J. INDIVIDUAL EMPLOYMENT RIGHTS, Vol. 11(4) 333-347, 2004-2005

AGILE METHODOLOGIES AND THE LONE SYSTEMS ANALYST: WHEN INDIVIDUAL CREATIVITY AND ORGANIZATIONAL GOALS COLLIDE IN THE GLOBAL IT ENVIRONMENT

JULIE E. KENDALL

KENNETH E. KENDALL

Rutgers University

ABSTRACT

The global IT environment has contributed to pressures on IT professionals to adopt methods that deliver innovative information systems rapidly. This pressure challenges traditional individual work styles and preferences often found in creative environments. In this article we use critical/historical analysis to address a current dilemma faced by systems analysts and programmers who are adopting new methods of information systems development called "agile methods" that require working closely with users and other analysts on IT projects. Our contribution is to raise awareness of the tension between deep-rooted systems development practices of individuals versus organizational demands to adopt agile methods. In addition, we cover the difficulty faced by organizations in trying to respond competitively to new market demands while respecting the creativity and skills of the individuals upon whom they are relying. We provide recommendations to organizations considering adopting agile methodologies for systems development as well as recommendations for analysts and programmers who are in the midst of changing methodologies.

Modern for-profit and nonprofit organizations alike house dozens of information systems (IS) and a plethora of information technology (IT) (consisting of hardware, software, people, databases, and manual procedures) that help them achieve

© 2006, Baywood Publishing Co., Inc.

their business goals and manage their internal operations and communications. Information systems development projects are a common occurrence in organizational life, and there are many reasons that information systems are updated or replaced. Chief among these reasons for systems change are the desire to be more competitive in one's industry nationally and internationally; the need to update systems to communicate more effectively with suppliers, vendors, customers, and strategic partners; meeting the requirement to comply with government-mandated regulations; and the necessity of replacing hardware and software that are no longer adequate to serve the organization's needs. Systems projects usually carry specifications on project length (in months or years, with 18 months being typical for a medium-sized project) and a dedicated budget (ranging from a small project budgeted at $10,000 to multimillion dollars for large-scale projects such as creating sophisticated e-commerce sites for retailers; or for specialized projects such as satellites developed by NASA for developing countries, or those that other governmental agencies might fund).

Examples of these types of systems development projects would be creating an innovative information system that sent special discounted prices to customers who had visited a corporate Web site; moving benefit plans for employees to a secure site on the Web or making records of current inventory available on the Web to prospective customers; posting privacy notices on all Web pages that collect and store information from the Web site user; and upgrading organizational computers from wired network configurations to implementing hardware and software for secure wireless networks.

Information systems are also classified by the functions they serve and the levels of the organization they serve. Wireless systems, ERP (enterprise resource planning) systems, e-commerce and Web systems cut across all functions, levels of organizations, and types of users. Specific types of information systems completed during information systems development projects include executive support systems (ESS) for support of strategic decisions; group decision support systems (GDSS) for supporting unstructured or semistructured decisions for groups; decision support systems (DSS) often used for modeling business intelligence; expert systems of knowledge-based systems (ES) and artificial intelligence (AI), which capture expertise of experts to solve a specific problem within a defined domain (e.g., how to negotiate with a hostage taker in a hostage situation); transaction processing systems (TPS), which process large amounts of data for routine business transactions such as payroll and accounting; office automation systems (OAS) that support office workers and knowledge work systems (KWS) that support knowledge creators such as scientists and engineers; and management information systems (MIS) that support managers through the use of databases and models to help interpret data for decision making [1].

The employees who develop these systems are either employees of systems consulting firms and/or paid IT employees of the organization who hold positions in the company that use these information systems applications. What are their

individual rights in terms of their working relationships in the face of increasing organizational pressures to work in pairs and teams as part of the adoption of new methodologies? This is the focus of our article.

In the upcoming sections we discuss what comprises typical information systems projects; the educational and personality attributes of systems analysts and programmers; the current global economic climate for IT; traditional versus alternative approaches to systems development; behaviors required of analysts working on agile projects; implications of organizational adoption of agile methods; refusal to participate on a team or as a part of a pair of programmers; the dilemma posed by adoption of new systems methods for analysts and organizations; potential risk and potential advantages for adopting agile methodologies. Upon analysis of these factors we then provide recommendations for analysts and organizations adopting agile methodologies, outline our contributions, and suggest future research to further clarify analysts' appropriate participation in new systems development methodologies.

THE SYSTEMS DEVELOPMENT PROJECT

Information systems projects can arise from any source, internal or external to the company. Often they come in the form of directives from upper management. Additionally, upper levels of IT professionals inside the organization are often charged with an educational function that demands that they educate others in the organization about information systems innovations, and they also champion projects that feature the latest IT developments. Often projects are suggested by those in charge of functional units who are in an excellent position to witness problems that recur repeatedly and who may appeal for an IT solution as a way to address them.

Projects are also undertaken to satisfy external demands such as new government reporting, disclosure, privacy, or storage guidelines that affect how organizational information must be handled. Other external demands include changes in the hardware or software of one's suppliers or vendors that often translate into obligatory changes. An extreme example is the mandate that Wal-Mart originally issued its suppliers to include RFID (radio frequency identification) tags on every pallet coming into Wal-Mart's warehouses. The RFID tags are extremely useful for inventorying purposes and when deployed worldwide could enable Wal-Mart to retain its edge among retailers by using inventorying data strategically.

However, Wal-Mart officers quickly learned that several small suppliers, if left to their own devices, would be unable to comply with the requirement and thus would be arbitrarily cut from the supply chain. Part of the reason was excessive cost of this innovation (most suppliers were not using RFID technology); part of it was the strict timeline that was imposed. Wal-Mart relented, set up a strategy and budget for helping suppliers implement the new technology, and also changed the

deadline and scope of the project to implement it more slowly over a longer period of time.

Other motives for commissioning new information systems include internal organizational politics, where new IT is used as a status symbol and accumulated as a symbol of political clout for those able to secure the project for their unit. In those cases the organizational benefits are usually minimal, and the projects are less likely to be successful. Although this is certainly an ethically "wrong" reason for information systems development, it does happen. Suffice it to say that there are internal and external reasons for an organization to alter, update, or replace its existing information technology.

Attributes of Systems Analysts

We focus here on the systems analyst or systems analyst/programmer in the process of completing an information systems project. Analysts typically are graduates of IS programs in business schools or of software development programs in computer science or engineering programs. Systems analysts and programmers who have earned business degrees are acknowledged to be better versed in business functions; those who did not often are more attentive to technical programming matters. Programmers may also be drawn from the ranks of students who attended technical schools or universities.

As a group of employees classified in a particular profession, certain attributes seem to describe the qualities of most successful systems analysts. These include, first and foremost, viewing systems problems as a challenge and receiving enjoyment from developing workable solutions. Analysts must also possess communication skills sufficient to enable them to communicate with organizational users and teammates. In addition, they must be technically proficient enough to communicate with machines via programming tools and techniques. Systems analysts often are described as self-disciplined, self-motivated employees capable of juggling many tasks of project management and managing numerous project resources including people, budgets, and schedules. One of the attributes analysts possess that springs from their need to multitask is their creativity in problem solving [1].

Most analysts acknowledge that they need infinite patience in staying with a systems problem until it is resolved. So systems analysts, programmers, and programmer/analysts in particular need to be self-motivated and capable of working alone. This is how most analysts have worked over the years, and most of their experience lies in working for untold hours in a very dedicated manner to untangle a problem and create a solution. Paradoxically, analysts and programmers also need team skills as they bring together the pieces of the IT solution they have worked on so that their work will interface with the work of teammates.

Current Global IT Economic Climate

The global IT environment, which seeks to support businesses in expanding their markets worldwide, expand their labor force to a global one, and expand operating hours to around-the-clock accessibility and service, has contributed to pressures on IT professionals to adopt methods that deliver innovative information systems rapidly. This pressure challenges traditional individual work styles and preferences often found in creative environments. Organizations face difficulty in trying to respond competitively to new market demands while respecting the creativity and skills of the individuals upon whom they are relying.

Systems Development Life Cycle

Most information systems projects are accomplished with a methodology called the Systems Development Life Cycle or SDLC for short. Through use of a series of seven phases (some of which can be done simultaneously), the analyst uses a systematic approach to identify problems, opportunities, and objectives that the organization is experiencing. As the project unfolds, analysts ascertain the information requirements of the organizational members and systems; design the agreed-upon information system; develop and/or document application and system software (which is often programmed by in-house programmers); and see the project through to implementation and finally evaluation, at which point analysts use predetermined benchmarks or metrics to assess the success and quality of the new system [1]. Technical competence coupled with superb verbal skills are critical for the analyst who must interact with upper and middle managers, system users, and other analysts and programmers assigned to the project.

Although the SDLC and the attendant methods are often criticized for their long development cycles (which can run anywhere from half a year to three years) and their fascination with graphically documenting every aspect of the old and new system before it is implemented, the systems development life cycle is still the most widely taught and practiced approach worldwide in systems development.

The Systems Development Life Cycle defines times during the development process when analysts work as a team and when programmers work alone, separate from users and even from each other. Teams of analysts and programmers, rather than pairs, are typical but not required, and certainly with small IS projects ($10,000-$75,000 USD budget approximately) one or two analysts will be completing all of the tasks for the entire project. Next, we turn to the contrasting approaches to developing information systems, which as a group are called “agile methods,” “agile methodologies,” or “agile approaches.” In the following section we define agile methodologies to examine how they can affect organizations and individual IT professionals who may be required to dramatically alter their culture of development and their individual ways of working as new methods are ushered in.

AGILE METHODOLOGIES

In response to some of the criticisms of SDLC and also in response to global demand for innovative information systems delivered in a timely way, agile methods were developed in the early 1990s. Many versions of agile development methods were introduced at that time, and there was an exciting cluster of approaches bursting on the development scene that was becoming available for analysts and programmers, including an agile methodology dubbed "XP" for "extreme programming [2], which has become a beacon in the agile community. Other agile approaches popularized at that time include feature-driven development [3], scrum [4], open source development [5], crystal family [6], as well as many others detailed in [7] and [8]. Approaches shared many features, and eventually an "agile alliance" [9] was formed to provide a united source for information about these kinds of development methods, as well as to provide a clearinghouse and naming conventions for them.

To that end, the Agile Alliance has spelled out the principles and values of agile methods and asserts that the success of IT projects with the use of agile methods can be realized when 1) real-life communication takes precedence over following rules or procedures; 2) the final product is more important than documenting the process; 3) systems development is customer-centered rather than manufacturing-centered; and 4) adapting is the correct state to be in, not planning [9].

The systems development life cycle emphasizes the understanding, diagraming, and design of organizational and system processes. On the other hand, agile approaches are centered on people, firm in the assertion that human error is at the bottom of most systems projects that fail. Further, researchers believe that human creativity should take over in instances where it is impossible to develop formal, specified solutions for every system problem that is encountered [10]. Others noted that agile methodologies only succeed in delivering on-time, on-budget, quality information systems when all of the stakeholders collaborate [11]. One of the other important aspects of agile methodologies is the importance of understanding the organizational culture that one is working in [12].

Agile methods, similar to rapid application development (RAD) and other iterative development methods, emphasize the use of short releases of a working version of a system or feature. There is an expectation of quality improvement each time an iteration is completed. Additionally, agile methods are less formal and less formally documented than systems created with structured methods. Agile methods underscore the importance of people in improving the quality of information systems based on increased communication, a belief in the value of flexibility, and the prompt and ongoing attention to systems.

Philosophy of Agile Approaches

The mainstay of the agile philosophy is that all of the humans involved in the process (users, analysts, programmers, management) deserve to be well-treated.

They are the center of the agile universe. Working in collaborative teams is very important for analysts and programmers using agile methods. This stems in part from believing that humans are inherently more complicated systems than machines, and as such they experience social and spiritual needs, need to express their creativity, are capable of responding to changes in their environment, and are quite adaptable in unpredictable ways. This translates into the agile philosophy of treating humans as humans, rather than replaceable parts in a grand machine.

Core Practices of Developers Using Agile Methods

Five core practices serve as the platform for supporting the manner in which programmers and analysts behave toward each other and others involved in a systems development project. They are quite distinct from those put forward in the systems development life cycle.

The five core practices are composed of short releases of developed software; the 40-hour workweek; the on-site customer; and pair programming. We briefly define each one and then focus on the core practice of pair programming as it relates to individual employment rights.

- **Short releases.** The programmer/analyst team will shorten the time between releases of newly developed pieces of software. Short releases refers to software that does not have all of the features that it will ultimately possess. (For example, a menu screen for an airline may permit the user to search on a flight number, but not on a city name originally. That feature will be added later.) Short releases are intended to come out rapidly, in quick succession as the programmers finish them. Essential features will be developed first, and when actually released, the product will contain critical features at first. The team agrees to work on improving it later in the process.
- **Forty-hour workweek.** Since the agile methods culture holds dear the importance of the long-term health of its developers, this core practice states that members ought to commit to intense work on a project, complemented by time off to prevent a common problem on projects, which is burnout and stress that diminish the effective accomplishment of intellectual tasks such as systems development.
- **On-site customer.** The human dimension is again noticeable here, since communication with the on-site customer is paramount. Business workers are required to stay on site during the entire development process. It is considered to be a way to create a deep, ongoing understanding of what the customer considers a high priority, and a way to develop a lasting client-customer relationship.
- **Pair programming.** This core practice requires that a duo of programmers who want to program together can choose to program together, run tests on

code together, communicate about how well the program is running, and so on. Interacting with another programmer in this way is intended to push the dyad to clarify their logic and promote rational thinking about the problem at hand. Some other benefits of pair programming are that very few errors remain in the program for long, since someone is there to catch them early on. Also, some supporters of pair programming believe that creativity is enhanced by working with another person, and that, although on its face it seems as if it would take more time, it can actually save time in development.

- **Timeboxing.** Timeboxes refer to the way developers construct their reality of time, by cutting the projects into several shorter time periods [13]. The time limit to finish each iteration (usually one or two weeks) is represented by what is referred to as a timebox. The timeboxes in turn depend on the specific project in an organization, including the scope of the project, its complexity, the size of the development team and their experience, project resources available, and other situation-specific factors.

In the upcoming section we examine in more detail the core practice of pair programming to better understand where the tension resides between organizational goals of adopting new methods and the individual interests of the programmer or analyst faced with changing his/her traditional methods of work to accommodate the employer.

PAIR PROGRAMMING AND EMPLOYEE RIGHTS

As a core practice of agile methods, pair programming has received quite a bit of interest from developers and organizations that are considering adopting new development methods, since its use has been championed as a path to guarantee a better quality of information systems [14]. Authors have noted that programmers are quite emotional when the topic of pair programming is broached [15-17]. The originator of the term XP claimed that the phrase "extreme programming" was used intentionally to evoke an emotional response about a radically different approach to development [17].

Using a Team for Programming

What happens when a programming team is constituted for a development project? Typically, one programmer, often a senior one, will choose a programming partner [18]. The protocol is that the invited programmer accepts the invitation and they work for a while together. If one pictures the physical setup (two programmers, one typing on their shared computer, the other thinking), one could recognize that in this setting errors are caught earlier and that each programmer serves as a double check on the work of the other in pair programming.

Each of the programmers codes, as well as tests, their programming output. Although coding in pairs is often initiated by the senior person, it is meant to be a give-and-take situation between peers. Programmers would alternate between tasks, depending on who is able to achieve the best "gestalt" or overall picture of what will be the pair's ultimate goal. Additionally, pairs can change often at the beginning of a project. The thinking is that this helps them gain new perspectives on their own coding capabilities and practices.

Working with another programmer helps each person clarify his/her thinking. Pairs may change frequently, especially during the exploration stage of the development process. This, too, can help programmers gain a fresh view of their coding habits. Programming chores can be split among the duo so that while one programmer is checking the accuracy of the code, another will pursue a goal of simplicity so that the code is clear and useful.

Pair programming can also be accomplished by having one programmer create an initial version, with the second programmer examining it principally for the quality of the code that has been written. So the second programmer verifies, tests, and improves the code without trying to add features to what has been written.

Benefits of Pair Programming

Of those who have used pair programming (including the authors), most say that it can save time, reduce the chance that sloppy thinking has crept in, serve as a springboard for creativity as each programmer builds on another's ideas, and is a fun way to program.

Pair programming is intertwined with the overall quality of the system being developed, while at the same time saving many days and weeks of development effort. Another advantage becomes obvious when one realizes at what point in the programming errors are detected. Using pair programming practices, errors are caught immediately [11]. In contrast, when using a traditional approach to code verification, errors are discovered after the programming is almost complete. Sometimes errors are found only after it is too late to correct them and deliver the project on time. As one can see, pair programming helps in delivering quality systems within the time frame agreed upon by programmers and managers.

There are many other advantages as well. One is the care with which one programs initially, when his/her code is subject to immediate inspection. Another benefit is that those who are less experienced can be paired with a more experienced programmer and quickly learn what to do. Williams argued that codes developed by pair programming teams often turn out to be better designed, freer of errors, easier to follow, and simpler rather than more complex, compared to systems developed with traditional methods [11].

Drawbacks of Pair Programming

Some of the drawbacks to pair programming include the inability to identify problems because of a lack of overall knowledge of a system [19]. Without traditional ways of verifying code that supply a new perspective, programmers may not be able to gain a distanced, critical perspective of what has been developed. Finally, it is also obvious that quality would suffer if programmers were not happy in their careers or particular jobs.

Turnover as a Traditional Problem of the IT Workforce

Over the years, IT professionals have been categorized as a restless group. Turnover in IT jobs occurs frequently (sometimes as high as 29 percent annually for IT people [20]), and it is difficult to retain analysts and programmers in an organization for an extended period of time. Their loyalty appears to be to the IT profession, rather than to any particular company. Also, many analysts and programmers find their initial assignments boring after the innovative projects they developed at the university, for new hires are routinely asked to maintain old systems or create documentation for those same legacy systems.

Organizations are continually searching for ways to attract and retain experienced and talented systems analysts and other IT people via bonuses; maintaining a "business casual" dress code; and offering additional social and health-related perks, such as organizational sports teams and 24-hour access to gym facilities which would appeal to younger analysts.

One of the reasons for high turnover must certainly be that analysts and programmers have a strong independent streak and see themselves as free to go where they can, often for a higher salary and better organizational perks. The other reason has to do with creativity in solving large and small problems. New systems problems (available in new workplaces) can provide a challenge unavailable in a more familiar or stable setting. Analysts and programmers often are creative individuals who do not necessarily march to the beat of the same drummer heard by others in the organization.

Refusing to Serve on a Team

Can systems analysts or programmers refuse to work on a systems development team or in a programming pair? Other professionals such as engineers have claimed the right to refuse to work on projects where they would not be free from legal liability if something goes wrong. And in the IT field, Web site developers and Web hosting organizations have reserved the right to refuse to post material to a Web site if the content is deemed inappropriate or if the authority to post does not seem to be appropriately gained.

The literature discusses the possibility of programmers who are unwilling to work in this agile method. "It's important to acknowledge that there are programmers (and analysts) who don't want to work side-by-side with someone else, and whose job satisfaction will certainly be affected by something like pair programming" [16]. Anecdotes are passed among colleagues to that effect. However, no formal studies of what happens when an analyst or programmer refuses to engage in one of the core practices of a new methodology are available. Their basis for refusal could be an individual work style as mentioned earlier or it could be a strong belief in a creative process that is a solitary, intellectual process, where arbitrary pairings are a hindrance rather than a help.

Dilemmas for Analysts and Programmers

The problem posed for analysts and programmers who prefer not to program in pairs or use other agile methodologies is multifaceted. Most have been inculcated with the idea that programming is a "solitary activity [21].

Additionally, although programmers and analysts may work in teams, they may not be prepared for some of the more extreme parts of pair programming, such as having one person type and the other work on the intellectual and creative aspects of the program. Others claim their code is "personal" [21].

Organizations hire analysts to perform certain systems development tasks to create and maintain various information systems. Much of what analysts traditionally accomplish on the job is taken up with maintaining legacy (older, existing) systems. The competitive environment is changing this. New hardware is available annually, and new software releases are available approximately every six months. In an additional push to be competitive, organizations are increasingly seeking development of innovative information systems that solve multiple information systems problems and address expanding businesses; doing business in a new way; addition of strategic partners; or moving to a new platform using a new language (i.e., instituting the use of open source software or migrating to the Web).

Dangers and Rewards in Adopting Agile Methods

The dangers inherent in adopting new agile methods are many. First and foremost is that they do not possess a long, proven track record of delivering quality information systems. Agile methodologies, as we have seen, have been around about 15 years. Firms that are interested in adopting them are just now testing the waters as agile methods slowly enter into the formal curriculum in business information systems programs and computer science programs. Rather, there is a great deal of anecdotal evidence that systems developed with agile methods are of better quality than those developed with traditional approaches.

A second danger in adopting agile methods is that experienced and mature programmers and analysts will not necessarily be familiar with them, so the organization may have to hire that expertise into the IT team. It is often difficult to add IT expertise at the time one notices the deficiency (i.e., when a project is ready to start).

Pair programming, one of the agile practices, aims to have one experienced programmer paired with one newcomer in all pairings [16]. However, it is ironic that "experienced" programmers may not be the long-tenured employees in this instance.

Agile methods are not without their rewards, and over the years, more developers and organizations are being drawn to them. One of the potential benefits for organizations in adopting agile methods is the hope of better designed systems. Another reason to adopt agile methods is to improve the rapidity of systems delivery. A third potential benefit is an "investment in quality assurance (before things are built), rather than quality control (after things are built)" [18].

Recommendations for Analysts

Analysts and programmers frequently learn new languages and new software tools that assist them in doing their jobs better. Most are eager to do so and recognize that the best way to maintain interest in one's work and to remain marketable in the workplace is to keep current. However, analysts need to evaluate their own working styles in relation to the use of agile methodologies. To better understand their attitude toward working in teams and pairs, we recommend analysts and programmers answer a brief series of questions.

1. What value do you place on working alone on a coding or systems project?
2. How important is the opportunity to work alone versus being part of a pair or a team?
3. What benefits can you identify from previously working in a team or a pair?
4. What are your experiences working with others on development projects?
5. What experiences working in teams or pairs would you avoid?
6. Which experiences in teams or pairs would you seek out?

All of these are questions targeted at analysts who are typically part of cultures where individual work is the hallmark of "doing your job."

As IT people face potential changes in their culture and way of working, they might also contemplate whether it is possible to suggest some rights that surround their work habits that can all be agreed to and subscribed to by participants in the IT process [15]. While Hayes outlined rights for customers and managers as well as programmers, we are primarily interested here in his suggestions for programmers (which we believe can be read as analysts/programmers), which include these programmer rights:

- The programmer has the right to estimate work and have those estimates respected by the rest of the team.
- The programmer has the right to honestly report progress.
- The programmer has the right to produce high-quality work at all times.
- The programmer has the right to know what is most important to work on next.
- The programmer has the right to ask business-oriented questions whenever they arise [15].

Also of interest is what is *absent* from the Extreme Programmers Bill of Rights, namely the idea that programmers have the right to refuse to work with a development team or in a pair programming situation. A suggested wording is, "The programmer has the right to refuse to work in a pair or team for creative or work-style reasons without fear of reprisal." We think this is a useful addition, and one that has important ramifications.

Recommendations for Organizations

We believe organizations should look carefully into the advantages and disadvantages of adopting agile methods as they relate to their IT workforce and their particular culture. They may provide the edge an organization needs to push past their competition. Organizations may have to "evolve" toward a balanced approach to choices of methods that suit the development project [22]. Other researchers have noted that organizations must "carefully assess their readiness" before adopting agile methods [10].

Finally, organizations must be aware of the disadvantages inherent in adopting agile methods, particularly in situations where valued employees, who may be unable to change their work habits away from independence and creativity to work in pairs, are part of the systems development team.

FUTURE RESEARCH

There are many fruitful aspects to researching questions of whether IT professionals have a right to refuse to work in a team or a pair due to past work habits, a desire to remain independent, or because of creative differences in approach or perspective. At what point should an organization impose team or pair membership? What is the effect on creative participation and how does it affect those senior members who are accustomed to working alone for long periods? At which point must the organization make a tradeoff to say that the importance of analysts' and programmers' participation in this practice is a key to the overall success of the systems development project? Currently, most programming teams are voluntary, and programmers choose with whom they would like to work. Research into where the boundary is between forward-looking adoption of methods and the pervasive current culture of creativity and independence (which supports opting out of team or pair participation) can assist

organizations and IT professionals alike to move toward an understanding of how changing methods ultimately will affect the information systems that are designed.

CONTRIBUTION

In this article we have raised awareness of the dilemma faced by individuals involved in IT systems development projects who may, for reasons of independence, creativity, or past work habits, refuse to work in teams or in pair programming, which is a core practice of agile methodologies. After considering the background and multiple facets of this issue, we recommended courses of action to both organizations and IT professionals as they work together to create better information systems. Awareness of the potential impact of changing systems development methods and the tensions produced by instituting a new group of practices (key among them working in pairs or in teams when the prior culture venerated the loner) are worthwhile for organizations to reflect and act upon.

CONCLUSION

Organizations and individuals in IT must work together to come to a mutual agreement about how quickly new methods for systems development are implemented and must also decide how critical it is to adopt the totality of agile core practices, versus selecting from among them in order to use only "what works." Each organization is in a unique situation regarding the systems development culture they create that helps foster programmers' and analysts' expectations and capabilities. The question remains whether the roles of analysts and programmers will evolve along with the progression of methods and the evolution of innovative information systems design, as agile methodologies become more deeply rooted in the organizations' culture.

ENDNOTES

1. K. E. Kendall and J. E. Kendall, *Systems Analysis and Design*, 6th Ed., Prentice Hall, Upper Saddle River, N.J., 2005.
2. K. Beck and C. Andres, *Extreme Programming Explained: Embrace Change*, 2nd Ed., Addison-Wesley, Boston, Mass., 2004.
3. P. Coad, E. Lefebvre, and J. De Luca, *Java Modeling in Color with UML*, Upper Saddle River, N.J.: Prentice Hall, 1999.
4. K. Schwaber and M. Beedle, *Agile Software Development with Scrum*, Upper Saddle River, N.J.: Prentice-Hall, 2002.
5. S. Sharma, V. Sugumaran, and B. Rajagopalan, A Framework for Creating Hybrid-Open Source Software Communities, *Information Systems Journal*, *12*(1), pp. 7-25, 2002.
6. A. Cockburn, *Agile Software Development*, Addison-Wesley, Boston, Mass., 2002.
7. P. Abrahamsson, P. O. Salo, J. Ronkainen, and J. Warsta, *Agile Software Development Methods*, vtt publications 478, 2002, http://www.inf.vtt.fi/pdf/publications/2002/P478.pdf

8. J. E. Kendall, K. E. Kendall, and S. Kong, "Improving Quality through the Use of Agile Methods in Systems Development: People and Values in the Quest for Quality", in *Measuring Information Systems Delivery Quality*, Evan Duggan and Han Reichgelt (eds.), Hershey, Pa.: Idea Group Publishing, pp. 201-222, 2006.
9. Agile Alliance, *Manifesto for Agile Software Development*, 2001, http://www.agilealliance.org/home
10. S. Nerur, R. Mahapatra, and G. Mangalaraj, "Challenges of Migrating to Agile Methodologies", *Communications of the ACM*, *48*(5), pp. 73-78, 2005.
11. L. A. Williams, "The Collaborative Software Process," unpublished doctoral dissertation, University of Utah, 2000.
12. D. P. Truex, R. Baskerville, and H. Klein, "Growing Systems in Emergent Organizations", *Communications of the ACM*, *42*(8), pp. 117-123, August 1999.
13. Dynamic System Development Consortium, http://www.dsdm.org/version4/timebox_plan.asp, 25 February 2004.
14. L. Wyssocky, *Pair Programming: Code Verification*, 2004, http://www.qualityprogramming.org/Implementation/CodeVerification/CodeVerification.htm
15. S. Hayes, ZDNet Australia, "The Extreme Programming Bill of Rights", November 11, 2003, http://www.builderau.com.au/program/development/soa/The_Extreme_Programming_Bill_of_Rights/0,39024626,20280741,00.ht, last accessed July 25, 2005.
16. S. Hayes, "Pair Programming—It Takes Twice as Long," from *Builder AU: Manage: At Work*, http://www.builderau.com.au/manage/work/0,39024674,39173564,00.htm, last accessed July 25, 2005.
17. K. Beck, *Extreme Programming Explained: Embrace Change*, Reading, Mass., Addison Wesley, 2000.
18. D. Chaplin, "Pair Programming and Quad Programming," *Byte-Vision ITD*, www.byte-vision.com/PairAndQuadPrint.aspx, last accessed July 25, 2005.
19. M. Stephens and D. Rosenberg, "Will Pair Programming Really Improve Your Project?" in *Methods and Tools*, 2004, http://www.methodsandtools.com/PDF/dmt0403.pdf
20. J. M. Wolfe, "Personnel Turnover Rates," http://www.waldentesting.com/about/article6.htm, last accessed on July 25, 2005.
21. A. Cockburn and L. Williams, "The Costs and Benefits of Pair Programming," http://collaboration.csc.ncsu.edu/laurie/Papers/XPSardinia.PDF, last accessed July 25, 2005.
22. B. Boehm, "Get Ready for Agile Methods, With Care", *Computer*, pp. 64-69, January 2002.

Direct reprint requests to:
Julie Kendall
Rutgers Univeristy
School of Business–Camden
Camden, NJ 08102
e-mail: julie@theKendalls.org