View metadata, citation and similar papers at core.ac.uk

brought to you by CORE

provided by AIS Electronic Library (AISeL)

Association for Information Systems
AIS Electronic Library (AISeL)

AMCIS 2004 Proceedings

Americas Conference on Information Systems (AMCIS)

December 2004

Philosophical Shifts in Software Development

Randall Brown
University of Texas at Arlington

Sridhar Nerur
University of Texas at Arlington

Craig Slinkman
University of Texas at Arlington

Follow this and additional works at: http://aisel.aisnet.org/amcis2004

Recommended Citation
Brown, Randall; Nerur, Sridhar; and Slinkman, Craig, "Philosophical Shifts in Software Development" (2004). *AMCIS 2004 Proceedings*. 516.
http://aisel.aisnet.org/amcis2004/516

This material is brought to you by the Americas Conference on Information Systems (AMCIS) at AIS Electronic Library (AISeL). It has been accepted for inclusion in AMCIS 2004 Proceedings by an authorized administrator of AIS Electronic Library (AISeL). For more information, please contact elibrary@aisnet.org.

Philosophical Shifts in Software Development

Randall Brown
University of Texas at Arlington
rwb6854@exchange.uta.edu

Sridhar Nerur
University of Texas at Arlington
snerur@uta.edu

Craig Slinkman
University of Texas at Arlington
slinkman@uta.edu

ABSTRACT

Drastic changes have occurred in the field of software development in the past few years. Concepts that have been proposed in recent times are very different from what has been done in the past. The philosophical shifts underlying these changes present many challenges – both technical and organizational. Such shifts are not peculiar to software development as parallels in other disciplines exist. The goal of this paper is to articulate the changes from a philosophical perspective and to examine the organizational implications that arise as a consequence. In particular, the article focuses on the conceptual differences between traditional and contemporary approaches within the context of inquiring systems, the philosophy of science, general systems theory, and soft systems methodology.

Keywords

Software development, agile methodologies, philosophical shifts, inquiring systems.

INTRODUCTION

Software development is a complex process that takes an inordinate amount of time, effort, and creativity. Software researchers, not unlike researchers in other disciplines, initially were preoccupied with an engineering approach, based entirely on the assumption that problem-solving is a mechanistic process where the steps can be pre-determined and fully specified. Further, they assumed that a rational approach would lead to an optimal solution. Some of the concepts that have been proposed in recent times are fundamentally opposed to this viewpoint (Highsmith, 2003).

Since the first computer was built, software has taken on an increasingly important role in our lives. The most obvious uses of software are the applications used on computers, such as office automation, accounting, tax preparation, educational, games, etc., Consumers are constantly demanding newer and more powerful versions of these software. This demand has put a lot of pressure on software developers to more rapidly produce larger numbers of applications in a shorter amount of time, and with fewer "bugs". Competition has also been increasing, further pressuring the need for more, faster, and higher quality applications. The advent of the internet and the increasing need to deploy e-commerce applications that exhibit very different characteristics, such as high resilience to change and quick turnaround cycles, further demonstrates the importance and complexity of software. Gaming systems, such as Nintendo, Game Boy, PS-2, etc. are also increasing in popularity and variety, and there are thousands of games available for them. Such systems would be inconceivable without the rapid strides that have been made in software development technologies.

Computer applications, however, are only the beginning. As computers pervade nearly all aspects of our lives, software must follow. Things that we practically use everyday, such as watches, cars, vacuum cleaners, cell phones, kitchen appliances, and stereo systems, are increasingly relying on sophisticated computer and software systems. In many cases, these technologies are seamlessly integrated with our lives and we are often unaware of their presence. There are even refrigerators that can monitor usage trends of items such as eggs and milk and can automatically place orders for groceries when supplies get low. These have been just a few of the ways in which computers and software are driving the way we live. Computers are ubiquitous, and so there is also software everywhere, and developers must continually question their assumptions and adopt new ways of thinking.

A study of the philosophical underpinnings of software development is useful for several reasons. First, it allows us to logically take stock of what has occurred thus far and where we are headed. Second, it enables us to have a better grasp of what we as researchers are doing. Third, it provides insights into the challenges and barriers that are presented to

organizations endeavoring to adopt these new approaches. Finally, by looking at similar epistemological patterns in other disciplines we will have some indication of what the future holds for us.

The study proceeds as follows. In the next section, we review the developments in the field of software. This is followed by a discussion on the conceptual differences between traditional and current philosophies of development, mainly from the perspectives Inquiring Systems as outlined by Churchman (1971) and Soft Systems Methodology (SSM) as described by Checkland (1981). The paper also touches on some of the concepts related to *General Systems Theory* as propounded by Bertalanffy (1968). Finally, the Kuhnian Philosophy of Science (1970) is used to argue that recent philosophical shifts in software development pose serious individual and organizational challenges.

EVOLUTION OF SOFTWARE APPROACHES

Fundamentally, there have been three broad approaches to software development. These are: 1) The structured approach, 2) Object-Oriented Development, and 3) Agile Methodologies. We discuss the characteristics of each of these below.

The Structured Approach

These can be broken down into process-oriented and data-oriented approaches (Henderson-Sellers and Edwards, 1990). Essentially, they treat data and procedures as separate entities. The process-oriented approach, epitomized by functional decomposition, assumes that the system has a main function that can be systematically broken down into a hierarchy of functions and sub-functions. There exists a boss-subordinate relationship between the functions and their sub-functions. This is essentially a reductionistic approach that deals primarily with parts, i.e., functions. Relationships are given scant attention. Deductive reasoning (moving from the general to the specific) is what guides this process. Analysis is the basis for this approach, with little or no emphasis on synthesis. In the process oriented approach, primary emphasis is on algorithms. Figure 1 shows an example of a banking application using the structured approach.

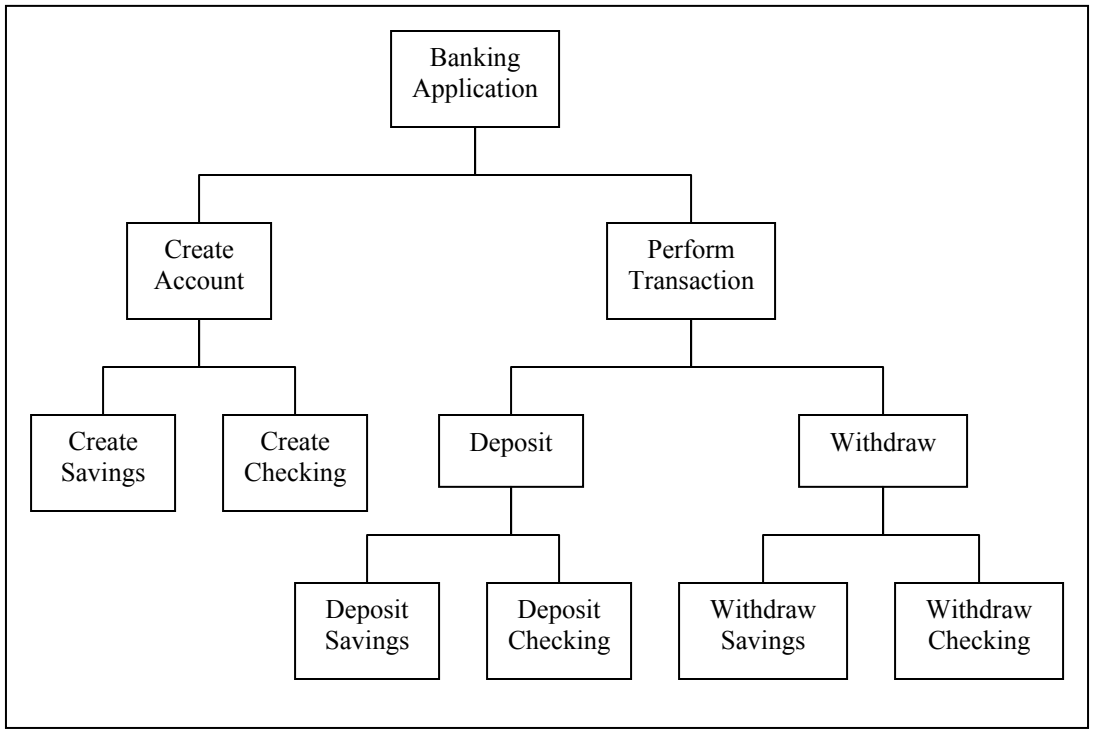

Figure 1: Traditional Design of a Simple Banking System

As shown in Figure 1, the system starts with one top-level function that is then systematically broken down into sub-functions, which in turn are further decomposed into functions. There exists a boss-subordinate relationship between a function and its sub-functions. The reductionistic bias with a predilection for "parts" is evident in the structured approach.

Object-Oriented Approach

As opposed to the structured approach, OO encapsulates data and procedures into a whole called an object (Henderson-Sellers and Edwards, 1990). Objects are the primary sources of concern to developers in an OO system. Objects interact with one another to accomplish the behaviors of a system. Objects that share common properties (i.e. attributes and behaviors) and semantic relationships are grouped into an abstraction called a class (Booch, Rumbaugh, and Jacobson, 1999). The decomposition of the system is in terms of classes and their relationships rather than as a hierarchy of functions (Booch, 1991). A tree-like representation of classes and their subclasses is the only hierarchy of interest in an OO system. Subclasses in an OO system inherit the attributes and methods of their super classes. Figure 2 demonstrates the OO method of decomposing the problem using the banking system as the domain of interest. The focus here is clearly on the relationships between the classes, which are high-level abstractions that map on to real-world concepts relevant to the domain being modeled (Richter, 1999).

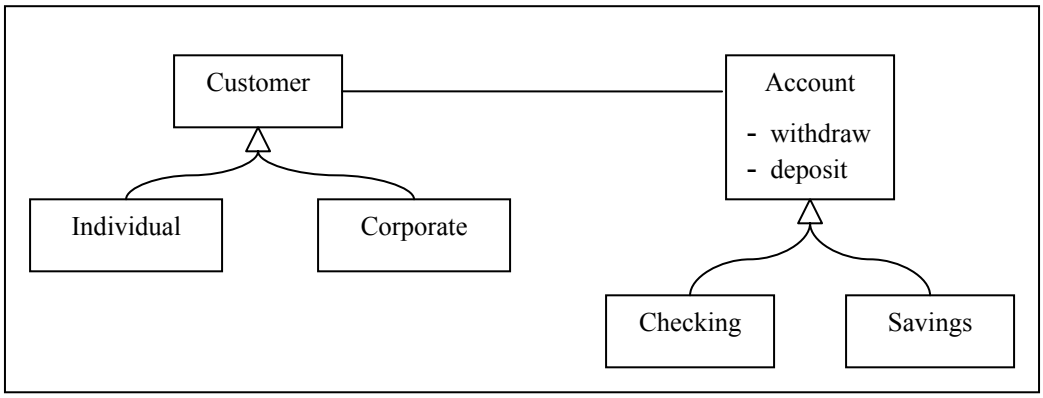

Figure 2: OO Design of a Simple Banking System

In the OO approach, both analysis and synthesis play a very important role. For example, analysis is required to specialize a class, while synthesis could be used to arrive at higher abstractions/generalizations by inducing common properties (i.e. attributes and behaviors) as well as semantic relationships of the lower level objects/classes (Richter, 1999). OO emphasizes modeling and is seen as a more holistic approach, with great emphasis on patterns/form than on substance or part. In other words, the understanding of relationships is paramount in comprehending the behavior of the whole system. The system is likely to display *emergent* properties that are not discernible by looking at a part (i.e. a class) in isolation. Abstraction, decomposition, and inheritance play important roles in OO modeling, and these differ considerably from the way they are employed in the structured approach.

In summary, OO exhibits the following characteristics that conceptually distinguish it from the structured approach discussed above (Henderson-Sellers and Edwards, 1990; Korson and McGregor, 1990; Booch, 1991; Richter, 1999).

1. OO emphasizes modeling.
2. Decomposition in OO results in a web of classes and their relationships.
3. Behaviors in the system are accomplished through interactions between objects.
4. Abstractions are closely identifiable with real-world concepts present in the domain of interest.

Principles such as inheritance, generalization, specialization, and composition allow us to provide a semantically stronger representation of the system being developed.

Agile Development

Increasing frustration with existing software practices compelled a few researchers to adopt an entirely new way of developing software. This new approach is influenced considerably by some of the principles articulated in the theory of complex adaptive systems and is geared towards imparting some of the characteristics of living systems to software (Highsmith, 2002). This viewpoint is philosophically different from previous design traditions and promises to afford the benefits of agility and adaptability in an environment where change is inevitable and constant (Cockburn and Highsmith, 2001a).

The philosophy of agile development values people over processes and tools, emphasizes working and deliverable software over unnecessary documentation, urges the active involvement and participation of customers rather than negotiating

schedule and cost contracts with them, and, above all, recognizes the need to create change as well as acknowledge its inevitability rather than attempt to eliminate it entirely by careful upfront planning (Cockburn and Highsmith, 2001a; Highsmith 2002, 2003). These tenets are embodied in a host of methods, collectively called Agile Methodologies. Extreme programming (XP), Scrum, Feature Design-Development (FDD), Crystal Methodologies, Dynamic Systems Development Methodology (DSDM), Adaptive Software Development, and Lean programming are some examples of these methods. While these ideas are new to the software development community, their antecedents may be traced back to systems thinking and the theory of living and complex adaptive systems, and more recently to lean manufacturing techniques (Highsmith, 2002).

	Traditional	Agile
Fundamental Assumptions	Long and careful planning yields a sequence of steps and their consequences Project proceeds in linear fashion Deterministic approach that eliminates uncertainty through reasoning	Short, intense periods of planning at the beginning of iteration Feature driven and proceeds in an iterative, evolutionary manner Ambiguity/uncertainty reduced through cycles of rapid feedback and continuous improvement
Goal	Optimization	Adaptability/agility
Role of the project manager	Traditional roles of controller, planner, organizer, staffer, decision-maker, scheduler	Facilitator/leader in a synergistic, collaborative environment
Mgmt Style	Hierarchical "command-and-control"	Leadership/collaboration
Process Model	Waterfall, spiral, etc. Emphasizes linear sequence of process steps.	Evolutionary-development model proposed by Gilb
Decision-making context	Unitary	Pluralistic
Systems Thinking	Hard-systems thinking	Soft-systems thinking
Inquiring System	Closer to Liebnitz and Locke	Closer to Singerian philosophical school
Organizational Structure	Hierarchical control-oriented, mainly centralized decision-making	Decentralized blending of cooperation and autonomy
Thought process	Driven by strict, predetermined rules established by the process model, which basically makes one react in predictable ways to unusual situations that might arise	Urges developers to use patterns to solve problems by relying on one's ability to innovate depending on the contingency.
Dealing with complexity	Assumes that complexity and ambiguities can be predicted, measured, and corrected.	Deals with complexity and uncertainties by using the ingenuity of people and relies on rapid feedback and adaptability.
Customer involvement	Not directly involved in the development process.	Mandatory, active participation throughout the development
Team composition	Relatively homogeneous	Self-organizing teams involving relevant stakeholders who may have diverse perspectives and disparate goals.
Assignment of roles	Specific responsibilities for each role (e.g., architect, analyst, programmer, etc.)	No clear separation of roles

Table 1. Conceptual Differences Between Traditional and Agile Approaches

The agile approach is a significant departure from the other two approaches mentioned above. These differences arise from their opposing conceptual and philosophical perspectives and have far-reaching implications for their assimilation in

organizations. Therefore, it is imperative that managers understand the consequences of these differences. Table 1, drawn from a variety of sources (Churchman, 1971; Burrell and Morgan, 1979, Checkland, 1979; Cavaleri and Obloj, 1993; Cockburn and Highsmith, 2001a, b; Boehm, 2002; Highsmith, 2001, 2002, 2003; Orr, 2002; Larman, 2004), delineates the fundamental dimensions that distinguish the agile approach from past practices. These distinctions provide the basis for our discussions on the philosophical shifts that have occurred in software development.

PHILOSOPHICAL SHIFTS IN SOFTWARE DEVELOPMENT

In the context of this paper, the term traditional development refers to the structured and object-oriented approaches. In reality, as shown in Figure 3, there exists a continuum from structured to OO to agile, with gradual philosophical shifts. The philosophical perspectives delineated in this figure are from Churchman (1971), Burrell and Morgan (1979), Checkland (1981), Cavaleri and Obloj (1993), and Kienholz (1999).

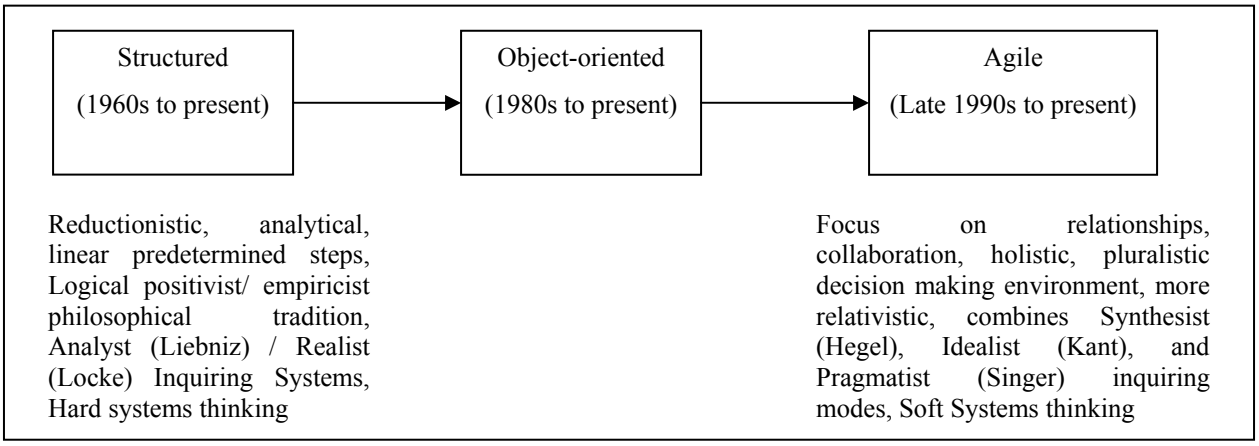

Figure 3. Philosophical Shifts in Software Approaches

The continuum shows that significant conceptual changes have occurred in the field of software development since its inception. Like many other disciplines (such as biology, linguistics, physics), the field has grappled with the tension between substance and form, reductionism and holism, measurement versus assessment, objectivity as opposed to subjectivity in design, and so forth. And, not unlike these other disciplines, there has been a gradual but evident shift in focus from parts to relationships between parts, from a mechanistic view to an organic one arising from the realization that truly emergent properties such as those that are present in living systems can be achieved only by imparting the characteristics of complex adaptive systems to the software system being designed.

Traditional approaches to software were considerably influenced by engineering and scientific methods of inquiry, based on the notion that all ambiguities could be resolved through reason and analysis. The approaches sought to find an optimal solution by articulating a sequence of steps and by dealing with any anomalies that might arise in a predetermined way. Also, these approaches placed a greater premium on processes and technologies than on people and their competencies. These characteristics are in the spirit of the philosophical traditions of hard system thinking, which aligns itself with logical positivism (Cavaleri and Obloj, 1993). From the perspective of Churchman's inquiring systems, the deterministic assumptions of the traditional development methods reflect the philosophical orientations of Liebniz and Locke (Churchman, 1971). The Liebnizian school with its reliance on analytical models and quantitative methods sought truth within the system, thus relying on reductionistic principles using deductive logic. Locke's realistic view sought confirmation of the truth outside the system using inductive processes. Both Locke and Liebniz rely on data and facts (van Gigch, 1978).

The agile approach requires a new way of thinking. It questions some of the fundamental assumptions and wisdom that have hitherto guided software development. The recognition that software systems are inherently complex and need to have adaptive properties, much like living systems, has been one of the motivating factors for this revolutionary way of thinking about software development. The antecedents may be traced to the works of Christopher Alexander (1979) on using patterns in architecture (Beck, 1999), to the tenets of complex adaptive systems (Highsmith, 2002), and to systems thinking (Highsmith, 2001).

The principles underlying agile development closely parallel the concepts characterizing the Soft Systems Methodology as elucidated by Checkland (1981). The most prominent of these are the emphasis on people, on shared learning, on pluralistic decision making contexts, and the idea of continuous improvement to solve the problem in incremental steps rather than

relying on engineering and scientific approaches (Cavaleri and Obloj, 1993). In agile, it is not a pre-written, fully anticipated requirements specification that guides development. Rather, requirements unfold in an incremental, iterative fashion based on a dynamic prioritization of the desired features of the software product as determined by the stakeholders. Short cycles of intense planning and development are followed by periods of reflection (sometimes referred to as *reflection workshops*) during which an assessment of what worked and what didn't is made (Beck, 1999; Highsmith, 2003). This is the mechanism for providing rapid feedback, a valuable means of overcoming uncertainties and ambiguities that arise along the way (Cockburn and Highsmith, 2001a). The increased emphasis on collaborative decision making and cooperative work habits helps to integrate a wide variety of views and at the same time fosters an environment that is conducive to learning.

Unlike the logical positivists who firmly believe that logic and the resulting universal laws can dispel the ambiguities surrounding a problem, the phenomenologists and relativists believe that the construction of reality occurs within oneself, primarily through the use of past experience, personal knowledge, and thinking (Cavaleri and Obloj, 1993). The principles behind Soft Systems Methodology (SSM) are based on phenomenological assumptions (Checkland, 1981). Checkland's clarification of the philosophical position of SSM also includes the philosophy of hermeneutics as proposed by Wilhelm Dilthey. The following description of the *hermeneutic circle* by Checkland is similar to the evolution of the software product through continuous improvement in the agile approach, a process that relies on shared learning through discovery.

> "…the method comprising a circular process of discovery called 'the hermeneutic circle', a means of perceiving social wholes as both wholes and parts. A preliminary overview of subject matter is used to guide an examination of what the parts denote; this clarifies the concept of the whole, which at the end of the cycle must be perceived so that all the parts can be related to it. Thus, there are no fixed or absolute starting points, only an iterative cycle which gradually leads to increased understanding of social reality." (p. 276).

Thus, the agile approach, with its strong conceptual ties to SSM, is more closely aligned with the phenomenological and hermeneutics schools of thought.

Agile encourages one to continually reexamine assumptions and to create change and use it to one's advantage (Highsmith, 2002, 2003). Thus, the Hegelian philosophy of synthesizing opposing viewpoints (through argumentation and dialectics) is embodied in agile practices. Effective decision making in the agile approach is overwhelmingly determined by the ability to integrate the views of a disparate group of stakeholders, closely related to the philosophical traditions of the Kantian inquiring system. Finally, the philosophy of not tackling the entire problem at once, but relying on iterative cycles of continuous improvement through innovation and adaptation is in line with the Singerian philosophical mode of inquiry.

Characteristics	Philosophical/Conceptual school or tradition
People-centric, focus on learning and experimentation, reflection workshops, incremental development	Soft Systems Methodology
Pluralistic decision making context	Soft Systems Methodology, Kantian (Idealist) philosophy
Questioning assumptions and encouraging change	Hegel (Synthesist) philosophy
Short-term incremental approach, continual refinement of the system using innovation and adaptation	Singer (Pragmatist) school of philosophy
Emphasis on relationships, synthesis of multiple perspectives, focus on form (i.e. pattern) rather than on substance (i.e. part)	Science of Design

Table 2: The Philosophical Orientation of Agile Development

The conceptual foundations of General Systems Theory (GST) include many of the notions previously discussed (van Gigch, 1978; Ulrich, 1980). In particular, the proponents of GST distinguish between the epistemology of the *Science of Analysis* and that of the *Science of Design*. The Science of Analysis is an analytical quest for the truth, with reductionism and objectivity being central to the modes of inquiry. On the other hand, the Science of Design looks at what the system ought to be, relying on principles of synthesis, holism, and the subjective construction of reality in order to arrive at the "truth". Several aspects of the *Science of Design* are reflected in agile practices. A summary of the key characteristics of agile methods and the philosophical schools with which they may be associated is shown in Table 2.

IMPLICATIONS/CONCLUSIONS

The increasing complexity of software is primarily because of all the changes that are occurring in the world of business. There is a great demand on organizations to quickly produce high-quality applications that are resilient to changes, particularly in a turbulent business environment. It is this need for alacrity in responding to various competing forces that has compelled researchers to explore radically different ways of developing software. At the forefront of this revolution is the agile methodology, an approach that questions the very fundamentals of everything that has been done before.

These new ways of thinking about software development have serious implications for decision making, inquiring into problems and solving them, organizational strategies and structures, management style, etc. Software development is essentially a social process in which the actions and behaviors of its participants are dictated to a large extent by beliefs, assumptions, biases and values that are built over time. As these get reinforced, they shape the culture that pervades all activities associated with software development. In addition, these values become ingrained in operational routines that are difficult to change. As we have argued in the previous section, the agile approach is built on a philosophical foundation that is different from the traditional one. Therefore, the transition to agile denotes a paradigm shift in the Kuhnian sense, one that requires careful thought and the expenditure of enormous amounts of time and money.

The philosophy of science as expounded by Kuhn details the difficulties of abandoning a dominant paradigm in favor of a new one. In particular, Kuhn's observation that assumptions and beliefs are very difficult to change makes the issues of adopting agile all the more challenging, as it entails very different ways of solving problems. Further, major organizational reorientations have to occur for agile to be successful. For example, an organization that thrives on strict hierarchical control may find it difficult to provide an environment that is conducive to collaborative decision making and cooperation among diverse stakeholders that epitomizes the new philosophy of software development. A change in fundamental assumptions also entails new problem-solving strategies, communication channels, roles and assignments, relationships, and so forth.

Margolis (1993) argues that habits of mind created from years of doing things in a particular way, within an established framework of assumptions and values, are hard to change. This is similar to the concept of signature skills which are skills that one becomes identified with in an organization. People are not easily persuaded to relinquish such skills because of their emotional attachment to them (Leonard-Barton, 1995). Clearly, the path to agility has many barriers, not just technical but also organizational. It is the latter that poses a greater challenge to the assimilation of agile methods.

REFERENCES

1. Alexander, C. (1979) *The Timeless Way of Building*, Oxford University Press, New York.
2. Beck, K. (1999) Embracing Change with Extreme Programming, *Computer*, 32, 10, 70-77.
3. Bertalanffy, L. von (1968) *General Systems Theory*, New York: Braziller.
4. Boehm, B. (2002) Get ready for agile methods, with care, *Computer*, 35, 1, 64-69.
5. Booch, Grady (1991) *Object Oriented Design with Applications*, The Benjamin/Cummings Publishing Company, Inc.
6. Booch, G., Rumbaugh, J., and Jacobson, I. (1999) *The Unified Modeling Language User Guide*, Addison-Wesley, Reading, MA.
7. Burrell, G. and Morgan, G. (1079) *Sociological Paradigms and Organizational Analysis*, Heinemann, London.
8. Cavaleri, S. and Obloj, K. (1993) *Management Systems: A Global Perspective*, Wadsworth Publishing Company, CA.
9. Charette, R. (2001) The decision is in: Agile versus heavy methodologies, *Cutter Consortium Agile Project Management Executive Update*, 2, 19.
10. Checkland, Peter (1981) *Systems Thinking, Systems Practice*, New York: John Wiley & Son.
11. Churchman, C. West (1971) *The Design of Inquiring Systems, Basic Concepts of Systems and Organization*, Basic Books, New York.
12. Cockburn, A. and Highsmith, J. (2001a) Agile Software Development : The People Factor, *Computer*, 34, 11, 131-133.
13. Cockburn, A. and Highsmith, J. (2001b) Agile Software Development: The Business of Innovation", *Computer*, 34, 9, 120-127.
14. Henderson-Sellers, Brian and Edwards, M. Julian, (1990) The Object-Oriented Systems Life Cycle, *Communications of the ACM*, 33, 9, 142-159.
15. Highsmith, J. (2001) Order for Free: An Organic Model for Adaptation, in L.L. Constantine, Ed., *Beyond chaos: the expert edge in managing software development*, Addison-Wesley, 251-257.

16. Highsmith, J. (2002) *Agile Software Development Ecosystems.* Addison-Wesley, Boston: MA.
17. Highsmith, J. (2003) *Cutter Consortium Reports: Agile Project Management: Principles and Tools*, 4, 2, Cutter Consortium, Arlington, MA.
18. Kienholz, A. (1999) Systems ReThinking: An Inquiring Systems Approach to the Art and Practice of the Learning Organization, www.cba.uh.edu/~parks/fis/inqre2a1.htm, Working Paper.
19. Korson, Tim and McGregor, D. John (1990) Understanding Object-Oriented: A Unifying Paradigm, *Communications of the ACM*, 33, 9, 40-60.
20. Kuhn, S. Thomas (1970) *The Structure of Scientific Revolutions*, second edition, University of Chicago Press, Chicago.
21. Larman, C. (2004) *Agile & Iterative Development: A Manager's Guide*, Pearson Education, Inc. (Addison-Wesley).
22. Leonard-Barton, D. (1995) *Wellsprings of Knowledge*: *Building and Sustaining the Sources of Innovation*, Harvard Business School Press, Boston, MA.
23. MacCormack, A. (2001) Product-development practices that work: How Internet companies build software, *MIT Sloan Management Review*, 42, 2, 75-84.
24. Margolis, H. (1993) *Paradigm and Barriers: How Habits of Mind Govern Scientific Beliefs*, University of Chicago Press: Chicago.
25. Orr, K. (2002) CMM Versus Agile Development: Religious Wars and Software Development, *Cutter Consortium Agile Project Management Executive Report*, 3, 7, Cutter Consortium, Arlington, MA.
26. Richter, C. (1999) *Designing Flexible Object-Oriented Systems with UML*, MacMillan Technical Publishing.
27. Ulrich, Werner (1980) The Metaphysics of Design: A Simon-Churchman 'Debate', *Interfaces*, 10, 2, 35-40.
28. van Gigch, John, P. (1978) *Applied General Systems Theory*, second edition, Harper, New York.