MIS Quarterly

RESEARCH NOTE

REVOLUTION OR EVOLUTION? A COMPARISON OF OBJECT-ORIENTED AND STRUCTURED SYSTEMS DEVELOPMENT METHODS[1]

By: Sumit Sircar
Center for IT Management
University of Texas at Arlington
Arlington, TX 76019
U.S.A.
sircar@uta.edu

Sridhar P. Nerur
Department of COIS
Central Missouri State University
Warrensburg, MO 64093
U.S.A.
Nerur@cmsu1.cmsu.edu

Radhakanta Mahapatra
Department of Information Systems and Operations Management
University of Texas at Arlington
Arlington, TX 76019
U.S.A.
mahapatra@uta.edu

Abstract

This paper examines the changes engendered when moving from a structured to an object-oriented systems development approach and reconciles the differing views concerning whether this represents an evolutionary or revolutionary change. Author co-citation analysis is used to elucidate the ideational and conceptual relationships between the two approaches. The difference in conceptual distance at the analysis and design level compared to that at the programming level is explained using Henderson's framework for organizational change. The conceptual shift during analysis and design is considered architectural, whereas for programming it is deemed merely incremental. The managerial implications of these findings are discussed and suggestions for improving the likelihood of success in the adoption of object-oriented systems development methods are provided.

Keywords: IS development, structured development approach, object-oriented approach, software development methodologies, author co-citation analysis.

ISRL Categories: FA 01, FA 11, FC 22, FC 25, FC 28, FC 29.

Introduction

In recent years, the software development community has been captivated by the reported benefits of object-oriented (OO) technology. In their struggle to find a panacea for the problems that often plague information systems (IS) pro-

[1]Ron Weber was the accepting senior editor for this paper.

jects, many software developers believe that object-orientation, although not a silver bullet (Brooks 1987), will potentially alleviate some troublesome issues that characterize structured approaches. Increased productivity gains, fewer maintenance problems, faster software systems development, increased resilience to change, and enhanced quality of systems are some OO virtues extolled in the literature.

In spite of the perceived benefits of OO development, the adoption of OO methodologies has progressed slowly. A recent survey of IS managers revealed that 39% of organizations have adopted OO technology in some form. Nonetheless, OO development methodologies are used in only 5% of IS projects (Glass 1999).

IS development is a complex, costly, and high-risk endeavor. In order to manage complexity and mitigate risk, organizations invest heavily in tools, technologies, and methodologies associated with IS development. A change in an IS development method, therefore, may involve not only changing the tools and technologies used but also retraining developers to institutionalize the new method, and the costs can be high. Thus, a significant managerial issue is whether migration to a new development approach involves extensions and/or adaptations of an existing approach, or whether it requires a wholesale change in mindset.

In this regard, considerable controversy exists in the literature about the magnitude and nature of the differences between OO and structured systems development methods. Some authors believe that the OO approach is merely an evolution from the structured systems development approaches (Booch 1991; Li 1991; Meli 1994; Page-Jones and Weiss 1991; Parodi 1995). Others claim that OO requires an entirely new approach and mindset (Cargill 1995; Due 1993; Henderson-Sellers 1992; Henderson-Sellers and Constantine 1991; Lilly 1993, Pun and Winder 1991; Sarna and Febish 1995). Much of the controversy appears to stem from differing views of the conceptual structure of the field of software development and the nature of the change that the OO phenomenon entails. The objectives of the research reported here are to: (1) elucidate the ideational and conceptual relationships between the OO and structured methods using a rigorous research framework, (2) reconcile the differing views regarding the nature of the change represented by the OO approach, and (3) offer guidance to organizations involved in the transition to OO computing.

The following sections describe the conceptual model and propositions that underlie our work, the research methodology we used, and the results of data analysis. The implications of the findings are then discussed, followed by a summary of the highlights of the study in the concluding section.

Background

To cope with a change from structured development methods to OO methods, organizations must understand the nature of the change and the conceptual barriers that it presents. Without this understanding, attempts to migrate to OO may be doomed to failure. Organizations that are equipped with a good grasp of the OO phenomenon and its conceptual relationships to structured philosophies will be better able to make informed decisions on how to make the shift to OO.

In the subsections below, we first present a framework developed by Henderson (1996) to categorize technological change. We then briefly summarize the major features of the structured and OO approaches to systems development. Next we develop propositions about the types of change associated with moving from structured to OO system development approaches.

Categories of Change

Organizational change has been studied extensively in the management literature (see, for example, Kochan and Useem 1992). A change phenomenon may be categorized by the intensity of change it engenders. A system, process, or product that is the subject of change can be described in terms of its components and its archi-

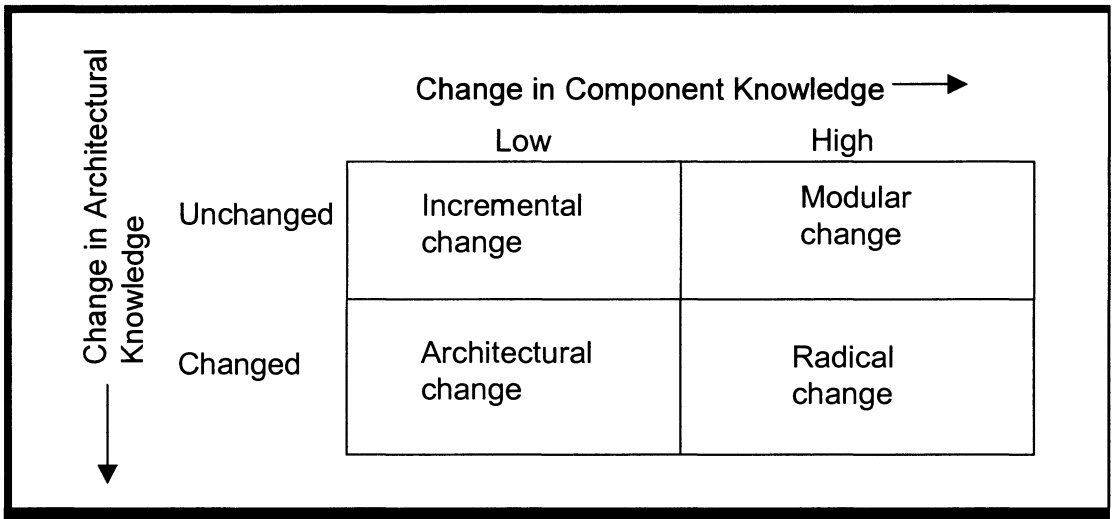

Figure 1. Changes in Architectural and Component Knowledge (Adapted from Figure 16.1 in Henderson 1996)

tecture. Components are parts of a system; together they constitute the whole. A system's architecture specifies how its components fit together to create the whole. Both component knowledge and architectural knowledge are required to develop a comprehensive understanding of a system. An innovation can be placed anywhere on a continuum from incremental (or evolutionary) to radical (or revolutionary) depending on the extent to which it affects the components or architecture of a system. Figure 1 describes the essence of this notion using a framework developed by Henderson (1996) and Henderson and Clark (1990).

Change is incremental when the components undergo minor changes but the architecture remains unchanged. Modular change causes radical change to how the components are designed, but the relationships or interaction among them remain unchanged. Architectural change modifies the structure without substantially altering the components of a system. Finally, radical change is characterized by dramatic changes in the components as well as in the architecture of the system.

To illustrate these notions, consider a room-cooling system comprised of large ceiling-mounted electric fans with motors that are hidden and insulated to dampen noise. Modifications of the motor or blade design to enhance power and/or efficiency cause changes to component knowledge and thus fall into the category of incremental change. A central air conditioning system is an example of radical change in cooling system technology. A portable fan represents an architectural change because, in this case, the component knowledge does not change but the way the components interact with each other (i.e., the architectural knowledge) changes radically. Replacing analog temperature and humidity sensors with digital sensors represents a modular change.

While evolutionary change is the easiest to accommodate, revolutionary change causes the most stress on an organization's coping mechanisms. These two categories of change have been studied extensively in the literature. The novelty of Henderson's framework is the notion of architectural change which, according to Henderson (1996), is also quite stressful because architectural knowledge often is embedded implicitly in various organizational processes.

Information Systems Development Processes

The major IS development tasks relevant to our discussion are analysis, design, and implementa-

tion. Systems analysis involves gathering and documenting the requirements for an IS. The primary task is capturing the essence of a real-world system through models. These requirements are then transformed into detailed specifications during design. The IS is implemented and tested during implementation. Although these tasks are incorporated into both the structured and the OO development approaches, significant differences exist in how they are accomplished within each systems development approach.

The structured systems development life cycle is based on the waterfall model (see for example, Hoffer et al. 1998). In this model the systems development phases—analysis, design, and implementation[2]—are executed in a predominantly sequential fashion with some iterations between phases. Thus it is a somewhat lockstep process where the activities in a phase begin at the completion of the activities in the previous phase. The two important facets of an IS—processes and data—are independently modeled in this approach. Process models represented by data flow diagrams capture the flow of control among processes and demonstrate how data get modified as they flow through the system. Data models are described using entity-relationship (ER) diagrams that depict data elements and their associations. Inter-model consistency may be checked using a data-to-process CRUD (create, read, update, delete) matrix. This strategy of independently modeling data and processes is continued during the design phase. The conceptual data model developed during analysis is normalized and transformed into the database schema. Simultaneously, the process model is transformed into structure charts showing the hierarchical relationship among the programs to be implemented. Detailed programming logic is also developed in this phase. The database is implemented using the database schema and populated with data during implementation. Application programs are then developed based on the structure charts and program logic. Finally, the IS is tested to verify that it meets the users' requirements.

[2]Planning and maintenance phases, which are not relevant to our discussion, have not been listed here.

Unlike the structured systems development process, OO development follows an iterative and incremental lifecycle. The four phases in the OO lifecycle are inception, elaboration, construction, and transition[3] (Jacobson et al. 1999). An IS is developed through several iterations, each consisting of all three systems development tasks; namely, analysis, design, and construction.[4] Requirements for an IS are specified through use cases that describe the interactions between the system and its users. These are then analyzed to identify objects, which model both data (attributes) and related processes (methods). Similar objects are grouped together to form a class. Classes may be organized in a generalization hierarchy where a subclass inherits the attributes and methods from its superclass. Classes and their relationships are documented through class diagrams. Analysis classes expand into design classes in the design phase. This step creates detailed specifications for classes that include their attributes and methods. New classes, usually control classes and/or abstract super classes (Jacobson et al. 1999), are added to the model to enhance maintainability of the IS and reuse of the class library. Since process logic is distributed among objects as methods, objects must collaborate with each other to accomplish the scenarios detailed in a use case. Such collaborations are documented through interaction diagrams, such as sequence and collaboration diagrams (Booch et al. 1999). Finally, the information system is implemented through a combination of reuse of classes that are already available in a class library and creation of new ones based on the specifications developed during design. An object-oriented database may also be implemented to store persistent objects (Khoshafian and Abnous 1990). The resulting system is then evaluated through testing.

[3]This discussion relies on the UML-based unified software development process, which has received wide acceptance among the OO developer community.

[4]Jacobson et al. (1999) specify five development tasks: requirements specification, analysis, design, construction, and testing. Here analysis subsumes requirements specification, and construction subsumes testing to facilitate discussion.

Table 1. Comparsion of OO with Structured Systems Development Method

	Structured	Object-Oriented
Life Cycle	Sequential with iterations permitted	Iterative and incremental
Single unifying concept across life cycle	None	Object
Analysis	Data models, process models	Object models (class diagrams), collaboration diagram
Design	Structure chart, detailed program specifications, database schema	Design class specifications, sequence diagram
Implementation	Program implementation, database implementation, integration and testing	Class implementation, database implementation for persistent objects, integration and testing

Proposition Development

Table 1 compares the salient features of the two systems development methods. The analysis and design phases are concerned primarily with modeling the information system. Analysis results in the requirements model, and design results in the design model. Analysis and design tasks can be categorized as ill-structured problem-solving tasks. Problem decomposition is used as a standard technique to deal with the complexity inherent in these tasks. A fundamental difference between the two systems development approaches is the strategy used for problem decomposition (Korson and McGregor 1990). As mentioned earlier, the structured approach separates processes from data and independently decomposes them into program modules and entity sets. In contrast, an object, which encapsulates data and related processes, is used as the fundamental unit of decomposition in the OO approach. Encapsulation and inheritance are two unique OO concepts that influence the object modeling approach (Korson and McGregor 1990). Data encapsulated in an object can only be accessed and/or modified by the methods associated with that object. Processing logic is thus distributed among objects instead of residing in one or more programs as in the structured approach. This requires objects to collaborate with each other to fulfill the functional requirements of a system.

As we move from structured analysis and design to object-oriented analysis and design, the components of the system—data and processes—do not change; however, the way a system is decomposed into its constituent parts and the way data and processes relate to each other (i.e., the architectural knowledge) change dramatically. Thus, we argue:

> *Proposition 1: The transition from structured systems analysis and design to object-oriented analysis and design constitutes an architectural change.*

Implementation tasks in both systems development approaches are similar in many respects. The main differences are due to new concepts in OO programming, such as inheritance, encapsulation, polymorphism, and abstract data types. While most of the basic programming concepts in a procedural language are still useful in an object-oriented programming language, the new concepts require the programmer to adopt a style different from the one followed in non-OO programming.

Learning a new programming language requires mastering a few additional keywords and the associated syntax. This is unlike learning a new systems analysis and design method, which imposes a significantly higher cognitive burden due to a change in one's views of the world

(Korson and McGregor 1990). Many computer professionals are well versed in several programming languages but know only a single approach to analysis and design. The transition from COBOL programming to C programming, which introduces several new concepts such as local variables, block structured programming, pointers, and memory management, may be considered an incremental change. Similarly, the transition from C to C++ or to another object-oriented language may be considered an incremental change. The language is merely a tool, and languages differ primarily in terms of syntax and certain other constructs that help one to implement the semantics of the domain model. The semantics really lie in the model. Thus we argue:

> *Proposition 2: The transition from structured systems implementation to object-oriented implementation constitutes an incremental change.*

Research Methodology

Author co-citation analysis (ACA) was used to examine the intellectual structure of the software development field and to identify conceptual distances among its subfields. ACA starts with the identification of a list of seminal authors who have made significant contributions to a discipline. Authors in ACA are viewed as surrogates for the concepts they write about. The co-citation count for each author pair (i.e., the count of papers that have cited the works of both authors) is then obtained from a bibliographic database. The raw co-citation matrix is converted into a Pearson's correlation matrix, which is analyzed using multivariate statistical techniques. McCain (1990) provides a detailed description of ACA.

ACA has been widely used to trace the evolution of thought and to map the intellectual structure of a field (e.g., Culnan 1986, 1987; Eom and Farris 1996; McCain 1983, 1984; Nerur 1994; White and Griffith 1981). It has the power to graphically render a conceptual map showing the distance of separation between authors by subjecting citation data to quantitative treatment. According to White (1990, p. 85),

> Within a given map, each point represents an author, and the proximity of points reflects relationships as perceived by multiple citers. Closely placed authors are seen as similar, generally in content and approach; distant authors, as dissimilar.

Because authors in ACA are nothing but substitutes for the concepts they have written about (Culnan 1986), the map of the intellectual structure of a field obtained using ACA is a map showing logical distances between various concepts in the field (McCain 1990). ACA was therefore considered an appropriate research method for this study.

Author Selection

Authors who have made significant contributions to the analysis, design, and programming stages of the structured and OO development approaches were considered for this analysis. The author list was prepared in two stages. A preliminary list was developed based on a detailed examination of textbooks and publications. This list was then evaluated by a group of experts with significant academic and/or industry experience in the analysis, design, and implementation of information systems. This resulted in the final list of 30 authors shown in Table 2. While we made every effort to include researchers who have made significant contributions to structured and/or OO system development theory and practice, we do not claim to have an exhaustive list; nonetheless, the ideas of an author not appearing in our list will still be included in our analysis through his/her co-citations to the authors in the list (Culnan 1986).

The citations for each author published between 1980 and 1994 were obtained in the form of *accession numbers* by searching the Science Citation Index (SCI). Each author in the list had more than 30 citations, which was a criterion for inclusion in the analysis (Culnan 1986). The citation data were scanned using a computer program to determine the co-citation count for each pair of authors.

Table 2. List of Authors Used in Author Co-citation Analysis

Boehm, B. W.	Goldberg, A.	Page-Jones, M.
Booch, G.	Henderson-Sellers, B.	Parnas, D.
Brooks, F. P.	Hoare, C. A. R.	Rumbaugh, J.
Buhr, R.	Jackson, M. A.	Shlaer, S.
Coad. P.	Liskov, B.	Stroustrup, B.
Constantine, L. L.	Martin, J.	Ward, P.
Dahl, O. J.	Mellor, S.	Wegner, P.
DeMarco, T.	Meyer, B.	Wirfs-Brock, R. J.
Dijkstra, E.	Myers, G.	Wirth, N.
Gane, C.	Orr, K.	Yourdon, E.

	BOEHM	BOOCH	BROOKS	BUHR	COAD
BOEHM	216.5	61	169	7	23
BOOCH	61	181	31	22	105
BROOKS	169	31	155.5	4	13
BUHR	7	22	4	23.5	3
COAD	23	105	13	3	130.5

Figure 2. Sample Raw Co-citation Matrix

Co-citation Analysis

The raw co-citation counts obtained were then used to form a matrix with identically ordered authors' names on the rows and columns. Figure 2 shows the matrix of raw co-citation counts for the first five authors in the study. For example, the count of 61 at the intersection of Boehm and Booch means that at least one work/paper of each of these authors was cited by 61 papers.

The diagonal values of the matrix (i.e., the intersection of an author with himself/herself) were computed by summing the three highest co-citation counts for each author and dividing the result by two (Culnan 1986; White and Griffith 1981). The co-citation data were normalized by converting the raw co-citation matrix into a matrix of Pearson correlations. This matrix was analyzed using factor analysis, cluster analysis, and multi-dimensional scaling (MDS).

Results

SPSS V6.1.4 and NCSS 2000 were used for our statistical analyses. The following subsections present our factor analysis, cluster analysis, and MDS results.

Factor Analysis

Factor analysis yields a set of factors, each of which includes a subset of authors used in the analysis. Each factor may be viewed as an intellectual perspective represented by the authors who load on it. The advantage of factor analysis over cluster analysis and multidimensional scaling is that it reveals the versatility of an author by showing the variety of conceptual areas/perspectives to which an author has contributed (McCain 1990).

Table 3. Factor Analysis Result

Factor Name	Structured Analysis and Design		Object-Oriented Analysis and Design		Structured Programming		Object-Oriented Programming		Software Management Principles	
Eigenvalue	7.61301		6.69034		5.60226		2.02078		1.70030	
% Total Variance	25.4		22.3		18.7		6.7		5.7	
Cumulative % Variance	25.4		47.7		66.4		73.1		78.8	
Loading > 0.40	Gane	0.85	Coad	0.93	Hoare	0.93	Goldberg	0.95	Brooks	0.92
	DeMarco	0.84	Rumbaugh	0.91	Dijkstra	0.93	Stroustrup	0.93	Boehm	0.91
	Ward	0.80	Shlaer	0.87	Wirth	0.87	Meyer	0.88	Myers	0.72
	Yourdon	0.79	Wirfs-Brock	0.85	Liskov	0.79	Wegner	0.84	Martin	0.56
	Page-Jones	0.79	Booch	0.81	Parnas	0.76	Dahl	0.61	Yourdon	0.56
	Mellor	0.78	Henderson-Sellers	0.73	Dahl	0.60	Booch	0.50	Parnas	0.52
	Orr	0.62	Buhr	0.50			Liskov	0.48	DeMarco	0.45
	Martin	0.59							Jackson	0.42
	Jackson	0.51								

Principal components analysis with varimax rotation was used in this study. Both the Kaiser criterion and the Scree test suggested that there were six factors to be extracted. While the first five factors accounted for 78.75% of the total variance, Factor 6 accounted for only 3.48% of the total variance. Factor 6 was dropped due to its poor explanatory power, and the analysis was run with only five factors. Table 3 presents the results. While loadings above ± 0.7 are particularly useful in interpreting a factor, most ACA researchers show loadings above ±0.4 or ±0.5 (McCain 1990). In this study, only loadings above ±0.4 are shown.

Naming of the first four factors was fairly straightforward and is consistent with the literature. The fifth factor, however, has authors who have contributed to a variety of topics, such as project management (e.g., Brooks), process models (e.g., Boehm), structured analysis (e.g., DeMarco and Yourdon), structured/composite design (e.g., Myers), structured techniques (e.g., Martin), modularity of complex systems and software reliability (e.g., Parnas), and system development/program design (e.g., Jackson). This factor, which was labeled *software management principles,* encompasses project management, software reliability, and testing.

Surprisingly, Constantine, who has made significant intellectual contributions to structured design, was excluded from all five factors due to relatively lower communality. A possible explanation for this outcome is the fact that citations are listed only by the authors named first (Bayer et al. 1990). Therefore, an author's citation count will be under-represented if he or she is not the first author of the articles co-authored by him/her.

Cluster Analysis

Cluster analysis using Ward's method of clustering was performed on the correlation matrix to identify clusters of conceptually similar authors. Both SPSS and NCSS produced identical clusters. NCSS, however, produced a visually more appealing dendrogram. Figure 3 shows the NCSS dendrogram with linkage distances among clusters. A cluster cutoff point of 1.0 produced five clusters, which are consistent with the factors identified by factor analysis. An interesting observation from the dendrogram is that structured programming is conceptually closer to OO programming than it is to structured analysis and design.

Multidimensional Scaling

Multidimensional scaling (MDS) provides a graphical rendition of authors in intellectual space. The proximity of one author to another, which is an indication of their conceptual similarity, can be assessed by examining the MDS map. Each group of authors on the map represents an intellectual perspective or subfield.

MDS was performed on the correlation matrix using the ALSCAL procedure in SPSS. Following McCain (1990), we used an ordinal level of measurement, the Euclidean distance model, and a cut-off value of –1.0 for missing values (in order to accommodate negative correlation values). Figure 4 shows the MDS map obtained by rotating the data through 45 degrees to improve interpretation of the result (McCain 1990). The stress value (Kruskal's stress formula 1) of 0.11473 and the R-Square of 0.926 indicate an acceptable "goodness of fit" (McCain 1990, p. 438).

The cluster boundaries shown on the MDS map are based on the five clusters produced by cluster analysis. The structured analysis and design cluster is positioned in the bottom-right quadrant. It is clearly separated from the OO analysis and design cluster that is located in the top-right quadrant. Both OO and structured programming clusters are located in the top-left quadrant. The software management principle cluster is located in the bottom-left quadrant. We have named the horizontal dimension (X-axis) "Programming (solution domain) versus Analysis and Design (problem domain)." We have named the vertical dimension (Y-axis) "OO versus Structured." The structured programming cluster is far removed from the structured analysis and design cluster. This result manifests a conceptual gap that exists between structured analysis and design on the one hand and structured programming on the other. The seamless integration between analysis and design and programming in the OO approach is reflected by the shorter conceptual gap between OO programming and OO analysis and design clusters.

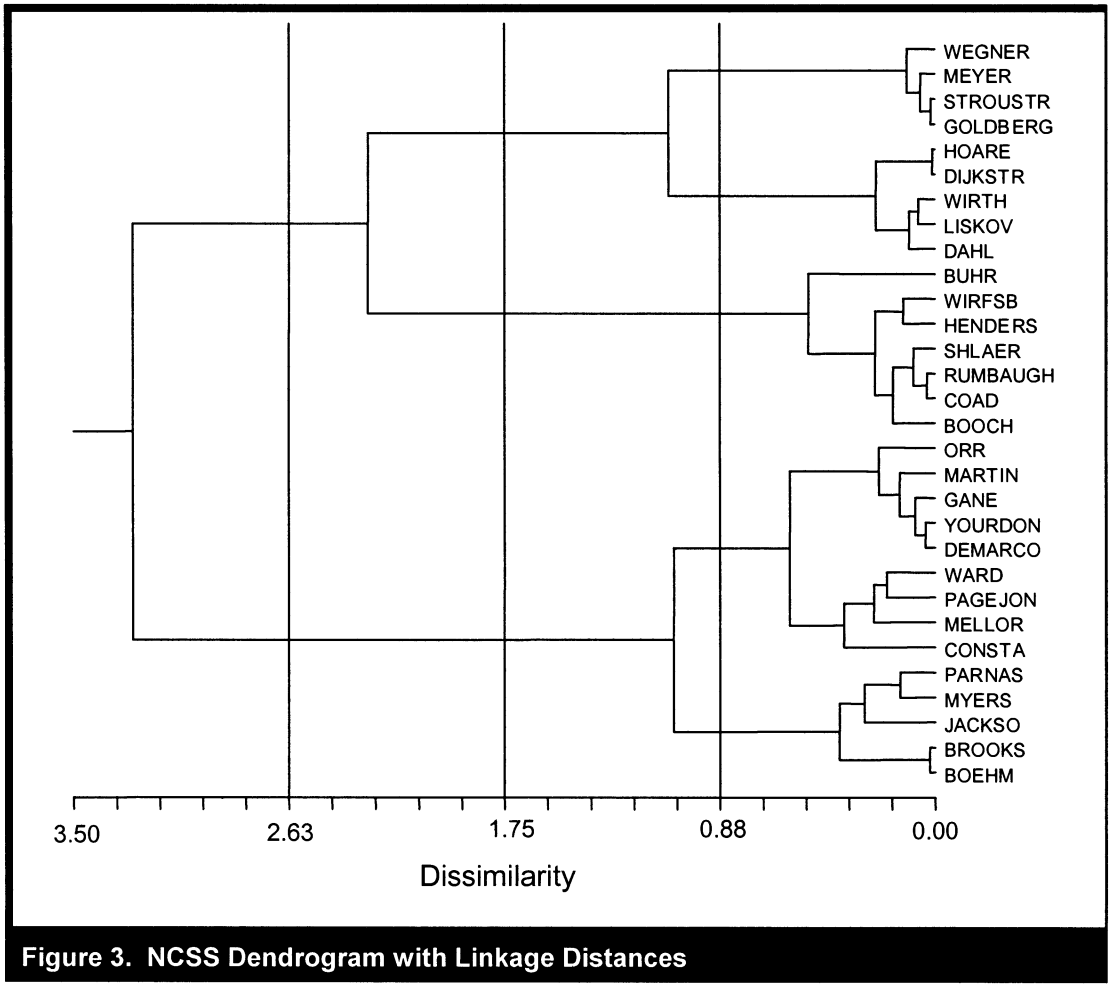

Figure 3. NCSS Dendrogram with Linkage Distances

Discussion

Understanding the nature of an innovation is a crucial first step in managing change associated with the innovation. Technological innovations in particular have a significant impact on the competitiveness of organizations (Tushman and Anderson 1986). Fortunately for existing firms in an industry, *incremental innovations* enable them to reinforce their competitiveness. Thus, they are termed *competence-enhancing*. *Radical innovations*, in contrast, are called *competence-destroying* because established firms have difficulty coping with them. *Architectural innovations*, which fall in between these two extremes, are difficult to discern. Furthermore, they pose vexing change management problems for organizations (Henderson 1996; Henderson and Clark 1990).

Our results show that OO and structured systems development methods differ substantially in some ways yet share many common concepts. The differences are much more pronounced in the analysis and design phases than in the implementation phase. In the analysis and design stages, the systems developer understands, conceptualizes, and represents the problem domain. The pronounced difference between OO and the structured view at these stages implies that those deeply entrenched in the structured approach need to undergo a conceptual transformation in order to make the transition to OO. Processing logic and data, the two fundamental components

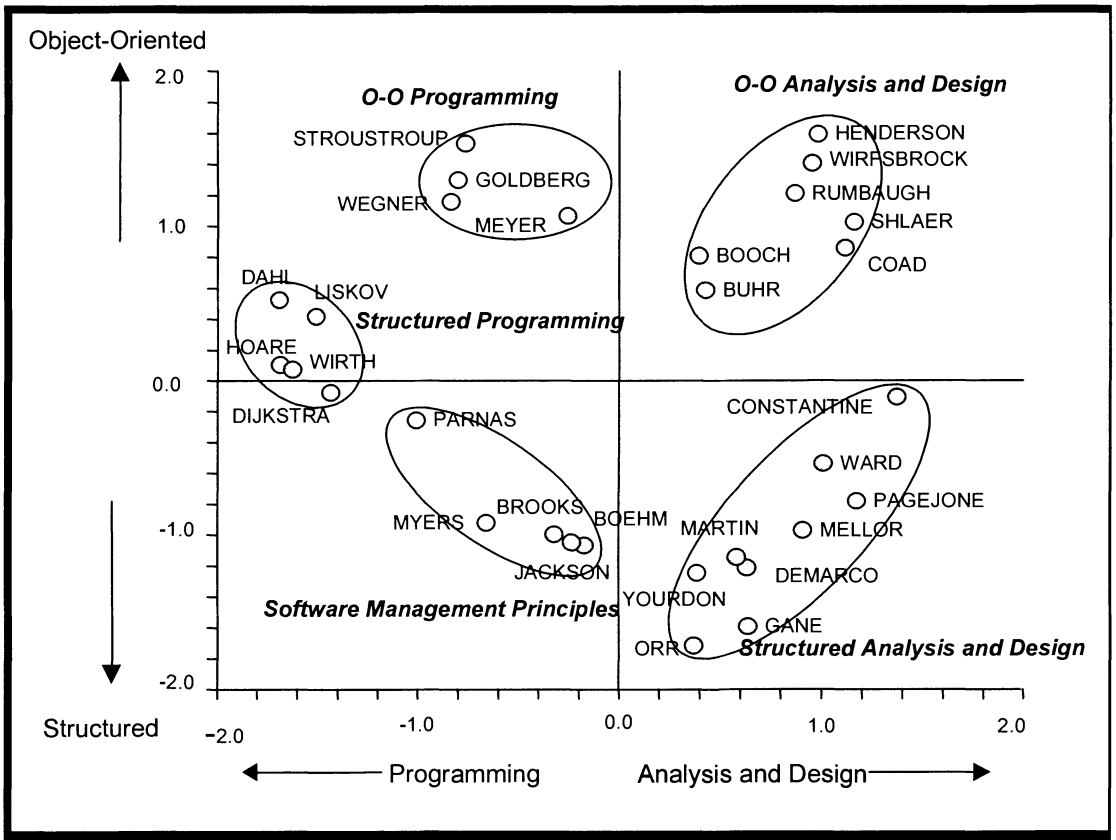

Figure 4. Multidimensional Scaling Map

of an information system, are modeled and designed independently in the structured approach. In contrast, the OO approach uses the object, which encapsulates data and relevant processing logic, as the basic unit for analysis and design. Thus, while knowledge about the components of an information system (i.e., processing logic and data) does not change, the relationship among these components undergoes complete transformation as one makes the transition from a structured to an OO approach. For this reason, using Henderson's (1996) model, we classify the change as an architectural change.

In the context of the implementation phase, fundamental programming logic (if...then, case statement, do...while, for, etc.) is an essential aspect of both structured and OO programming approaches. OO programming incorporates several new concepts such as encapsulation, inheritance, and polymorphism. These concepts build on basic programming principles, however, and thus are incremental in nature. The software model that is ultimately implemented in a programming language should be as faithful a representation of reality as possible; however, what is critical in this mapping of reality to the software model is not the programming language per se. Instead, it is the conceptualization and representation of the problem domain that is accomplished in the analysis and design stages of the software development lifecycle that is critical. For example, Jacobson et al. (1995) used an assembler to implement an OO design model. They claim that "the programming language itself is not crucial in modeling software" (p. 46). The results of our empirical research further emphasize this point. For this reason, using Henderson's model, we classify the change from structured implementation to OO implementation as an incremental change.

There is also anecdotal evidence to support our findings. For example, the number of articles devoted to OO pedagogy (Bolloju 2001; Dodani 1996; Mann and Schneider 1997) as well as the search for pedagogical patterns (Manns et al. 1998) to effectively teach OO are evidence of the steep learning curve involved in migrating to OO. Sheetz et al. (1997) also found that mastering OO modeling and design skills was more difficult than mastering OO programming languages. This view is further supported by a panel of OO experts (Pancake 1995). According to a panelist, "If languages continue to drive this shift [to object technology], then we are really barking up the wrong tree. The promise of OT (object technology) is not in what language we choose, but in how we attack the problem."

Our findings have important implications for organizations, especially for IS managers who are in the process of adopting the OO approach. Assisting systems developers to assimilate OO analysis and design knowledge should be given a high priority in their migration plans. The change in the architectural knowledge, which has been loosely described as the *mindset*, can be facilitated through a combination of training, mentoring, and adoption of the right CASE tools. Training helps individuals acquire knowledge. Mentoring is useful in putting this new knowledge into practice because it involves the guidance of an experienced person in a real-world project. Concepts learned through training become institutionalized when they are applied under the supervision of the mentor. The mentor, therefore, plays a critical role in helping an organization migrate to OO technology (Booch 1996; Ramaswamy 2001). Adoption and use of OO CASE tools also are likely to facilitate this transition (Quatrani 2000). These tools embody the principles underlying the OO systems development method and can help the systems developer stay focused on the OO approach.

This discussion will be incomplete without an exposition of the limitations of ACA as a research method. For example, citations are listed by the first author only (Bayer et al. 1990), thus the citation count is not affected when papers have a single author. The contributions of authors who are not the lead authors of co-authored papers, however, are under-represented in the data set. In addition, the reason for citing an author is not considered in ACA. This can skew the data when a paper that compares two different approaches co-cites authors who represent disparate concepts. The intellectual structure of a field as discovered by ACA also may not be current due to publication delays. Finally, ACA assumes that all citations have the same value. It might be argued that some citations are *more equal* than others. Nevertheless, despite these drawbacks, ACA has been a widely accepted research methodology.

Conclusions

This study endeavored to resolve the debate in the literature about whether OO systems development represents a revolutionary change for software developers trained in structured methods, or merely an evolution. We believe that we have been able to throw significant light on this issue by demonstrating that views on the extent of change depend on which aspect of OO development is being observed: analysis, design, or implementation. Furthermore, we believe that Henderson's concept of architectural change is more relevant and succinct in the case of OO development than the notion of revolutionary change. The beauty of the idea of an architectural change is that it explains a category of change that cannot be classified as either revolutionary or evolutionary. This framework also helps focus the attention of the change manager on factors that are at the root of the change.

Author co-citation analysis (ACA), a form of bibliometrics, was used to map the intellectual structure of the field. It was deemed especially appropriate for this study because of its capacity to identify subspecialties within a field and reveal the conceptual links among them. Our results demonstrated a major conceptual gap between OO and the structured approach at the analysis and design stages. These differences are neither evolutionary nor revolutionary in nature. They fall in a continuum between these two extremes and are best described as architectural innovations.

While the component knowledge remains valid, the architectural knowledge undergoes a major transformation. This means that there is hope for structured systems analysts and designers in the sense that their knowledge and skills will not be obliterated by the new method. At the same time, it is important for the change manager to understand that architectural changes are stressful to manage. Significant efforts must be made to help analysts and designers change their views of the world and effectively adopt the new development method. To make the most of the existing skill levels among systems developers, training should focus more on modeling issues rather than on teaching the syntax of an OO language. This will help developers with structured analysis and design skills bridge the conceptual gap that exists between OO and the structured approach while making the best use of their existing knowledge and skills. Appropriate use of mentoring and the adoption of OO case tools should facilitate this transition.

Acknowledgments

The authors are indebted to the senior editor, the associate editor, and the reviewers for their insightful comments that have significantly enhanced the quality of the manuscript. Also, many thanks to our colleague, Mark Eakin, for his suggestions.

References

Bayer, E. A., Smart, C. J., and McLaughlin, W. G. "Mapping Intellectual Structure of a Scientific Subfield through Author Co-citations," *Journal of the American Society for Information Science* (41:6), 1990, pp. 444-452.

Bolloju, N. "Using Group Systems in Teaching and Learning an Object-Oriented Analysis and Design Course," *Journal of Object Oriented Programming*, January 2001.

Booch, G. *Object Oriented Design with Applications*, The Benjamin/Cummings Publishing Company, Inc., Redwood City, CA, 1991.

Booch, G. *Object Solutions: Managing the Object Oriented Project*, Addison-Wesley, Reading, MA, 1996.

Booch, G., Rumbaugh, J., and Jacobson, I. *The Unified Modeling Language User Guide*, Addison-Wesley, Reading, MA, 1999.

Brooks, F. P. "No Silver Bullet—Essence and Accidents of Software Engineering," *Computer*, April 1987, pp. 10-19.

Cargill, C. "Understanding the Rules of OT," *Uniforum Monthly* (15:1) January 1995, pp. 44-45.

Culnan, J. M. "The Intellectual Development of Management Information Systems, 1972-1982: A Co-Citation Analysis," *Management Science* (32:2), February 1986, pp. 156-172.

Culnan, J. M. "Mapping the Intellectual Structure of MIS, 1980-1985: A Co-Citation Analysis," *MIS Quarterly*, September 1987, pp. 341-350.

Dodani, M. "Object-Oriented Shock Therapy," *Journal of Object Oriented Programming* (9:4), July/Aug. 1996, pp. 17-19.

Due, R. T. "Object-Oriented Technology—The Economics of a New Paradigm," *Information Systems Management*, Summer 1993, pp. 69-74.

Eom, S. B., and Farris, R. S. "The Contributions of Organizational Science to the Development of Decision Support Systems Research Subspecialties," *Journal of the American Society for Information Science*, December 1996, pp. 941-952.

Glass, R. L. "A Snapshot of Systems Development Practice," *IEEE Software*, May/June 1999, pp. 112-111.

Henderson, R. M. "Technological Change and the Management of Architectural Knowledge," in *Organizational Learning*, M. D. Cohen and L. S. Sproull (eds.), Sage Publications, Thousand Oaks, CA, 1996, pp. 359-375.

Henderson, R. M., and Clark, K. B. "Architectural Innovation: The Reconfiguration of Existing Product Technologies and the Failure of Established Firms," *Administrative Science Quarterly*, (35), 1990, pp. 9-30.

Henderson-Sellers, B. *A Book of Object-Oriented Knowledge*, Prentice Hall, Englewood Cliffs, NJ, 1992.

Henderson-Sellers, B., and Constantine, L. L. "Object-Oriented Development and Functional Decomposition," *Journal of Object-Oriented Programming—Focus on Analysis and Design*, 1991, pp. 18-23.

Hoffer, J. A., George, J. F., and Valacich, J. S. *Modern Systems Analysis & Design*, Addison-Wesley, Reading, MA, 1998.

Jacobson, I., Booch, G., and Rumbaugh, J. *The Unified Software Development Process*, Addison-Wesley, Reading, MA, 1999.

Jacobson, I., Ericsson, M., and Jacobson, A. *The Object Advantage—Business Process Reengineering with Object Technology*, Addison-Wesley, Reading, MA, 1995.

Kochan, T. A., and Useem, M. *Transforming Organization*, Oxford University Press, New York, 1992.

Khoshafian, S., and Abnous, R. *Object Orientation: Concepts, Languages, Databases, User Interfaces*, John Wiley & Sons, New York, 1990.

Korson, T., and McGregor, J. D. "Understanding Object-Oriented: A Unifying Paradigm," *Communications of the ACM* (33:9), September 1990, pp. 40-60.

Li, X. "Integration of Structured and Object-Oriented Programming," *Journal of Object-Oriented Programming—Focus on Analysis and Design*, 1991, pp. 54-60.

Lilly, S. "The Structure of Software Revolutions (and the Difficulty of Teaching Them)," *Object Magazine* (3:3), September-October 1993, pp. 77-79.

Mann, J., and Schneider, R. "'Aha' Experiences in Object-Oriented Education: Searching for a Theoretical Foundation," *Proceedings of the Third Americas Conference on Information Systems*, Indianapolis, IN, 1997, pp. 768-770.

Manns, M. L., Sharp, H., Prieto, M., and McLaughlin, P. "Capturing Successful Practices in OT Education and Training," *Journal of Object Oriented Programming*, March/April 1998, pp. 29-34.

McCain, K. W. "The Author Cocitation Structure of Macroeconomics, *Scientometrics* (5), 1983, pp. 277-289.

McCain, K. W. "Longitudinal Author Cocitation Mapping: The Changing Structure of Macroeconomics," *Journal of the American Society for Information Science* (35), 1984, pp. 351-359.

McCain, K. "Mapping Authors in Intellectual Space: A Technical Overview," *Journal of the American Society for Information Science* (41:6), 1990.

Meli, M. "Object Orientation—Real or Hype?," *Data Management Review* (4:7), July 1994, pp. 24-26.

Nerur, S. *Paradigmatic Issues in Software Development: The Case of Object-Orientation*, Unpublished Doctoral Dissertation, The University of Texas at Arlington, 1994.

Pancake, C. "The Promise and the Cost of Object Technology: A Five Year Forecast," *Communications of the ACM* (38:10), October 1995, pp. 33-49.

Page-Jones, M., and Weiss, S. "Synthesis: An Object-Oriented Analysis and Design Method," *Journal of Object-Oriented Programming—Focus on Analysis and Design*, 1991, pp. 133-135.

Parodi, J. "Distributed Object Technology: The Long View," *Uniforum Monthly* (15:1), January 1995, pp. 34-36.

Pun, W., and Winder, R. "Design Method For Object-Oriented Programming," *Journal of Object-Oriented Programming—Focus on Analysis and Design*, 1991, pp. 61-73.

Quatrani, T. *Visual Modeling with Rational Rose 2000 and UML*, Addison-Wesley, Reading, MA, 2000.

Ramaswamy, R. "Mentoring Object-Oriented Projects," *IEEE Software*, May/June 2001, pp. 36-40.

Sarna, D., and Febish, G. "Think Objects? Think Big," *Datamation*, July 15, 1995, pp. 25-27.

Sheetz, S. D., Irwin, G., Tegarden, D. P., Nelson, H. J., and Monarchi, D. E. "Exploring the Difficulties of Learning Object-Oriented Techniques," *Journal of Management Information Systems*, (14:2), Fall 1997, pp. 103-131.

Tushman, M. L., and Anderson, P. "Technological Discontinuities and Organizational Environments," *Administrative Science Quarterly* (31), 1986, pp. 439-465.

White, D. H. "Author Co-Citation Analysis: Overview and Defense," *Scholarly Communication and Bibliometrics*, C. L. Borgman (ed.), Sage Publications, Thousand Oaks, CA, 1990, pp. 84-106.

White, D. H., and Griffith, C. B. "Author Cocitation: A Literature Measure of Intellectual Structure," *Journal of the American Society for Information Science* (32), May 1981, pp. 163-171.

About the Authors

Sumit Sircar is professor and director of the Center for IT Management in the College of Business Administration at the University of Texas at Arlington where he previously served as chairman of the Department of Information Systems and Operations Management as well as associate dean. He obtained his doctorate in information systems from the Harvard Business School. His area of specialization is the management of information technology in which he has published extensively in journals such as *Communications of the ACM*, *Journal of MIS*, *MIS Quarterly*, and *Information and Management*. He serves on the Editorial Board of *Information Systems Research*.

Sridhar Nerur is an assistant professor of Information Systems at Central Missouri State University. He holds a B.E. degree in Electronics from Bangalore University, a PGDM from the Indian Institute of Management, Bangalore, India, and a Ph.D. in Business Administration from the University of Texas at Arlington. His research interests are in the areas of object-oriented analysis and design, software reuse, cognitive aspects of programming, computer security, database design, knowledge management, and software maintenance.

Radhakanta Mahapatra is an assistant professor of Information Systems at the University of Texas at Arlington. He holds a B.Sc (Hons.) degree in Electrical Engineering from Regional Engineering College, Rourkela, India, a PGDM from the Indian Institute of Management, Ahmedabad, India, and a Ph.D. in Information Systems from Texas A&M University. His research interests are in the areas of physical database design, data mining, case-based reasoning, systems analysis and design, software reuse and knowledge management. His research publications have appeared in several journals, including *Decision Support Systems*, *Information & Management*, *Journal of Database Management*, and *Data Base*.