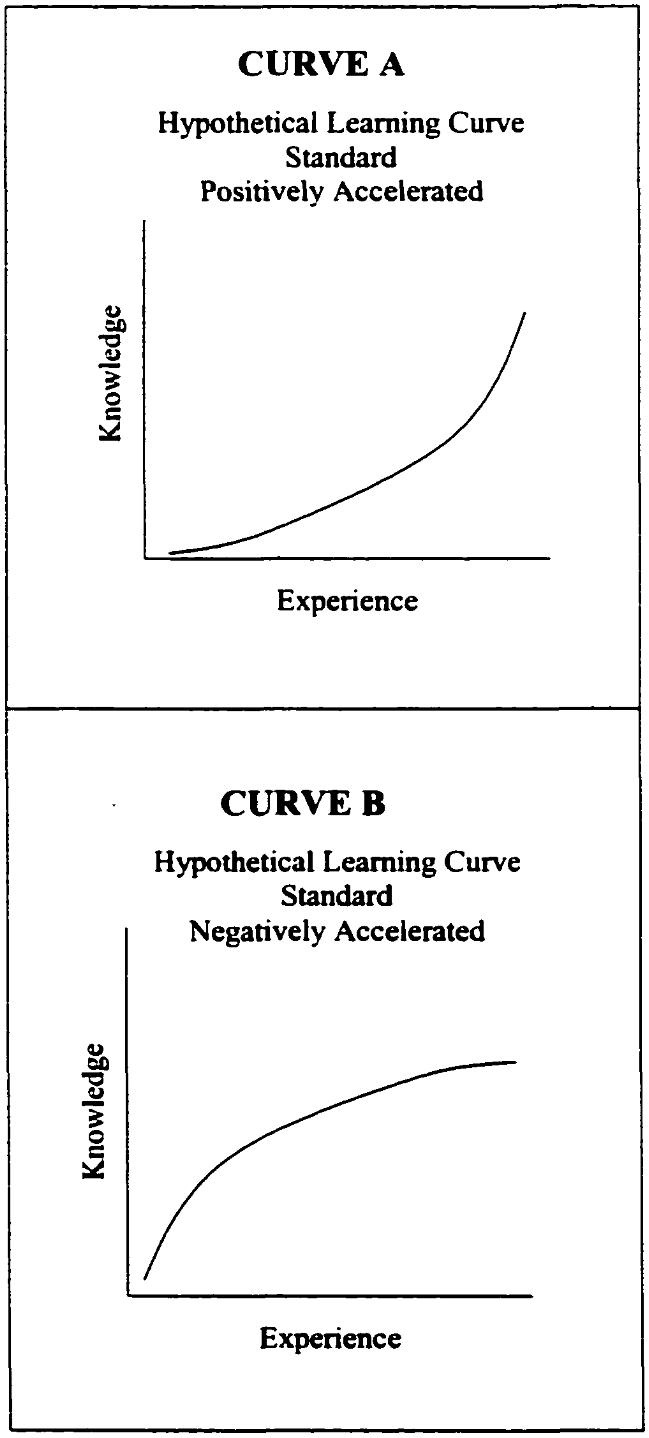


Figure 2. Hypothetical Learning Curves.

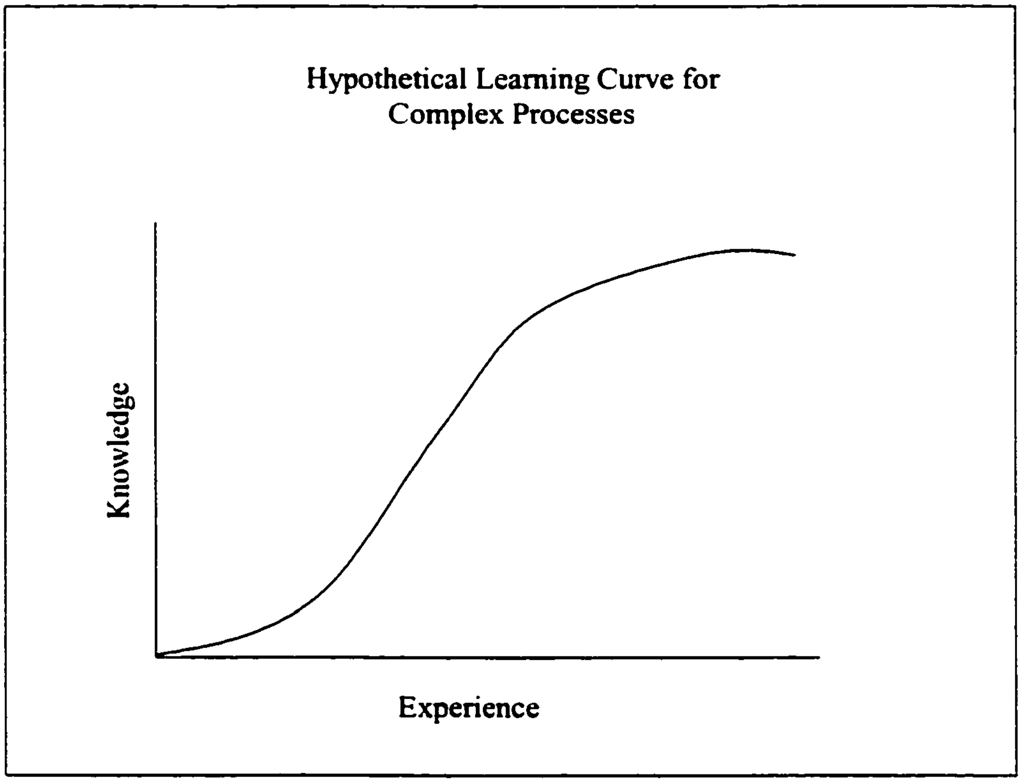


Figure 3. Hypothetical Learning Curve – Complex Processes.

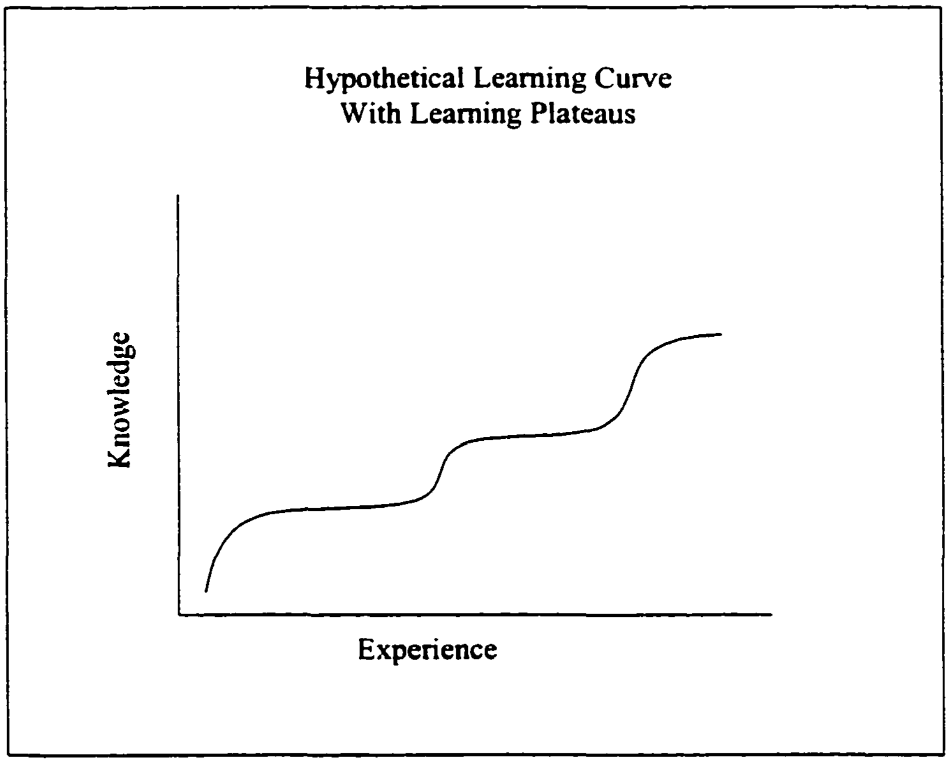
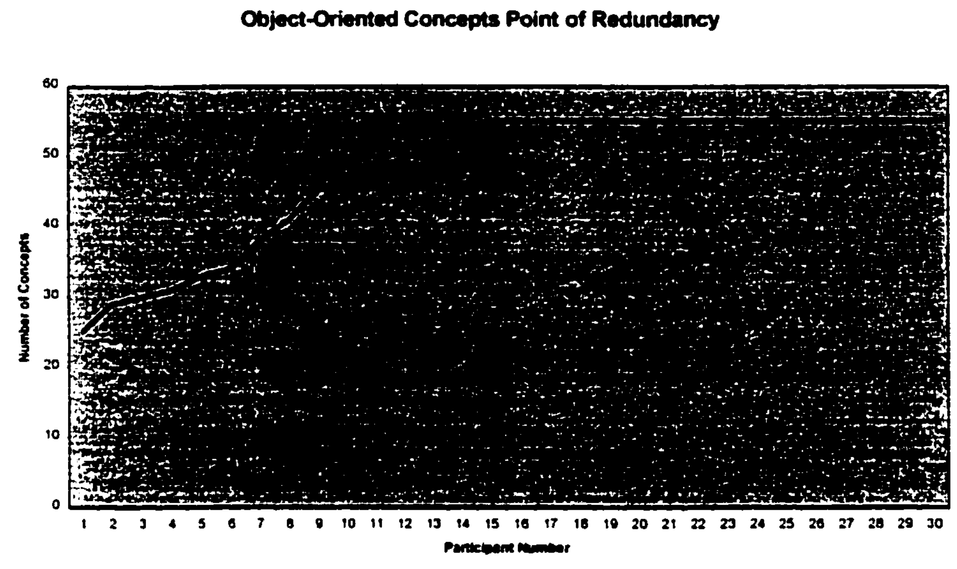


Figure 4. Hypothetical Learning Curve With Plateaus.



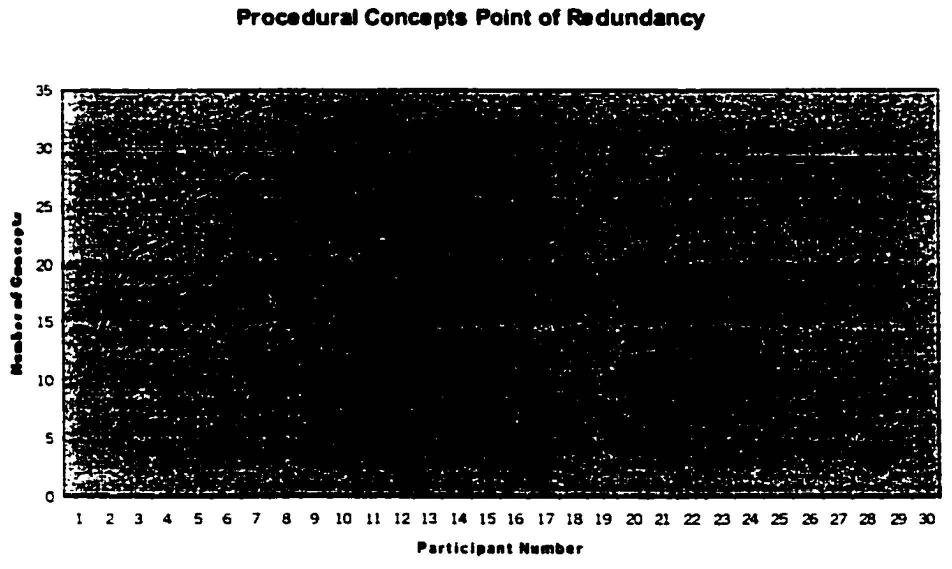


Figure 5. Point Of Redundancy For Procedural And Object-Oriented RCMs.

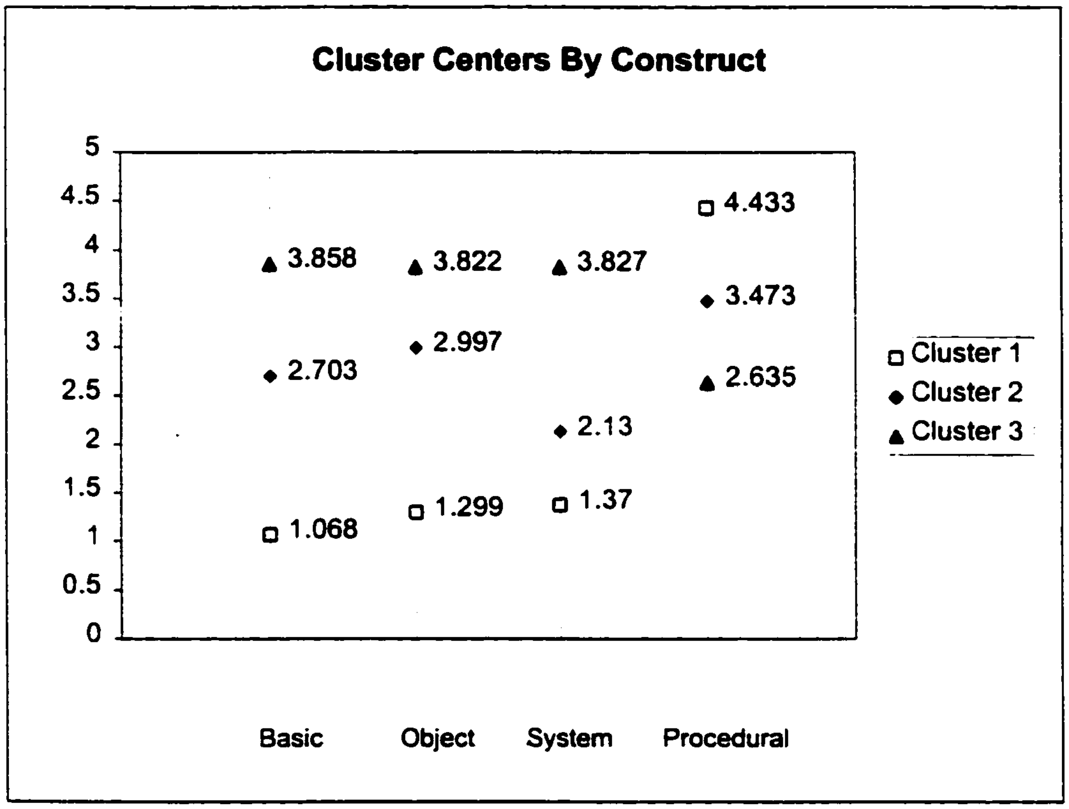


Figure 6. K-Means Cluster Analysis Centers.

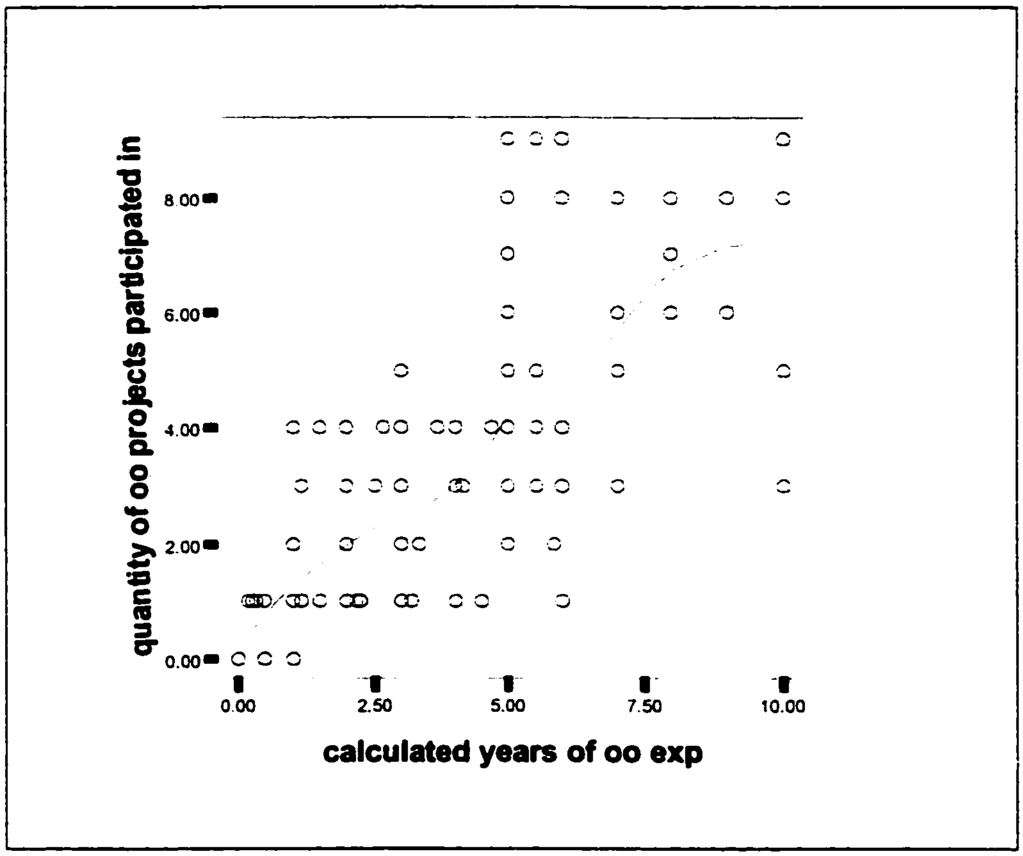


Figure 7. Correlation Graph, Years versus Number of Projects.

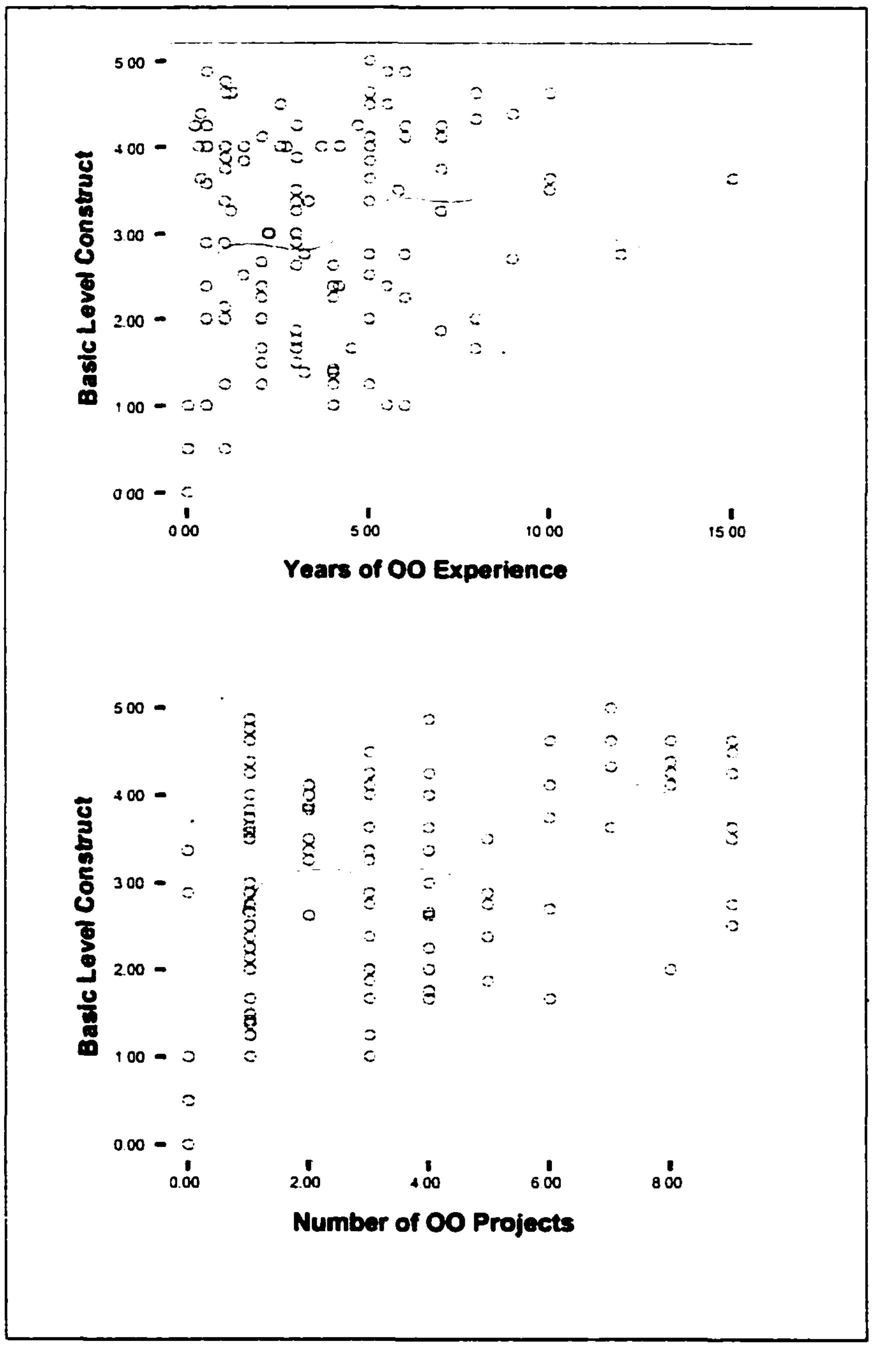


Figure 8. Basic Level Construct.

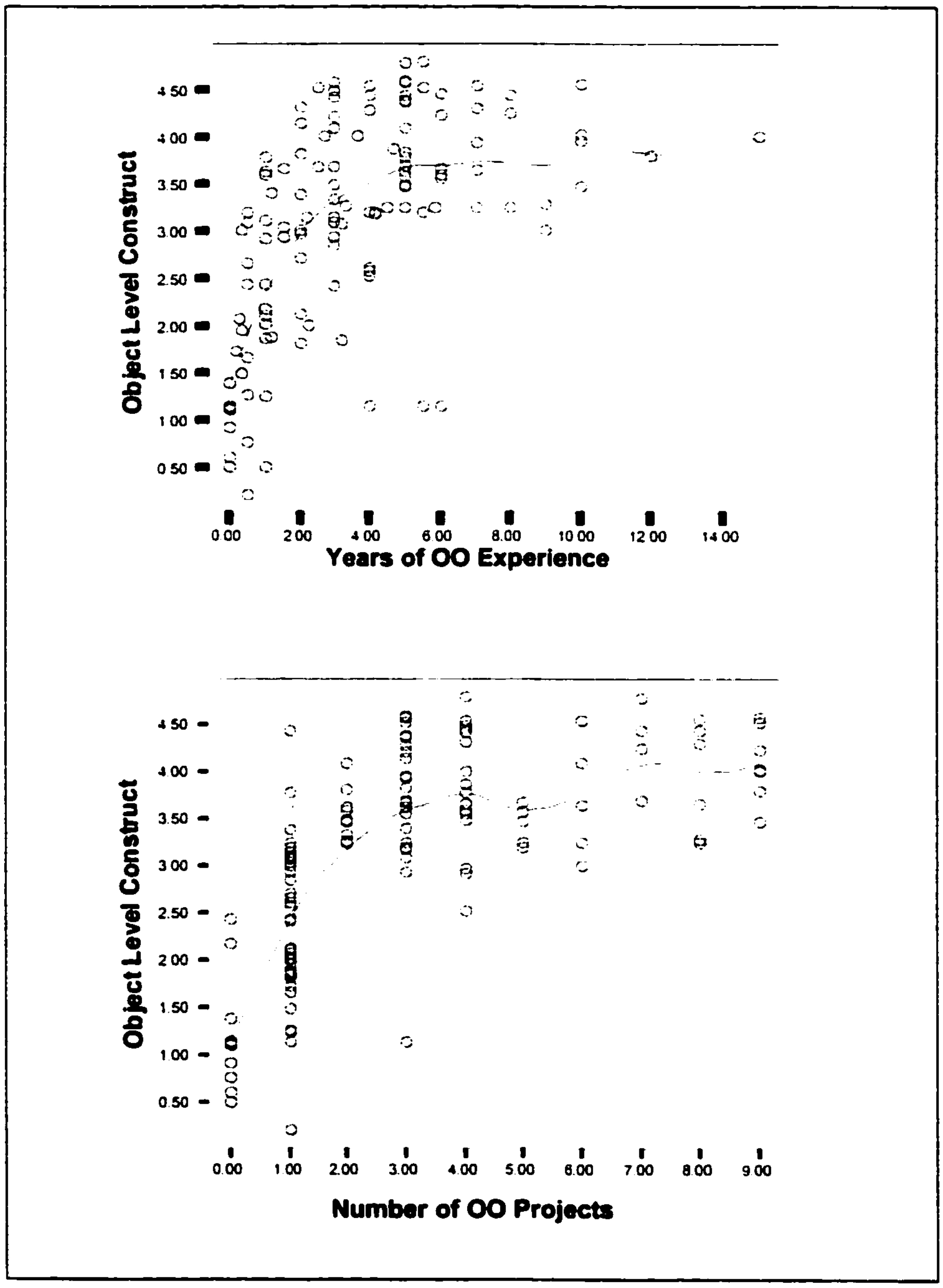


Figure 9. Object Level Construct.

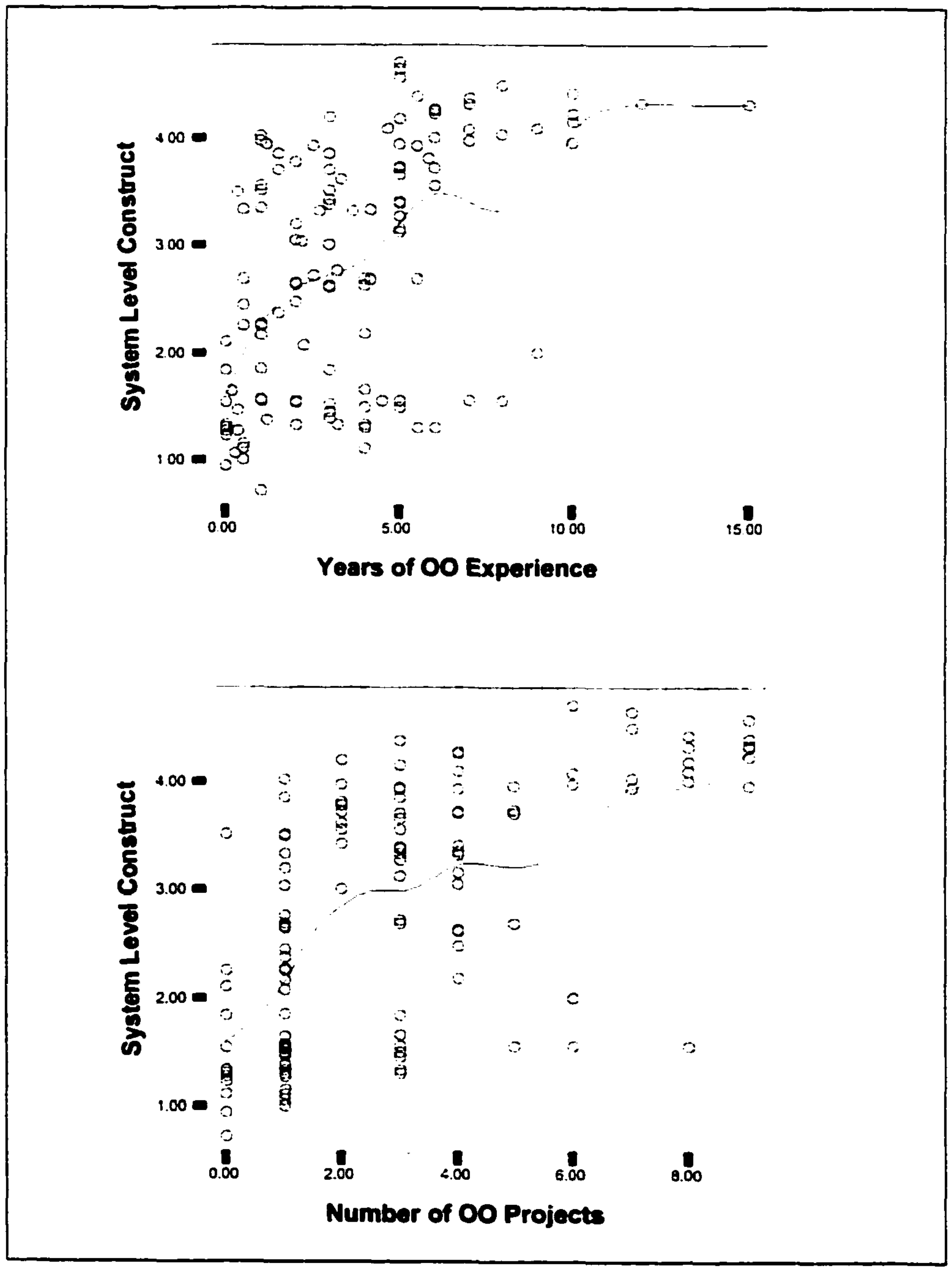
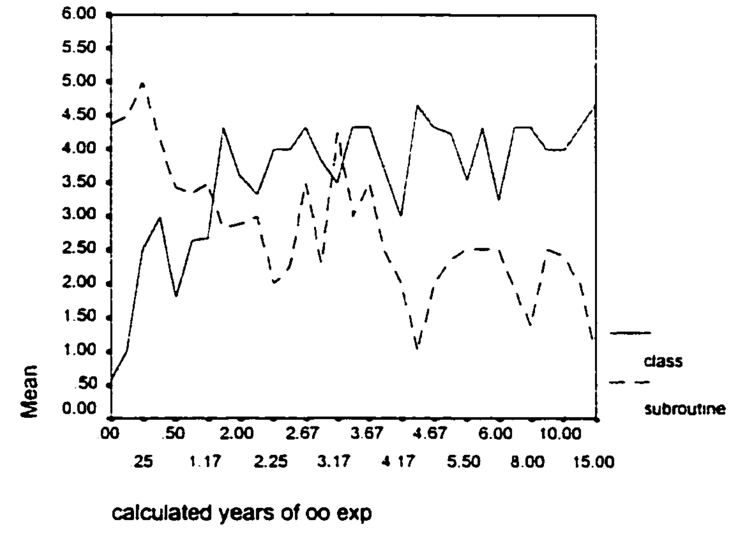


Figure 10. System Level Construct.



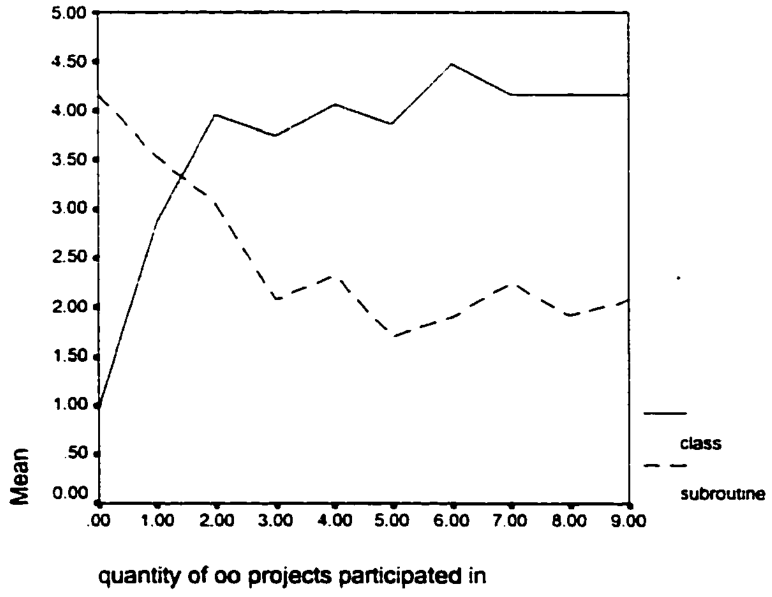
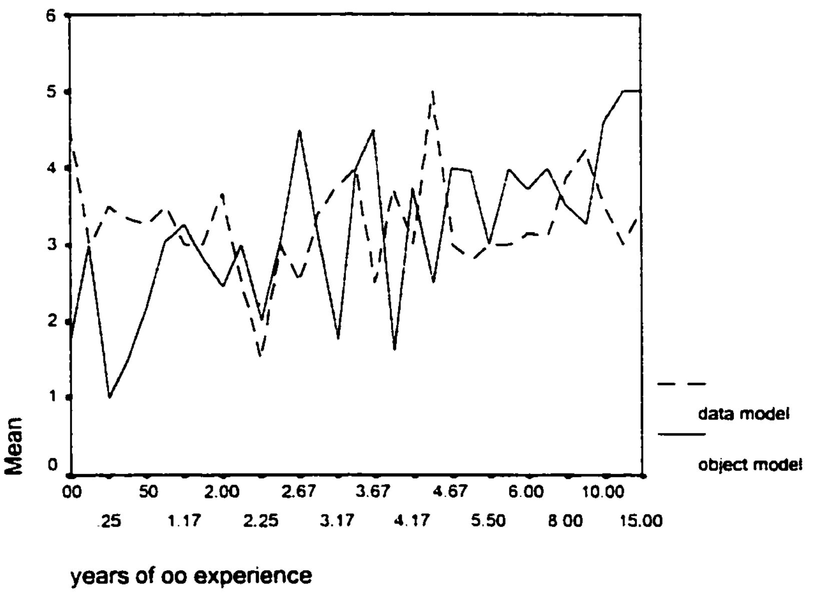


Figure 11. Comparison Of Class Concept Versus Subroutine Concept Using Years Of OO Experience And Number Of OO Projects As The X-Axis.



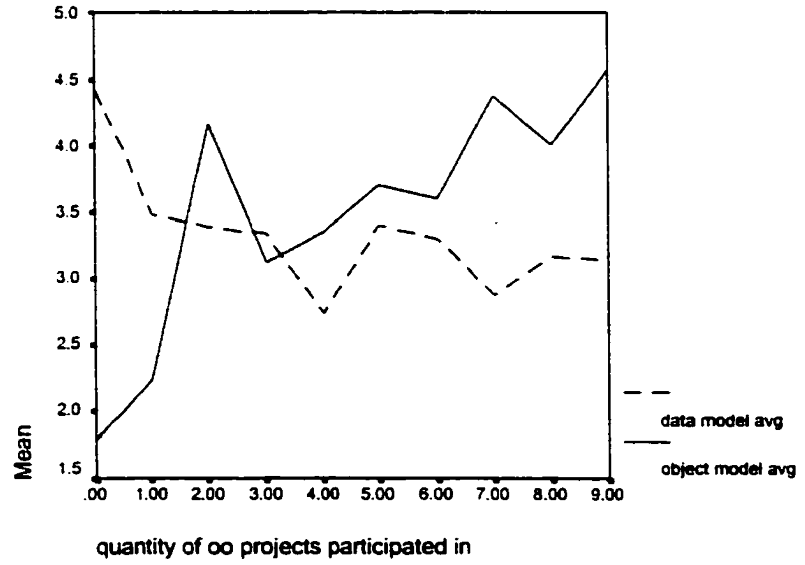
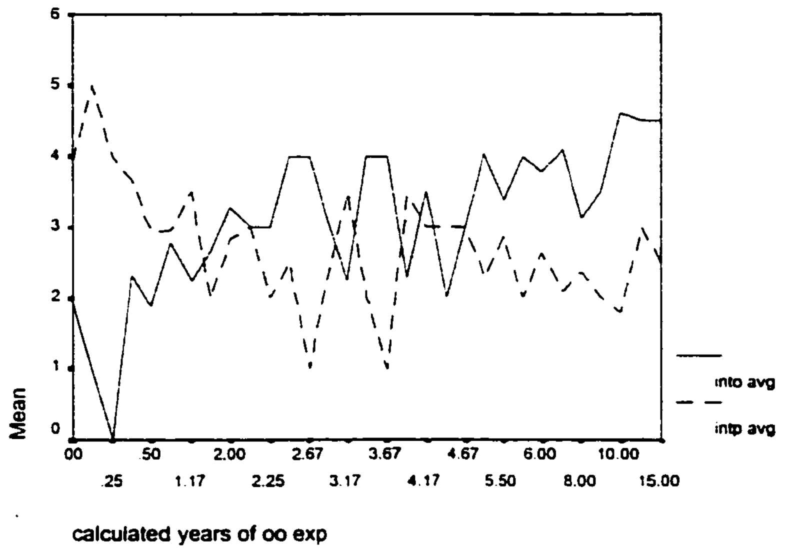


Figure 12. Comparison Of Data Model And Object Model Concepts Using Years Of OO Experience And Number Of OO Projects As The X-Axis.



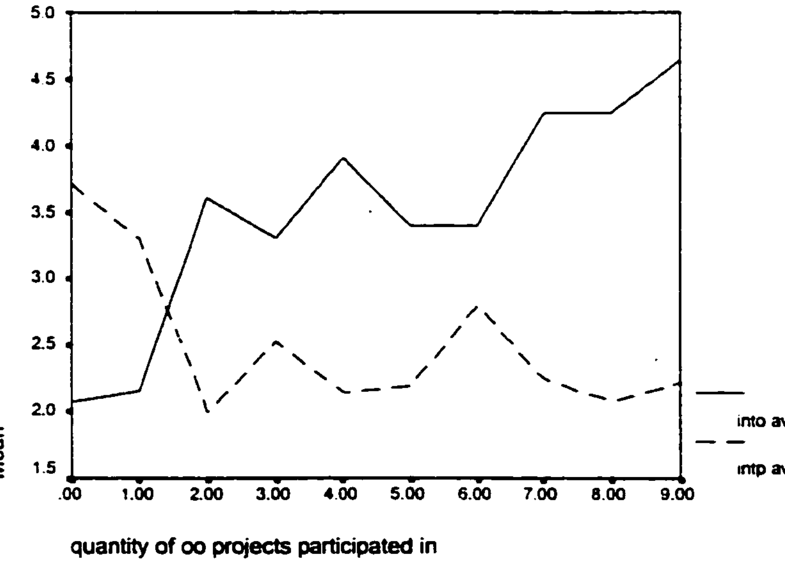
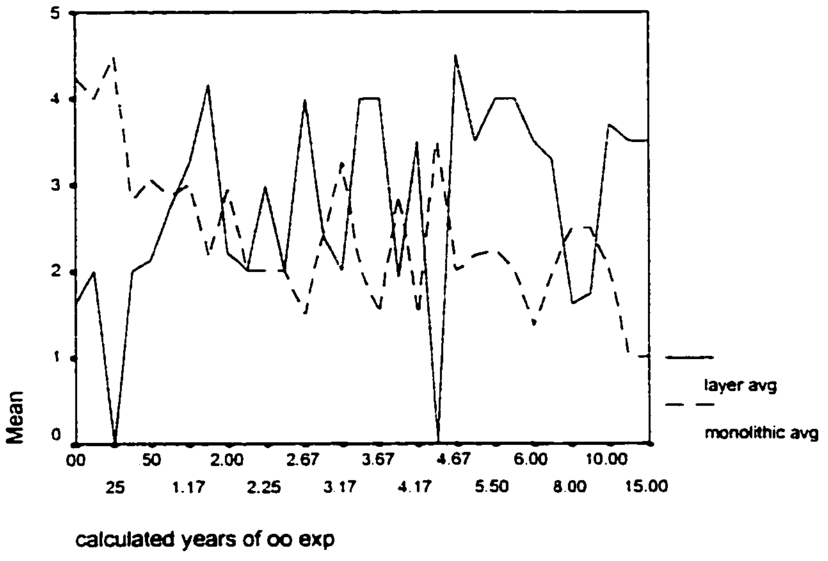


Figure 13. Comparison Of Procedural And Object-Oriented Interaction Concepts Using Years Of OO Experience And Number Of OO Projects As The X-Axis.



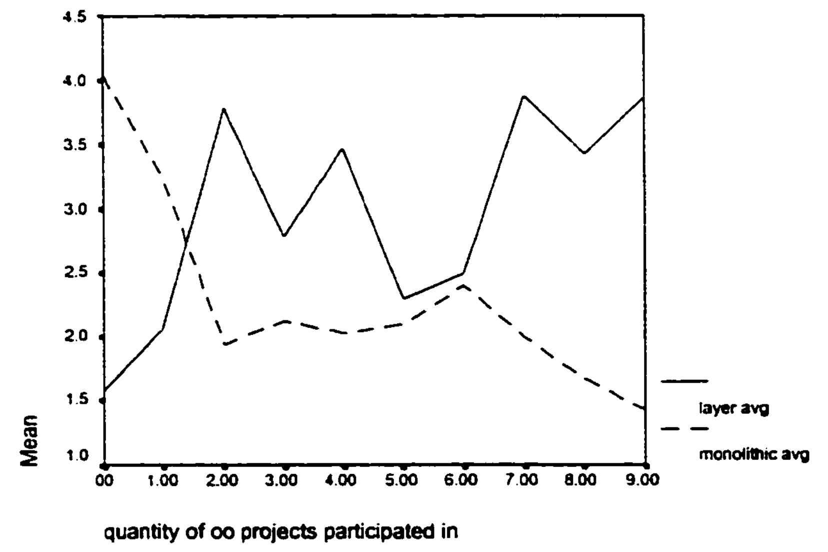


Figure 14. Comparison Of the Layer And Monolithic Concepts Using Years Of OO Experience And Number Of OO Projects As The X-Axis.

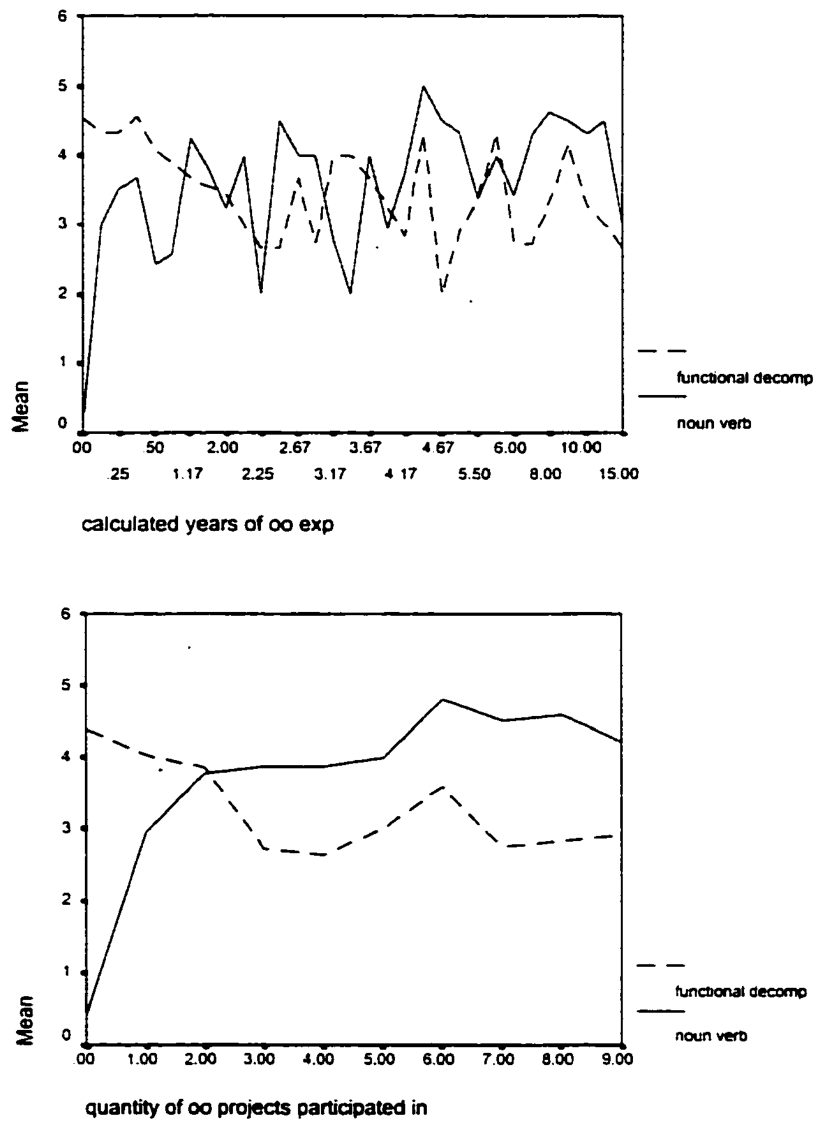


Figure 15. Comparison Of Functional Decomposition And Noun-Verb Analysis Concepts Using Years Of OO Experience And Number Of OO Projects As The X-Axis.

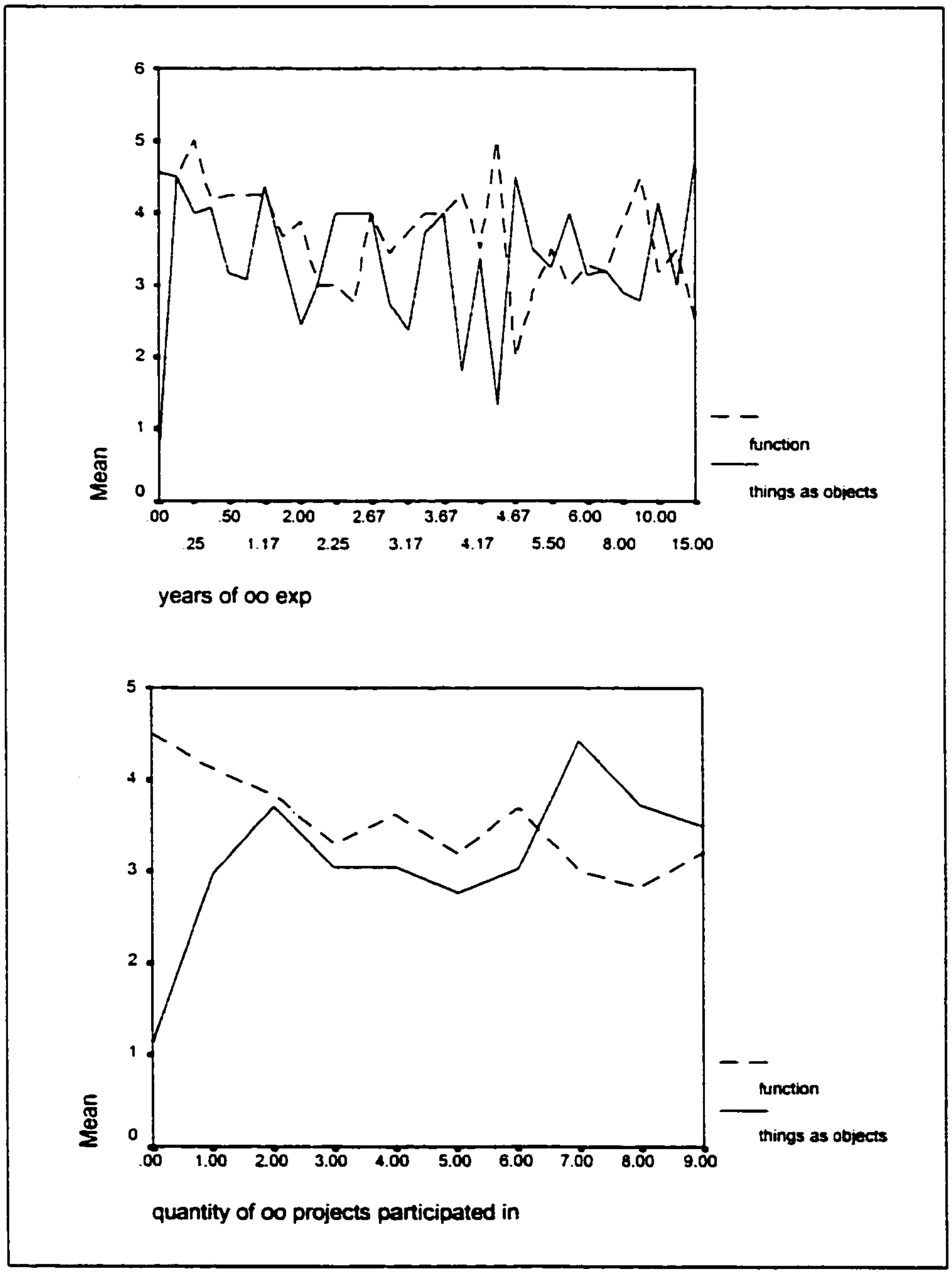


Figure 16. Comparison Of Function And Things As Objects Concepts Using Years Of OO Experience And Number Of OO Projects As X-Axis.

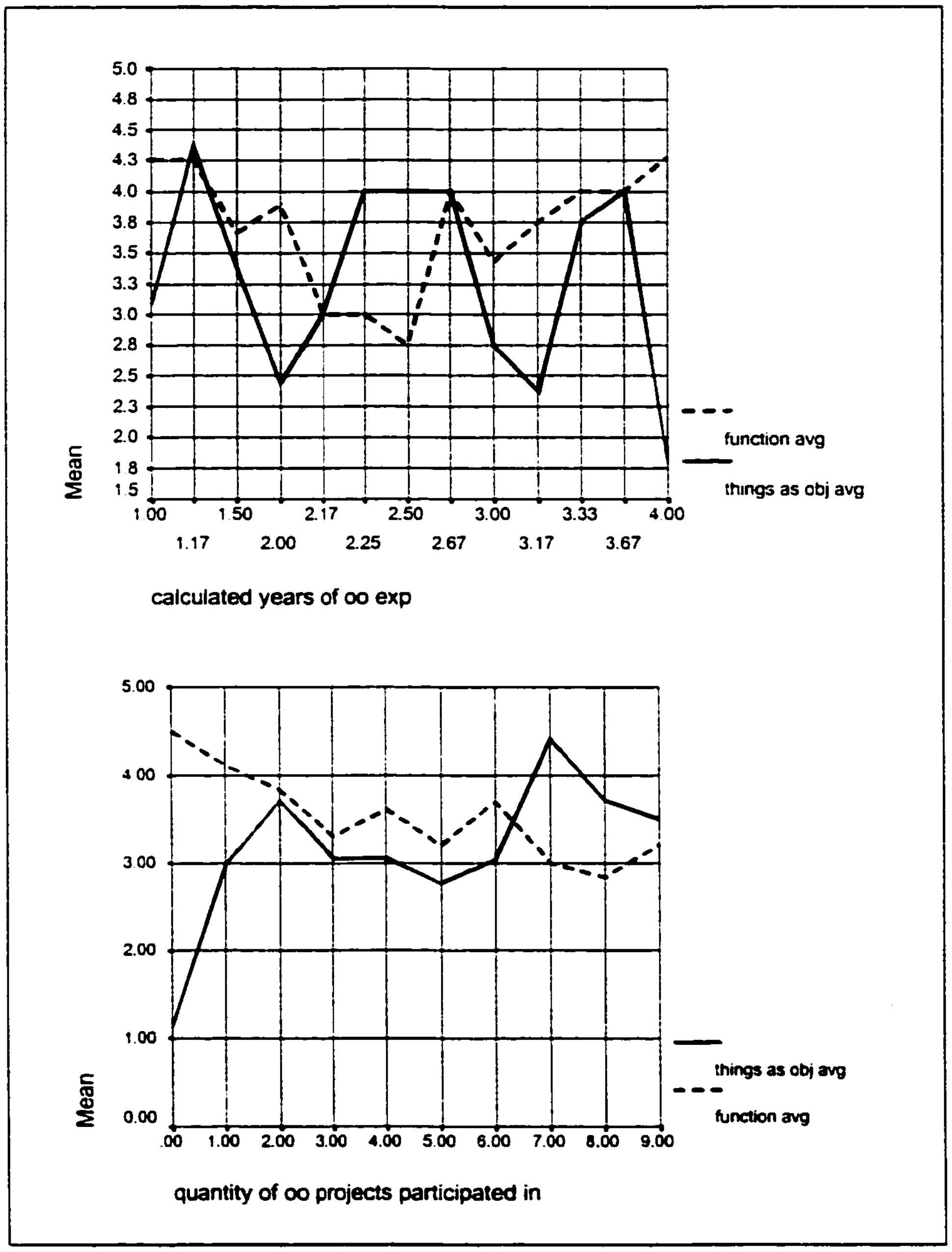


Figure 17A. Comparison Of Function And Things As Objects Concepts Using Years Of OO Experience And Number Of OO Projects As X-Axis.

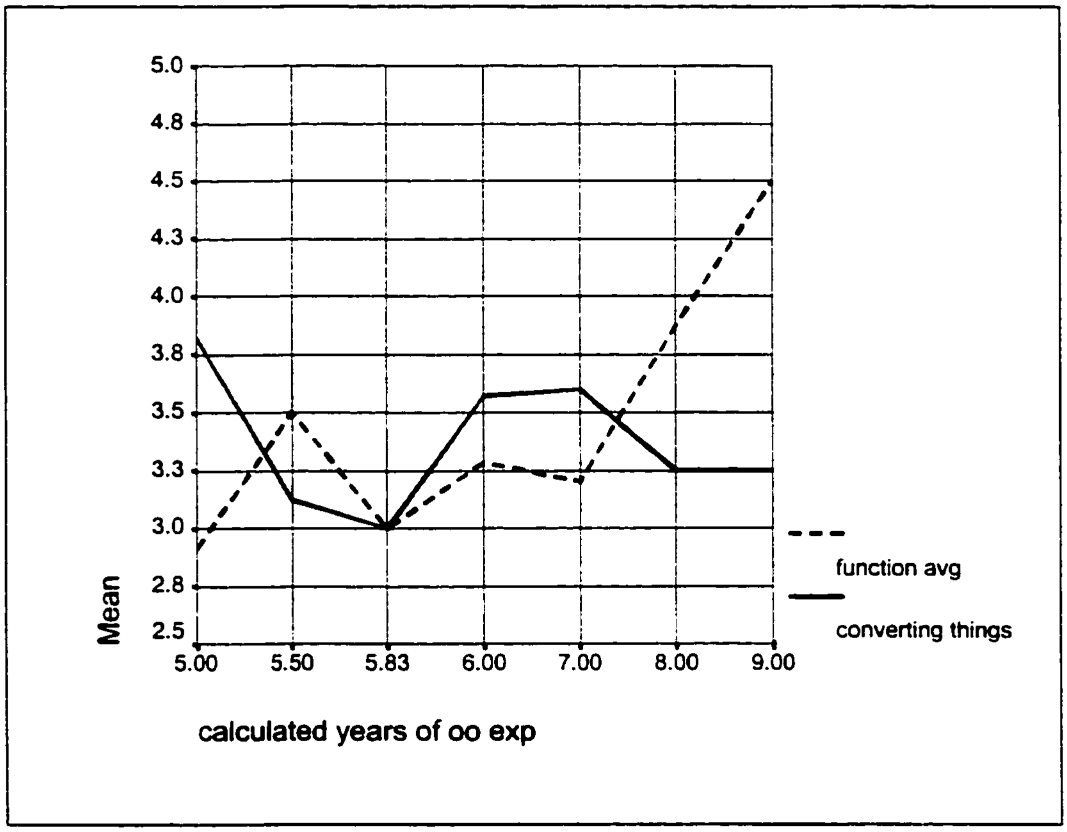


Figure 17B. Comparison Of Function And Converting Things Into Objects Concepts Using Years Of OO Experience As The X-Axis.

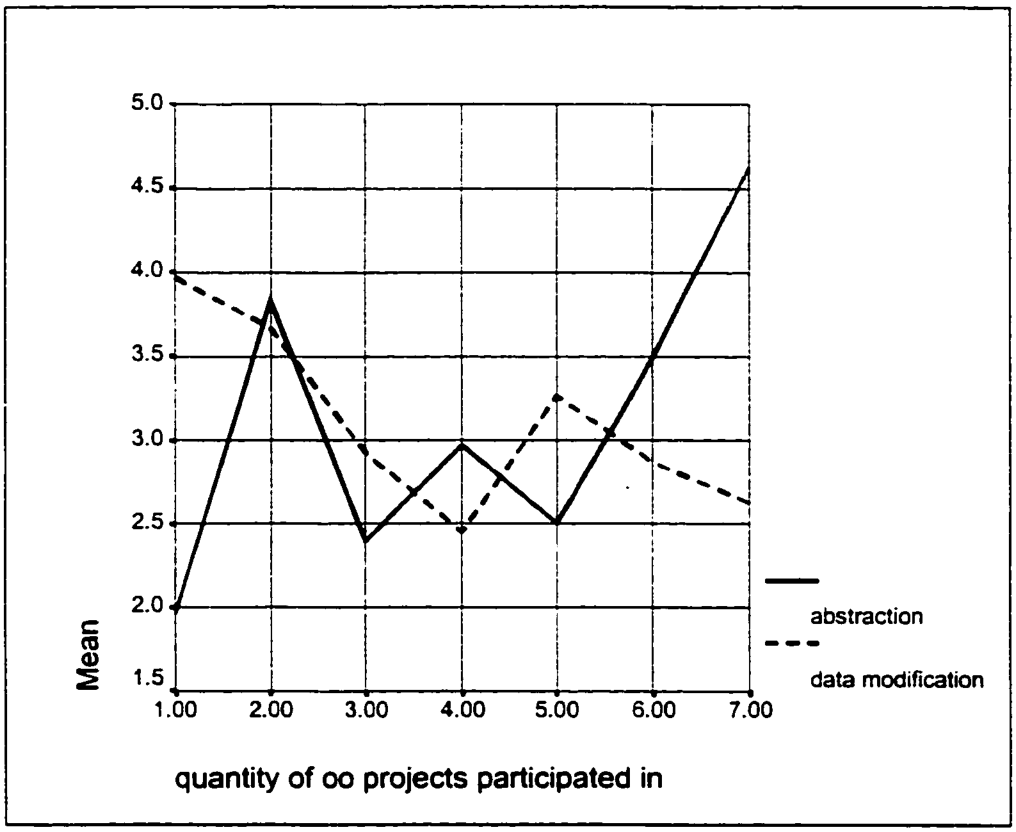


Figure 18. Comparison Of Abstraction And Data Modification Concepts Using Number Of OO Projects As The X-Axis.

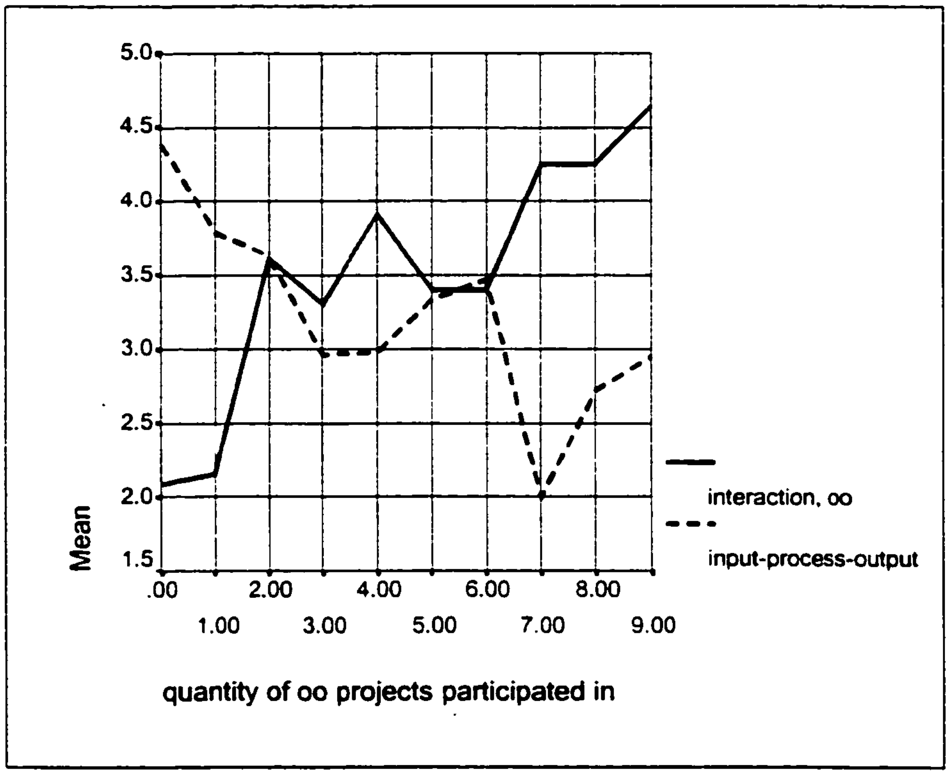


Figure 19. Comparison Of Interaction And Input-Process-Output Concepts Using Number Of OO Projects As The X-Axis.

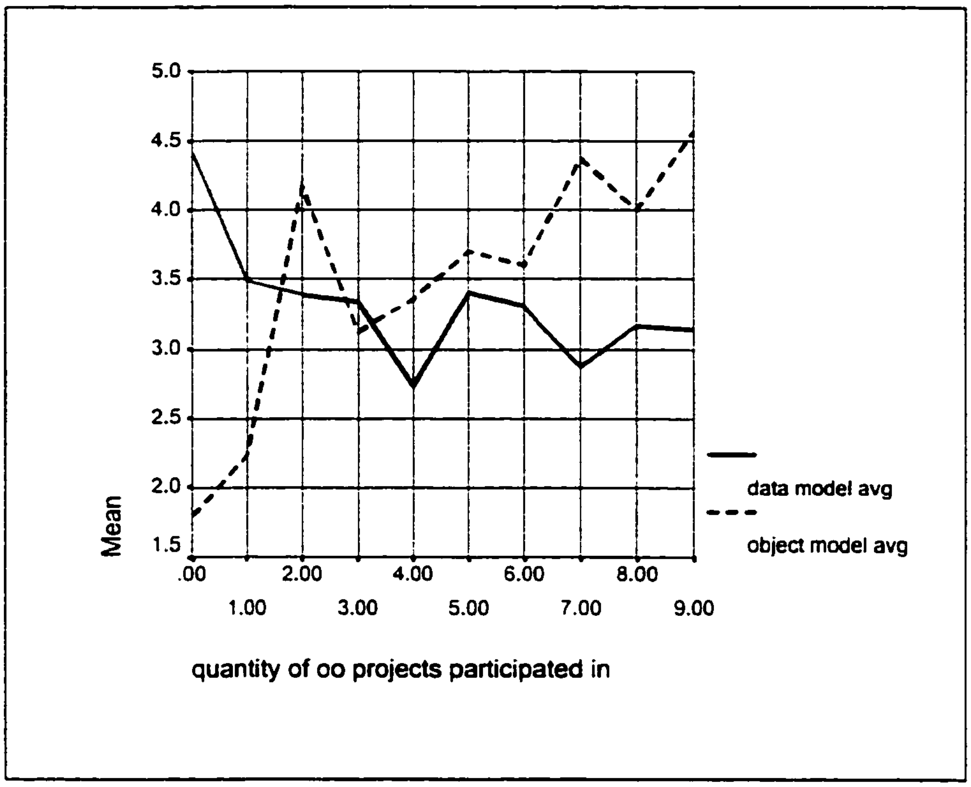


Figure 20. Comparison Of Data Model And Object Model Concepts Using Number Of OO Projects As The X-Axis.

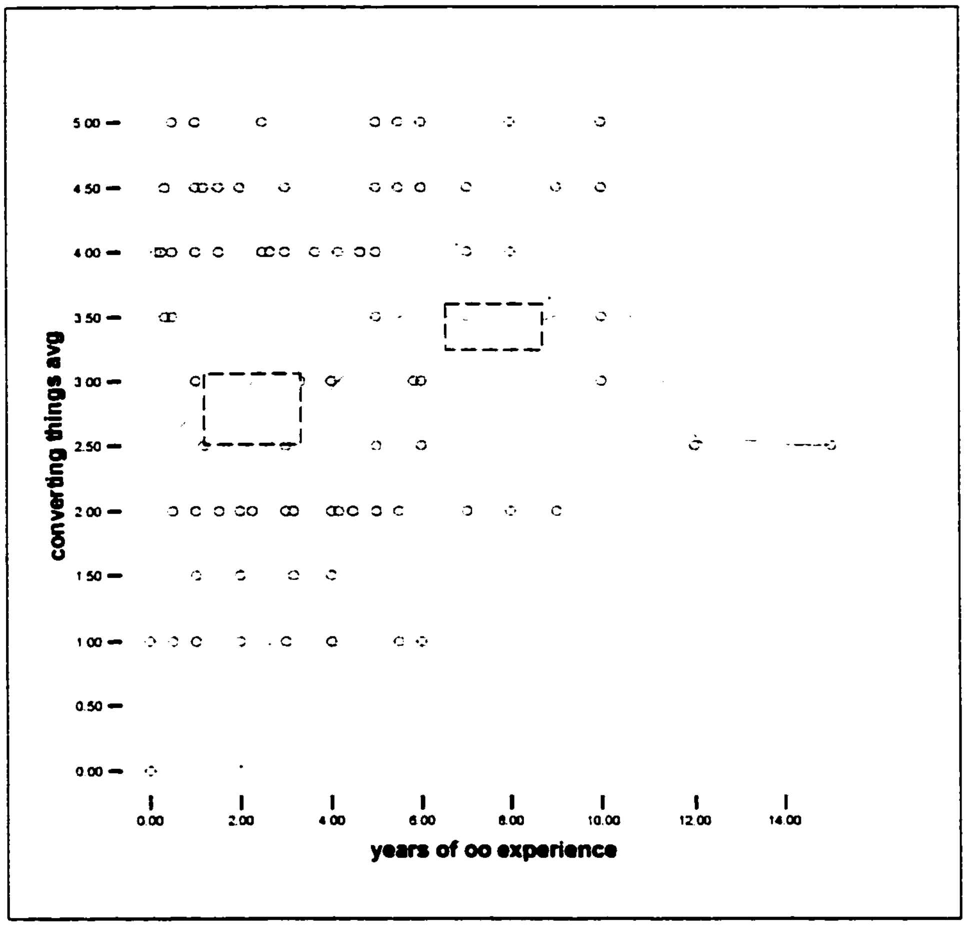


Figure 21. Multiple Plateau Location.

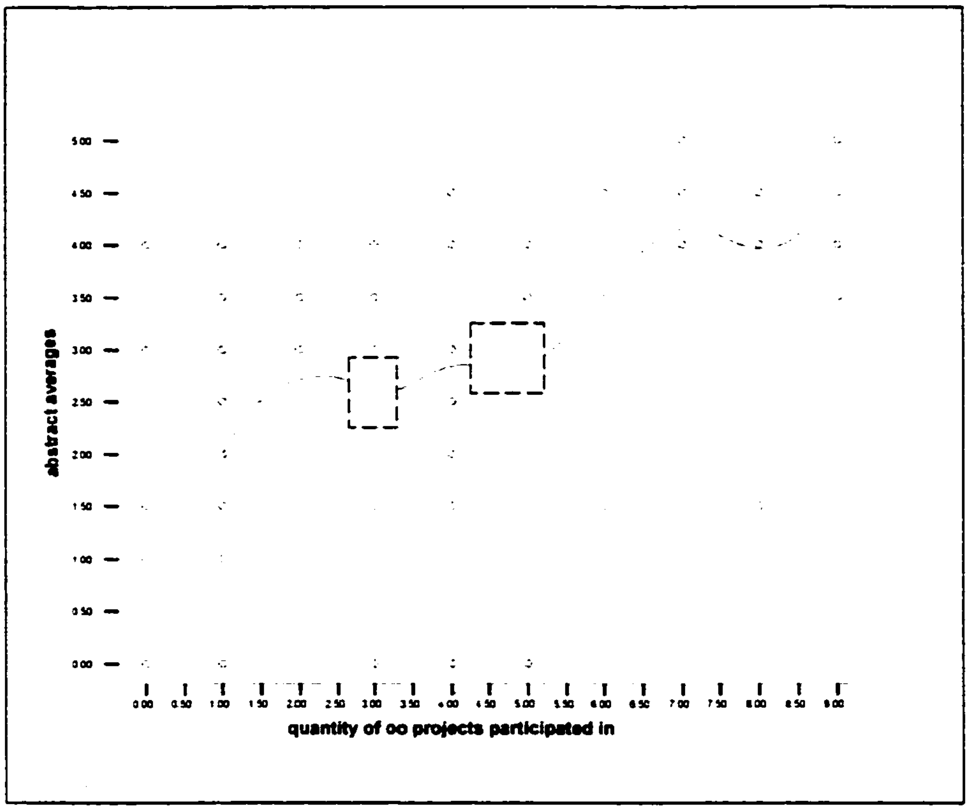


Figure 22. Multiple Plateau Location.

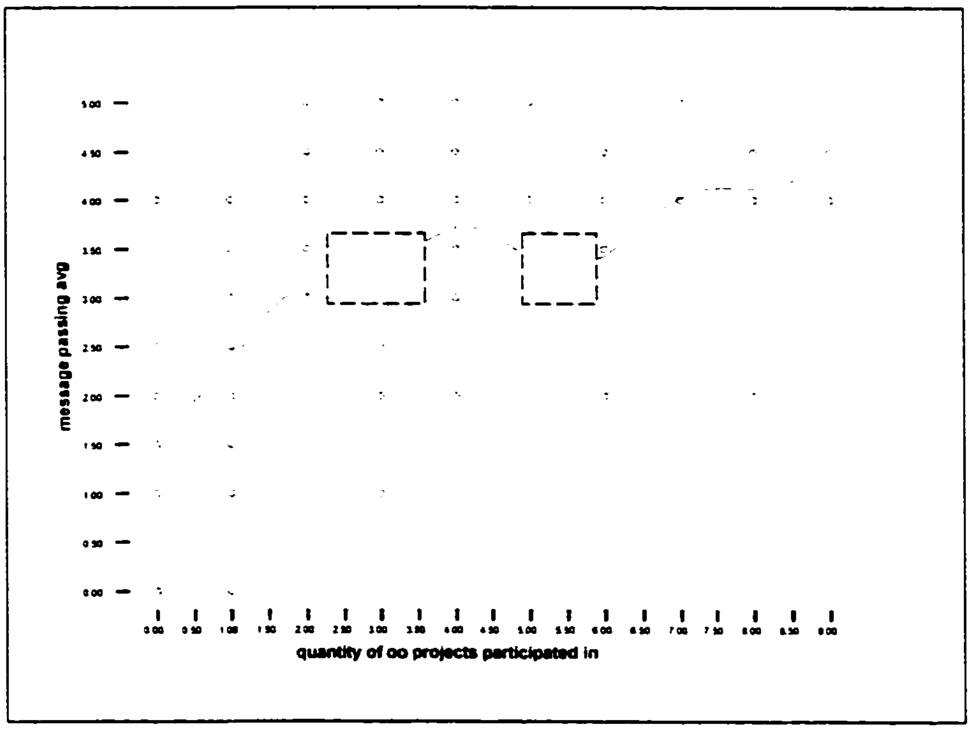


Figure 23. Multiple Plateau Location.

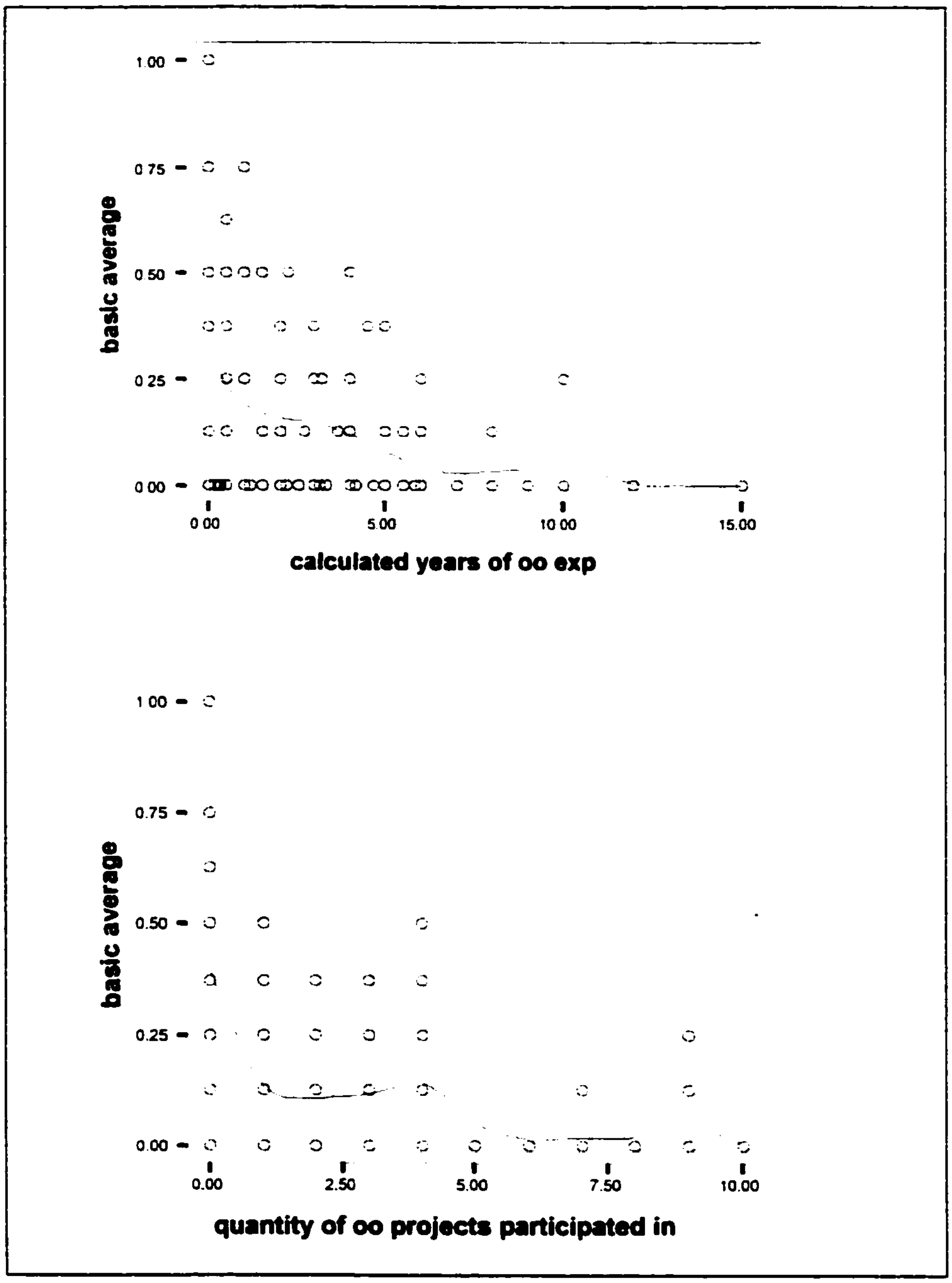


Figure 24. Basic Level Construct For "Don't Know" Analysis.

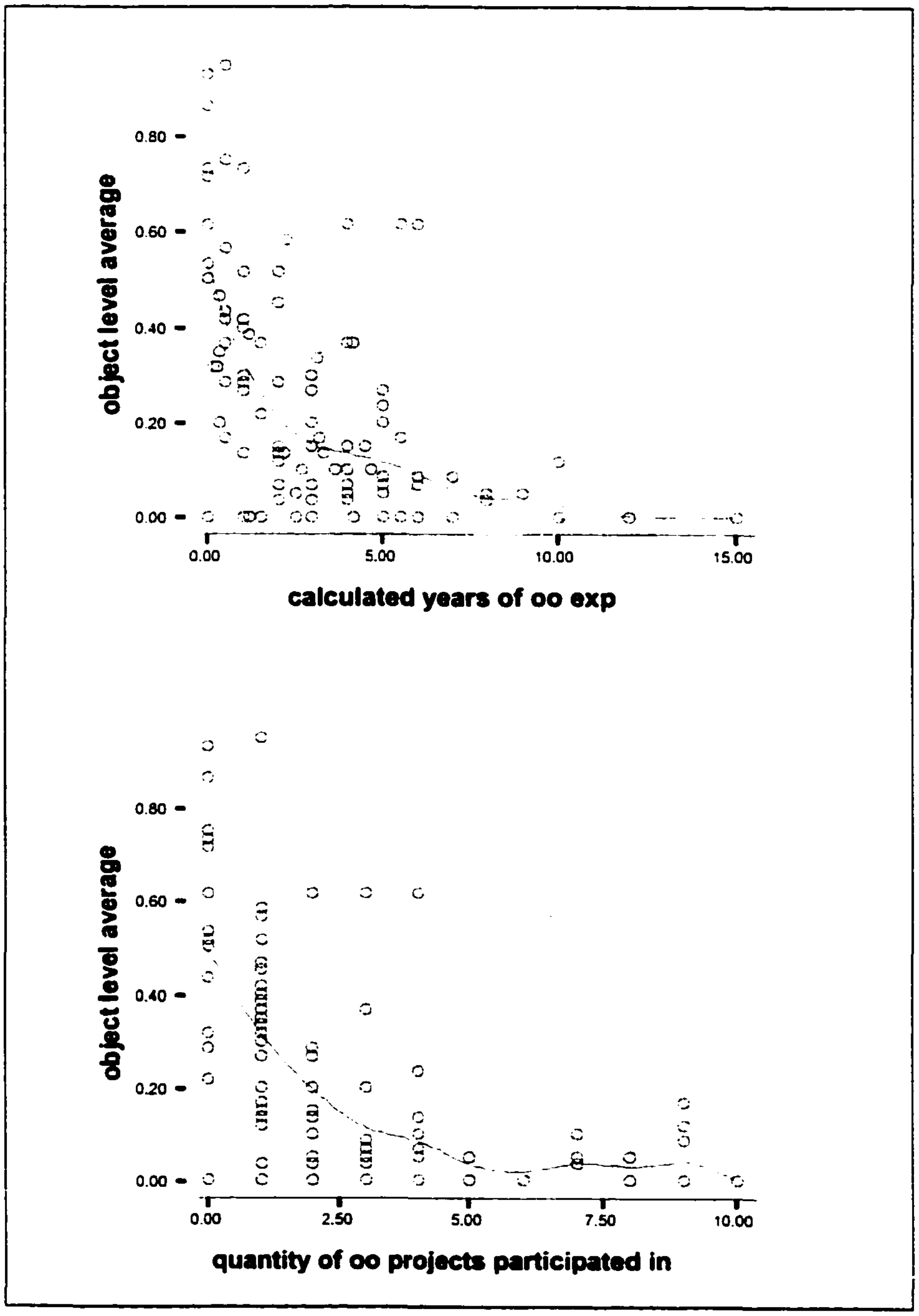


Figure 25. Object Level Construct For "Don't Know" Analysis.

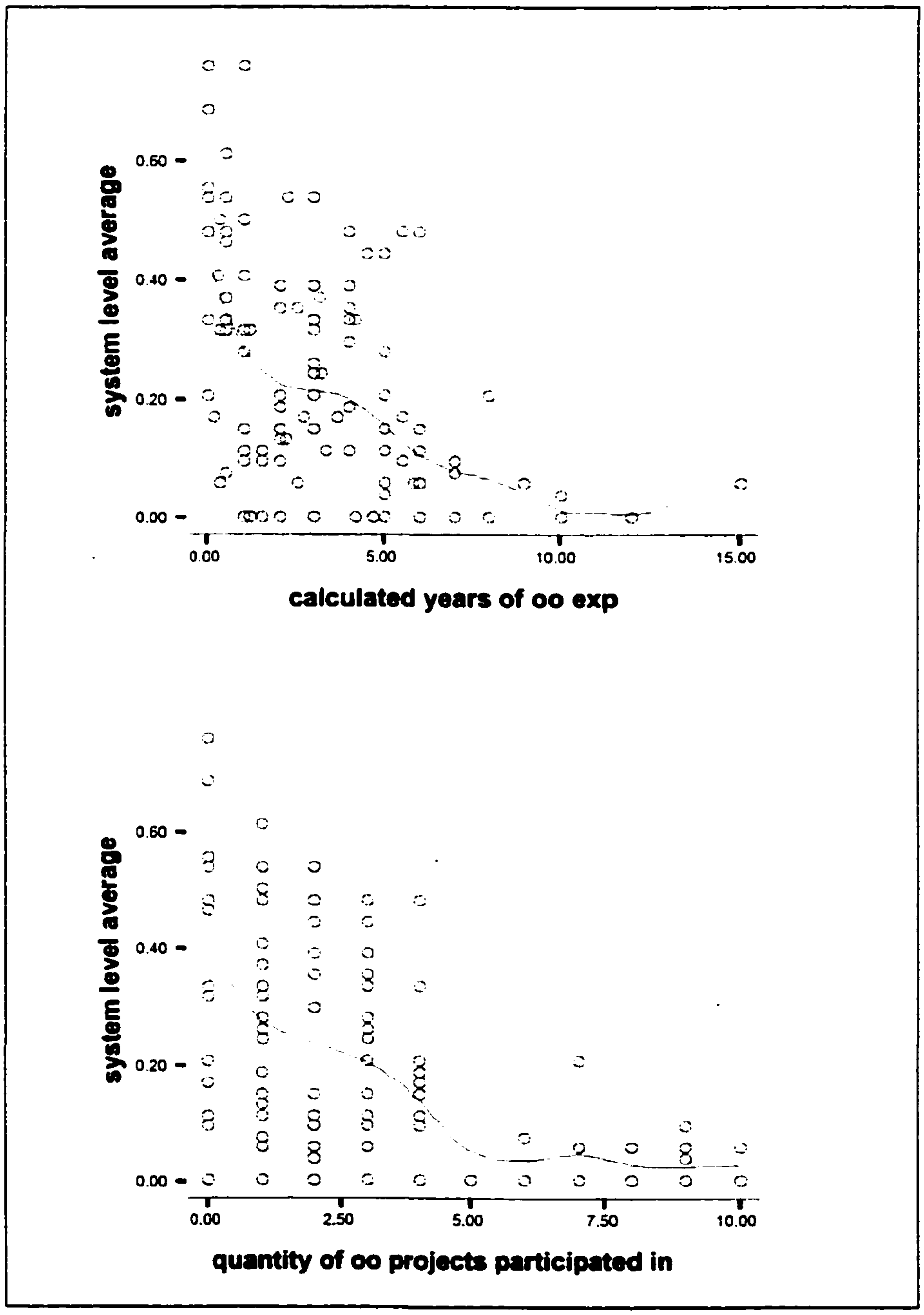


Figure 26. System Level Construct For "Don't Know" Analysis.

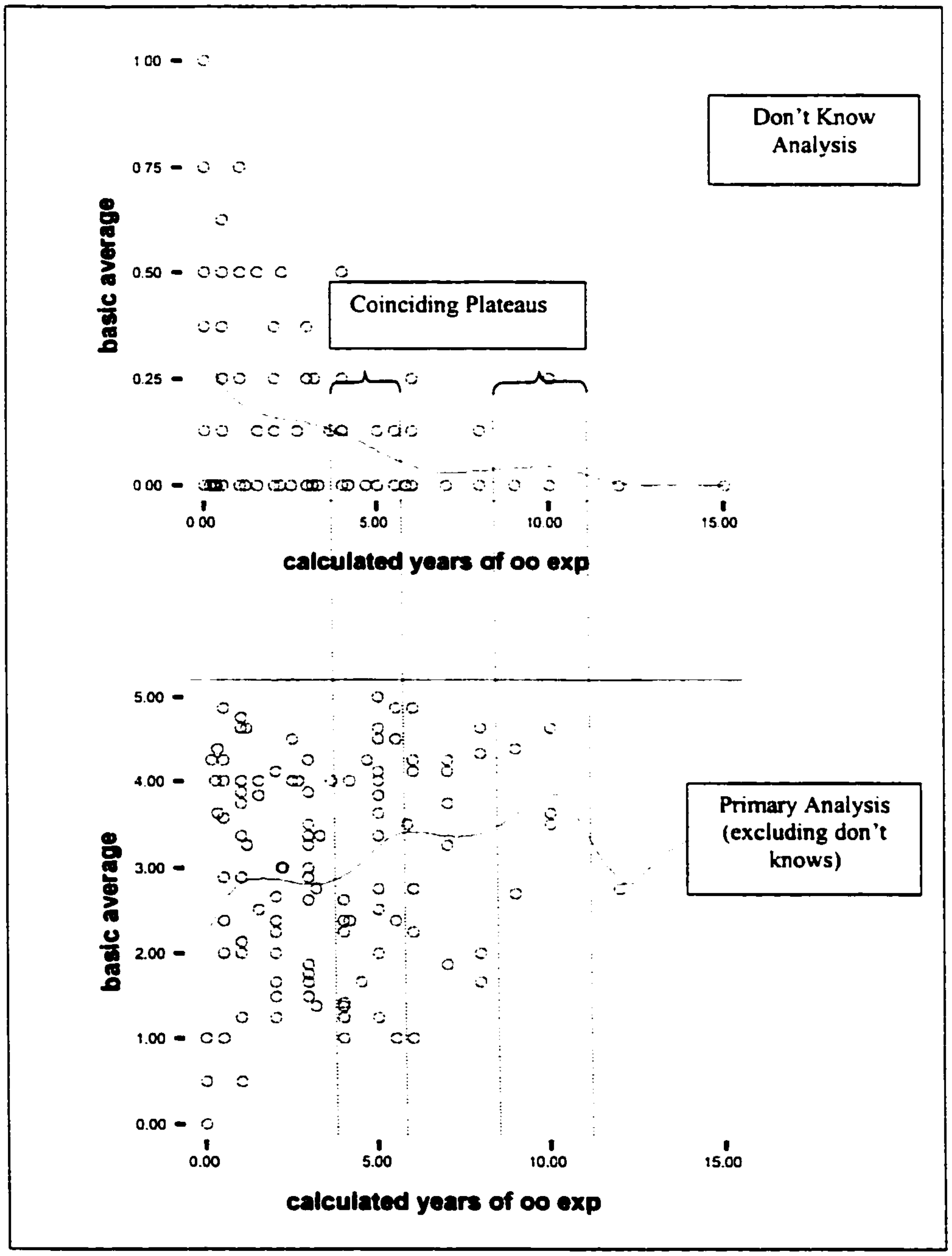


Figure 27. Comparison Of Basic Level Construct Plateaus.

Table 1

Approaches to Skill Acquisition

Category	Theory	Definition	Authors
Production Systems	ACT Theory Adaptive Control of Thought	Three stages of skill acquisition are the cognitive (declarative) stage, in which the learner makes an initial approximation of the skill; the associative stage (knowledge compilation), in which performance is refined; and the autonomous (procedural) stage in which performance is well refined but continues to improve slightly.	Anderson (1982; 1987; 1993), Fitts (1964)
	Larkin's ABLE Model	Characterize the minimal knowledge a learner might acquire from a textbook and then propose a means by which practice might facilitate the application of primitive knowledge to solve problems.	Larkin (1981)

table continues

Table 1

Approaches to Skill Acquisition

Category	Theory	Definition	Authors
Production Systems	Learning by Doing	Focuses on the development of strategies in problem solving.	Anzai and Simon (1979)
	Holland's framework	Acknowledges that a degree of parallel processing (conscious and subcogntive) occurs. A number of rules may fire simultaneously but only some will register.	Holland, Holyoak, Nisbett and Thagard (1986)
	Hunt and Lansman's Production Activation Model	Does not separate declarative and procedural information. Productions can be triggered either by spreading activation between them or by matching with information in working memory.	Hunt and Lansman (1986)
Mental Models	Mental Models	An individual creates a model of a situation and supposed conclusion.	Johnson-Laird (1989)

table continues

Table 1

Approaches to Skill Acquisition

Category	Theory	Definition	Authors
Propositional Based	Frames	Micro units of knowledge used to represent a stereotyped situation.	Minsky (1975)
	Schema	Cognitive processes continually evaluate incoming information and compare it to existing knowledge structures.	Bartlett (1932)
	Scripts	Individuals use stereotyped sequences of events that take place over time. Activated in highly specific situations.	Abelson (1976), Schank and Abelson (1977)

Table 2

Phase I Demographics

Demographic	Object-Oriented	Procedural
Number of Participants	35	20
Age	39.09	36.89
Gender (% male)	96.4%	75.0%
Years in IT	14.26	11.28
Years with current organization	4.61	8.11
Years of procedural experience	10.16	13.22
Number of procedural projects participated in	18.00	42.20
Years of object-oriented experience	5.76	0.88
Number of object-oriented projects participated in	6.00	0.67

Table 3

Causal Statements

Statement Type	Sentence
Object-Oriented Explicit	"I'm putting behaviors on it because I have to do things to make the use cases work."
Object-Oriented Implicit	"The first thing I think about- what are the things we need to keep track of, and how do they interact."
Procedural Explicit	"From a requirements document, the first thing I think about is the data and so we would get those data items out and we would design a nice, clean, logical data model."
Procedural Implicit	"We used to think lets listen to the users talk and then lets extract from that conversation what information items they really need."

Table 4

Evoked Statements

Type	**Statement**
Object-Oriented	An object should be able to hide all of its private information and all of its data from all other objects.
Procedural	You group the requirements document items based on functions.

Table 5

Expert Classification Scheme (Phase I, Object-Oriented)

Design Characteristics	Design Relationships	Execution Characteristics	Execution Interaction
Abstraction	Collaboration	Instantiation	Interaction
Converting Things	Inheritance		Message Passing
Framework	Polymorphism		
Layer	Aggregation		
Relationship	Class Hierarchy		

Object Characteristics	Analysis Techniques	Application Design
Attribute	Noun/Verb Analysis	Components
Encapsulation	Object Model	OO Development
Info Hiding		Class
Method		
"Things" As Objects		

table continues

Table 5

Expert Classification Scheme (Phase I, Procedural)

Design Characteristics	Design Relationships	Execution Characteristics	Execution Interaction
Functions	Linear Structure	Linear Program	Data Modification
Linear Form		Subroutine	Input-Process-Output
Monolithic			Interaction

Analysis Techniques	Application Design
Data Model	Linear Flow
Functional Decomposition	

Table 6

Phase II Demographics

Demographic	**Sample**
Number of Participants	31
Age:	
Under 21	0
21-30	7
31-40	9
41-50	7
51-60	7
Over 60	1
Gender (% male)	54.8%
Years with current organization	6.55
Years of procedural experience	12.71
Number of procedural projects participated in	69.29
Years of object-oriented experience	2.18
Number of object-oriented projects participated in	1.07

Table 7

Phase III Demographics

Demographic	**Sample**
Number of Participants	131
Age:	
Under 21	0
21-30	6
31-40	57
41-50	42
51-60	25
Over 60	1
Gender (% male)	88.6%
Formal Education (highest level):	
High school diploma	0
Some college, but no degree	5
Associates degree	9
Technical degree	6
Bachelors degree	48
Some graduate coursework, but no degree	32
Masters degree or MBA	11
Some doctoral work, but no degree	13

table continues

Table 7

Phase III Demographics

Demographic	**Sample**
PhD or equivalent	5
Other	2
Number of professional organizations belong:	
0	116
1	10
2	4
3	1
Years of procedural experience	15.64
Number of procedural projects participated in	34.21
Years of object-oriented experience	3.70
Number of object-oriented projects participated in	3.13
Number of programming languages used on the job:	
0-2	11
3-4	60
5-6	37
7-9	20
10-15	3

table continues

Table 7

Phase III Demographics

Demographic	**Sample**
Learning methods (could check multiple items):	
Attending conferences	69
College courses	93
Company training session	79
Internet research	41
Online courses	27
On-the-job training	111
Reading books/manuals	113
Reading technical journals/magazines	55
Seminars	37
Self-taught	117
Years with current organization	9.13
Industry:	
Aerospace (2 organizations)	38
Agriculture	2
Consulting	1
Education	4

table continues

Table 7

Phase III Demographics

Demographic	Sample
Financial	11
Government	8
Manufacturing	26
Medical	4
Retail	3
Services	18
Software development	2
Telecommunications	12
Utilities	2

Table 8

Learning Conceptual Scheme

Basic Level	**Object Level**	**System Level**	**Procedural**
= .8949	= .9534	= .9252	= .9580
Converting Things	Attribute	Abstraction	Data Model
Things As Objects	Class	Component	Data Modification
	Collaboration	Framework	Functional Decomposition
	Encapsulation	Information Hiding	Function
	Inheritance	Interaction	Interaction
	Instantiation	Layer	Input-Process-Output
	Method	Message Passing	Linear Flow
	Noun Verb Analysis	Object Model	Linear Form
	OO Development	Relationship	Linear Program
	Polymorphism		Linear Structure
			Monolithic
			Subroutine

Table 9

K-Means Cluster Analysis

	Cluster 1	**Cluster 2**	**Cluster 3**
N	21	48	62
Cluster Center – Basic Level	1.068	2.703	3.858
Cluster Center – Object Level	1.299	2.997	3.822
Cluster Center – System Level	1.370	2.130	3.827
Cluster Center - Procedural	4.433	3.473	2.635
Years OO Experience	1.746	2.771	5.086
Number of OO Projects	.952	2.333	4.484

Table 10

t-tests Comparing Clusters

Cluster Comparison	Factor	t	df	Sig (2-tailed)
Clusters 1 & 2	Basic	6.891	67	.000
	Object	7.819	67	.000
	System	4.789	67	.000
	Procedural	-7.501	67	.000
Clusters 2 & 3	Basic	-7.342	108	.000
	Object	-6.207	108	.000
	System	-15.384	108	.000
	Procedural	7.498	108	.000
Clusters 1 & 3	Basic	-18.789	81	.000
	Object	-18.189	81	.000
	System	-23.151	81	.000
	Procedural	12.982	81	.000

Table 11

Cluster ANOVA

	Cluster		Error			
	Mean Square	df	Mean Square	df	F	Sig.
Basic	65.040	2	.611	128	106.514	.000
Object	51.308	2	.483	128	106.174	.000
System	65.570	2	.293	128	224.100	.000
Procedural	28.283	2	.307	128	92.062	.000

Table 12

Coefficient of Determination (r^2)

Factor	Regression Type	Years of OO Experience	Number of OO Projects
Basic Level	Linear	6.1%	15.2%
	Quadratic	6.9%	17.8%
	Curvilinear	70.3%	68.3%
Object Level	Linear	28.6%	39.3%
	Quadratic	41.4%	55.5%
	Curvilinear	69.9%	69.4%
System Level	Linear	23.7%	37.4%
	Quadratic	26.0%	39.8%
	Curvilinear	79.4%	80.1%
All OO	Linear	22.8%	38.5%
	Quadratic	27.8%	45.9%
	Curvilinear	89.2%	89.8%
Procedural	Linear	15.9%	24.3%
	Quadratic	22.3%	34.6%
	Curvilinear	59.7%	59.5%

Table 13

Standard Error (Se)

Factor	**Regression Type**	**Years of OO Experience**	**Number of OO Projects**
Basic Level	Linear	1.2248	1.1639
	Quadratic	1.2243	1.1503
	Curvilinear	.6996	.7224
Object Level	Linear	.9474	.8730
	Quadratic	.8610	.7508
	Curvilinear	.5152	.6301
System Level	Linear	.9941	.9002
	Quadratic	.9825	.8863
	Curvilinear	.5251	.6249
All OO	Linear	.8872	.7917
	Quadratic	.8616	.7453
	Curvilinear	.3367	.3274
Procedural	Linear	.7812	.7411
	Quadratic	.7536	.6916
	Curvilinear	.5492	.5507

Table 14

Comparison of Response Choices

Class

Response	**N**	**Years of OO exp**	**# OO projects**
Don't Know	9	0.278	0.222
1.00	7	2.310	1.286
1.67	1	1.000	1.000
2.00	7	1.524	1.286
2.50	3	0.417	1.667
2.67	1	2.000	1.000
3.00	11	2.212	1.727
3.33	19	4.290	3.368
3.67	4	4.793	4.750
4.00	26	3.914	2.962
4.33	21	4.437	4.381
4.50	2	5.250	6.500
4.67	15	6.300	4.533
5.00	5	5.300	4.600

Subroutine

Response	**N**	**Years of OO exp**	**# OO projects**
1.00	22	5.136	4.136
1.50	5	4.700	5.400
2.00	26	5.356	4.692
2.50	15	3.333	3.333
3.00	15	3.500	2.400
3.50	11	2.849	2.545
4.00	14	3.250	2.070
4.50	19	1.527	0.842
5.00	4	0.228	0.750

Table 15

Plateaus using Years of OO Experience

Concept	**# of Plateaus**	**Coinciding Plateaus**
Abstraction	2	A
Collaboration	2	A
Components	2	A
Converting Things into Objects	2	A
Information Hiding	2	A
Interaction – OO	2	A
Layer	2	A
Message Passing	2	A
Method	2	A
Noun-verb Analysis	2	A
Object Model	2	A
Things as Objects	2	A
Attribute	1	B
Class	1	B
Encapsulation	1	C
Instantiation	1	B
OO development	1	B
Relationship	1	C

Table 16

Plateaus using Number of OO Projects

Concept	# of Plateaus	Coinciding Plateaus
Abstraction	2	A
Components	2	B
Converting Things into Objects	2	A
Inheritance	2	B
Interaction – OO	2	B
Layer	2	B
Message Passing	2	B
Noun-verb Analysis	2	C
OO development	2	
Relationship	2	B
Encapsulation	1	D
Polymorphism	1	D
Things as Objects	1	C

Table 17

Percentage of "Don't Know" Responses

Construct	**Concept**	**%**
Object	Collaboration	30.53%
	Inheritance	21.37%
	Method	22.14%
	OO Development	21.37%
	Polymorphism	24.81%
System	Components	21.63%
	Framework	49.24%
	Layer	24.81%

Table 18

Summary of Findings

Hypothesis	Description	Results
H1	Basic Level Construct	Multiple plateaus supported using time on x-axis.
	Object Level Construct	Single plateaus on both time and projects.
	System Level Construct	Multiple plateaus supported on number of projects on x-axis.
H2	Basic Level Construct	Supported for Functions/Things as Objects
	Object Level Construct	Supported for Class/Subroutine, Functional Decomposition/Noun-Verb Analysis
	System Level Construct	Supported for Data Model/Object Model, Interaction, Monolithic/Layer
H3	Basic Level Construct	Supported for Functions/Things as Objects, Functions/Converting Things into Objects
	Object Level Construct	Not supported

table continues

Table 18

Summary of Findings

Hypothesis	Description	Results
H3	System Level Construct	Supported for Abstraction/Data Modification, Framework/Linear Structure, Object-Oriented Interaction/Input-Process-Output, Layer/Data Modification, and Object Model/Data Model
H4	Basic Level Construct	Supported for Converting Things into Objects
	Object Level Construct	Supported for Noun-Verb Analysis
	System Level Construct	Supported for Abstraction, Components, Interaction, Layer and Message Passing

Table 19

Construct/Concept Plateaus

Construct	Concept	Multiple Plateaus Y= Years, P= Projects, B=Both	Single Plateaus Y= Years, P= Projects, B=Both
Basic	Converting things into objects	B	
	Things as objects	Y	P
Object	Attribute		Y
	Class		Y
	Collaboration	Y	
	Encapsulation		B
	Inheritance	P	
	Instantiation		Y
	Method	Y	
	Noun-verb analysis	B	
	OO Development	P	Y
	Polymorphism		P

table continues

Table 19

Construct/Concept Plateaus

Construct	**Concept**	**Multiple Plateaus** **Y= Years, P= Projects, B=Both**	**Single Plateaus** **Y= Years, P= Projects, B=Both**
System	Abstraction	B	
	Components	B	
	Information hiding	Y	
	Interaction	B	
	Layer	B	
	Message passing	B	
	Object model	Y	
	Relationship	P	Y

References

Abdolmohammadi, M. J., & Shanteau, J. (1992). Personal attributes of expert auditors. Organizational Behavior and Human Decision Processes, 53, 158-172.

Abelson, R.P. (1976). Script processing in attitude formation and decision making. In J. S. Carroll, & J. W. Payne (Eds.), Cognition and Social Behavior. Hillsdale, NJ: Lawrence Erlbaum Associates.

Adelson, B. (1981). Problem solving and the development of abstract categories in programming languages. Memory and Cognition, 9, 422-433.

Adelson, B. (1984). When novices surpass experts: The difficulty of a task may increase with expertise. Journal of Experimental Psychology: Learning, Memory and Cognition, 10, 483-495.

Aldenderfer, M. S., & Blashfield, R. K. (1984). Cluster Analysis. Newbury Park, CA: Sage Publications.

Anderson, J. C., & Gerbing, D. W. (1991). Predicting the performance of measures in a confirmatory factor analysis with a pretest assessment of their substantive validities. Journal of Applied Psychology, 76, 732-740.

Anderson, J. R. (1982). Acquisition of cognitive skill. Psychological Review, 89, 369-406.

Anderson, J. R. (1987). Methodologies for studying human knowledge. Behavioral and Brain Sciences, 18, 467- 505.

Anderson, J. R. (1993). Rules of the Mind. Hillsdale, NJ: Erlbaum Associates, Inc.

Anzai, Y., & Simon, H. A. (1979). The Theory of Learning by Doing, *Psychological Review*, 86, 124-140.

Axelrod, R. (1976). Structure of decision: The Cognitive Maps of Political Elites. Princeton, NJ: Princeton University Press.

Babbie, E. R. (1973). The Practice of Social Research. Belmont, CA: Wadsworth Publishing Company, Inc.

Babbie, E. R. (1979). The Practice of Social Research.(2nd ed.). Belmont, CA: Wadsworth Publishing Company, Inc.

Bagozzi, R. P., Yi, Y., & Phillips, L. W. (1991). Assessing construct validity in organizational research. Administrative Science Quarterly, 36, 421-458.

Baldwin, T. T., Magjuka, R. J., & Loher, B. T. (1991). The perils of participation: Effects of choice of training on trainee motivation and learning. Personnel Psychology, 44, 51-65.

Bartlett, F. C. (1932). Remembering: A Study in Experimental and Social Psychology. Cambridge, England: University Press.

Billett, S. (1994). Situating learning in the workplace: Having another look at apprenticeships. Industrial and Commercial Training, 26, 9-16.

Blashfield, R. K. (1980). The growth of cluster analysis: Tryon, Ward, and Johnson. Multivariate Behavioral Research, 15, 439-458.

Boland, R.J., Tenkasi, R.V., & Te'eni, D. (1994). Designing information technology to support distributed cognition. Organization Science, 5, 456-475.

Bolton, R.N., Chapman, R.G. & Zych, J.M. (1990). Pretesting alternative survey administration designs. Applied Marketing Research, 30, (Third Quarter 1990), 8-13.

Bougon, M., Weick, K., & Binkhorst, D. (1977). Cognition in organizations: An analysis of the Utrecht jazz orchestra. Administrative Science Quarterly, 22, 606-639.

Briggs, G. E. (1954). Acquisition, extinction and recovery functions in retroactive inhibition. Journal of Experimental Psychology, 47, 285-293.

Bruce, R. W. (1933). Conditions of transfer of training. Journal of Experimental Psychology, 16, 343-361.

Bryan, W. L., & Harter, N. (1897). Studies in the physiology and psychology of the telegraphic language. Psychological Review, 4, 27 - 53.

Bryan, W. L., & Harter, N. (1899). Studies on the telegraphic language. The acquisition of a hierarchy of habits. Psychological Review, 6, 345 - 375.

Campbell, R. L., Brown, N. R., & diBello, L. A. (1992). The programmer's burden: Developing expertise in programming. In R.R. Hoffman (Ed.) The Psychology of Expertise: Cognitive Research and Empirical A.I. New York: Springer-Verlag,

Cassidy, W. (1997). Good skilled help is hard to find, ITAA study says of IT employees. TrafficWorld, 3, 41.

Colley, A. M., & Beech, J. R. (1989). Acquiring and performing cognitive skills. In A. M. Colley & J. R. Beech (Eds.) Acquisition and Performance of Cognitive Skills. (pp. 1-16). Chichester, UK: John Wiley and Sons,.

Conrad, R., & Hull, A. J. (1968). The preferred layout for numerical data entry sets. Ergonomics, 11, 165-173.

Conway, T., & Wilson, M. (1988). Psychological studies of knowledge representation. In G. A. Ringland & D. A. Duce (Eds.) Approaches To Knowledge Representation: An Introduction. Letchworth: Research Studies Press.

Cronbach, L. J., & Gleser,G. C. (1953). Assessing similarity between profiles. Psychological Bulletin, 50, 456-473.

Culbert, S. (1996). Mind-Set Management. Oxford, UK: Oxford University Press.

Cureton, E., & D'Agostino, R. (1983). Factor Analysis: An Applied Approach. Hillsdale, NJ: Lawrence Erlbaum Associates.

Detienne, F. (1985). Programming expertise and program understanding. Ninth Congress of the International Ergonomics Association, Bournemouth, UK. 1985.

Detienne, F. (1990). Difficulties in designing with an object-oriented language: An empirical study. Human Computer Interaction, 5, 971-976.

Detienne, F. (1995). Design strategies and knowledge in object-oriented programming: Effects of experience. Human Computer Interaction, 10, 129-169.

Due, R. T. (1993). Object-oriented technology: The economics of a new paradigm. Information Systems Management, 10, 59-73.

Dumas, J., & Parsons, P. (1995). Discovering the way programmers think about new programming environments. Communications of the ACM, 38, 45-57.

Eason, R. L., Smith, T. L, & Plaisance, E. (1989). Effects of proactive interference on learning the tennis backhand stroke. Perceptual and Motor Skills, 68, 923-930.

Eaton, T. V., & Gatian, A. W. (1996). Organizational impacts of moving to object-oriented technology. Journal of Systems Management, (March-April, 1996), 18-24.

Eden, C., Ackerman, F., & Cropper, S. (1992). The analysis of causal maps. Journal of Management Studies, 29, 309-324.

Eden, C., Jones, S., Sims, D., & Smithin, T. (1981). The intersubjectivity of issues and issues of intersubjectivity. Journal of Management Studies, 18, 37 - 47.

Ericcson, K. A., & Simon, H. A. 1980. Protocol Analysis: Verbal Reports as Data. Cambridge, MA: MIT Press.

Everitt, B. (1980). Cluster Analysis. London: Halsted Press.

Fiol, C. M., & Huff, A. S. (1992). Maps for managers: where are we? Where do we go from here? Journal of Management Studies, 29, 267-285.

Fitts, P. M. (1964). Human Performance. Belmont, CA: Brooks/Cole.

Ford, J. D., & Hegarty, W. H. (1984). Decision maker's beliefs about the causes and effects of structure: An exploratory study. Academy of Management Journal, 27, 271-291.

Fossum, J. A., Arvey, R. D., Paradise, C. A., & Robins, N.E. (1986). Modeling the skill obsolescence process: A psychological/Economic integration. Academy of Management Review, 11, 362-374.

Fowler, F. (1993). Survey Research Methods. (2nd ed.) Thousand Oaks, CA: Sage Publications, Inc.

Gagne, R. M., & Foster, H. (1949). Transfer of training from practice on components in a motor skill. Journal of Experimental Psychology, 39, 47-68.

Gerencher, K. (1999). How to say 'farewell.' InfoWorld, 21, 83-84.

Gibson, E. (1991). Flattening the learning curve: Educating object-oriented developers. Journal of Object Oriented Programming, 3, 24-29.

Gibson, E. J. (1940). A systematic application of the concepts of generalization and differentiation to verbal learning. Psychological Review, 47, 196-229.

Gist, M., Rosen, B., & Schwoerer, C. (1988). The influence of training method and trainee age on the acquisition of computer skills. Personnel Psychology, 41, 255-265.

Goodner, S. T. (2000). How to create a culture of retention. Human Resource Professional, 13, 13-15.

Gordon, A. D. (1999). Classification. Boca Raton, FL: Chapman and Hall.

Gordon, P.C. (1992). In H. H. Brownell & Y. Joanette (Eds.) Narrative discourse in neurologically impaired and normal aging adults. San Diego, CA: Singular Publishing Group.

Guadagnoli, E., & Velicer, W. F. (1988). Relation of sample size to the stability of component patterns. Psychological Bulletin 103, 265-275.

Guttman, M. K., & Matthews, J. R. (1992). Managing a large project: Case study of a long-term project at NCR. Object Magazine, 2, 50-55.

Harrel, E. C., & McLean, E. R. (1985, June). The effects of using a nonprocedural computer language on programmer productivity. MIS Quarterly, 109-119.

Heberlein, T. A., & Baumgartner, R. (1978). Factors affecting response rates to mailed surveys: A quantitative analysis of the published literature. American Sociological Review, 43, 447-462.

Hendrick, H. W. (1983). Pilot performance under reversed control stick conditions. Journal of Occupational Psychology, 56, 297-301.

Hilgard, E. R., & Bower, G. H. (1970). Applicability of Models and Learning Theories. In W. S. Sahakian (Ed.) Psychology of Learning: Systems, Models, and Theories (pp. 384-387). Chicago, IL: Markham Psychology Series.

Holland, J. H., Holyoak, K. J., Nisbett, R., & Thagard, P. R. (1989). Induction: Processes of Inference, Learning, and Discovery. Cambridge: MIT Press.

Huck, S. W. (2000). Reading Statistics and Research. New York: Addison Wesley Longman, Inc.

Huff, A. (1990). Mapping the process of problem reformulation: Implications for understanding strategic thought. In A. Huff (Ed.) Mapping Strategic Thought. New York: John Wiley and Sons.

Hunt, E., & Lansman, M. (1986). Unified Model of Attention and Problem Solving. Psychological Review, 93, 446-461.

Jobber, D., & Sanderson, S. (1983). The effects of prior letter and colored survey paper on mail survey response rates. Journal of the Market Research Society, 25, 339-349.

Johnson-Laird, P.N. (1989). Mental models. In M. I. Posner (Ed.) Foundations of Cognitive Science (pp. 469-499). Cambridge, MA: MIT Press.

Kahney, J. H. (1983). Problem solving by novice programmers. In The Psychology of Computer Use: A European Perspective. London: Academic Press.

Kaufman, L., & Rousseeuw, P. J. (1990). Finding Groups in Data: An Introduction to Cluster Analysis. New York: Wiley.

Knoke, D., & Kuklinski. J. H. (1982). Network Analysis, Newbury Park, CA: Sage Publications.

Kraiger, K., Ford, K. J., & Salas, E. (1993). Application of cognitive, skill based and affective theories of learning outcomes to new methods of training evaluation. Journal of Applied Psychology, 78, 311-328.

Koubek, R. J., Salvendy, G., Dunsmore, H. E., & LeBold, W. K. (1989). Cognitive issues in the process of software development: Review and reappraisal. International Journal of Man-Machine Studies, 30, 171-191.

Larkin, J. H. (1981). Enriching formal knowledge: A model of learning to solve textbook physics problems. In J. R. Anderson (Ed.), Cognitive skills and their acquisition (pp. 311-334). Hillsdale, NJ: Erlbaum.

Lawley, D. N., & Maxwell, A. E. (1971). Factor analysis as a statistical method. London: Butterworth and Co.

Lee, A., & Pennington, N. (1994). The effects of paradigm on cognitive activities in design. International Journal on Human-Computer Studies, 40, 577-601.

Leonard, D. (1995). Wellsprings of Knowledge. Cambridge, MA: Harvard Business School Press.

Lewis-Beck, M. (1980). Applied Regression, An Introduction. Newbury Park, CA: Sage Publications, Inc.

Lincoln, Y. S., & Guba, E. G. (1985). Naturalistic Inquiry. Newbury Park, CA: SAGE Publications.

Liu, C., Goetze, S., & Glynn, B. (1992). What contributes to successful object-oriented learning? In A. Papacke (Ed.), Conference on Object-Oriented Programming Systems, Languages, and Applications (OOPSLA) 1992 (pp. 77-86). Vancouver, BC: ACM Press.

Lorr, M. (1983). Cluster Analysis for Social Scientists. San Francisco, CA: Jossey-Bass Publishers.

Manns, M. L., & Nelson, H. J. (1996, November-December). Retraining procedure-oriented developers: An issue of skill transfer. Journal of Object-Oriented Programming, 6-10.

Markus, M. L. (1990). Toward a "critical mass" theory of interactive media. In J. Fulk & C. Steinfield (Eds.) Organizations and Communication Technology (pp. 194-218). Newbury Park, CA: Sage Publications.

Mayer, R. E. (1983). Thinking, Problem Solving, Cognition. New York: W.H. Freeman and Company.

McCray, P., & Blakemore, T. (1989). A guide to learning curve technology to enhance performance prediction in vocational evaluation. Research Utilization Report. Menomonie, WI: University of WI, Stout, Research and Training Center.

McGeoch, J. A. (1952). The Psychology of Human Learning. (2nd ed.) New York: Longmans, Green..

McKeithen, K., Reitman, J., Rueter, H., & Hirtle, S. (1981). Knowledge organization and skill differences in computer programmers. Cognitive Psychology, 13, 307-325.

Melton, A. W., & Irwin, J. (1940). The influence of the degree of interpolated learning on retroactive inhibition and the overt transfer of specific responses. American Journal of Psychology, 53, 173-203.

Mey, M. (1982). The cognitive paradigm: cognitive science, a newly explored approach to the study of cognition applied in an analysis of science and scientific knowledge. Boston: D. Reidel Publishing Co..

Minsky, M. (1975). A framework for representing knowledge. In P. H. Winston (Ed.), The Psychology of Computer Vision. New York: McGraw-Hill.

Murphy, G. L., & Wright, J. C. (1984). Changes in the conceptual structure with expertise: differences between real world experts and novices. Journal of Experimental Psychology: Learning, Memory and Cognition, 10, 144-155.

Nachmias, D., & Nachmias, S. (1981). Research Methods in the Social Sciences. New York: St. Martin's Press.

Narayanan, V. K., & Fayhey, L. (1990). Evolution of revealed causal maps during decline: A case study of Admiral. In A. Huff (Ed.) Mapping Strategic Thought (pp. 109-133). London: John Wiley and Sons.

Nelson, H. J., Armstrong, D., & Ghods, M. (in press). Teaching old dogs new tricks. Communications of the ACM.

Nelson, H. J., Irwin, G., & Monarchi, D. E. (1997). Journeys up the mountain: Different paths to learning object-oriented programming. Accounting, Management and Information Technology, 7, 53-85.

Nelson, K. M., Nadkarni, S., Narayanan, V. K., & Ghods, M. (200). Understanding software operations support expertise: A causal mapping approach. MIS Quarterly, 24, 475-507.

Newell, A,. & Simon, H. A. (1958). Elements of a theory of human problem solving. Psychological Review, 65, 115-166.

Newell, A., & Simon, H. A. (1972). Human problem solving. Englewood Cliffs, NJ: Prentice Hall.

Nunnally, J. C. (1967). Psychometric Theory. New York: McGraw Hill.

Nunnally, J. C., & Bernstein, I. H. (1994). Psychometric Theory (3rd ed.). New York: McGraw-Hill.

Ormerod, T. (1990). Human cognition and programming. In J. Hoc, T. Green, R. Samurcay, & D. Gilmore (Eds.) Psychology of Programming (pp. 63-82). London: Academic Press.

Osgood, C. E. (1949). The similarity paradox in human learning: A resolution. Psychological Review, 56, 132-143.

Page-Jones, M. (1994). Education and training for real object-oriented shops. Journal of Object-oriented Programming, 7, 51-53.

Payne, S. (1951). The Art of Asking Questions. Princeton, NJ: University Press.

Pei, D., & Cutone, C. (1995). Object-oriented analysis and design. Information Systems Management, 12, 54-60.

Pennington, N. (1987). Stimulus structures and mental representations in expert comprehension of computer programs. Cognitive Psychology, 19, 295-341.

Pennington, N., Lee, A., & Rehder, B. (1995). Cognitive activities and levels of abstraction in procedural and object-oriented design. Human Computer Interaction, 10, 171-226.

Pool, R. (1997). Beyond Engineering: How Society Shapes Technology. NY: Oxford University Press.

Relmann, P., & Chi, M. T. (1989). Human expertise. In K. J. Gilhooly (Ed.) Human and Machine Problem Solving (pp. 161-191). Hillsdale, NJ: Lawrence Erlbaum Associates.

Rist, R. (1989). Schema creation in programming. Cognitive Science, 13, 389- 414.

Rossi, P. H., Wright, J. D., & Anderson, A. B. (1983). Handbook of Research Methods. Orlando, FL: Academic Press Inc.

Rosson, M., & Alpert, S. R. (1990). The cognitive consequences of object-oriented design. Human Computer Interaction, 5, 345-379.

Rosson, M., & Carroll, J. (1990). Climbing the Smalltalk mountain. SIGCHI Bulletin, 21, 76-79.

Rosson, M., & Gold. (1989). Problem-solution mapping in object oriented design. In Conference on Object-Oriented Programming Systems, Languages, and Applications (OOPSLA) 1987 (pp. 7-10). New York: ACM Press.

Ryan, S. D. (1999). A model of the motivation for IT retraining. Information Resources Management Journal, 12, 24-32.

Sahakian, W. S. (1970). Psychology of Learning: Systems, Models, and Theories. Chicago: Markham Psychology Series.

Saltz, E. (1971). The Cognitive Bases of Human Learning. Homewood, IL: The Dorsey Press.

Schank, R. C., & Abelson, R. P. (1977). Scripts, Plans, Goals and Understanding: An Inquiry into Human Knowledge Structures. Hillsdale, NJ: Lawrence Erlbaum Associates.

Schenk, K. D., Vitalari, N. P., & Davis, K. S. (1998). Differences between novice and expert systems analysts: What do we know and what do we do? Journal of Management Information Systems, 15, 9-50.

Schmidt, R. A. (1988). Motor Control and Learning: A Behavioral Emphasis. Champaign, IL: Human Kinetics.

Scholtz J., & Wiedenbeck, S. (1990). Learning second and subsequent programming languages: A problem of transfer. International Journal of Human-Computer Interaction, 2, 51-72.

Scholtz J., & Wiedenbeck, S. (1992). The use of unfamiliar programming languages by experienced programmers. In A. Monk, D. Diaper, & M. Harrison (Eds.) People and Computers VII. Cambridge, England: Cambridge University Press.

Schuman, H., & Presser, S. (1981). Questions and Answers in Attitude Surveys. New York: Academic Press.

Shanteau, J. (1992). Competence in experts: The role of task characteristics. Organizational Behavior and Human Decision Processes, 53, 252-266.

Shanteau, J. (1987). Psychological characteristics of expert decision makers. In J. L. Mumpower, O. Renn, L. D. Phillips, & V. R. Uppuluri (Eds.), Expert Judgment and Expert Systems (pp. 289-304). Berlin: Springer-Verlag.

Shneiderman, B. (1976). Exploratory experiments in programmer behavior. International Journal of Computer and Information Sciences, 5, 123-143.

Siegel, S. (1956). Nonparametric Statistics for the Behavioral Sciences. New York: McGraw-Hill Book Company, Inc.

Siipola, E. M., & Israel, H. E. (1933). Habit-interference as dependent upon stage of training. American Journal of Psychology, 45, 205-227.

Simon, J. L., & Burstein, P. (1985). Basic Research Methods in Social Science. New York: Random House.

Singley, M. K., & Anderson, J. R. (1989). The Transfer of Cognitive Skill. Cambridge: Harvard University Press.

Sokal, R., & Michener, C. (1958). A statistical method for evaluating systematic relationships. University of Kansas Scientific Bulletin, 38, 1409-1438.

Soloway, E., & Ehrlich, K. (1984). Empirical studies of programming knowledge. IEEE Transactions on Software Engineering, 10, 595-609.

Spohrer, J.C., & Soloway, E. (1986). Novice mistakes: Are the folk wisdoms correct? Communications of the ACM, 29, 624-632.

Steele, T. J., Schwendig, W. L., & Kilpatrick, J. A. (1992). Duplicate responses to multiple survey mailings: A problem? Journal of Advertising Research, 37, 26-34.

Stone, E. (1978). Research Methods in Organizational Behavior. Glenview: Scott, Foresman and Co.

Sudman, S. (1976). Applied Sampling. New York: Academic Press, Inc.

Sudman, S., & Bradburn, N. M. (1982). Asking Questions. San Francisco: Jossey-Bass Inc., Publishers.

Tarpy, R. M., & Mayer, R. E. (1978). Foundations of Learning and Memory. Glenview, IL: Scott, Foresman and Company.

Thurstone, L. L. (1919). The learning curve equation. Psychological Monographs, 26, 1-51.

Tomaskovic-Devey, D., Leiter, J., & Thompson, S. (1994). Organisational survey non-response. Administrative Science Quarterly, 39, 439-457.

Travers, R. (1963). Essentials of Learning: An Overview for Students of Education. New York: Macmillan Company.

Underwood, B. J. (1957). Interference and forgetting. Psychological Review, 64, 49-60.

Vessey, I., & Conger, S. A. (1994). Requirements specification: Learning object, process and data methodologies. Communications of the ACM, 37, 102 - 113.

Villeneuve, A.O., & Fedorowicz, J. (1997). Understanding expertise in information systems design, or, What's all the fuss about objects? Decision Support Systems, 21, 111-131.

Vitalari, N. P. (1985). Knowledge as a basis for expertise in systems analysis: An empirical study. MIS Quarterly, 9, 221-240.

Ward, J. (1963). Hierarchical grouping to optimize an objective function. Journal of the American Statistical Association, 58, 236-244.

Weiser, M., & Shertz, J. (1983). Programming problem representation in novice and expert programmers. International Journal of Man Machine Studies, 19, 391-398.

Williams, W., Lance, G.N., Dale, M.B., & Clifford, H.T. (1971). Controversy concerning the criteria for taxometric strategies. Computer Journal, 14, 162-165.

Wu, Q., & Anderson, J. R. (1991). Knowledge transfer among programming languages. Proceedings of the 13th Conference of the Cognitive Science Society. Lawrence Erlbaum Associates.

Yammarino, F. J., Skinner, S., & Childers, T. L. (1991). Understanding mail survey response behavior. Public Opinion Quarterly, 55, 613-639.

Zmud, R. W., Anthony, W. P., & Stair, R. M. (1993). The use of mental imagery to facilitate information identification in requirements analysis. Journal of Management Information Systems, 9, 175-191.

Appendix A

Interview Guide

1. When a friend asks you "What is object-oriented (procedural) development?" what do you tell him or her?
2. What are the main ideas that define object-oriented (procedural) development?
   a. Explain each one.
3. What is the easiest concept to learn?
4. What is the most difficult concept to master?
5. How is that different from procedural (object-oriented) development?
6. Think of a time when you have been given a requirements document (for example, say to develop an accounting system) and asked to produce an object-oriented (procedural) solution. What was the first thing you did? How did you proceed from there?
7. What problems do you think experienced procedural developers have as they learn object-oriented development?
8. How could the transition from procedural to object-oriented development be made easier?
9. How do you know if an object-oriented (procedural) developer is an expert?

Appendix B

Glossary of Software Development Terms

Object Oriented Terms

- Abstract Class:
  - A class with no instances that is created only for the purpose of organizing a class hierarchy by defining methods and variables that will apply to lower-level classes.
  - Synonym: abstract data type; object type
- Abstraction:
  - The act of removing certain distinctions between objects so that we can see commonalties.
  - Determining the essential characteristics of an object.
  - Abstraction is one of the basic principles of object-oriented design, which allows for creating user-defined data types, known as objects.
  - Simplification of complex objects; we do this all the time naturally.
  - Real Life Example: We think of a "Cat" as a single thing without dwelling on the details of all the parts that go into making up a "Cat".
  - Software Example: A "database" is an abstract concept for a collection of tables, columns, constraints, etc.
- Aggregation:
  - Form a whole new object using other objects as the parts.

- Treat many objects as one object.
- Reduces complexity
- Describes a "has a" relationship
- The lifetime of the whole and its parts are independent of each other.
- Real Life Example: A garden exists independent of plants, but independent plants could exist as part of the garden.
- Software Example: A "company" object exists independent from "employee" objects.

➢ Architecture:

- The design of software that incorporates protocols and interfaces for interacting with other programs and for future flexibility and expandability.
- Synonym: Software architecture

➢ Association:

- A means to link object types (abstract classes) meaningfully.
- A relationship between different objects
- Can be navigable - one way or two ways.
- General and weak - no aggregation or ownership is implied.
- Real Life Example: A person rents and drives a vehicle. Neither owns or is part of the other, and there is no inheritance.
- Software Example: Several products are part of a sale.

➢ Attribute:

- An identifiable association that an object has with some other object or set of objects that is represented within an object type.
- For example the color of a car is an attribute of the car.

➢ Behavior:

- The processing that an object can perform, how the objects interact.
- The way objects are accessed or changed over time.

➢ Class:

- A user-defined data type that defines a collection of objects that share the same characteristics. An object, or class member, is one instance of the class. Concrete classes are designed to be instantiated. Abstract classes are designed to pass on characteristics through inheritance.
- A template for defining the methods and variables for a particular type of object.
- All objects of a given class are identical in form and behavior but contain different data in their variables.
- classes are blueprints; objects are instances of blueprints.
- Classes should have well-defined responsibilities; responsibilities should be balanced between classes, ie. there shouldn't be one overall class that does all the work with other classes just looking on and helping occasionally.

- Real Life Example: A house blueprint would be a class. One or more houses (objects) are instances of the blueprint.
- Software Example: A date class is the blueprint. Date objects would be MemorialDay, MyBirthday, etc.
- Synonyms: object, noun, thing, abstract data type, user defined data type

➢ Class Hierarchy:

- Classes are created in hierarchies.
- A tree structure representing the inheritance relationships among a set of classes. A class hierarchy has a single top node and may have any number of levels with any number of classes of each level.
- Hierarchy defines generalization/specialization or "Is A"
- used to simplify our view of the world.
- Real Life Example: A housecat IS A type of cat which IS A type of animal.
- Software Example: A circle is a type of shape that is a type of drawing object.

➢ Collaboration

- Classes work together to solve problems.
- Identifying collaborations between them can assist in better design.
- Real Life Example: A facilities manager and a teacher collaborate to schedule courses.

- Software Example: A date class and a calendar class work together to manage schedules.

➢ <u>Collection class:</u>

- A class that is designed to hold a variable number of references to other objects.
- Synonym: array; vector.

➢ <u>Component:</u>

- One element of a larger system. Software components are routines or modules within a larger system.
- Component software implies the use of small modules that allow applications to be quickly customized. Rather than launch the huge feature-rich applications in common use today, it is envisioned that users will run smaller, tighter applications in the future, calling in additional features (components) only when needed.

➢ <u>Composition</u>

- A "part of" relationship.
- The whole cannot exist without it's parts.
- Real Life Example: A human cannot exist without the heart, lungs, brain, etc.
- Software Example: A "rectangle" object cannot exist without the "points" that define it.

➢ <u>CRC Cards</u>

- OO design method that uses 3x5 cards.
- A card is made for each class containing responsibilities (knowledge and services) and collaborators (interactions with other objects).
- Cards provide a way for a group to work on an OO design together.

➢ <u>Design Patterns</u>

- Design patterns are class design solutions for common and well-understood problems.
- Real Life Example: Underwater diving - you need air, timers, known descent and ascent rates, etc.
- Software Example: Using proxies and skeletons in order to facilitate remote object communication.

➢ <u>Encapsulation</u>

- Making the data and processing within the object private, which allows the internal implementation of the object to be modified without requiring any change to the application that uses it.
- Data is packaged together with its corresponding procedures.
- The creation of self-sufficient modules that contain the data and the processing.
- We want to encapsulate a collection of related methods and data into a single cohesive object.
- Real Life Example: A computer monitor is fully encapsulated behind a case. Everything it needs to work is inside it.

- Software Example: A date class encapsulates all the methods and data it needs into a single unit.
- Synonym: information hiding, packaging.

➤ Incremental and Iterative Development

- Instead of delivering 100% of all the functionality at the end, a portion of the system is delivered over a smaller time period.
- In the first time increment you deliver 100% of 10% of the functions. In the next increment you would deliver 100% of the next 20% of the functions. Then 100% of the next 30%. So by the time you've reached the third increment 60% of the functions have been delivered at 100%.

➤ Information Hiding:

- Technique of making the internal details of a module inaccessible to other modules, protecting the module from outside interference, and protecting other modules from relying on details that might change over time.
- Keeping details of a routine private.
- Programmers only know what input is required and what outputs are expected.
- The details are hidden in an object.
- We want to protect our objects from having their state modified without their permission.

- Real Life Example: Nobody should be able to reach in and take your heart without permission.
- Software Example: You shouldn't be able to set the month of a date to 15 by going straight into the internals of a date class.
- Synonym: encapsulation; data hiding

➢ <u>Inheritance:</u>

- A mechanism whereby classes can make use of the methods and variables defined in all classes above.
- The ability of one class of objects to inherit properties from a higher class.
- Hierarchical structure
- Involves object relationships
- Synonym: delegation

➢ <u>Instance</u>

- When an instance is created, the initial values of its instance variables are assigned
- A member of a class: for example, "Lassie" in an instance of the class "dog.".
- Synonym: member, object

➢ <u>Instantiate</u>

- To create an object of a specific class

➢ <u>Interface:</u>

- Any communication surface that determines the signals that can pass through the surface.
- A message interface.
- Interfaces provide the ability to have modular components which can be plugged-in and unplugged as desired without adversely affecting a solution.
- Interfaces for a class should be complete, but should also be minimal.
- Real Life Example: Television sets have interfaces for power and video input.
- Software Example: An edit control exposes an interface allowing it to be plugged in or unplugged as desired.

➢ Layered Approach

-
- Synonym: layers of abstraction

➢ Library:

- A set of ready-made software routines (class definitions) that programmers use to write OO programs.

➢ Loose coupling:

- Classes should be able to stand on their own as much as possible.
- Real Life Example: A computer and a monitor are connected only through standard interfaces, not via internal wires.

- Software Example: A collection of shape objects can be drawn by a drawing class, but the drawing class doesn't need to know the internals of the shape objects.

➢ <u>Message:</u>

- A signal from one object to another that requests the receiving object to carry out one of its methods.
- Consists of three parts: the name of the receiver, the method it is to carry, and any parameters the method may require to fulfill its charge.
- Synonym: request

➢ <u>Method</u>

- A procedure defined within a class.
- The processing that an object performs.
- When a message is sent to an object the method is implemented.
- Object methods embody the behavior of the system.
- Synonym: responsibilities

➢ <u>Naturalness</u>

- With OO a system can be designed as familiar business functions, and the design can be carried all the way down to the programming level. In traditional systems the programs are decomposed into procedures that are more alien to the business model.

- You model your application system using the same verbiage that they use in their business process description.
- Models the way people understand reality
- OO development flexes and changes with the business process.

➢ <u>Object</u>

- A self-contained module of data and its associated processing.
- A software packet containing a collection of related methods and data.
- Independent programming modules
- Objects are the software building blocks of object technology.
- An instance of a class
- Synonym: class, noun, thing, entities, component

➢ <u>Object-Oriented Technology</u>

- Focus on the things
- A set of principles guiding software construction together with languages, databases and other tools that support those principles.
- Languages: Smalltalk, Java, C++

➢ <u>Object Model</u>

- A description of an object architecture, including the details of the object structure, interfaces between objects and other OO features and functions.
- An object-oriented description of an application.

➢ <u>Overriding</u>

- A special case of polymorphism in which the same name is given to a method or variable at 2 or more levels on the same branch of a class hierarchy.
- The name that is lowest in the hierarchy takes precedence, overriding the more generic definitions further up the hierarchy.

➢ <u>Parameter</u>

- An object or a data element that is included in a message to provide the requested method with information it needs to perform its task.

➢ <u>Polymorphism</u>

- The ability of a generalized request (message) to produce different results based on the object that it is sent to.
- The ability to hide different implementations behind a common interface, simplifying the communications among objects.
- Requires objects derived from a common base.
- Real Life Example: All vehicles on the road can be "told" to "go" by turning a light green.
- Software Example: All derived classes of a "shape" can be told to draw and each will in it's own way.

➢ <u>Redundancy</u>

- Have redundancy everywhere.
- Redundant data attributes all over the place because more concerned with behavior and not data.

- <u>Reusability</u>
  - The ability to use all or the greater part of the same programming code or system design in another application.
  - If object is too small (granular, fine) then can't reuse it.
- <u>Roles</u>
  - Classes should have specific roles in providing services.
  - The more specific and well defined the role, the more useful and re-usable the class.
  - The more roles there are, the more schizophrenic the class, and the less re-usable.
  - Real Life Example: A manager, receptionist, file clerk, developer, all have different and distinct roles. It is easier to find others who can "fill in" a position.
  - Software Example: A UI component, Business Object and Data Object all have different roles that are specific, and more reusable.
- <u>State</u>
  - Objects have state; complex objects may have more complex states; understanding the valid class states makes for classes with fewer failures.
  - Real Life Example: A person can be awake or asleep, running, walking or sitting.

- Software Example: A robot welding arm controller can be moving, on or off, in different positions.

- <u>Strong Cohesion</u>
  - All the elements of a class are closely related.
  - Real Life Example: A computer monitor contains only those parts that work toward it's purpose. It doesn't contain a hard drive or sound card.
  - Software Example: A Date class should not contain methods for computing sine and cosine.
- <u>System Qualities</u>
  - Things a developer is trying to maximize in an application.
  - Security, speed reliability, extensibility and flexibility
- <u>Unified Modeling Language (UML)</u>
  - A single standard OO design language.
  - Standard diagramming method.
  - Graphical representation method.
- <u>User Interface</u>
  - The combination of menus, screen design, keyboard commands, command language and online help, which create the way a user, interacts with a computer.
- <u>Use Cases</u>

- Breaking down requirements into user functions. Each use case is a transaction or sequence of events performed by the user. Use cases are studied to determine what objects are required to accomplish them and how they interact with other objects.
- Synonym: use-case analysis

Traditional/Structured/Procedural Terms

- Business Logic
  - The part of an application that performs the required data processing of the business.
  - It refers to the routines that perform the data entry, update, query and processing behind the scenes.
  - Synonym: Business rules
- Data flow diagram:
  - description of the data and the manual and machine processing performed on the date.
- Data model:
  - A description of the organization of a database.
  - Often created as an entity-relationship (ER) diagram. ER diagram describes the attributes of entities and the relationships among them.
- Functional Decomposition:
  - A technique for analyzing a set of requirements and designing a program to meet those requirements.

- An overall goal for the program is broken down into a series of steps to meet that goal.
- Each step is then decomposed into more elementary steps and so on.
- Each of the resulting components is programmed as a separate module.

➢ GL Definitions

- 1GL – machine languages
- 2GL – machine dependent assembly language
- 3GL – high level programming language
- 4GL – English-like language, commands that don't require traditional input-process output logic. They often have GUI's

➢ 3GL (3rd generation language)

- high level programming languages Fortran, COBOL, C, Basic

➢ 4GL (4th generation language)

- more advanced than traditional high level languages

➢ Input/output (I/O):

- Transferring data between the CPU and a peripheral device.
- Every transfer is an output from one device and an input into another.

➢ Library:

- A collection of programs or data files.
- A set of ready-made software routines (functions) for programmers.

- <u>Logic:</u>
  - Sequence of operations performed by the software, sequence of instructions in a program
  - Synonym: algorithm
- <u>Procedural programming</u>
  - requires programming discipline
  - must have a proper order of actions in order to solve problem
  - e.g. Fortran COBOL, Basic
  - also called 3GL
  - all the logic has to be explicitly programmed
  - focus on the processes
  - easy to read the code, can see the sequence of events
  - Synonym: linear; structured programming
- <u>Source code</u>
  - Programming statements and instructions that are written by a programmer.
  - What a programmer write but it is not directly executable by the computer. It must be converted into machine language by compilers, assemblers or interpreters.
  - Synonym: lines of code
- <u>Structured analysis</u>
  - includes data flow diagrams, data models,

- implementation independent graphical notation for documentation

➢ <u>Structured design</u>

  - design guidelines and recipes

➢ <u>Structured programming</u>

  - Techniques that impose a logical structure on the writing of a program.
  - A collection of techniques designed to increase the rigor of software development and to improve the quality of development systems.
  - Large routines broken down into smaller ones
  - Focused on the data items, data centered, data driven
  - Standardized
  - Goto is discouraged
  - Use walkthroughs
  - e.g. Pascal, Ada, dBase

➢ <u>Top down programming:</u>

  - Imposes hierarchical structure on design of program
  - Design starts at the highest level of an idea and works its way down to the lowest level of detail

➢ <u>User Requirements:</u>

  - The details and needs of the customer.
  - Usually in document form.

Appendix C

Scale Reliabilities

Concept	**Pilot Study Reliability**	**Study Reliability**
Abstraction	0.7100	0.8298
Attribute	**0.6795**	**0.6403**
Class	**0.6795**	**0.6375**
Class Hierarchy	0.1081	Eliminated
Collaboration	0.8011	N/A[1]
Components	0.7363	**0.6036**
Control	0.4417	Eliminated
Converting "things" into objects	0.7980	0.7924
Data Model	0.7014	**0.6743**
Data Modification	0.8398	0.7074
Encapsulation	0.7898	0.7912
Framework	0.7391	0.7632
Functional Decomposition	0.8347	0.8240
Functions	0.8097	0.7436
Information Hiding	**0.6866**	0.8468
Inheritance	0.7470	0.8716
Input-Process-Output	0.7918	**0.6428**

table continues

Appendix C

Scale Reliabilities

Concept	**Pilot Study Reliability**	**Study Reliability**
Instantiation	0.7294	0.8386
Interaction (OO)	0.7559	0.8541
Interaction (Procedural)	0.7498	**0.6610**
Interface	0.4563	Eliminated
Layer	0.7804	0.6238
Linear Flow	0.7485	0.8172
Linear Form	0.7923	**0.6795**
Linear Program	**0.6413**	0.7018
Linear Structure	0.7021	0.8486
Message Passing	0.7107	0.8210
Method	0.7866	**0.6642**
Monolithic	0.8011	0.7221
Noun/Verb Analysis	0.8686	0.7636
Object	0.7208	0.2732[2]
Object Model	0.7248	0.7712
OO Development	0.783	0.8039
Polymorphism	0.7533	0.7194

table continues

Appendix C

Scale Reliabilities

Concept	**Pilot Study Reliability**	**Study Reliability**
Procedural Programming	0.7595	Eliminated
Relationships	0.7363	0.7765
Structured Development	**0.6616**	Eliminated
Subroutine	0.7465	0.8026
Systems Development Life Cycle	0.7764	Eliminated
Things as objects	0.809	0.8799

[1] Only one item used.

[2] Variable was eliminated from study results.

Appendix C

Scale Reliabilities

Mindset	**Construct**	**Pilot Study Reliability**
Object-Oriented	Application Design	.7693
Object-Oriented	Design Characteristics	**.6227**
Object-Oriented	Design Relationships	.7250
Object-Oriented	Execution Characteristics	**.6555**
Object-Oriented	Execution Interaction	.7107
Object-Oriented	OO Characteristics	.8226
Object-Oriented	Analysis Techniques	.7472
Procedural	Application Design	.7485
Procedural	Design Characteristics	.7795
Procedural	Design Relationships	.7021
Procedural	Execution Characteristics	**.6962**
Procedural	Execution Interaction	.8055
Procedural	Analysis Techniques	.8217

table continues

Appendix C

Scale Reliabilities

Mindset	**Construct**	**Study Reliability**
Object-Oriented	Basic Level	.8949
Object-Oriented	Object Level	.9534
Object-Oriented	System Level	.9252
Procedural		.9580

table continues

Appendix C

Scale Reliabilities

Mindset	**Meta-Construct**	**Pilot Study Reliability**
Object-Oriented	Application	.7693
Object-Oriented	Design	.7631
Object-Oriented	Runtime	.7135
Object-Oriented	OO Characteristics	.8226
Object-Oriented	Analysis	.7472
Procedural	Application Design	.7485
Procedural	Design	.8365
Procedural	Runtime	.8528
Procedural	Analysis	.8217

Mindset	**Meta-Construct**	**Study Reliability**
Object-Oriented		.9685
Procedural		.9580

Appendix D

Form Loading Screen

Loading form.
Please wait... This will take about 60 seconds.

If the survey does not load - check your browser version.

It must have java script capability
(e.g. Internet Explorer v4.0 or higher, Netscape v4.7 or higher)

Appendix E

Survey Screen

SOFTWARE DEVELOPMENT SURVEY

The purpose of this study is to understand how people are approaching software development (and significant maintenance enhancements). In order to gather data that is a more accurate reflection of what actually happens in organizations, we are interested in collecting data from individuals out in the field. There are no right or wrong answers; we are interested in gathering information about a variety of approaches to software development.

Thank you for participating in this study. **Your individual answers will be kept completely confidential,** and will be used only as part of a summary of all responses gathered. Individual responses will not be singled out at any time. **An overall summary of the results will be made available to you and your organization** when statistical analyses are completed. Filling out this survey indicates that you are at least 18 years of age and that you are giving your informed consent to participate in this study.

Knowing your busy schedule, the survey has been designed to minimize the effort required to respond. The survey should take approximately 20 minutes. *Please be as complete as possible in responding to the questions*, as skipping questions or incomplete answers may invalidate your responses.

SECTION I:

Think about your most recent software development experiences and how you approach software development. Use that as your reference. Please indicate on a scale of 1-5 the extent that your approach to software development agrees with each statement.

Appendix E

Survey Screen

DK	**1**	**2**	**3**	**4**	**5**
Don't Know, no knowledge or experience in this area	Strongly Disagree	Disagree	Neither Disagree Nor Agree	Agree	Strongly Agree

_____1. Object-oriented development is concerned with the nouns (things) and the verbs (action) of business.

_____2. A class specifies the behavior of its instances.

_____3. Everything goes in one central location, or brain, and that logic runs the system.

_____4. You start finding some very high level objects, maybe discarding some, and go down deeper from there.

Appendix F

Thank You Screen

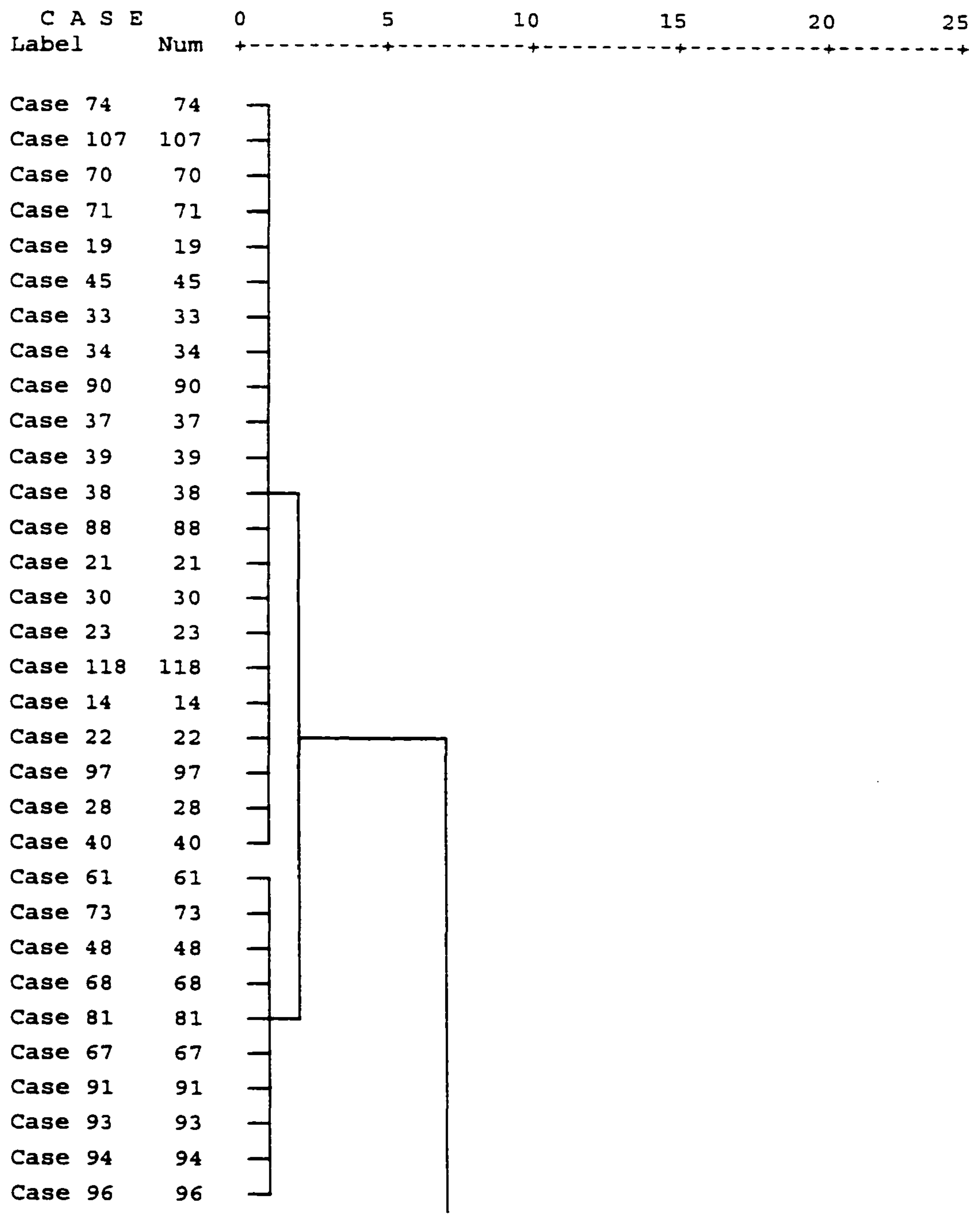
Thank you,

your response has been sent.

Please close your browser window now.

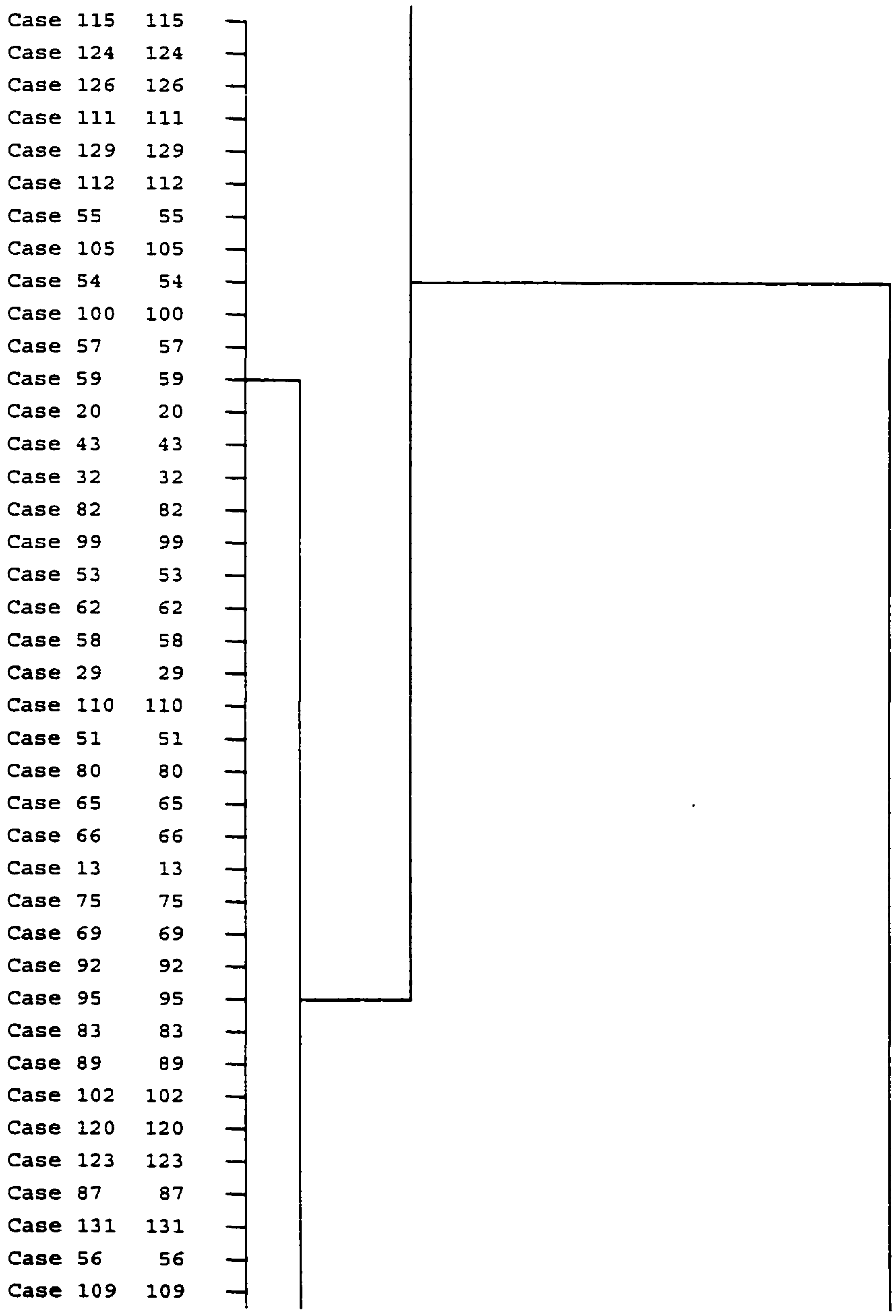
Appendix G

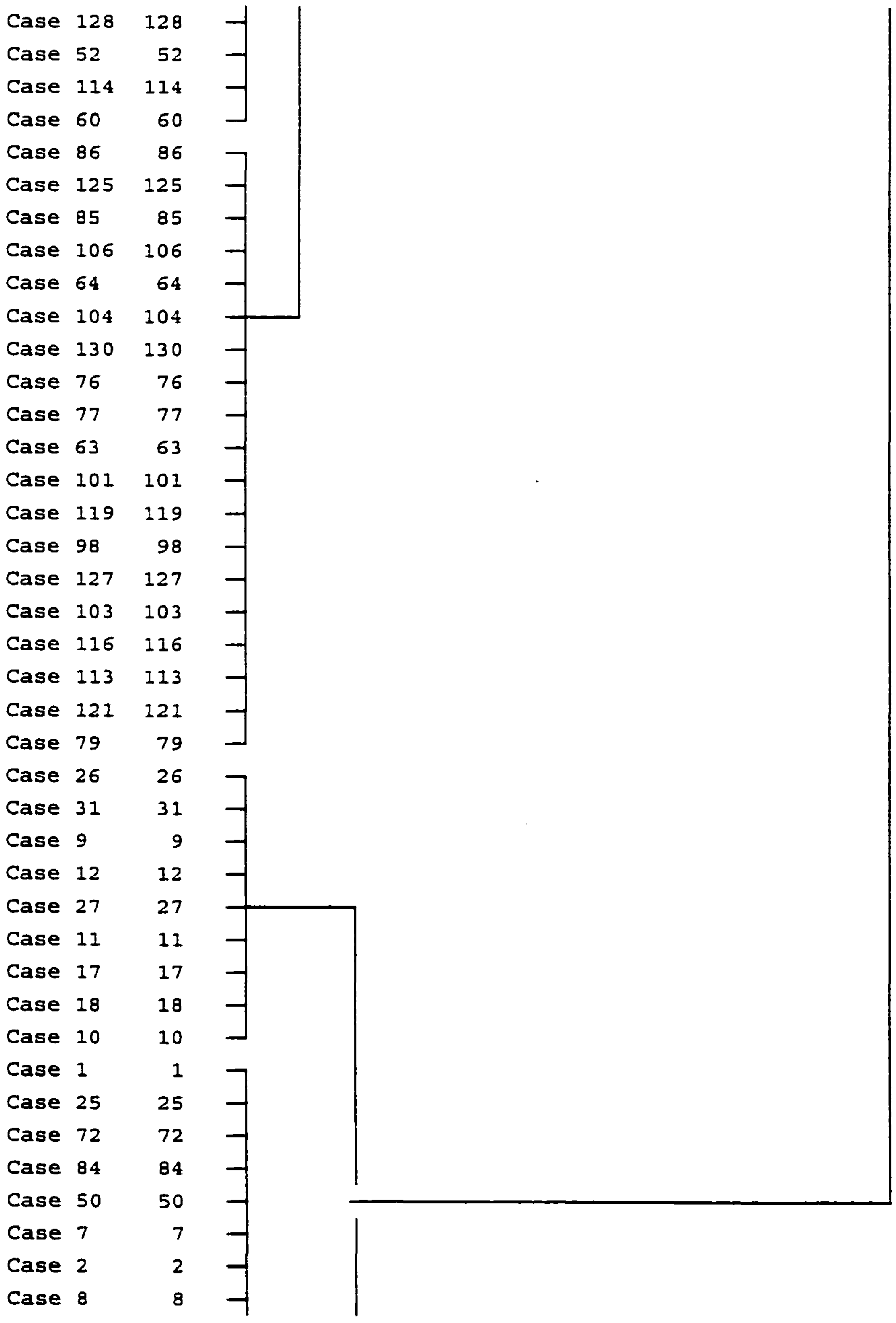
Hierarchical Cluster Analysis Dendrogram Using Ward's Method

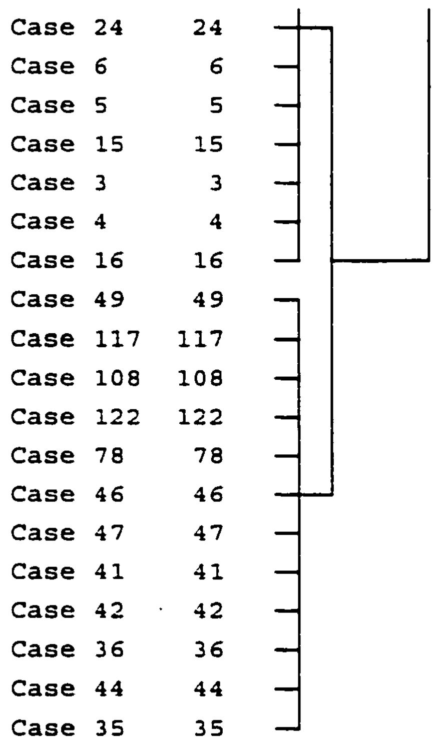


```
   C A S E         0         5        10        15        20        25
  Label      Num   +---------+---------+---------+---------+---------+

  Case 74     74
  Case 107   107
  Case 70     70
  Case 71     71
  Case 19     19
  Case 45     45
  Case 33     33
  Case 34     34
  Case 90     90
  Case 37     37
  Case 39     39
  Case 38     38
  Case 88     88
  Case 21     21
  Case 30     30
  Case 23     23
  Case 118   118
  Case 14     14
  Case 22     22
  Case 97     97
  Case 28     28
  Case 40     40
  Case 61     61
  Case 73     73
  Case 48     48
  Case 68     68
  Case 81     81
  Case 67     67
  Case 91     91
  Case 93     93
  Case 94     94
  Case 96     96
```


Case 115 115
Case 124 124
Case 126 126
Case 111 111
Case 129 129
Case 112 112
Case 55 55
Case 105 105
Case 54 54
Case 100 100
Case 57 57
Case 59 59
Case 20 20
Case 43 43
Case 32 32
Case 82 82
Case 99 99
Case 53 53
Case 62 62
Case 58 58
Case 29 29
Case 110 110
Case 51 51
Case 80 80
Case 65 65
Case 66 66
Case 13 13
Case 75 75
Case 69 69
Case 92 92
Case 95 95
Case 83 83
Case 89 89
Case 102 102
Case 120 120
Case 123 123
Case 87 87
Case 131 131
Case 56 56
Case 109 109


Case 128 128
Case 52 52
Case 114 114
Case 60 60
Case 86 86
Case 125 125
Case 85 85
Case 106 106
Case 64 64
Case 104 104
Case 130 130
Case 76 76
Case 77 77
Case 63 63
Case 101 101
Case 119 119
Case 98 98
Case 127 127
Case 103 103
Case 116 116
Case 113 113
Case 121 121
Case 79 79
Case 26 26
Case 31 31
Case 9 9
Case 12 12
Case 27 27
Case 11 11
Case 17 17
Case 18 18
Case 10 10
Case 1 1
Case 25 25
Case 72 72
Case 84 84
Case 50 50
Case 7 7
Case 2 2
Case 8 8


Case 24 24
Case 6 6
Case 5 5
Case 15 15
Case 3 3
Case 4 4
Case 16 16
Case 49 49
Case 117 117
Case 108 108
Case 122 122
Case 78 78
Case 46 46
Case 47 47
Case 41 41
Case 42 42
Case 36 36
Case 44 44
Case 35 35

Appendix H

Cluster Analysis Agglomeration Schedule

	Cluster Combined		Coefficients	Stage Cluster First Appears		Next Stage
Stage	Cluster 1	Cluster 2		Cluster 1	Cluster 2	
1	76	77	.000	0	0	63
2	63	65	.000	0	0	30
3	58	59	.000	0	0	14
4	48	49	.000	0	0	89
5	37	38	.000	0	0	106
6	21	22	.000	0	0	50
7	6	7	.000	0	0	45
8	1	3	.000	0	0	77
9	10	31	1.557E-02	0	0	74
10	51	52	3.519E-02	0	0	35
11	98	99	5.883E-02	0	0	78
12	129	130	8.378E-02	0	0	28
13	112	115	.110	0	0	58
14	58	60	.140	3	0	103
15	120	121	.174	0	0	62

table continues

Appendix J

Cluster Analysis Agglomeration Schedule

	Cluster Combined		Coefficients	Stage Cluster First Appears		Next Stage
Stage	Cluster 1	Cluster 2		Cluster 1	Cluster 2	
16	72	75	.208	0	0	67
17	43	50	.257	0	0	72
18	23	26	.312	0	0	54
19	13	16	.373	0	0	51
20	102	103	.433	0	0	83
21	15	17	.500	0	0	40
22	67	69	.570	0	0	86
23	95	96	.643	0	0	47
24	100	101	.718	0	0	83
25	81	92	.793	0	0	55
26	118	119	.869	0	0	62
27	2	20	.948	0	0	87
28	128	129	1.028	0	12	68
29	124	125	1.110	0	0	59
30	63	66	1.196	2	0	48

table continues

Appendix J

Cluster Analysis Agglomeration Schedule

	Cluster Combined		**Coefficients**	**Stage Cluster First Appears**		**Next Stage**
Stage	**Cluster 1**	**Cluster 2**		**Cluster 1**	**Cluster 2**	
31	86	87	1.282	0	0	57
32	64	73	1.375	0	0	73
33	107	114	1.470	0	0	69
34	116	117	1.569	0	0	65
35	46	51	1.673	0	10	72
36	70	78	1.778	0	0	67
37	27	32	1.884	0	0	80
38	18	29	1.995	0	0	61
39	47	57	2.106	0	0	53
40	4	15	2.219	0	21	56
41	106	108	2.338	0	0	58
42	105	113	2.457	0	0	69
43	56	62	2.576	0	0	71
44	104	109	2.699	0	0	88
45	6	14	2.834	7	0	94

table continues

Appendix J

Cluster Analysis Agglomeration Schedule

	Cluster Combined		**Coefficients**	**Stage Cluster First Appears**		**Next Stage**
Stage	**Cluster 1**	**Cluster 2**		**Cluster 1**	**Cluster 2**	
46	44	55	2.980	0	0	71
47	95	97	3.145	23	0	70
48	63	88	3.313	30	0	110
49	79	85	3.490	0	0	105
50	21	36	3.670	6	0	91
51	5	13	3.850	0	19	81
52	80	83	4.042	0	0	90
53	40	47	4.235	0	39	60
54	23	25	4.429	18	0	80
55	81	89	4.622	25	0	90
56	4	8	4.838	40	0	91
57	71	86	5.055	0	31	76
58	106	112	5.277	41	13	88
59	124	126	5.503	29	0	84
60	40	45	5.749	53	0	97

table continues

Appendix J

Cluster Analysis Agglomeration Schedule

	Cluster Combined		**Coefficients**	**Stage Cluster First Appears**		**Next Stage**
Stage	**Cluster 1**	**Cluster 2**		**Cluster 1**	**Cluster 2**	
61	18	24	5.999	38	0	92
62	118	120	6.252	26	15	99
63	76	93	6.509	1	0	115
64	123	127	6.779	0	0	84
65	116	122	7.049	34	0	99
66	19	35	7.321	0	0	96
67	70	72	7.595	36	16	73
68	128	131	7.870	28	0	107
69	105	107	8.147	42	33	109
70	94	95	8.440	0	47	78
71	44	56	8.741	46	43	85
72	43	46	9.043	17	35	79
73	64	70	9.361	32	67	86
74	10	41	9.686	9	0	112
75	82	91	10.031	0	0	95

table continues

Appendix J

Cluster Analysis Agglomeration Schedule

	Cluster Combined		Coefficients	Stage Cluster First Appears		Next Stage
Stage	Cluster 1	Cluster 2		Cluster 1	Cluster 2	
76	71	84	10.382	57	0	105
77	1	34	10.735	8	0	87
78	94	98	11.094	70	11	119
79	42	43	11.509	0	72	103
80	23	27	11.926	54	37	102
81	5	11	12.346	51	0	93
82	28	33	12.803	0	0	92
83	100	102	13.267	24	20	119
84	123	124	13.752	64	59	107
85	44	61	14.238	71	0	106
86	64	67	14.763	73	22	100
87	1	2	15.289	77	27	101
88	104	106	15.820	44	58	109
89	39	48	16.376	0	4	97
90	80	81	16.987	52	55	104

table continues

Appendix J

Cluster Analysis Agglomeration Schedule

	Cluster Combined		Coefficients	Stage Cluster First Appears		Next Stage
Stage	Cluster 1	Cluster 2		Cluster 1	Cluster 2	
91	4	21	17.631	56	50	101
92	18	28	18.304	61	82	102
93	5	12	19.010	81	0	108
94	6	54	19.729	45	0	108
95	68	82	20.533	0	75	110
96	19	30	21.340	66	0	114
97	39	40	22.259	89	60	116
98	9	53	23.220	0	0	113
99	116	118	24.193	65	62	123
100	64	74	25.185	86	0	117
101	1	4	26.257	87	91	112
102	18	23	27.336	92	80	114
103	42	58	28.439	79	14	116
104	80	90	29.626	90	0	115
105	71	79	30.842	76	49	117

table continues

Appendix J

Cluster Analysis Agglomeration Schedule

Stage	Cluster Combined Cluster 1	Cluster 2	Coefficients	Stage Cluster First Appears Cluster 1	Cluster 2	Next Stage
106	37	44	32.186	5	85	12
107	123	128	33.614	84	68	12
108	5	6	35.051	93	94	11
109	104	105	36.503	88	69	11
110	63	68	38.059	48	95	12
111	110	111	39.671	0	0	11
112	1	10	41.331	101	74	12
113	5	9	43.789	108	98	12
114	18	19	46.780	102	96	12
115	76	80	49.922	63	104	12
116	39	42	53.568	97	103	12
117	64	71	57.534	100	105	12
118	104	110	61.546	109	111	12
119	94	100	65.705	78	83	12
120	63	64	71.057	110	117	12

table continues

Appendix J

Cluster Analysis Agglomeration Schedule

	Cluster Combined		**Coefficients**	**Stage Cluster First Appears**		**Next Stage**
Stage	**Cluster 1**	**Cluster 2**		**Cluster 1**	**Cluster 2**	
121	37	39	76.803	106	116	127
122	1	5	82.566	112	113	125
123	104	116	92.204	118	99	128
124	63	76	104.631	120	115	126
125	1	18	123.398	122	114	127
126	63	94	148.351	124	119	129
127	1	37	184.084	125	121	129
128	104	123	227.383	123	107	130
129	1	63	313.476	127	126	130
130	1	104	650.000	129	128	0