Association for Information Systems
AIS Electronic Library (AISeL)

AMCIS 2002 Proceedings

Americas Conference on Information Systems (AMCIS)

December 2002

PARADIGMATIC CHANGES IN SOFTWARE DEVELOPMENT: A PHILOSOPHY OF SCIENCE PERSPECTIVE

Sridhar Nerur
Central Missouri State University

Radha Mahapatra
University of Texas at Arlington

Sumit Sircar
University of Texas at Arlington

PARADIGMATIC CHANGES IN SOFTWARE DEVELOPMENT: A PHILOSOPHY OF SCIENCE PERSPECTIVE

Sridhar P. Nerur
Department of COIS
Central Missouri State University,
Warrensburg
nerur@cmsu1.cmsu.edu

Radha K. Mahapatra
Department of Information Systems &
Operations Management
University of Texas at Arlington
mahapatra@uta.edu

Sumit Sircar
Center for IT Management
University of Texas at Arlington
sircar@uta.edu

Abstract

By all accounts, the structured approach to software development is no longer the dominant paradigm. Object orientation (OO) is at the forefront of technology today, and a deep understanding of this phenomenon is required to effectively manage the transition to this approach. While there are many who assert that OO is a fundamentally different approach to software development and, therefore, constitutes a paradigm shift, there are others who assert that OO has evolved from prior ideas in software engineering. This research uses a philosophy of science framework to explore paradigmatic changes that may have occurred in the field of software development. Specifically, it applies the Kuhnian notion of paradigm shift to understand the evolution of software development processes. It also proposes Author Co-Citation Analysis (ACA) as a viable method for empirically validating concepts such as paradigms and consensus in a field.

Keywords: Paradigm shift, OO, structured, ACA

Introduction

Despite the tremendous strides that have been made in software development, the inherent complexity of software systems continues to boggle the imagination of researchers and developers. The sophistication of today's software tools and techniques has made programming much easier, but the fact remains that projects are still hard to manage, seldom on time, considerably beyond budget, and invariably fail to provide the highest level of satisfaction to customers (Ledbetter and Cox 1985). These inadequacies in approaches to software development have precipitated a crisis, the nature of which is more daunting than the one developers had to cope with in the 1960s.

The structured approach has thus far been the dominant paradigm in software development. Research within this paradigm has spawned some great software engineering ideas that helped to alleviate the software crisis to some extent. However, increasing anomalies in the structured paradigm and the virtual exhaustion of ideas within it have compelled some researchers to break away from the structured tradition. Such a break in the cumulative tradition of a dominant paradigm is often followed by the emergence of a new paradigm (Kuhn 1970). While OO is no panacea, it is believed by many that it has the potential to resolve some or most of the anomalies that have plagued the structured paradigm.

There seems to be a lack of understanding of the conceptual structure of the field of software development and the nature of the change that the OO phenomenon entails. The OO approach emphasizes problem understanding and the organization of an OO

system is akin to the way humans think and reason about problems in the real world (Coad and Yourdon 1990, Booch 1991). Further, the OO approach encourages the evolutionary growth of software systems by employing existing blocks in their construction. However, from a software development viewpoint, it may be very different from prior practices and may entail a radical change in mindset. While some believe that OO is just an evolution of thought in software engineering (e.g., Page-Jones and Weiss 1991), there are others who assert that OO is a revolutionary approach (Cargill 1995, Pun and Winder 1991, Lilly 1993, Henderson-Sellers and Constantine 1991). The question of whether OO actually involves a paradigm shift in a Kuhnian sense is yet to be addressed. The primary objectives of this paper are:

1. To study the evolution of the software development field using Kuhn's view of paradigms and paradigm shifts.
2. To identify and elucidate a method that has the potential to empirically validate abstract concepts such as paradigms.

Paradigms and Paradigm Shifts – A Kuhnian Perspective

Kuhn's notion of paradigms is the conceptual basis for this research. Thomas Kuhn defined a paradigm as "universally recognized scientific achievements that for a period of time provide model problems and solutions to a community of researchers" (Kuhn 1970). Research in a discipline is guided by certain fundamental rules, values, assumptions, and beliefs that are instilled from past experiences. The inability of scientists, especially in mature scientific disciplines, to perceive the world in ways different from what they are accustomed to stems primarily from their deeply entrenched values and beliefs (Toulmin 1963 as cited in Crane 1972). Such assumptions and beliefs shared by researchers in a specialty constitute a paradigm, and any attempt to break away from the shared commitments would be viewed as tradition-shattering. The fundamental purpose and objective of a discipline, the manner in which problems are formulated, represented, and solved, the sources used to elicit the truth, the methods, tools, and techniques used to validate the truth, and the anomalies that plague the discipline are but a few things that a paradigm helps to clarify.

A paradigm's increasing inability to solve problems results in a crisis. The despair and despondency brought about by such a crisis compels researchers to seek out alternatives beyond the confines of their paradigm. Such a bold effort to break the cumulative tradition of an existing paradigm is often crowned with the successful development of a revolutionary idea that may have the potential to resolve the anomalies and quell the crisis that exists. This revolutionary idea is a cognitive event that heralds the emergence of a new paradigm, the values and beliefs of which may be in sharp contradistinction to those of the existing paradigm. The new paradigm competes with the old one and presents a conceptual barrier, which, though not insuperable, requires drastic changes at the thinking or epistemological level. The transition between paradigms, often termed as Paradigm Shift, is not an incremental, step-by-step change (Kuhn 1970).

Ideas similar to Kuhn's concept of paradigms have been expressed by theorists in other areas, such as individual adult development, group development, organizational evolution, evolutionary biology, and self-organizing systems (van Gigch 1991, Gersick 1991). All of them agree that there are protracted periods of equilibrium followed by infrequent revolutions. From an organizational perspective, the key to successfully managing the transition lies in understanding the ramifications of a paradigm shift.

The life cycle of a paradigm follows the s-shaped logistic curve (Barker 1989, Crane 1972). Figure 1 captures the notion of a paradigm shift. The initial pre-paradigmatic period (segment A) is a trying time during which confusion is rife, with the pioneers of the potentially new paradigm endeavoring to prove that the revolutionary ideas of the new paradigm can resolve the anomalies of the existing dominant paradigm. The proponents of the existing paradigm not only defend the assumptions of the existing paradigm, but also attempt to integrate (often unsuccessfully) the new concepts with their existing ones. Problem solving efficiency, which tends to follow the paradigm curve, is low, and the cost of solving a problem in this phase is therefore high (Barker 1989).

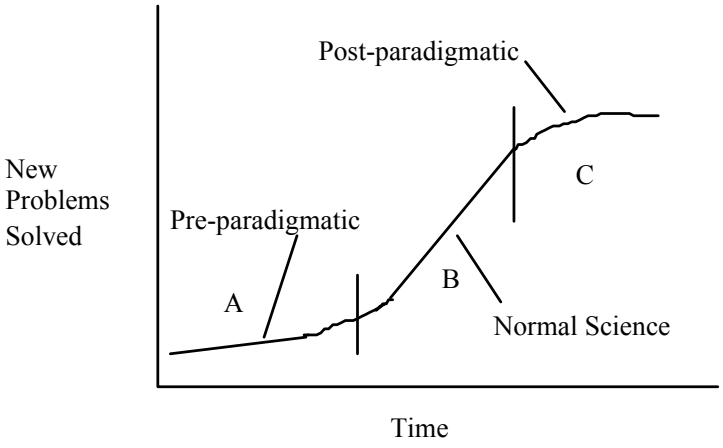

Figure 1: The Paradigm Curve
Adapted from Barker (1989)

Once the controversies and uncertainties of the pre-paradigmatic period are allayed, there is increased collaboration and a clearer understanding of definitions and values that comprise the new paradigm. The period, represented by segment "B" in figure 1, known as normal science (Kuhn 1970) is the one in which there is an underlying paradigm guiding research in a discipline. During this period, increased collaboration among scientists in the discipline and their efforts to build on each others' works engenders a cumulative tradition that emphasizes the conceptual consensus among them. Such a conceptual consensus is a reflection of the shared values and commitments that constitute the paradigm, which influences the perceptions, actions, and behavior of these researchers. As the practitioners within the new paradigm get more efficient, the cost of solving problems decreases rapidly.

The post-paradigmatic phase (segment C) is when problem solving efficiency drops, owing primarily to the inability of the constricted assumptions of a paradigm to deal with newer and more complex problems. Alternatives are sought when costs become prohibitive to a point of being a crisis. After exhausting ideas and strategies to overcome the crisis, researchers look beyond the confines of their paradigm, and that heralds the emergence of a new paradigm. The essence of this phenomenon is illustrated in Figure 2.

To summarize, we find two important concepts that may be used to identify paradigm shifts and emergence of a new paradigm. First, fundamental concepts that guide the intellectual activities in a new paradigm must be very different from the concepts and principles that were prevalent in the preceding paradigm. Second, emergence of a new paradigm and a period of normal science are preceded by a pre-paradigmatic period characterized by creation and dissolution of new and competing ideas and concepts that vie for acceptance. Some of these survive and act as seeds, or fundamental concepts, that provide the basis for further creative development during the period of normal science. In the following section we compare structured systems development methods with OO methods using the Kuhnian view of paradigm shift.

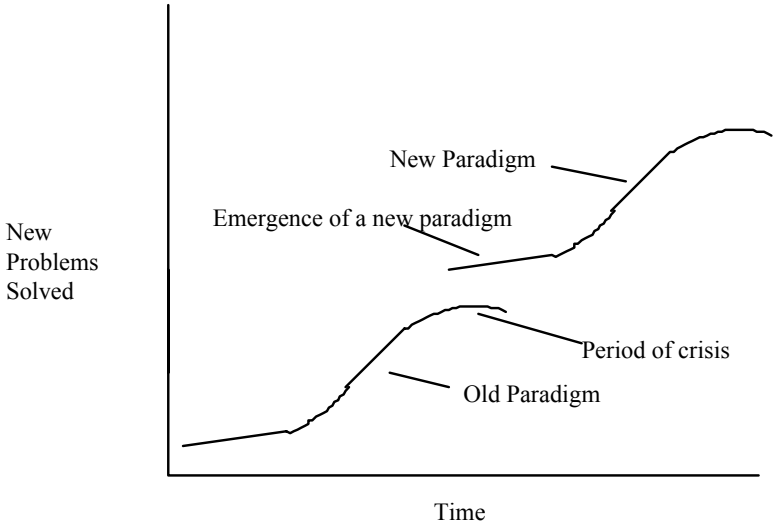

Figure 2: The Emergence of a New Paradigm
Adapted from Barker (1989)

Structured vs OO Paradigms

Structured Systems Development Process

This section provides a general description of the structured systems development process and is not based on any specific methodology. The waterfall model (or some variation of it) forms the basis of the systems development lifecycle. In this model, the different phases of systems development – analysis, design, implementation, and maintenance – follow a sequence with some iterations between phases. We restrict our discussion to the first three phases that are most relevant to our discourse.

The primary task during analysis is to capture the requirements with a view to map the real-world problem to a problem representation. The focus is on what the system is supposed to do rather than on how the system does it. Processes and data are treated as separate entities and are modeled independently. The process model comprises Data Flow Diagrams, which capture the flow of data and their transformations by various processes. Data models are described using Entity Relationship (ER) diagrams that depict entities or data elements, and their associations or relationships. The only commonality between a data model and a process model is that entities in the data model correspond with data stores in the process model. Inter-model consistency may be checked using a data-to-process CRUD (create, read, update, delete) matrix.

The goal of systems design is to choose from alternatives and develop a feasible solution to the problem documented during the analysis phase. The designer maps the requirements obtained in the analysis phase into design models, keeping in mind the constraints imposed by the solution environment. The conceptual data model developed during requirements analysis is normalized and transformed into relational schema. Simultaneously the process model is transformed into structure charts showing the hierarchical relationship among programs that will constitute the IS. Detailed programming logic is also developed using pseudocode in this phase. Design models are mapped into implementation models during system implementation. The

database is created using the schema developed during design. Application programs are then coded based on the structure charts and program logic. Finally, the information system is tested to verify that it meets the users' requirements.

Object-Oriented Systems Development Process

While structured development methods have been around for several decades, the OO approach is still evolving. Our discussion of the OO development process is based on the general principles embodied in the unified development process (Jacobson et al. 1999), an approach that has found favor with a vast majority of the OO community.

The four phases of the OO development lifecycle - inception, elaboration, construction, and transition – are done in an iterative and incremental manner. An information system is developed through several iterations, each consisting of all three systems development tasks, namely, analysis, design, and construction. While Jacobson et al. (1999) describe five development tasks – specification, analysis, design, construction, and testing – the analysis task in our discussion includes requirement specification and construction encompasses testing. Use cases, which represent coherent units of functionality, are used to model the functional requirements of a system. They represent the interaction between the system and all its users. Use cases form the basis for the base-lining of an architecture, and for the identification of classes and their interactions. The OO approach, unlike the structured approach, encapsulates data and procedures into one single unit called an object. Objects represent real world concepts and are organized into classes. Relationships among classes are represented using a class diagram. Multiplicities (akin to cardinalities in Entity-Relationship diagrams) are typically shown on the class diagram. Aggregation, composition, generalization, and specialization relationships make class diagrams semantically richer than data models in the structured approach. Some of the benefits of OO, such as flexibility, extensibility, and reuse, may be achieved by the proper use of such relationships. An object diagram may be used to show relationships between objects at a point in time.

Classes identified during the analysis phase are often refined in the design phase. Attributes and methods now bear a closer resemblance to their implementations. Objects, which are instances of classes, collaborate with one another via message-passing. Interaction diagrams (sequence and collaboration) detail the interactions between objects in a scenario, which is an instance of a use case. The key in OO development is to reuse existing classes, frameworks, components, and design patterns, rather than to build from scratch. In addition to meeting functional requirements, the system must also conform to non-functional requirements such as security, performance (e.g., throughput), scalability, reliability, and robustness.

Object-oriented implementation involves building and testing the designed system. A component-based development approach is often used. Since there is a great emphasis on reuse, the identification, creation, retrieval, and maintenance (e.g., versioning and configuration management) of classes is a crucial activity.

The Unified Modeling Language (UML) has been adopted as a standard by the Object Management Group. To get the most out of the UML, Booch et al. (1999) recommend that developers use a process that is use case driven, architecture centric, and proceeds in iterative and incremental steps. Thus, we see that the OO approach is quite a change from the structured approach to systems development. Table 1 summarizes the comparison between the two approaches.

Pre-paradigmatic Phase and Normal Science in Software Development

The structured approach was the dominant design until the 1980s. During its period of supremacy, a number of programming languages (e.g., C, 4GLs) and programming styles (e.g., procedural, declarative, event-driven) were developed. Yet, these were merely incremental changes that were *tradition-bound* and competence enhancing. They were quite appropriate for the nature and complexity of problems being solved at that time. Therefore, anything that was *tradition-shattering* (e.g., OO concepts), albeit conceptually superior, did not compel the attention of researchers in the structured paradigm.

Although the fundamental tenets of OO were established in the 1960s, it was not until the mid-1980s that developers started looking at it as a viable alternative to the structured approach. This was the period of crisis for the structured paradigm because of its inability to effectively deal with increasingly complex processing and data requirements. The assumptions and values underlying the structured paradigm were inadequate for building complex systems that had to be resilient to changes, reliable, robust, extensible, and flexible. The concept of reusing components was not encouraged. The resulting crisis manifested itself in the form of delayed projects, high maintenance costs, and poor quality of systems.

Table 1. Comparison of OO with Structured Systems Development Process
Sources: Sircar et al. (2001) and Nerur et al. (2002)

	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	Use case model, Class diagram, Interaction diagram
Design	Structure Chart, Detailed program Specifications, Database schema	Design class specifications, Organization of classes into packages, Interface design
Implementation	Program implementation, database implementation, integration and testing	Class implementation, database implementation for persistent objects, integration and testing
Abstraction (Henderson-Sellers and Edwards 1990)	Procedural or algorithmic	Classes
Decomposition (Booch 1991)	Top-down functional decomposition that results in a hierarchy of functions.	OO decomposition, i.e., objects and their interactions. This decomposition does not imply a hierarchy of objects, but rather a collaboration of objects to accomplish tasks.
Hierarchy	Hierarchy of functions - implies a boss-subordinate relationship between a function and its sub-functions immediately below it.	A class hierarchy. The class hierarchy allows one to derive subclasses and specific instances/objects from a class that is at a higher level of abstraction through the process of specialization. The class itself, which may be abstract, is a generalization of subclasses and specific instances
Process Model	The waterfall model or some variation of the waterfall model. This model appeared to be appropriate because of the non-evolutionary nature of software development encouraged by the structured approach.	Boehm's Spiral Model (1988) or Henderson-Sellers' Fountain Model (1990). OO is primarily a prototyping approach that encourages evolutionary software development. Indeed, its basic philosophy is to extend existing systems using pre-defined classes rather than creating a system from scratch.

For almost a decade thereafter, there were efforts to integrate the OO and structured approaches (e.g., Constantine 1989, Ward 1989). There was a proliferation of methods (e.g., the Booch method, OMT by Rumbaugh, the Fusion Method, the Buhr method, etc.) to develop systems using the OO approach (for example, see Arnold et al. 1991). This period was characterized by a great deal of uncertainty about the emergence of OO as a dominant systems development paradigm.

In the mid-1990s, a consensus began to emerge after Booch and Rumbaugh (and subsequently Jacobson) got together. The result of their efforts was the development of the Unified Modeling Language (UML), which was ratified by the Object Management Group as a standard for OO modeling. Today, there appears to be a strong cumulative tradition built around the principles of OO, examples of which include Design Patterns and Component-based development. OO's initial struggle to dethrone the structured approach followed by the emergence of a consensus, and the new guiding values and commitments that are now driving research directions in this field are fairly consistent with Kuhn's assessment of the emergence of new paradigms. The essence of this subsection is summarized in Table 2.

Table 2. A Time Line of Evolution of the OO Paradigm

Phase	Key occurrences in Software Development
Pre-paradigmatic (mid 1980s to mid 1990s)	• Mid-1980s: The software crisis compelled developers to explore alternatives. OO tenets, which received little attention in the late 1960s and the most part of the 1970s, suddenly appear to be a solution to the anomalies such as inflexible systems, high maintenance costs, and long development time. Smalltalk and C++ are already making a lot of news. • Mid-1980s to Mid-1990s: Confusion is rife. Even fundamental definitions of terms are inconsistent across the body of OO literature. Many OO methods are proposed, some of which attempt to integrate OO with the structured approach.
Normal Science (Post mid 1990s)	• The unified approach to modeling as outlined by the Unified Modeling Language (UML) emerges • The UML clarifies many of the terms and introduces notations for modeling that are quickly accepted • The UML is ratified as a standard by the Object Management Group • The UML is supported by many of the tools, and is being used not only for OO modeling, but also for enterprise and business modeling • evelopment of Design Patterns, Frameworks, and Components suggests that a cumulative tradition is being built in the OO paradigm.

Author Co-Citation Analysis (ACA) – A Method to Study Paradigm Shifts

For OO to qualify as a new paradigm from a Kuhnian perspective, it has to be non-cumulative (Hacking 1981). Therefore, if OO really produced a paradigm shift, there should be very little or no ideational links between OO researchers and the community of scientists who have made seminal contributions to the structured paradigm. Kuhn suggests the use of linkages among citations of scientists in a discipline as a way of identifying formal and informal communication networks (Kuhn 1970, pp. 178). The idea that citations and their patterns can be employed to detect the emergence of a paradigm has been expressed by a number of authors (e.g., Kuhn 1970, Sullivan et al. 1980, Weber 1987). The beliefs and values shared by authors in a discipline can be elicited from their writings. The ideational relationships among scientists, whose seminal contributions epitomize their paradigm, essentially establish an invisible structure (Crane 1972).

Author Cocitation Analysis (ACA) is a form of bibliometrics that uses authors as the units of analysis (Culnan 1986). The term author refers not to the author per se, but to the concepts embodied in the author's writings and publications (White and Griffith 1981, Culnan 1986). Therefore, any relationship between two authors in ACA is really a relationship between the concepts expressed by the two authors in their respective writings. ACA uses the frequency of cocitations between pairs of authors as a measure of the degree of closeness between the two authors as perceived by other authors who cite their works.

The gap view of paradigm shift, proposed by Margolis (1993, p.29) defines a "revolutionary paradigm shift as a shift across an intrinsically large space between new and old theories or practices or concepts." ACA has the power to graphically render a conceptual map showing distance of separation between authors by subjecting citation data to quantitative analysis. Since authors in ACA are nothing but substitutes for the concepts they have written about, the map of intellectual structure of a field obtained by using ACA is really a map showing logical distances between various concepts in the field. The presence of a large gap in the intellectual map may provide some clue to the existence of a new paradigm. ACA, therefore, can be used to study the evolution of knowledge in a field and identify paradigm shifts. A brief description of the ACA method is provided in Appendix A. Interested readers may refer to McCain (1990) for a more complete description.

Sircar et al. (2001) used ACA to analyze citation data in the software development field during the time period 1980 to 1994. This study revealed a large conceptual distance between OO and structured methods in the analysis and design phase, but found a relatively shorter gap separating the two approaches in the implementation phase. While this analysis does not conclusively demonstrate a paradigm shift in the software development field, the large conceptual gap between OO and structured analysis and design methods shows preliminary evidence of a revolutionary change, according to the gap view of paradigm shift expounded by Margolis.

Summary and Conclusions

Software development is a dominant activity within the IS discipline and holds significant interest for both researchers and practitioners. Software development processes have undergone radical change over time. OO development methods are replacing structured development methods that were dominant for a long time. While some view this change as a natural evolution others argue this to be a paradigm shift. This research employed the Kuhnian view of paradigm change to study the evolution of software development methods. The OO approach appears to be conceptually very different from the structured approach. An analysis of the history of development of the OO approach also clearly reveals a pre-paradigmatic period (mid 1980s to mid 1990s) followed by a period of normal science (post mid 1990s). There seems to be reasonable evidence to label the OO approach to software development as a new paradigm. This study also proposes and describes ACA as an appropriate technique to study paradigm changes.

The philosophical aspects of software development need more attention than they have received in the past. A good understanding of the epistemology, ontology, and methodology of software development will help researchers to better anticipate future conceptual shifts. It is highly recommended that an in-depth analysis of the ontology and epistemology of software development approaches should be done with a view to developing more effective ways to design information systems.

References

Arnold, P., Bodoff, S., Coleman, D., Gilchrist, H., and Hayes, F., "Evaluation of five object-oriented development methods," in *Journal of Object-Oriented Programming: Focus on Analysis and Design*, SIGS Publication, Inc., New York, 1991, pp. 101-121.

Barker, J., *Discovering the Future: The Business of Paradigms*, ILI Press, 1989.

Boehm, B. W., "A Spiral Model of Development and Enhancement," *Computer*, May 1988, pp. 61-72.

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

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

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

Coad, P. and Yourdon, E., *Object-Oriented Analysis*, Prentice-Hall, Inc., 1990.Constantine, L. L., "Object-Oriented and Structured Methods - Toward Integration," *American Programmer*, Vol. 2, Number 718, August 1989, pp. 34-40.

Crane, D., *Invisible Colleges - Diffusion of Knowledge in Scientific Communities*, The University of Chicago Press, 1972.

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

Gersick, J. G., Connie, "Revolutionary Change Theories: A Multilevel Exploration of the Punctuated Equilibrium Paradigm," *Academy of Management Review*, Vol. 16, No. 1, 1991, pp. 10-36.

Hacking, I., *Scientific Revolutions*, Oxford University Press, 1981.

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

Henderson-Sellers, B. and Edwards, M. J., "The Object-Oriented Systems Life Cycle," *Communications of the ACM*, Vol. 33, Number 9, September 1990, pp. 142-159.

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

Kuhn, S. T., *The Structure of Scientific Revolutions*, second edition, University of Chicago Press, Chicago, IL, 1970.

Ledbetter, L. and Cox, B., "Software ICs," *Byte*, June 1985, pp. 307-315.

Lilly, S., "The Structure of Software Revolutions (and the difficulty of teaching them)," *Object Magazine* (3:3), Sept.-Oct. 1993, pp. 77-79.

Margolis, H., *Paradigms and Barriers*, The University of Chicago Press, Chicago, IL, 1993.

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

Nerur, S., Ramanujan, S., and Kesh, S., "Pedagogical Issues in Object Orientation," *JCSE Online*, April 2002 (http://www.iste.org/sigcs/community/jcseonline/2002/4/nerur.html).

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

Pun, W., and Winder, R., "Design method for object-oriented programming," *Journal of Object-Oriented Programming - Focus on Analysis and Design*, SIGS Publication, Inc., New York, 1991, pp. 61-73.

Sircar, S., Nerur, S., and Mahapatra, R., "Revolution or Evolution: A Comparison of Object-Oriented and Structured Systems Development Methods," *MIS Quarterly*, 25(4), December 2001, pp. 457-471.

Sullivan, D., Koester, D., White, D. H., and Kern, R., "Understanding Rapid Theoretical Change in Particle Physics: A month-by-month Co-Citation Analysis," *Scientometrics*, 2, 1980, pp. 309-319.

Toulmin, S., *Foresight and Understanding*, Harper and Row (Torchbook Series), 1963.

van Gigch, John, P., *System Design Modeling and Metamodeling*, Plenum Press, 1991.Ward, P. T., "How to Integrate Object-Orientation with Structured Analysis and Design," *IEEE Software*, March 1989, pp. 74-82.

Weber, R., "Toward a Theory of Artifacts: A Paradigmatic Base for Information Systems Research," *Journal of Information Systems*, Spring 1987, pp. 3-19.

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.

Appendix A
Author Cocitation Analysis

The first step in ACA is to identify a list of authors who have made seminal contributions to the field. This list could be obtained from experts in the field, editorial boards of journals, or from researchers who have been accepted as fellows by an institute. The next step is to obtain cocitation counts (i.e., the count of all papers **cociting** the works of each pair of authors) for each author pair for the time period of interest. Thus, if there are **n** authors included in the study, cocitation counts/frequencies for **n * (n - 1) / 2** pairs of authors would have to be obtained. Depending on whether the discipline of interest falls within the realm of the sciences, the social sciences, or the arts and humanities, one may use the Science Citation Index, the Social Sciences Citation Index, or the Arts and Humanities Citation Index respectively. This can be done by accessing an online bibliographic database such as Dialog. This approach, however, is very expensive and time-consuming because of the number of author pairs involved. For example, if 30 authors are included in the study, the number of author pairs will be 435. A relatively faster and inexpensive way to obtain the cocitation counts between each pair of authors is to first retrieve the accession numbers for citing/source articles associated with each author and then to count the co-occurrences of these accession numbers between each pair of authors.

Once the cocitation count between each pair of authors is obtained, a raw cocitation matrix (n x n) is created. The raw cocitation matrix is then converted to a Pearson's correlation matrix. The data can now be subjected to multivariate analysis. Cluster analysis, factor analysis, and multidimensional scaling are some of the multivariate techniques frequently used. The raw cocitation matrix is used for factor analysis, whereas the Pearson's correlation matrix is subjected to cluster analysis and multidimensional scaling. The Ward's method and Complete-Linkage are often used in cluster analysis. Principal Components with Varimax rotation seems to be the favored approach for factor analysis. Factor loadings above 0.4 are typically used to identify factors. Oblimin may be used if one wishes to obtain the correlations between factors. Multidimensional scaling gives a graphical rendition of the authors in intellectual space. Since maps with three or more dimensions may be difficult to interpret, most researchers tend to use a two-dimensional map. The Stress value and R-Square may be used as measures of "goodness of fit".

Interpretation and validation of the cluster compositions and factors is the last step in ACA. The results of multivariate analyses should be consistent. The interpretation and validation of the compositions is relatively easy if one is very knowledgeable about the domain. An objective way to interpret and validate the results would be to seek expert opinion.