By Sridhar Nerur, RadhaKanta Mahapatra, *and* George Mangalaraj

Challenges of Migrating to Agile Methodologies

Organizations must carefully assess their readiness before treading the path of agility.

Software development methodologies are constantly evolving due to changing technologies and new demands from users. Today's dynamic business environment has given rise to emergent organizations that continuously adapt their structures, strategies, and policies to suit the new environment [12]. Such organizations need information systems that constantly evolve to meet their changing requirements—but the traditional, plan-driven software development methodologies lack the flexibility to dynamically adjust the development process.

Illustration by Richard Downs

While object-oriented (OO) approaches provide a viable method to incrementally develop information systems, a host of new methods called agile development methodologies or lightweight methodologies claim to go a step further in overcoming the limitations of traditional plan-driven ones. A number of software development methods such as extreme programming (XP), feature-driven development, crystal clear method, scrum, dynamic systems development, and adaptive software development, fall into this category [1, 4, 7]. Our objective here is to articulate the challenges that CIOs and project managers must be cognizant of in their endeavors to embrace the agile philosophy of software development.

While traditional methodologies, such as life cycle-based structured and OO approaches, continue to dominate the systems development arena, numerous opinion pieces and several surveys clearly demonstrate the growing popularity of agile methodologies. The advent of these methodologies has divided the software development community into opposing camps of traditionalists and agilists, with each group proclaiming the superiority of its own methodology. A more balanced view of the two competing methodologies is offered by a few who suggest that each method has its strengths as well as limitations, and is appropriate for specific types of projects [1, 2, 6, 10]. According to Boehm, "organizations must carefully evolve toward the best balance of agile and plan-driven methods that fits their situation [1]." Clearly, most organizations cannot ignore the agile wave, but for organizations steeped in the traditional systems development methodologies, adoption of agile methodologies will likely pose several challenges, since the two software development methodologies are grounded in opposing concepts.

Past research shows that software development process changes represent complex organizational change phenomena and cannot be accomplished merely by replacing current tools and technologies with new ones [11]. Such changes may impact several aspects of the organization including its structure, culture, and management practices. Therefore, understanding organizationwide ramifications of a change phenomenon is a critical first step in planning and managing such changes. In this article, we have consciously taken an organizational and managerial perspective of this change phenomenon primarily because such a perspective, although critical in implementing organizational change, is largely missing from the current discourse on adoption of agile methodologies. We provide a brief comparison of agile development methodologies with traditional systems development methodologies, and discuss the challenges of adopting agile methodologies.

MOST ORGANIZATIONS CANNOT IGNORE THE AGILE WAVE, BUT FOR THOSE STEEPED IN TRADITIONAL SYSTEMS DEVELOPMENT, ADOPTION OF AGILE METHODOLOGIES WILL LIKELY POSE SEVERAL CHALLENGES.

Software development is a complex activity characterized by tasks and requirements that exhibit a high degree of variability [2, 8]. Uncertainties are further compounded by the diversity and unpredictability of people who engage in such tasks [5, 6]. The changing nature and sophistication of tools (for example, a development environment including programming languages, techniques, and so on) may also exacerbate development problems. A rationalized, engineering-based approach has dominated software development almost since its inception. Such an approach, grounded in the principles of hard systems thinking, assumes that problems are fully specifiable, and that an optimal and predictable solution exists for every problem [3]. Extensive upfront planning is the basis for predicting, measuring, and controlling problems and variations during the development life cycle. The traditional software development approach is process-centric, guided by the belief that sources of variations are identifiable and may be eliminated by continually measuring and refining processes [4]. The primary focus is on realizing highly optimized and repeatable processes. Thus, planning and control accomplished by a command and control style of management provide the impetus for developing a software product [8].

Systems development in the traditional approach is guided by a life cycle model such as the waterfall model, the spiral model, or some variations of these.

The life cycle model specifies the tasks to be performed and the desired outcomes of each phase, and assigns roles (such as systems analyst, programmer) to individuals who will perform these tasks. In addition to the end product of working code, these methodologies also produce a large amount of documentation that codifies process and product knowledge. Communication among project participants is formalized through these documents. Customers play an important role during specification development, but their participation is minimal in other activities. These methodologies are appropriate both for OO and non-OO technologies.

	Traditional	Agile
Fundamental Assumptions	Systems are fully specifiable, predictable, and can be built through meticulous and extensive planning.	High-quality, adaptive software can be developed by small teams using the principles of continuous design improvement and testing based on rapid feedback and change.
Control	Process centric	People centric
Management Style	Command-and-control	Leadership-and-collaboration
Knowledge Management	Explicit	Tacit
Role Assignment	Individual—favors specialization	Self-organizing teams—encourages role interchangeability
Communication	Formal	Informal
Customer's Role	Important	Critical
Project Cycle	Guided by tasks or activities	Guided by product features
Development Model	Life cycle model (Waterfall, Spiral, or some variation)	The evolutionary-delivery model
Desired Organizational Form/Structure	Mechanistic (bureaucratic with high formalization)	Organic (flexible and participative encouraging cooperative social action)
Technology	No restriction	Favors object-oriented technology

Table 1. Traditional versus agile software development.

Unlike the traditional methodologies, agile methodologies deal with unpredictability by relying on people and their creativity rather than on processes [5]. They are characterized by short iterative cycles of development driven by product features, periods of reflection and introspection, collaborative decision making, incorporation of rapid feedback and change, and continuous integration of code changes into the system under development [4, 7, 8]. A project is broken down into sub-projects, each of which typically involves planning, development, integration, testing, and delivery. Developers work in small teams with customers (representing system users) as active team members. The features to be implemented in each development cycle are jointly decided by the customer and the rest of the development team. Collaborative decision making involving stakeholders with diverse backgrounds and goals is thus a characteristic of agile development. Agile methodologies favor a leadership-and-collaboration style of management where the project manager's role is that of a facilitator or coordinator [8].

The deliverable of each development cycle is working code that can be used by the customer. Agile methods discourage documentation beyond the code. Product knowledge, therefore, becomes tacit. Rotation of team membership ensures this knowledge is not monopolized by a few individuals. Further, the iterative strategies that characterize agile methodologies are best supported by OO technologies. The evolutionary-delivery model proposed by Gilb provides a meaningful framework to guide agile software development [9].

To summarize, agile development is characterized by social inquiry in which extensive collaboration and communication provide the basis for collective action [5, 6, 8]. Diverse stakeholders including developers and end users go through repeated cycles of thought-action-reflection that foster an environment of learning and adaptation. Team members, empowered with more discretionary and decision-making powers, are not confined to a specialized role. This increases the diversity/variety of the teams and enables them to self-organize and respond with alacrity to emergent situations. Table 1 summarizes the comparison between traditional and agile methodologies.

Making Agile Methodologies Work

For decades organizations have relentlessly pursued the goal of creating optimized and repeatable processes [4, 7, 8]. The stability they yearned for presents one of the biggest hurdles to adopting agile development methodologies. The variations between traditional and agile methodologies previously detailed suggest that organizations must rethink their goals and reconfigure their human, managerial, and technology components in order to successfully adopt agile methodologies. Here, we identify the key management, organizational, people, process, and technological issues in adopting agile methodologies. Table 2 summarizes the main issues involved under each component.

Management and organizational issues. Organizational culture has a significant impact on the social

structure of organizations, which in turn influences the behavior and actions of people [3, 6, 8]. The values, norms, and assumptions of an organization are stabilized and reinforced over time, and are reflected in the policies embodied in organizational routines. Culture exerts considerable influence on decision-making processes, problem-solving strategies, innovative practices, information filtering, social negotiations, relationships, and planning and control mechanisms. Neither culture nor mind-sets of people can be easily changed, which makes the move to agile methodologies all the more formidable for many organizations [2].

As mentioned, agile methodologies require a shift from command-and-control management to leadership-and-collaboration. The organizational form that facilitates this shift needs the right blend of autonomy and cooperation to achieve the advantages of synergy while providing flexibility and responsiveness [3]. The project manager's traditional role of planner and controller must be altered to that of a facilitator who directs and coordinates the collaborative efforts of those involved in development, thus ensuring that the creative ideas of all participants are reflected in the final decision [8]. The biggest challenge here is to get the project manager to relinquish the authority he/she previously enjoyed.

Knowledge management is of vital importance to organizations. Traditional development approaches create much documentation. Such records serve as useful artifacts for communication and traceability of design. Agile methodologies, on the other hand, encourage lean thinking and cutting down on overhead, particularly documentation. Much of the knowledge in agile development is tacit and resides in the heads of the development team members [1, 8]. This can make the organization heavily dependent on the development teams and can potentially shift the balance of power from the management to the development teams. Such a situation may not be acceptable to many organizations. This impasse can be resolved by determining which knowledge should be codified and what may remain tacit.

Agile development relies on teamwork, as opposed to individual role assignment that characterizes traditional development. Performance measurement and reward systems, therefore, must be suitably designed for successful adoption of agile methodologies.

People-related issues. A cooperative social process characterized by communication and collaboration between a community of members who value and trust each other is critical for the success of agile methodologies [5, 8]. For programmers accustomed to solitary activities or working with relatively homogeneous groups of analysts and designers, the ideas of shared learning, reflection workshops, pair-programming, and collaborative decision making may be overwhelming.

Management and organizational
• Organizational Culture • Management Style • Organizational Form • Management of Software Development Knowledge • Reward Systems
People
• Working effectively in a team • High level of competence • Customer relationships—commitment, knowledge, proximity, trust, respect
Process
• Change from process-centric to a feature-driven, people-centric approach • Short, iterative, test-driven development that emphasizes adaptability • Managing large, scalable projects • Selecting an appropriate agile method
Technology (Tools and Techniques)
• Appropriateness of existing technology and tools • New skill sets—refactoring, configuration management, JUnits

Table 2. Key issues in migrating to agile.

At the present time, there is little evidence to suggest that agile principles will work in the absence of competent and above-average people [1, 2]. This can pose serious problems related to staffing and morale. First, it will be difficult to find enough personnel to staff software development teams that use agile methodologies. Second, it will create a culture of elitism within the systems development group that may affect the morale of non-agile developers.

In an agile environment, the development team comprising software developers and the customer makes most of the decisions. This creates a pluralist decision-making environment due to the diverse backgrounds, attitudes, goals, and cognitive dispositions of the team members [3]. Decision making in this environment is more difficult compared to the traditional approach where the project manager is responsible for most decisions. It may take an organization enormous effort, time, and patience to build a culture of trust and respect among its employees to facilitate such collaborative decision making.

The success of agile development hinges on finding customers who will actively participate in the development process. Further, the customers are expected to be "Collaborative, Representative, Authorized, Committed, and Knowledgeable" [2]. It is not an easy task to find such persons, especially for complex systems.

Process-related issues. The problem of changing attitudes and practices from process-centric to people-

centric is acute. Organizations that have for years attempted to achieve higher levels of CMM are particularly susceptible to this problem. The idea of changing a process to fit the capabilities and competencies of people and the characteristics of the project, rather than using a rigid process encompassing standardized activities may be sound [5], but can be achieved only through significant investment of time, effort, and capital.

Traditional processes are compliance-driven and activities- and measurement-based, aimed at providing assurance [1, 2, 7]. Agile methodologies rely on speculation, or planning with the understanding that everything is uncertain, to guide the rapid development of flexible and adaptive systems of high value [7, 8]. They stress the importance of assessing as opposed to measuring, and are highly tolerant of change. One of the biggest barriers to migration is the change in a process model from a life cycle model to one that supports feature-based development using evolutionary and iterative development. Such a change entails major alterations to work procedures, tools and techniques, communication channels, problem-solving strategies, and roles of people.

WHILE THE OPPORTUNITIES AND BENEFITS THAT AGILE METHODOLOGIES AFFORD MAKE THEM ATTRACTIVE, ORGANIZATIONS SHOULD BE CIRCUMSPECT IN EMBRACING THEM OR IN INTEGRATING THEM WITH EXISTING PRACTICES.

Agile methodologies place a premium on testing, urging developers to develop the test code upfront [8]. The notion of test-driven development (TDD) is becoming increasingly popular within the agile community. TDD is motivated by the fact that thinking about and writing tests prior to coding will make the code more understandable and maintainable. TDD also facilitates continuous integration of new code and/or changes without adversely affecting the existing code base. However, the firmly entrenched tradition of writing code prior to testing must be overcome to institutionalize the practice of early and frequent testing. TDD also redefines the role of the quality assurance function in systems development.

Questions have also arisen about the efficacy of agile approaches with regard to large projects where scalability is paramount [2, 10]. Due to the novelty of agile methodologies, very little empirical data is available regarding each method. The biggest challenge facing the project manager, therefore, is the selection of an appropriate method from the host of agile methods currently available. This is very much like the predicament that early adopters of object technology faced. While all agile methods, to some extent, conform to the tenets outlined in the agile manifesto, they are not all alike in every respect. They differ in terms of team size, code ownership, duration of each iterative cycle, emphasis on upstream and downstream activities, and the mechanisms for rapid feedback and change [2, 7]. In the absence of a unified agile approach, organizations must decide which is most compatible with their existing practices.

Technological issues. An organization's existing technology can impact the efforts to migrate to agile methodologies. Companies that rely solely on mainframe technologies may find it difficult to assimilate agile methods compared to those that use OO development techniques. Tools play a critical role in successful implementation of a software development methodology. Organizations planning to adopt agile methodologies must invest in tools that support and facilitate rapid iterative development, versioning/configuration management, JUnits, refactoring, and other agile techniques. Of course, tools alone cannot make software development successful. People must be trained to use them correctly.

Conclusion

The principles of agile methodologies parallel the ideas delineated in Checkland's Soft Systems Methodology and Ackoff's Interactive Planning [3]. These reflect the essential characteristics of complex adaptive systems [4, 7, 8], and have the potential to endow organizations and systems with emergent properties. While the opportunities and benefits that agile methodologies afford make them attractive, organizations should be circumspect in embracing them or in integrating them with existing practices [1].

Agile methodologies are ideal for projects that exhibit high variability in tasks (because of changing requirements), in the capabilities of people, and in the technology being used [8]. As Highsmith notes, they are also appropriate for projects where the value of the product to be delivered is very important to customers. Organizational forms and cultures conducive to innovation may embrace agile methods more easily than those built around bureaucracy and formalization. Organizations must carefully assess their readiness before treading the path of agility. The issues raised in this article are invaluable in making this judgment. C

References

1. Boehm, B. Get ready for agile methods, with care. *Computer* (Jan. 2002), 64–69.
2. Boehm, B. and Turner, R. *Balancing Agility and Discipline: A Guide for the Perplexed*. Addison-Wesley, Boston, MA, 2004.
3. Cavaleri, S. and Obloj, K. *Management Systems: A Global Perspective*. Wadsworth Publishing Company, CA, 1993.
4. Cockburn, A. and Highsmith, J. Agile software development: The business of innovation. *IEEE Computer* (Sept. 2001), 120–122.
5. Cockburn, A. and Highsmith, J. Agile software development 2: The people factor. *IEEE Computer* (Nov. 2001).
6. Cockburn, A. *Agile Software Development*. Addison-Wesley, Boston, MA, 2002.
7. Highsmith, J. *Agile Software Development Ecosystems*. Addison-Wesley, Boston, MA, 2002.
8. Highsmith, J. *Cutter Consortium Reports: Agile Project Management: Principles and Tools 4*, 2 (Feb. 2003), Cutter Consortium, Arlington, MA.
9. MacCormack, A. Product-development practices that work: How Internet companies build software. *MIT Sloan Management Review* (Winter 2001), 75–84
10. Orr, K. CMM versus Agile Development: Religious Wars and Software Development. *Agile Project Management Executive Report 3*, 7 (2002), Cutter Consortium, Arlington, MA.
11. Sircar, S., Nerur, S.P., and Mahapatra, R. Revolution or Evolution? A Comparison of Object-Oriented and Structured Systems Development Methods. *MIS Quarterly 25*, 4 (Dec. 2001), 457–471.
12. Truex, D.P., Baskerville, R. and Klein, H. Growing systems in emergent organizations. *Commun. ACM 42*, 8 (Aug. 1999), 117–123.

Sridhar Nerur (snerur@uta.edu) is an assistant professor of information systems at the University of Texas at Arlington. **RadhaKanta Mahapatra** (mahapatra@uta.edu) is an associate professor of information systems at the University of Texas at Arlington. **George Mangalaraj** (mangalaraj@uta.edu) is a doctoral student at the University of Texas at Arlington.