2007

Best Practices for Implementing Agile Methods: A Guide for Department of Defense Software Developers

Ann L. Fruhling
University of Nebraska at Omaha, afruhling@unomaha.edu

Alvin E. Tarrell
University of Nebraska at Omaha

Best Practices for Implementing Agile Methods: A Guide for Department of Defense Software Developers

Ann L. Fruhling
Associate Professor
Information Systems and Quantitative Analysis
Peter Kiewit Institute
College of Information Science and Technology
University of Nebraska at Omaha

Alvin E. Tarrell
Doctoral Student in Information Technology
Peter Kiewit Institute
University of Nebraska at Omaha

IBM Center for
The Business of Government

Best Practices for Implementing Agile Methods: A Guide for Department of Defense Software Developers

Ann L. Fruhling
Associate Professor
Information Systems and Quantitative Analysis
Peter Kiewit Institute
College of Information Science and Technology
University of Nebraska at Omaha

Alvin E. Tarrell
Doctoral Student in Information Technology
Peter Kiewit Institute
University of Nebraska at Omaha

IBM Center for
The Business of Government

T A B L E   O F   C O N T E N T S

F O R E W O R D

On behalf of the IBM Center for The Business of Government, we are pleased to present this report, "Best Practices for Implementing Agile Methods: A Guide for Department of Defense Software Developers," by Ann L. Fruhling and Alvin E. Tarrell.

The Department of Defense needs to respond quickly to changing threats and requirements. Increasingly, this rapid response capability requires an ability to field new software applications quickly as well. Traditional software development methods can take too long, cost too much, or lead to a solution to a requirement that is not in fact what the user really needed. Agile software methods offer many advantages, including speed. They also have the ability to evolve quickly to meet users' real, as opposed to apparent, needs. In addition, they are cheaper than more traditional methods. Though not a panacea, agile methods offer a solution to an important class of problems faced by organizations today.

This report offers a guide to how Department of Defense (DoD) organizations can use agile methods to meet DoD's mission more quickly and effectively at a lower cost. The best practices outlined in this report come from interviews with 11 project teams that have used agile methods to meet mission requirements. The techniques described have applicability to any organization facing fast-changing problems, the need to act and adjust quickly, and limited budgets.

Agile software methods, like any software methodology, require technical sophistication, but it would be a mistake to focus only on the technical aspects. Because agile software methods represent a change from traditional approaches, organizational factors are very important. The culture of the organization needs to be open to change, not entrenched in traditional approaches. Communication must be open, and information should flow easily among participants. The information technology infrastructure must be robust. Using agile software methods will take sustained leadership from senior executives to be successful. It is too large a change from traditional practice to simply be delegated and delivered.

Albert Morales

David Amoriell

The authors of this report found that a best practice is to form a small team, give team members good tools, start on small projects, and expand based on early successes. The team needs to have a "can do" attitude, be experienced problem solvers, and work well together in an atmosphere of mutual trust. These factors are as important as domain knowledge. As the organization builds experience with agile techniques and success builds credibility, their use can spread to other areas. The organization will then have the opportunity to make further improvements based on firsthand knowledge of what works.

The best practices for implementing agile software methods are the same as for rolling out any new business process. The difficult issues are not technical. Many organizations have effectively used agile programming techniques to enhance their mission. The techniques are known and can be replicated. The difficult part is applying the technique to the right problem, dealing with the change management issues that arise from a new way of working, and finding a path from small early successes to its use as a standard business process.

We hope this collection of best practices based on the experience of the Department of Defense will help other organizations become more agile.

Albert Morales
Managing Partner
IBM Center for The Businessof Government
albert.morales@us.ibm.com

David Amoriell
General Manager, Federal Sector
IBM Global Business Services
amoriell@us.ibm.com

EXECUTIVE SUMMARY

Government leaders and defense contractors are constantly seeking faster and better methods to get critical information, knowledge management applications, and decision support tools into the hands of decision makers and warfighters. These issues can be especially challenging for Department of Defense (DoD) entities due to rapidly changing operational requirements, preoccupation with other more pressing needs, ever-advancing technological capabilities, and a continued emphasis on a more streamlined U.S. military.

These factors are driving many defense-related organizations to examine their fundamental information technology (IT) system development processes in search of opportunities for improvement. Agile system development methodologies offer one solution. Widely used in the commercial sector, they were originally developed to address similar operational environments.

Agile methods purport to streamline the systems development process in general and to bring significant improvements such as more reliable delivery of required functionality within a shorter elapsed time. These benefits will only fully accrue if the underlying methodologies are properly implemented.

In this study, 11 IT project teams from multiple organizations that were experienced in agile development practices were interviewed and surveyed to identify their best practices. This involved seven DoD project teams located at the United States Strategic Command (USSTRATCOM), three industry project teams, and one university team.

The experience of the organizations using agile methods ranged from one to four years. Nine of the 11 teams were centralized. One team was distributed among USSTRATCOM and the prime contractor's headquarters; another team was distributed among several COCOMS (Combatant Commands). Team members from all levels (for example, developers, business owners, analysts, project leaders, government program managers, and government functional managers) were interviewed.

The report addresses the following:

- Key activities organizations implemented that positively impacted the introduction and management of the agile software development process

- Specific agile best practices and principles that were followed

- The ways in which several best practices were modified to fit the customer-specific development environment

We also review the key steps that agile teams take to ensure that a quality IT product is delivered to the customer for implementation. In short, we provide a synopsis of the currently accepted best practices for use of agile software development methodologies within a DoD-related environment, thereby assisting and supporting DoD organizations and contractors in their acceptance of and migration to agile methods.

Organizational Readiness

Several key factors must be considered before beginning the transition to the use of agile methods. First and foremost, there must be a pre-existing underlying organizational culture that is receptive to change. Second, there must be an underlying IT infrastructure that can support the demands of the

agile development effort and help ensure the viability of fluid product releases. Management commitment to this effort is also critically important, and can be demonstrated through the previous two factors as well as other means. Finally, the scope and criticality of the projects included in the initial transition to agile methodologies should also be carefully considered.

Organizational culture. Transitioning to agile methods will require changes in the thought processes at the operational levels of IT organizations. This will involve a significant cultural shift in many organizations. Management needs to assess questions such as: How does our organization as a whole react to change? How entrenched are the members of our organization in the current process? How good is communication within the organization? Does information flow freely up and down the hierarchy?

Nearly all of the companies we interviewed and surveyed for this research report highlighted organizational culture as the key determinant of successful movement to agile methods, and answers to these (and similar) questions will indicate the readiness of the organization to accept the transition.

IT infrastructure. The underlying IT infrastructure is always important, but that importance is magnified in the agile environment. Most of the organizations we contacted pointed to the availability and use of automated testing and tracking tools as one of the main contributors to their successful transition to and employment of agile methods, and the impact of those resources is lessened if they are not state-of-the-art.

Leadership commitment. The level of commitment and endorsement from the top leadership and management also determines success. There must be both horizontal and vertical dedication to the agile process across the organization. Several of the teams reported the importance of both preliminary and ongoing training and consultation services provided by "agile experts" as key to their successful transition and continued operation. These start-up and ongoing maintenance costs create some not insignificant overhead costs to transitioning to the use of agile methods, and those costs can only be justified with full management support and a sponsor who is committed to championing the agile movement.

Starting small. Certain projects are more compatible with agile methods than others. Projects that have emergent and rapidly changing requirements, highly experienced technical and knowledgeable IT staff available, and collaborative system owners are the best candidates for successful agile projects. The key is to start small and grow from there. Essentially all of the organizations we interviewed initially employed agile methods on a relatively small project, generally not a mission-critical or life-threatening one and not at an enterprise level.

Best Practices

We compiled a collection of best practices based on our surveys and interviews. We categorized these practices chronologically based on traditional project development phases: initial start-up, project implementation, and ongoing development.

Best Practices for Initial Start-Up
At the initiation of the transition to agile development methods, the organization's leadership team and managers must demonstrate their commitment to the process. This includes supporting and participating in agile team building workshops and sponsoring agile process training across the entire organization.

The agile team should consist of IT professionals who are both technically experienced and possess in-depth domain knowledge. Excellent communication and interpersonal skills are a must. Management should ensure that the agile development effort is the only assignment of the agile team members and that system owners are assigned to the agile development project full-time. The project team must also possess a high level of mutual trust, so team-building activities are important.

The IT infrastructure must include a variety of IT support tools, a trusted version control management system, easy and quick access to technical experts, and, whenever possible, access to collaboration engineering techniques and group support systems.

Best Practices for Project Implementation
The agile implementation should include holding initial and incremental planning meetings, introducing the agile process on a pilot project, and carefully matching the project tasks to each developer's talents.

It's important to scale the development process to match the project's size. Shared versus individual responsibility is heavily emphasized in agile method-ologies, so managers should focus on the progress of the task, not the performance of individuals.

The project manager should keep the best practices of the past while also considering an à la carte approach when implementing agile development practices. Also, agile practices may in themselves need to be tailored. For example, if pair program-ming is adopted, it may be that assigning a lead pair programmer is beneficial. Flexibility is the watchword in agile development.

Best Practices for Ongoing Development
As the agile development process proliferates throughout the organization, lessons learned can be shared by designating an "agile champion team." This team works together and across project teams to help share successful strategies and develop tech-niques to address roadblocks and barriers to progress.

Additional best practices include automating testing, employing a migration control expert, and schedul-ing open time to wrap up loose ends. The program manager and project director need to leverage mul-tiple forms of communication, find ways to provide access to the Internet to research solutions, and address classified environment challenges inherent in any DoD-related development environment.

Conclusion
The adoption of agile development methods by defense-related organizations is important in that their use will assist these groups as they help DoD transform itself into a more modern, more adapt-able, and more service-oriented entity. Agile methods can and should be at the core of that transformation, and improved understanding and appreciation of agile methods will facilitate and foster their increased use within DoD.

Introduction

Challenges of Transforming Department of Defense Information Systems

The term *agility* is a key attribute when describing measurement of a military unit's ability to meet varying sets of mission requirements. Agility in the context of the military environment is the ability to quickly adapt and adjust to changing conditions. According to Department of Defense Chief Information Officer John G. Grimes:

> … succeeding in the new strategic environment requires levels of responsiveness and agility never before demanded of our forces. The U.S. Defense Department must transform from its historical emphasis on ships, guns, tanks and planes to a focus on information, knowledge, and actionable intelligence" (Grimes, 2006).

Thus, the U.S. Department of Defense (DoD) is in the midst of an information technology (IT) infrastructure transformation and is moving toward a net-centric, service-oriented architecture (SOA) that promotes information sharing among all users. Instead of a strict hierarchy where decisions are pushed down, networked warfighters (decision makers) are cooperatively pursuing the strategic goals of the commander in a much more decentralized fashion. Systems that support the net-centric goal must be real-time and be able to assimilate a rapidly changing environment in order to provide the information necessary to support quick, accurate recommendations and the resulting decisions. In addition, these systems need to be adaptive and able to quickly adjust to new technological capabilities and the changing political landscape. For example, this transformed environment would provide individuals an architecture that links "get weather" with "get target list" and "get asset status" services together to allow construction of a mission plan without having to go to separate information systems. Hence, authorized users throughout the organization could employ various services in combination to achieve the desired end state (Grimes, 2006).

Movement toward this SOA represents a great leap forward for DoD. Today's DoD information architecture is stovepiped, and movement away from that construct is necessary to allow realization of the full value and benefit of the SOA. Thus, one of the main challenges facing DoD is creation of an enterprise infrastructure that enables people, processes, and technologies to work together to provide timely and trusted access to information through information sharing and collaboration. This shift to SOA will impact not only the organizational culture, but also the way information systems are developed and the speed at which they are available to DoD. While the technology change is significant, changing the thought processes—the cultural shift and the evolving methodologies in which these information systems are developed, maintained, and enhanced—may be even more challenging.

In response to these challenges, various organizations within DoD are working together to build new information systems that provide seamless information sharing. Examples of these new types of information systems and services include the following:

- Wiki capabilities are now available where the DoD community can come together and share information in real time.
- The Strategic Knowledge Integration Web, or SKIWeb, is an event tracking and blogging tool

developed at U.S. Strategic Command that is finding increased use throughout DoD to facilitate the sharing of information regarding military and world events among members of the command and intelligence communities.

- An Intellipedia-based information architecture supports distributed information sharing.

The migration toward information-based warfare, combined with a need for greater agility stemming from rapidly changing requirements, calls for changes in the way military defense information systems are developed. New approaches to implementing these systems in a timely manner are also required.

Due to the rapidly changing operational requirements, preoccupation with other more pressing needs, ever-advancing technological capabilities, and a continued emphasis on a more streamlined U.S. military, both government leaders and defense contractors are seeking faster and better ways of getting critical information and decision support tools into the hands of decision makers and warfighters. This has led to a strategy of decentralization of information, a key component of supporting the migration toward net-centric, service-oriented architectures for current DoD information systems (Net-Centric Checklist, 2004). While this type of infrastructure seems well-suited to the task of supporting emergent requirements, the linear nature and high-density documentation required by plan-driven software development methods traditionally employed by the current DoD IT development and support teams typically do not (Alleman et al., 2003). Attempts to address this drawback have come in the form of spiral development approaches (Boehm, 1988) that essentially break the traditional waterfall method (Royce, 1970) down into smaller iterations based on risk or priority of functionality (Potok, 1992).

Advantages of Agile Development Methods

More recently, agile software development methods—for example, Scrum (Schwaber and Beedle, 2001) and eXtreme Programming (Beck, 2000)—have aimed to further streamline the development process and bring significant improvements such as timely delivery of the required functionality of various SOA and command and control (C2) systems.

'Agile' Spells Success for Strategic Command's IT Integration Efforts

When the Global Operations Center at the U.S. Strategic Command (USSTRATCOM) needed a new capability that would integrate 21 web-enabled data sources across a collaborative network, the acquisition team went to one of their core contractors to determine its feasibility. The response: at least one full year at the earliest to develop and implement, with several spiral years if needed. This long turn-around time was not acceptable to the USSTRATCOM commander, a strong advocate for quick turn-around methodologies and innovative solutions.

He directed the acquisition team to consider other options, so they contacted another contractor who was noted for recent agile IT development success. This contractor carefully reviewed the request and reported that they could provide the requested capabilities within three months using agile development practices. Their plan was to deliver the new capabilities in iterative cycles and adjust effort and focus based on user feedback.

The second proposal was accepted, and the agile development team successfully integrated 19 of the 21 data sources within the promised 90 days. The project manager attributes the 75 percent reduction in development time from the original proposal—and significant savings—to the utilization of agile development practices. "We see great value in what 'agile development' offers to a program and we will continue to use it within our engineering practices," explained the project manager.

Agile system development has several anticipated benefits:

- Agile methods can improve the design process. Iterative releases being used by customers, even those having had little design input, can serve as ongoing usability tests.

- Agile development also allows automated usage tracking and testing tools to be brought into play sooner and in a more relevant context.

- Lower-risk release cycles can encourage design experiments and reduce, if not eliminate, the writing of lengthy specification documents.

- The fast release pace gives an ongoing sense of accomplishment.

- Since there are closer team interactions, shared goals, and less solitary time invested in elaborate design, individuals are less defensive and territorial about their designs (Armitage, 2004).

Agile system development styles also purport to embrace situations involving changing or unclear requirements, while also promoting user and developer interaction (Beck et al., 2001). Perhaps one of the most important reasons for the agile movement is the acknowledgment that large, long-term projects often change course during development. Agile methods recognize that reality and seek to produce finished, working, reliable code in an iterative, incremental fashion in response. These results can be more valuable to a customer than an unfinished, poorly written, and unreliable product derived from a set of poorly written or poorly understood requirements based on the initial design (Armitage, 2004). In other words, agile methods appreciate that providing partial solutions earlier can be more valuable than providing full solutions later, even though they may not be fully finished (Sutherland, 2005).

Furthermore, agile methods exist to mitigate product development risk. They are more empirical than other methods, essentially using trial and error to reduce the risk of building the wrong thing. Since users are often likely to alter their requirements once they see and test the system, successful projects involve customer feedback on a regular and frequent basis. Developers accept the expense of having more routine code rework and having to maintain all development code close to release quality to achieve the gains that come from the customer-centered approach, where the customer plays a central role during the design process. Essentially a series of very-high-fidelity design experiments, these projects achieve low-level certainty by accepting high-level uncertainty (Armitage, 2004).

Armitage (2004) suggests that large projects be subdivided into a series of mini-projects, each of which can be quickly started, finished, tested, and delivered to the customer. Each mini-project in turn would form the basis for the next iteration, allowing customers to provide ongoing feedback as the project marches toward its ultimate completion. This iterative approach should minimize risk, since the frequent iterations with included reliability testing

and feedback would allow the system to "grow" into existence while actually being used by the customer. This approach would provide a system that more closely matches user requirements than one whose requirements were "determined" at project outset.

While agile methods such as Scrum and eXtreme Programming (XP) are used in some government and military projects, they have yet to reach their full potential in those environments. Traditional plan-driven methods are the mainstay today, partly because they are considered by some to have less risk and because they support CMMI Level-5 certification (Armstrong, 2007).

DoD projects are required to be CMMI Level-3 compliant, essentially forcing use of the more mature development methodologies. However, recent research findings show that it is possible, with strategic tailoring, for Scrum projects to achieve CMMI Level-5 compliance, so that barrier to entrance may be falling for agile methods (Sutherland, 2008).

Plan-driven methods were widely used in part because they are more consistent with the attitudes and approaches traditionally applied in the hierarchical, highly structured military culture. Nonetheless, there is a noticeable movement toward agile approaches, and further development successes using these methods will only hasten that migration. Next, we further explain various reasons supporting the paradigm shift from plan-driven to agile methodologies.

Paradigm Shift from Plan-Driven to Agile

Information systems that are designed and developed efficiently, accurately, and reliably—and that meet the intended needs and expectations of the stakeholders—are important goals of all organizations today. This is especially important for command and control systems that support information warfare, network defense, missile defense, global strike and integration, global intelligence, surveillance and reconnaissance, as well as space and combating weapons of mass destruction missions.

No one universal methodology for system development will work for all projects (Iivari, 2001) and in all environments. The traditional plan-driven system

Understanding the Military/Government IT Development Environment

Systems development projects in the military/government environment have several unique attributes. DoD IT teams are assembled and managed by independent contractors, with the main contractor known as the prime contractor and supporting contractors known as subcontractors. Software development and system support projects are competitively bid on by IT defense contractors, and the contract lengths vary from one to four or five years. This establishes an environment where the major defense contractors are both competing with each other to become the prime contractor on a project every few years, but are also cooperating with each other to offer subcontractor support as much as possible in the intervening years.

Contractors hire civilians to work on the IT development projects, and many of the positions require an active security clearance. These civilians may or may not be retired military employees, but oftentimes are because of the security clearance requirement. These contractors also may or may not have military experience directly related to the project being worked. The IT development team reports to a government program manager (PM) and works directly with one or more government functional managers (FMs).

The PM is responsible for the overall health and funding of the program, and generally focuses on concerns more than 12 months in the future. The PM oversees the project, ensuring that the project deliverables are completed on time and within budget, and meet the contractual requirements. Thus, the PM would be considered the system owner.

On the other hand, the FM is generally concerned with day-to-day operations of the system, and is also charged with determining needs to be addressed within the next 12 months. The FM office generally constitutes the domain expertise for the information system, and helps determine the requirements for the system and signs off on user acceptance testing of the system.

The PM and FM work together to determine what functionality will be in the system and prioritize the delivery of the functionality for the short and long term. Various weekly status meetings occur between the DoD contractors and the government program and functional managers. The PM and FM are frequently active duty military members, so their positions are often rotated as frequently as every 24 months.

development methodology requires extensive planning, codified processes, and rigorous reuse (Boehm, 2002). This methodology works best when developers can determine the requirements in advance, including prototyping, and when the requirements are relatively stable. The plan-driven model is often used in practice because of its straightforward and methodical, structured nature. However, in practice, the plan-driven model has a number of key shortcomings that have been widely reported, including the inability to effectively handle changing requirements and the tendency to be significantly over budget and behind schedule (see, for example, Biffl et al., 2005; Boehm, 2002; Watson et al., 1997). As new technologies, infrastructure, and expectations evolve at Internet speed, the plan-driven system development methodologies struggle to keep pace.

To address some of these shortcomings of plan-driven methodology, new system development models were proposed, including the spiral model (Boehm, 1988) and agile approaches such as Scrum, eXtreme Programming, and Crystal (Highsmith and Cockburn, 2001). Scrum is a team management process for agile development and can be used as a wrapper for existing methodologies. eXtreme Programming is an agile software development process consisting of 12 principles and techniques. Crystal is referred to as a family of software development methods that are tailored to different team sizes, and is a philosophy that embraces frequent delivery, reflective improvement, and osmotic communication (Cockburn, 2002). All Crystal approaches share common features and have three priorities: safety (in project outcome), efficiency, and habitability (developers can live with Crystal).

Meeting stakeholders' expectations accurately and in a timely manner during a DoD software development project is a complex task. Most of these projects involve multiple stakeholders from various DoD organizations with different, often competing, needs and goals. One way to address the competing needs

is to work closely with stakeholders (system owners, end users, and other customers) to ensure needs are determined and conflicts are resolved. Thus, moving toward a system development methodology where daily interaction and even co-location with the system owner/user occurs and one that continuously assesses implementing the highest priority and most useful capabilities for the stakeholders just makes sense in this development environment.

Such new approaches focus on fast deliverables, dynamic management of requirements, and rapid iteration and incrementation. Although these new approaches promise many benefits, the adoption of agile system development approaches has been sporadic within DoD, partly due to uncertainties regarding the best ways to implement them in hierarchical and traditional system development cultures that are slow and often resistant to change. Shifting to agile methods will require changes at the operational levels of IT organizations. IT managers need advice on when to use which methodology (Boehm, 2002; Glass, 2004) and how to operationalize a particular methodology for their particular development environment.

Research Method and Structure of the Report

The intent of this report is to provide information, insights, and practical, actionable advice to IT managers and analysts to help them prepare their organization to move to an agile system development environment. In addition, the report highlights the best practices of information system development teams that are successfully conducting agile development.

As part of our research for this study, 11 IT project teams from multiple organizations that were experienced in agile development practices were interviewed and surveyed to identify their best practices. This involved seven DoD project teams at USSTRATCOM, three industry project teams, and one university team. The experience of the organizations using agile methods ranged from one to four years. All of the teams except two were centralized. One team was distributed among USSTRATCOM and the prime contractor's headquarters; another team was distributed among several COCOMs, or Combatant Commands. Team

The U.S. Strategic Command

In 2002, the USSTRATCOM transitioned from a one mission-centric command focused on nuclear issues to one having eight missions: Information Operations; Cyberwarfare; Missile Defense; Global Strike and Integration; Global Intelligence, Surveillance and Reconnaissance; Space; Combating Weapons of Mass Destruction; and Nuclear Issues. This transition has understandably caused major adjustments in the USSTRATCOM organizational structure.

USSTRATCOM is a unified command composed of members from all four branches of the U.S. military (Army, Marines, Navy, and Air Force). These branches collaborate to provide specialties of all the services to optimize performance of the diverse missions of USSTRATCOM. Its role in the armed services is a "global integrator," with responsibilities including information operations, strategic warning and missile defense, and global command and control functions (USSTRATCOM, 2006). Many mission-critical information systems are maintained at USSTRATCOM.

USSTRATCOM is an organization that must react quickly and accurately to global events. To succeed at all these missions, which in some cases are quite diverse, the command needed to move to an even more adaptive, flexible, and collaborative environment that includes shifting to an agile information system development and support environment.

members at all levels (developers, system owners, analysts, project leaders, government program managers, and government functional managers) were interviewed.

Our research was conducted primarily at the U.S. Strategic Command, located at Offutt Air Force Base in Bellevue, Nebraska (see the sidebar for more on USSTRATCOM).

Defense contractors who can effectively and efficiently implement agile system development processes will have a competitive advantage when vying for DoD contracts. In support of this effort, the report provides details on how contractors might best implement agile development methods. We begin with a short discussion of the agile movement and more detailed background information on two popular agile methodologies: Scrum and XP. We then examine key factors organizations need to

examine to determine readiness for transition to agile processes. Then we present the best practices employed by system development teams that are currently practicing various techniques of agile software development. We examine the characteristics of the team and the attributes of the team members, and also describe how an organization can positively impact the management of the agile process. We also focus on the XP principles that are most often adopted, how Scrum and XP principles are adapted to the specific environment, and key steps IT managers are taking today to ensure a quality information system is delivered to the customer.

Agile Development Methodologies

The Agile Movement

In recent years, agile software development approaches have received a great deal of attention. They aim to make software development more flexible and focus on highly dynamic environments with quickly changing requirements (Cockburn, 2002). The word *agile* implies effectiveness and maneuverability; an agile process is light—which is a means of staying maneuverable—and sufficient, and implies being able to *stay in the game* throughout (Cockburn, 2002).

Agile development methods began to get public attention in the late 1990s. In 2001, 17 software professionals and consultants formed the Agile Alliance and produced *The Agile Manifesto* (Beck et al., 2001). The Agile Alliance movement was motivated by the observation that software teams in many corporations seemed entrapped in an ever-increasing amount of processes and documentation, with customers generally dissatisfied with the software the developers were creating. Continued dissatisfaction with the available development methods led to the introduction of various agile approaches, for example, eXtreme Programming (Beck, 1999; Beck, 2000), Scrum (Schwaber and Beedle, 2001), and Crystal (Cockburn, 2002). These approaches are based on the philosophy of the *The Agile Manifesto*. Agile development methods focus on four key ideas (Beck et al., 2001; Martin, 2003):

- **Individuals and interactions over processes and tools.** People are the most important success factor. Too much emphasis is often placed on coding knowledge and development tools. Instead, team members and their communication with each other should carry a much larger role.

- **Working software over comprehensive documentation.** Software documentation is important, yet information transfer is more effective through the code itself and through human interaction.

- **Customer collaboration over contract negotiation.** Successful software development requires frequent communication and collaboration between the user and the developer, rather than a traditional statement of work.

- **Responding to change over following a plan.** Long-term project plans are not adaptable; short-term plans provide more flexibility in responding to change. It is considered more effective to devise a detailed plan for a two-week period and a general plan for a three-month time period.

In addition, an agile approach encompasses a set of principles stressing valuation of early and continuous delivery of functionality, collaboration, and responsiveness to change over heavy documentation, negotiation, and plan-driven project management (Beck et al., 2001). On the one hand, some claim that this is in direct contrast to traditional plan-driven development models and is essentially "undisciplined hacking" (Paulk, 2001). Conversely, others argue that agile methods focus on short delivery cycles and continuous refinement (Beck, 2000; Beck et al., 2001) and therefore are just as rigorous as plan-driven approaches. *The Agile Manifesto* points out that formal planning and documentation are still essential, but should be pared down to the essentials in creating environments that are more responsive to change (Highsmith and Cockburn, 2001).

Differences Between Plan-Driven and Agile Approaches

The key differences between the plan-driven and agile approaches become apparent when comparing the attributes of seven so-called *home ground* parameters of a software project (see Table 1). The home ground parameters can help determine which development approach is best suited for the particular software project (Boehm, 2003). Table 1 summarizes each of the home ground attributes.

Developers

The critical attributes of the developers for both plan-driven and agile development teams are amicability, talent, skill, and communication (Cockburn and Highsmith, 2001). Developers following the plan-driven approach often rely on external sources of knowledge, have less system development experience, and may be decentralized. In contrast, the agile approach recommends that the developers have substantial previous experience in system development and that they be co-located with the customer as much as possible.

Customers

Customers include system owners and users, and their attributes are also important to the success of the system development process. The agile approach stresses that customers should be committed, knowledgeable, collaborative, representative, and empowered (Boehm, 2002). As mentioned previously, they should be co-located with the system development project team.

In the context of this discussion, *committed* means that the customer's primary work assignment is to be a member of the system development project team and to be available to provide guidance and knowledge for the system development process. In contrast, the plan-driven approach usually relies on a customer whose primary work responsibilities are assigned outside of the development project, and so he or she is not always immediately available.

Requirements

The requirements traits of the agile approach are in stark contrast to those of the plan-driven approach. Plan-driven methods are most effective when requirements are stable and known in advance (Schwaber and Beedle, 2001), whereas the agile approach embraces volatile and emerging requirements. Agile methodologies address the issue of how to better handle the inevitable changes that arise throughout the system development life cycle. In fact, agile methods are not just about accommodating change, but focus on embracing it without compromising quality. System owners demand and expect innovative, high-quality software that meets their needs and expectations, and agile methods help meet those demands.

Architecture

With the agile approach, the architecture is produced and refactored as needed, as opposed to the plan-driven approach, which determines the ideal system architecture up front for current and foreseeable requirements (Astels et al., 2002).

Table 1: Home Ground Attributes for Plan-Driven vs. Agile Methods

Home Ground Project Parameters	Plan-Driven Attributes	Agile Attributes
Developers	Plan-oriented, adequate skills; access to external knowledge	Agile, knowledgeable, co-located, and collaborative
Customers (system owners and users)	Access to knowledgeable, collaborative, representative, and empowered customers	Dedicated, knowledgeable, co-located, collaborative, representative, and empowered
Requirements	Knowable early; largely stable	Largely emergent; rapid change
Architecture	Designed for current and foreseeable requirements	Designed for current requirements
Size	Larger teams and products	Smaller teams and products
Refactoring	Expensive	Inexpensive
Primary objective	High assurance	Rapid value

Source: Boehm, 2003.

However, recently many organizations are utilizing initial envisioning and requirements modeling, referred to as *Iteration 0*. This practice is particularly useful for scaling the project to larger, more complex, and globally distributed development efforts (Ambler, 2007b).

Size

Agile methodologies are intended for smaller, less complex information system projects that consist of fewer than 10 team members. In the past, agile methods were thought to be difficult to scale up to large projects because of a lack of sufficient architecture planning and over-focusing on early results (Boehm, 2002). However, in recent years this shortcoming has been reported less and less. For example, there have been several organizations that have successfully implemented agile practices with large project teams, some as high as 200 developers (Benefield, 2008; Adkins and Baldwin, 2008; Ambler, 2007a).

Refactoring

The assumption is that with skilled developers and small, less complex systems, the cost of refactoring is essentially free. Conversely, refactoring costs in larger, more complex systems with increased requirements and less experienced developers "can be expensive" (Boehm, 2002).

Primary Objective

The highest priority of agile methodologies is to satisfy the customer by quickly delivering the most valuable features early in the development project (Beck et al., 2001). The intention of the agile movement was to break the cycle of process inflation and to focus on simple techniques for helping teams quickly reach their goals (Beck et al., 2001). In an agile environment, this is accomplished by implementing more or less as a series of small projects (called stories in the XP lexicon), strung together into a larger, ongoing project. Each story has its own requirements, specifications, and project phases (Armitage, 2004).

Ambler (2001) points out that the agile approach is an attitude, not a prescriptive process. The agile method is a supplement to existing methods; it is not a complete methodology. The agile approach is

a way to work together effectively to meet the needs of project stakeholders. It is effective and is about being effective. He also contends that the agile method is something that works in practice; it isn't an academic theory (Ambler, 2001).

Overview of Scrum

The Scrum Process

Scrum is a project management method for agile software development. The first Scrum meeting occurred in 1993 at the Easel Corporation (Sutherland, 2004). A development process was needed to support enterprise teams where visualization of design immediately generated working code, and Scrum was developed as the solution to that problem. The "Scrum process" consists of a combination of daily meetings, Scrum "sprints" to the next incremental delivery point, time boxing ongoing development work, and processes for managing the backlog. More details on each of these concepts follow. Figure 1 on page 18 shows the agile development process using Scrum.

Daily Scrum Meetings

Scrum emphasizes short daily meetings involving all agile project team members, including the system owner. These meetings are designed to address immediate problems and keep tabs on progress on a frequent basis. The daily meetings are led by a ScrumMaster, who is charged with keeping track of the status of the tasks and who is sometimes also a technical contributor. The daily meetings provide an opportunity for problems or misunderstandings to be reviewed by managers and discussed before they become damaging and impact schedules (Bradbury, 2007). Having a cross-functional team is also beneficial. Daily Scrum meetings generally center on answering three key questions (Sutherland, 2005):

- What did you do yesterday?

- What makes sense to do tomorrow/today?

- What is blocking the way?

Scrum Sprints

All development for an increment release occurs within a time box known as a "sprint" within Scrum. Scrum sprints are usually in 30-day increments, and the set of requirements the team works

on during a sprint are frozen at the outset. Changes to these requirements are allowed only if the delay is acceptable to everyone involved. At the end of each sprint, the software is reviewed to ensure ongoing feedback to the project team and system owner (Martin et al., 2003), and changes are then included in the next code delivery.

In some organizations, the timelines for individual sprints vary depending on the purpose of the deliverables (Sutherland, 2005). Table 2 is the scheduled timeline some organizations have adopted.

Time Boxing

The Scrum methodology divides requirements into a set of discrete tasks assigned to groups. Tasks are organized into small time boxes, and all required functionality is to be delivered within this set time box. This technique promotes completion of verifiable functionality at a faster rate. To accomplish this, daily drops of code iterations are submitted to a shared server. Automated tests are run regularly to

Table 2: Sample Timeline for Scrum Delivery

Code Implementations	Timeline
System maintenance	Weekly sprints
Customer enhancements	Monthly sprints
New application releases	Quarterly sprints

quickly and consistently verify the code. This keeps the development staff from being bogged down with continuous testing and adds a number of measurable interim progress points.

Processes for Managing the Backlog

Some of the more important guidelines for effective implementation of Scrum include managing the system backlog. The system owner controls and manages the business plan, functional specifications, system backlog, and prioritization. As a member of the team, he or she works side by side with the ScrumMaster. The system backlog should contain one central prioritized list of all requirements. Only

Figure 1: The Agile Development Process Using Scrum
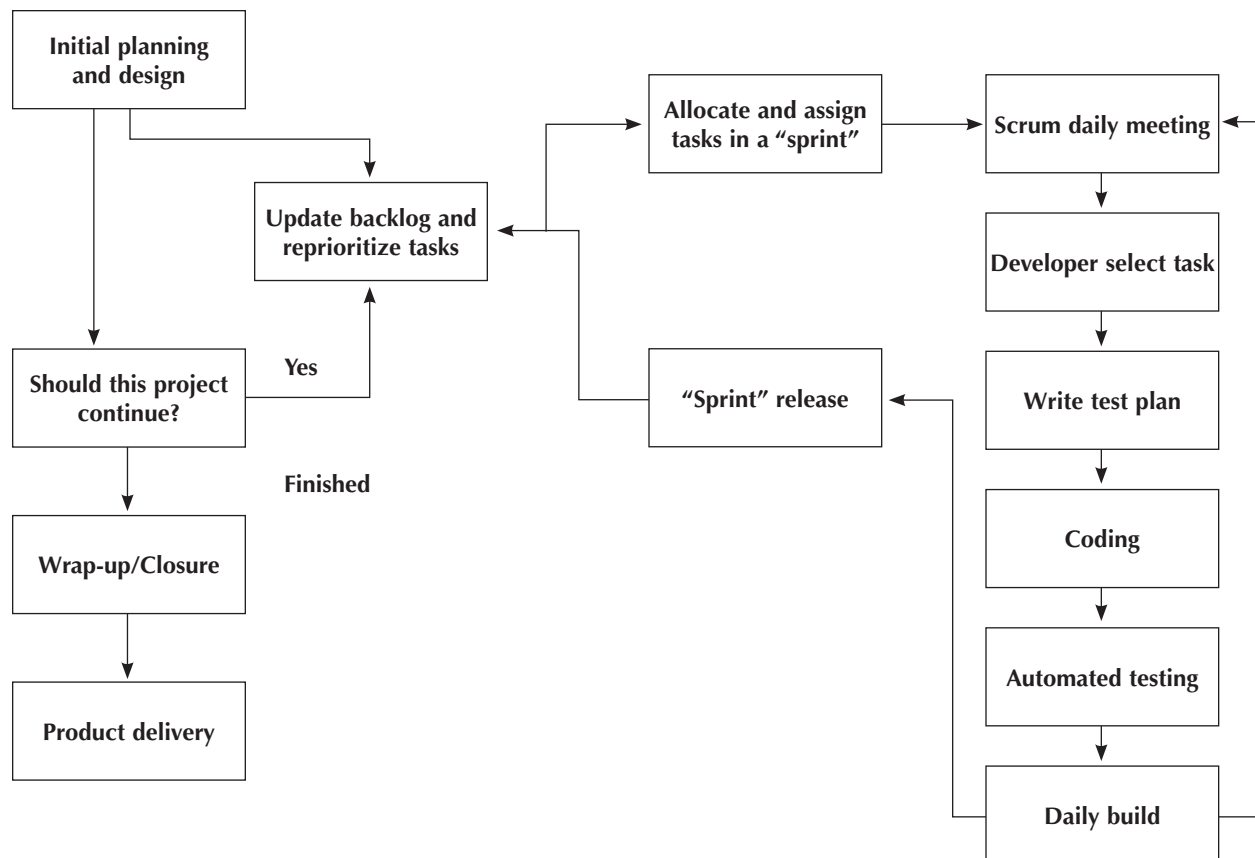
Source: Based on Rising and Janoff (2000).

one person, known as the system owner, controls the system backlog. However, everyone can contribute to the list (Martin et al., 2003). The team confers on items in the system backlog and identifies the backlog items that are assigned to the next sprint, and then the technical team members break down the backlog items into manageable tasks. In some teams, these tasks are written on sticky notes and posted on a board for developers to select.

Between sprints, it is important for developers to get enough clarity about the user requirements to start coding for the next sprint. This is critically important, since lack of understanding may cause the developer to code well into a sprint before he or she really understands the user experience well enough to implement a solution, and that can be a problem. One way to address this problem is to wrap the Scrum process around the XP software development practices. User stories and scenarios can help

sort out the missing requirement information and improve understanding among the developers.

Another option is to overlap sprints and work on requirement refinement in parallel with a sprint involving previously defined requirements (Sutherland et al., 2007). This gives the customer continuous updates while also addressing the incomplete requirements issue. Furthermore, the requirements definition for system backlog items should be worked concurrently with the current sprint to enhance overall efficiency (Sutherland, 2005).

Table 3 provides a summary of the key elements of the Scrum philosophy.

Benefits of Scrum
Organizations can achieve several benefits when effectively implementing Scrum. It has been reported that sprints can double system develop-

Table 3: Principles of Scrum

Principle	Description
Daily Scrum meetings	Daily status discussions with the entire project team. Often these are "stand up" meetings and occur first thing in the daily work cycle.
Inclusive customer (system owner) involvement	Customers (system owner, user) are part of the development team.
Incremental releases	Frequent intermediate deliveries of software with working functionality. Incremental releases that include an opportunity to validate and verify at shorter intervals, rather than at the end, providing time to fix problems and thus reducing the cost of fixes.
Risk management	Risk mitigation plans are created by the developers and implemented at every stage of the project.
Collective sharing of tasks and status	Transparency in planning and module development. Everyone should know who is accountable for what and by when.
Frequent stakeholder meetings	Frequent stakeholder meetings to monitor progress and provide visibility of potential slippage or deviation ahead of time.
Immediate problem disclosure	No problems are swept under the carpet. No one is penalized for recognizing or describing an unforeseen problem.
Energized work environment	Workplaces and working hours must be energizing. Developers have a "can do" attitude.
Partial solution encouraged	Appreciate that providing partial solutions earlier can be more valuable than providing full solutions later.
Simple design	Design solutions and work products that are the simplest version of the idea and that can be easily changed and/or built on later.
Sprint/time boxing	30-day increments where a set of requirements are developed and released and organized into time boxes.

Source: *Based on Bradbury, 2007; Sutherland, 2005.*

ment productivity, that development teams repeatedly deliver projects on time and within budget, and that the functionality is precisely targeted to end-user demands (Sutherland, 2005). Recently it has been reported that the implementation of Scrum in a data-rich CMMI Level 5 company simultaneously running waterfall, incremental, and iterative projects showed that the productivity of Scrum teams was at least double that of waterfall teams, even with CMMI Level 5 reporting overhead (Jakobson, 2007).

These benefits are partly a result of reduction or elimination of the development bottleneck because of the improved management of the product backlog during the sprints (Sutherland, 2005). Scrum also assists in streamlining the requirements communication among the team members, aligning individual and organization objectives, creating a culture driven by performance, supporting stakeholder value creation, achieving stable and consistent communication of performance at all levels, and enhancing individual development (Sutherland et al., 2007).

Evolution of the Scrum Sprint

There are variations on how Scrum has been operationalized. Teams new to Scrum follow Type A, and as teams become more experienced and efficient they evolve to Type B. A third type of Scrum (Type C) is also proposed (Sutherland et al., 2007). One oft-cited concern for the Scrum process is the downtime between the iterations, although downtimes in the plan-driven approach are often comparable.

- **Type A Scrum** is composed of isolated cycles of work, which is suitable for groups that have just started using Scrum since they can use the downtime between sprints to adjust.

- **Type B Scrum** introduces the overlap between sprints, so that the backlogs can be prepared at the end of the last sprint, and the development team has more time to figure out the functional specification.

- **Type C Scrum** is fast-paced and adds more overlap between sprints. MetaScrum has been introduced to allow company leadership to manage multiple simultaneous product releases. Using a MetaScrum approach could help with the small size and scalability issues identified.

In one company where Type C Scrum was applied, the company reported increased productivity and enhanced project quality, and achieved more stable and consistent communication (Sutherland et al., 2007).

Overview of eXtreme Programming

One of the more prominent approaches adhering to the principles of agile software development is eXtreme Programming, or XP. A lightweight software development methodology, XP was originally designed for teams of up to 10 people and for projects that need to develop software quickly in an environment of vague or rapidly changing requirements (Beck, 1999). Although XP was originally envisioned for smaller projects, some groups are finding success using XP on larger projects by using a number of separate teams of up to 10 people to address scalability; one project currently in progress for DoD is implementing XP by using five 10-person teams on a large development project (Adkins, 2008).

The XP process itself can be characterized by the use of short development cycles, incremental planning, evolutionary design, and an ability to respond to changing business needs. XP emphasizes productivity, flexibility, teamwork, minimal documentation, and the limited use of technology outside of programming. XP promotes a discipline of software development that is "people-oriented" (Beck, 1999; Beck, 2000). System owners are responsible for identifying the features that the developer must implement, assigning priority to them, and then providing detailed acceptance tests for those stories chosen for work. Developers constantly review system scenarios—known as "stories" in XP—that are of highest priority to the customer, and then focus on quickly delivering the functionality described in those scenarios (Beck, 2000).

Figure 2 illustrates the XP process, including the frequent iterative development cycles of small releases of functionality with constant end-user or system-owner consultation and engagement (Beck, 1999; Beck, 2000). Developers should implement the stories the system owner wants, in the order the system owner wants, and verify that the software passes any tests that the end user and/or system owner specifies. Each cycle begins with gathering end-user stories representing system requirements. The stories

Figure 2: The XP Process
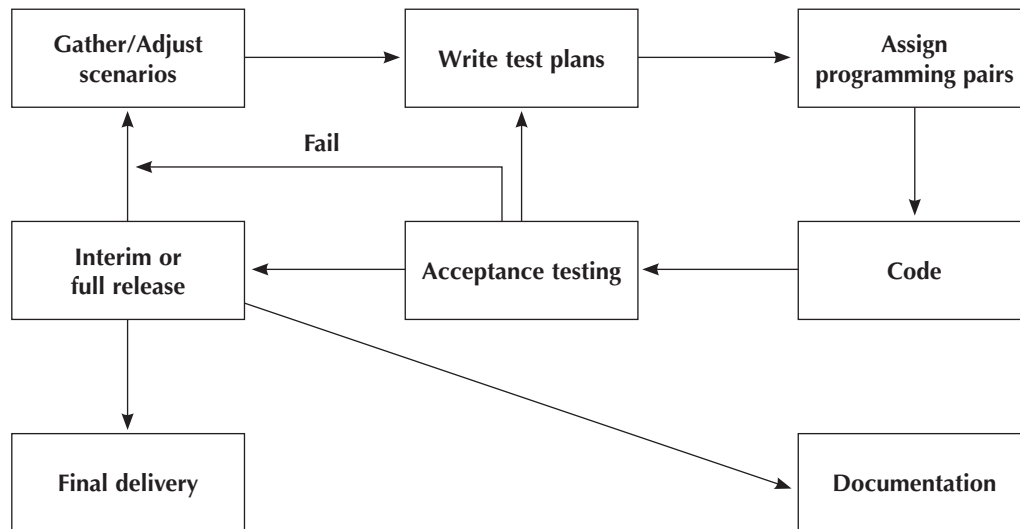
Source: Based on Beck, 2000.

and their associated requirements are simple. System owners and developers then determine which requirements will be developed in the next iteration by prioritizing the stories. Developers are assigned to specific stories or tasks within the stories, and test plans are written before any coding begins. Developers generally work in pairs, with one person coding while the other verifies correctness and conformity with standards. Roles are reversed frequently within the paired teams to ensure that each developer gets equal exposure, and team composition is sometimes rotated as well to maximize cross-training opportunities.

Upon completion of the code, each section of code is tested according to the defined functional test cases. The end user and/or system owner must also be engaged in testing. If the acceptance test fails, the end user and/or system owner and developer will meet again to adjust the end-user stories as necessary, and the process will repeat. Once the testing is completed with expected results and the system owner accepts the results, the enhancements to the system are released. New enhancements are generally released in two-week cycles. Following the final incremental release, documentation is completed and a final delivery of the system is made to the end user.

The difference between XP and iterative design is that XP builds and releases smaller systems strictly

at extremely high fidelities, whereas iterative design typically seeks to model, assess, and revise larger systems at low and high fidelities (Armitage, 2004).

The 12 Principles of XP

XP itself is defined by a set of 12 principles. Table 4 on page 22 lists the principles and provides a short description of each. These principles define XP, and their use is core to the implementation of the XP method. Research suggests that following these principles leads to several advantages over traditional software engineering methods (Beck, 1999). However, critics of XP state that the 12 principles are highly interdependent and that each principle cannot stand on
its own because of its reliance on at least one other principle for support. Other studies have found that tailoring the XP principles can be problematic (Stephens and Rosenberg, 2004). However, our research shows that organizations can successfully tailor the 12 XP principles to best fit their development environment, and that an XP project can be successful even if not all of the principles are adopted and used.

System Metaphor
System metaphors are a powerful way to relate a difficult idea in an unfamiliar area by defining a concept or feature using a simplistic representation (Astels et al., 2002). For example, a system that is

Table 4: The 12 Key Principles of XP

Principle	Principle Overview
System metaphor	Metaphors are used for communicating an overarching description of the system's functionality, purpose, and conventions, thus providing customers and developers with common ground on which they can stand. A metaphor does not convey an understanding of what something is, but rather what it is like. A metaphor is designed to allow customers who cannot conceptualize a system to describe its likeness.
Incremental planning or the planning game	Work is planned in chunks, with the on-site customer playing a large part in requirements determination and work prioritization. Planning is continuous and progressive. Developers estimate the cost of the candidate features, and customers prioritize features based upon cost and business value. The customer is heavily involved from the beginning and selects the priority of the work to be done. Decisions are based on brief estimates of work and cost provided by the developers.
Small releases	Frequently releasing simple systems, and releasing new versions on a very short cycle (one to three weeks). Development team puts core functions into production early and then builds upon them using feedback from users.
Simple design	Keeping the system design as simple as possible and finding and removing extra complexity. Simple and small pieces of design allow frequent changes to be made as necessary. A program built with XP should be the simplest program that meets the current requirements. There is not much building for the future. Instead, the focus is on providing business value.
Test-first development	XP teams focus on validation of the software by writing tests first, then software that fulfills the requirements reflected in the tests. Frequent user acceptance tests ensure the system is fulfilling user requirements.
Refactoring	A technique used to improve code without altering functionality. Focuses on simple, clean, non-repeating code that can be easily changed.
Pair programming	Two programmers developing production code at the same time on one machine. Pair programming has been shown by many experiments to produce better software at similar or lower cost than programmers working alone.
Collective ownership	Everyone owns all of the code, allowing necessary changes to be made by anyone at any time. This lets the team go at full speed, because when something needs changing, it can be changed without delay.
Continuous integration	Integrating new changes with current code as they are completed to detect system failures as soon as possible. Perhaps surprisingly, integrating more frequently tends to eliminate integration problems.
Sustainable pace	Developers keep a normal work schedule to remain productive and interested in the project. Tired developers make more mistakes. XP teams do not work excessive overtime, thus keeping them fresh, healthy, and effective.
On-site customer	A customer sits with the development team full-time. An on-site customer is available for requirements clarification and business decisions that should not be made by the developer. The effect of being there is that communication improves, with less hard-copy documentation, which is often one of the most expensive parts of a software project.
Uniform coding standards	Developers write all code in accordance with the standards agreed upon by the team to ensure that communication is made through code. Coding standards (language, formatting, syntax) are agreed upon by the team at the start of a project. This facilitates communication and ease of development.

Source: *Based on Beck, 2000.*

designed to sell books via the Internet may use a shopping cart as a metaphor to understand the process whereby customers can virtually select a book and take it to the virtual checkout for purchase. This gives the developers and users a common understanding of how the system is expected to perform.

Incremental Planning (or the Planning Game)

XP recognizes that not everything is known at the beginning of the project. The XP planning game makes a rough plan quickly and refines the plan as things become more and more clear. Its goal is to quickly derive a high-level plan for the next release or iteration. Planning is continuous, and so occurs throughout the system development process (Auer and Miller, 2002).

Small Releases

Small releases and the continuous design process allow for frequent review of the system by the developers and the users (Shore, 2004). XP depends on rapid feedback from the customer to establish the accuracy of the functionality of the scenario that is being implemented (Beck, 2000). This constant and quick feedback helps identify errors, supports ongoing usability tests (Martin, 2000; Muller and Padberg, 2003), and builds trust between the user and the developer.

Simple Design

The simple design approach enables frequent changes to be made to the system easily and encourages design experimentation.

Test-First Development

According to Beck (2000), testing should occur every time a code change is made, and efficiency is improved when testing is automated. The implementation of an automated testing method and use of test-driven development (where unit tests are developed before actual code is written) have been shown to increase confidence in functional and system reliability (McMahon, 2003; Smith and Stoecklin, 2001). Yet, test-first development does not always scale well (Ambler, 2008).

Refactoring

Refactoring is a key principle of XP that allows for the pieces of code to fit together better and adapt to

inevitable changes. Used to improve code without altering functionality, refactoring is designed to produce programming units with a strong internal structure. Refactoring can be thought of as XP's eraser; it involves taking time out from adding new features in order to rework and integrate what has already been accomplished (Armitage, 2004). These refactored programming units are generally more reusable, object-oriented, pattern-oriented, maintainable, and simple (Smith and Stoecklin, 2001). Further, refactoring reduces the complexity of the code by removing unused code and helping to implement consistent patterns (Elssamadisy and Schalliol, 2002). Refactoring also helps developers respond quickly to changing requirements (Smith and Stoecklin, 2001). However, there are mixed opinions on the costs involved with refactoring. Refactoring is expected to be done on an ongoing basis, and this may lead to extra cost and higher system overhead.

Pair Programming

Pair programming occurs when two people develop tests and create code side-by-side. The thought is that there is a free flow of ideas, a richer experience base, and early defect detection, resulting in improved overall quality of the tests and code (Muller and Padberg, 2003). There is an ongoing debate on the value of pair programming (Keefer, 2005). In some cases, empirical research has found that pair programming produces higher quality code at a lower cost (McMahon, 2003). Further, it is argued that the increase in the cost of development using XP is offset by productivity gains (for example, a pair of programmers has a higher development speed than a single programmer) and increased quality of the code through the continuous checking of the code against test cases by the second programmer (Cockburn and Williams, 2000; Elssamadisy and Schalliol, 2002). Yet pair programming could also increase system overhead if higher code production and quality code do not occur. In addition, in one case study it was a hard concept to implement; team members were not comfortable working as pairs (Poole et al., 2001). Despite the resistance, quantitative improvements in productivity occurred (Poole et al., 2001).

Collective Ownership

XP maintains collective ownership in which everyone owns all the code and coding changes

are made by anyone at any time. Close team interaction, shared goals, and limited time invested in elaborate designs provide for an environment where there is less defensiveness and territoriality (Armitage, 2004). Using XP as the development methodology has been shown to increase team morale (Poole et al., 2001).

Continuous Integration

XP is said to improve overall product stability and maintainability (Poole et al., 2001). It is believed to enable effective software development by allowing organizations to deliver and change requirements quickly during the software engineering process. This is partially due to the continuous integration of new code into the collective code base. Coding assignments are broken into small tasks, preferably of no more than one day of effort each. When each task is completed, it is integrated into the collective code base. At all times, regardless of the level of functionality, the system compiles, runs, and passes all the tests. When the new code is integrated into the current collective code base, the system must meet all these criteria. Continuous integration often leads to making new system builds multiple times per day.

Sustainable Pace

The sustainable pace principle recognizes the importance of a reasonable workweek in which XP team members can sustain quality. It is important to come to an agreement within the team on expectations for the hours team members work to keep a sustainable pace.

On-Site Customer

XP is serious about customer involvement—so serious that it is mandated that the customer be a full-time member of the project and co-located with the development team. The on-site customer is very important to the success of the project, and provides valuable, timely feedback. Without this feedback, the system development process becomes trial and error.

Uniform Coding Standards

The value of implementing coding standards has long been recognized when developing software. Coding standards serve as a means to produce software that has a consistent style, independent of the author, resulting in software that is easier to understand and maintain.

Organizational Readiness and Best Practices

This section details the knowledge gleaned from our interviews and review of the literature on agile processes. Eleven IT project teams experienced in agile development practices were interviewed and surveyed for this study. As mentioned earlier, this involved seven DoD project teams, three industry project teams, and one university team. Team members at all levels (developers, system owners, analysts, project leaders, government program managers, and government functional managers) were interviewed. Throughout this section we highlight quotes from some of these interviews.

Organizational Readiness

The experiences of the agile project teams underlined the need for leaders to examine four fundamental aspects of their organization before embarking on agile transformation. They are:

- The state of the current organizational culture

- The current IT infrastructure

- Management and leadership commitment

- The transitional first project

Current Organizational Culture: Ready to Embrace Change?

The current organizational culture will impact the readiness for agile methods. Management needs to assess questions such as:

- How does our organization as a whole react to change?

- How entrenched are the members of our organization in the current process?

- How good is communication within the organization?

- Does information flow freely up and down the hierarchy?

Answers to these and similar questions will indicate the readiness of the organization to accept agile methods.

Current Organizational IT Infrastructure: Ready to Invest?

The current IT infrastructure that is available is important to the overall success of agile development. IT developers and managers need to have access to various technology and technical resources such as code development and testing tools, fast Internet, and state-of-the-art workstations and testing environments. This is important so that they are not hampered by a less-than-adequate IT infrastructure. Regardless of whether the system development process is optimized, the end result will not meet expectations if the IT infrastructure is marginal. Therefore, there must be a pre-existing underlying organizational culture and infrastructure that is ready to support the agile development transition before the best practices can be fully effective.

Management and Leadership Commitment: Ready to Endorse?

Every team stressed the importance of management and leadership backing to the success of their agile development process. Endorsement of the agile system development movement needs to be communicated from the top down. There must be both horizontal and vertical dedication to the agile process across the organization. Management can further demonstrate its support by providing any additional

software and technical resources needed by the project team, including project management tools, software testing tools, and Scrum and XP training. The literature also acknowledges that one of the most important factors for successful agile implementation is to have a sponsor who is committed to championing the agile movement (Schatz and Abdelshafi, 2005).

"In one organization, everyone from the CEO to the end user participated through a three-day agile education and training course. This created unity and a common understanding of where the institution was going throughout the entire organization."

Transitional Project Selection: Ready to Begin?

The scope, importance, size, and criticality of the transitional projects using agile methodologies need to be considered carefully. The key is to start small and focus on a compatible project. Just as a doctor does not start his medical training by conducting open-heart surgery on a critically ill patient, it is unrealistic to start the movement to agile methods on a project supporting mission-critical or enterprise-level projects. Essentially all of the organizations we interviewed initially employed agile methods on a relatively small project within their organization and then progressed from there.

Best Practices for Initial Startup

Select Agile Team Members with the Specific Attributes Needed

Second to the pre-existing underlying organization culture that is receptive to change, the agile team members' characteristics are the most important factor to the success of agile software development, according to the teams interviewed. Team members should possess the following attributes:

- **Can-do attitude.** Team members should have an attitude of not being afraid to fail, a willingness to learn from mistakes, and an ability to critically assess a situation before pressing forward. They should also have an outlook that embraces new technology and technical challenges. There should be a feeling of readiness among the team

members for the agile software development process and new technology. Team members are expected to be self-starters.

- **Experienced problem solvers.** IT developers assigned to agile teams that were reported as successful were intelligent, talented, strategic thinkers and excellent problem solvers. Some managers interviewed said the teams could still be effective as long as at least half of the development staff have a strong technical understanding of various system architecture options and a solid understanding of the application process. Experienced and inexperienced developers could be paired to minimize weaknesses and facilitate mentoring. They also mentioned it is advantageous if the ScrumMaster is also a technical expert.

- **Mutual trust.** Mutual trust among team members was reported as one of the top keys to the success of the agile process. In fact, most project teams reported a high level of trust and respect among team members.

- **Excellent communication and interpersonal skills.** Another characteristic of highly effective teams is that team members have excellent communication skills. They need to be able to effectively communicate with technical and non-technical people, engineers, managers, and customers. Team members need to have the ability to work effectively in a team environment and demonstrate team-building attitudes and skills, interact positively with others, and have a strong understanding of customer service philosophies.

- **Domain-knowledge expertise.** Mission-critical systems can benefit from agile development as long as the majority of the team members are experts who fully understand the complex processes and expected results of the system they are constructing.

Promote Team Building

Team building is especially important when team members are unfamiliar with each other or when new communication processes are introduced. An important prerequisite for an agile development project is to build trust and learn how team members communicate. Many of the teams interviewed allocated time to work on intra-team communication

exchanges at the beginning of the project. One- and two-day off-site workshops were employed. Project managers reported that this further promoted a team culture of collective ownership. In some cases, it helped change the culture from an environment where individuals were responsible for the various tasks to a culture where the "team as a whole" was responsible for the project. Collective ownership has been found to be a key XP principle to project success (Fruhling and McDonald, 2008). Without strong collective ownership, the flexible and dynamic code development process is hindered.

Train Team Members in the Agile Process

Formal training in agile practices such as XP and Scrum across the organization is highly recommended. This includes all levels of IT professionals and non-IT management, the system owner(s), customers, business analysts, and any other agile development team stakeholders. IT managers stated it was very important that all parties have a common understanding of the agile practices and how they will be implemented within their organization.

Practical training can also be an effective tool. Agile project teams interviewed recommended starting with a pilot project and then implementing it throughout the organization, project by project, as new projects begin. Some organizations also hired an external, experienced ScrumMaster to run their first agile pilot.

Provide Information Technology Support Tools

The teams must have the right development tools available at the outset. This may seem like an obvious requirement, but in the government environment, procurement, evaluation, and security scrutiny of various system development aides and tools may take substantial lead time and additional levels of approval. In one case, a DoD system manager reported it would take over 18 months to go through procurement and security audits to acquire a requirements management tool. The selected IT support tools must be included in the IT project proposal budget submitted by the DoD contractor. The most common tools teams recommended as being especially effective and essential for agile development were automated testing and collaboration tools

like Requisite Pro, Rational Robot, Websphere, and Sharepoint. Rational Robot improved the speed and quality of software testing for many teams, and Sharepoint was identified as a good tool for issue tracking of defects and changes. In addition, it was extremely important to have a solid software version management tool.

Ensure Trusted Version Control Management

It is especially important in a dynamic agile environment that developers be able to do daily builds of a system that links all recently configured files. The developers need to have complete confidence that all the checked-in files are complete and included in the daily build. Therefore, it is key that a robust version control system is utilized.

One team especially benefited from having the team's technical lead create scripts that combined all of the scripts to build the entire system daily from scratch and keeping these scripts up-to-date and available to all team members. This was for a mission-critical, complex system, and thus ensured everyone was working with the same set of modules at all times. This improved productivity and also allowed the two-week code drops to continue even as the size and complexity of the system grew.

Consider Carefully the Organization and Size of the Team

Preparing for agile software development may require a reconfiguration of the IT project teams. IT managers agreed that the ideal number of developers on a team was four; however, there could be more if needed. The consensus was that the members of each agile team should consist of:

- Developers (four)

- Quality Assurance tester

- Human factor engineer

- Database administrator

- System architect

- System owner/end user/customer (two in a DoD environment: one primary and one backup)

- Floating technical writer

The Role of the Quality Assurance Tester

The Quality Assurance (QA) tester position is essential to the agile process because testing is so important to the overall quality of the system, and the amount of testing needed grows with each iteration. The IT managers stated that they automate what they can and what makes sense for regression purposes. The QA tester's role is basically to assist the project developers, analysts, and other project team members in writing test cases. They then use scripting to automate those tests, usually in QTP, although one group also used Fitnesse. They create driver scripts to allow the tests to be run at the push of a button or by an automated build process.

The QA tester needs to have the following qualities: a strong understanding of customer service philosophies and the role of information management with regard to quality assurance and testing automation, in addition to knowledge of and experience in structured analysis, application development strategies, testing automation tools and processes, quality assurance strategies, and technical and analytical skills. QA testers should have hands-on experience with software testing at all phases (unit, module, integration, regression, system/acceptance, load, performance, end-to-end). They should have experience in both manual and automated testing environments and be able to develop quality assurance and testing strategies; develop and document test scripts and cases using an automated testing tool; execute test scripts in accordance with the test plans; verify and document test results utilizing testing tools and utilities; provide first-level support of the testing tools and testing environment; and configure and support testing automation tools.

> *"Quality Assurance testers cannot assure quality, but they can assist in quality. It takes an entire team to build a quality product, which very much includes the developers. What this person brings is 'smart' testing, which is very different from the testing a developer would do. Developers tend to test the happy path. Also, there is an element of human testing that a computer cannot do. The goal is to automate what a human cannot do (for example, run thousands of tests in a short time, simulate many users at once, et cetera). Another reason to automate is to try to make sure the project team didn't break something that once worked (regression testing)."*
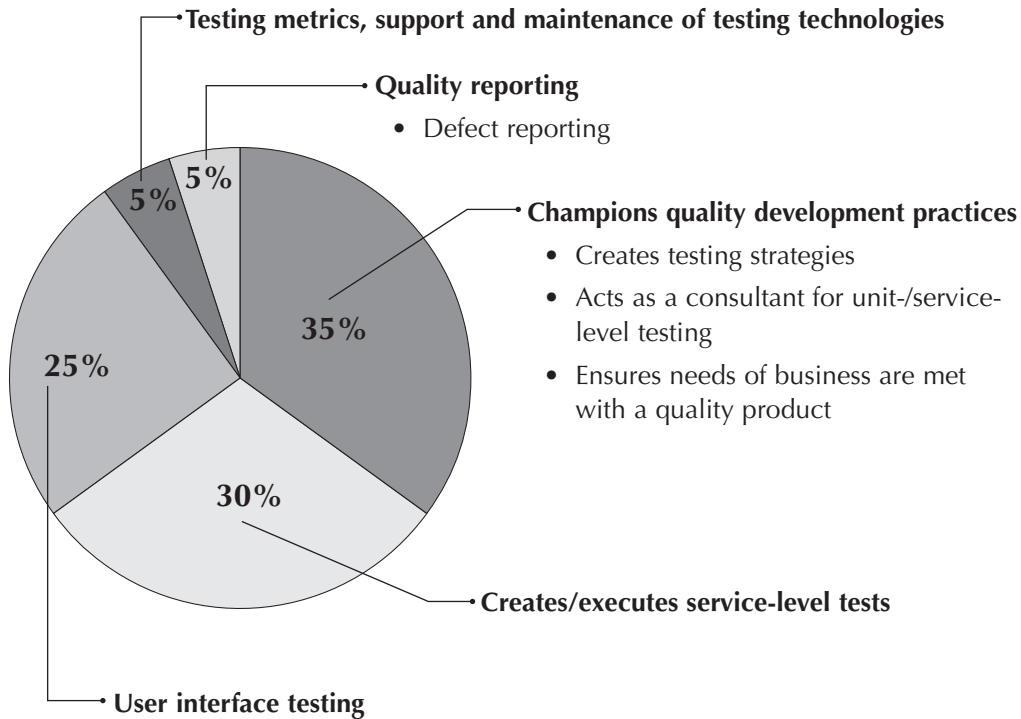
One organization tried sharing QA resources and found it to be too much of a strain. Therefore, IT managers recommend that each team needs at least one dedicated QA resource, plus the help of the rest of the team in testing.

The QA tester is responsible for:

- **Customer service.** Takes the necessary steps to ensure the customers' needs are met to the maximum extent possible in an accurate and timely manner. Communicates with technical and non-technical people.

- **Quality assurance.** Proactively interacts with all agile project teams to develop and promote quality assurance strategies to improve overall information system quality assurance. Recommends new processes, software and/or systems to improve the organization's information systems development effectiveness. Assesses and monitors adherence to software change control and other process standards. Identifies and communicates when QA processes are not being adhered to or when insufficient quality conditions exist in a project or process.

- **Software testing.** Displays a "test to break" attitude and an ability to take the point of view of the customer, a strong desire for quality, and an attention to detail. Creates test scripts and test plans to enable streamlined application testing practices and trains other employees in how to do this. Ensures that all software is tested for defects, meeting end-user requirements. Works with testers, system owners, and agile project teams to ensure that all problems are documented and resolved.

- **Various other project tasks.** Acts as a consultant for projects, processes, project implementation, and application development.

Workload Breakdown of the Quality Assurance Tester

In one organization, directors recently met to discuss the workload breakdown of this role and came up with the following description (Schmidt, 2008):

Testing metrics, support and maintenance of testing technologies

Quality reporting
- Defect reporting

Champions quality development practices
- Creates testing strategies
- Acts as a consultant for unit-/service-level testing
- Ensures needs of business are met with a quality product

5%

5%

35%

25%

30%

Creates/executes service-level tests

User interface testing

Assign Two Government User-Collaborators (System Owners/End Users)

On government agile projects where an end user assigned to the agile project may unexpectedly be re-assigned to a different task (due to military assignment rotations) in the middle of the project, it was recommended to have two government users serve as the main contacts for the project. The agile team leaders said that having a second dedicated end user kept the project moving forward when one was unavailable, and thus the developers could continue the short iterative software releases.

"We quickly realized that in our environment, where our end users may be called away with little notice due to the mission of the DoD, that assigning two user-collaborators just made good sense. Consistent feedback from the end user stayed intact from assigning two user collaborators."

Provide Easy, Quick Access to Technical Experts

Various technical experts—for example, the security manager, network administrator, chief architect, and database administrator—need to be available as needed to keep things moving. Developers need to have an attitude that they are willing to say, "I don't know this" and "I need help." Likewise, technical experts need to have an attitude that they are willing to help and not be judgmental.

Make the Agile Development Effort the Only Assignment

Both the agile project leader and the agile team members must have the agile IT project as their *only* primary responsibility. This is a known agile requirement, but is often not the case when developers are assigned to both maintenance and development projects, or in some cases when they are responsible for more than one information system. To reach optimum efficiency, there should be no exceptions to this best practice.

A distributed team lead stated that having the developers off-site freed them from being caught up in the day-to-day politics of the office, reducing distractions and allowing them to focus on the work at hand.

Use Collaboration Engineering Techniques

Both XP and Scrum appear to under-emphasize the challenging job facing the on-site customer and system owner in gathering and prioritizing the requirements for the project. To overcome these challenges, one team used collaboration engineering techniques and group decision support system tools to do initial planning and prepare a "way ahead" plan of requirements (Fruhling et al., 2007). System owners and other stakeholders expressed that collaboration engineering practices helped give the agile project a jumpstart and was an extremely effective way to validate, elicit, and prioritize requirements for the entire agile development project.

Establish Documentation Expectations

Team leads and developers stated it is acceptable to have less documentation as long as coding standards, such as naming conventions and in-line comments, are adhered to. Documentation can further be accomplished through user stories, test cases, and conditions of satisfaction. IT managers advised organizations to determine what standards are mandatory versus what standards are guidelines or recommendations.

Reach Agreement on the Iteration Cycle Timeline

All stakeholders need to come to a consensus on the timeline of the release cycles. Most teams agreed that the release cycle "sweet spot" is three-week intervals, although two-week intervals are commonly mentioned in the literature. It is easier to do agile development in two-week iterations when the project is only in development mode and there is not a production system in parallel. When the system is released to production and becomes more complex and the user community grows, three-week intervals are more doable.

Best Practices for Project Implementation

Conduct Initial and Incremental Planning Meetings

For every new agile project, IT managers suggested a two-day strategic planning meeting to develop a vision of the new system and to establish an overall workable, stable system architecture. The meeting should also define the scope, requirements, and design for the next three months. System owners should select three to five capabilities that the system must have and three to five capabilities that the system owner would like to have. As the XP process unfolds, new and smaller ideas are implemented within the three to five larger capabilities.

Conduct a Pilot Agile Project

Several IT managers recommended organizations begin by piloting the agile process on one team and holding weekly meetings for a period of time to discuss how to fine-tune the process within the organization. In these weekly meetings, members should continuously examine XP practices and strategize on adoption and adaptation of each practice. They also advised implementing a few practices at a time, letting the agile teams work through the process and become more effective, and then adding more practices.

Consider à la Carte Introduction of XP Practices

In a recent study, a survey was administered that collected information on the actual amount of usage of various XP principles and the perceptions of their importance to project success. Some of the surveys were returned by members of the teams interviewed. Other respondents were from large Midwest organizations that had also implemented XP. Results that are significant to this report are:

- XP principles identified as most often used were collective ownership, continuous integration, and planning game.

- XP principles identified as the most important to project success were continuous integration, test first, planning game, collective ownership, on-site customer, and sustainable pace.

Thus, the XP principles that were deemed the most important *and* used most often are collective ownership, continuous integration, and planning game. Therefore, it is logical that these XP principles are the first ones implemented.

XP principles that are less commonly used, but still perceived as important, include test first, on-site customer, and sustainable pace (Fruhling and Zhang, 2007). Based on this study, these principles should be considered for inclusion in the next phase of implementation of XP principles. The remaining principles could be considered later. Table 5 summarizes the results in priority order.

Focus on the Task, Not the Individual Status

One of the more experienced agile teams said that they had very positive feedback when the reporting at the Scrum meeting focused on the task rather than the individual. This shifts the focus from the individual to the team and, in effect, promotes collective ownership and collective problem solving.

"By emphasizing the status of the task, rather than the individual, managers noticed that when roadblocks were identified, more information sharing and problem solving occurred among team members instead of just the person with the problem being solely responsible for doing the problem solving."

Adjust Scrum to Match Project Scale

In some situations, project teams modified the Scrum standup meeting schedule for scalability (for example, team size) or if they were on a tight deadline. The daily Scrum meeting was adjusted to every other day. The meetings would occur on Tuesday/Thursday one week and then Monday/Wednesday/Friday the next week. Also, some project teams split the team and had two Scrum meetings for the same project.

Implement Time Boxing Productivity Management

Formal time boxing was one of the advanced project management practices employed by experienced agile teams. Time boxing helped system owners and

Table 5: Summary of Recommendations for Implementation of XP Principles by Priority

XP Principle	Most Common	Most Important
Collective ownership	1	1
Continuous integration	1	1
Planning game	1	1
Test first		2
On-site customer		2
Sustainable pace		2

other stakeholders evaluate if the development team was meeting their goals and the level of productivity. IT managers could assign the tasks in a sprint based on the estimated amount of effort and then evaluate if those tasks were completed accordingly. This may be especially useful in the government contract environment, where contractors are awarded based on their team performance.

Assign a Lead Pair Programmer

Pair programming was used sparingly, but one team that actively uses pair programming modified it by assigning one of the pair programmers to be the lead and the reporter of progress on the task. Lead programmers alternated throughout the project. This technique was implemented because in a few cases pair programming caused problems with accountability on task completion. There was not "one" person accountable to get the task completed or to report back when the pair was "stumped" technically and not making progress. There was a tendency to hide or not disclose that the pair team needed additional technical expertise. At times, neither member wanted to admit his or her lack of technical expertise, so in these cases peer pressure caused problems.

Outsource Documentation

Several teams enlisted a technical writer for creation of the required documentation. Their philosophy was to leave the software development to the developers and the documentation to a technical writer. For example, the technical writer was responsible for the system implementation documentation, maintaining the architecture documentation, and updating the training manual.

Match Tasks to Talents

Project managers emphasized the importance of having the agile task assignments match team members' talents to achieve optimum results. For example, developers with user interface design skills and experience were assigned to design and develop the graphical user interfaces, and developers with database knowledge were assigned database query functions.

Keep the Best Practices of the Past

IT managers cautioned to be careful not to throw away everything from structured methodology practices, such as design walkthroughs and over-the-shoulder code walkthroughs. One project team continued to use system-wide design and software walkthroughs periodically and included user interface usability evaluations in the process.

Best Practices for Ongoing Development

Designate an Agile Champion Team

Mature organizations using agile throughout their organization have designated a "champion" team to address problems of the agile Scrum and XP process. For organizations that have multiple agile projects, each agile team selects a representative to be part of a champion team that brings issues to this group to resolve. The membership of this team is rotated on an annual basis. The team members share strategies that are working well on their team as well as roadblocks or barriers their team is encountering with the agile process. The team works together to identify possible solutions. IT managers recommend the membership of this team be a cross-section of Scrum team members.

This team is also charged with the task of inspecting and adapting current techniques, in addition to reviewing Scrum and XP processes that are not employed to determine if they fit or can be modified within their organization. The interviewees emphasized it was important to recognize that adjustments of the agile practices will be on a team-by-team basis. Further, organizations must recognize that the agile process will need to be continuously fine-tuned (for example, *continuous refactoring of the agile process)*.

Schedule Open Time

Managers and developers mentioned that scheduling open time between 30-day sprints or quarterly was beneficial so that developers could tie up loose ends such as code cleanup, refactoring, miscellaneous small tasks, and documentation. This is another way to ensure that quality is being integrated into the final product.

Automate Continuous Testing

Teams who have adopted the practice of writing the test plan first are glad they did and stated they have greatly benefited. Also, teams that use automated testing tools say it has further enabled them to deliver new capabilities in short iterations and to ensure a quality product is delivered. Rational Robot was noted as an especially useful testing tool. It helped increase developer productivity and the quality of the product. Agile teams should plan on system testing being continuous.

Employ a Migration Control Expert

A distributed project team employed a migration control expert that managed the unclassified to classified code migration process. Developing in an environment that is unclassified and implementing in an environment that is classified has its own set of unique challenges. It was more efficient for the developers to work in an unclassified environment, thus allowing them access to Internet tools and resources, and then moving the code to the classified environment. The migration control expert was both a technical expert and application knowledge expert. The final classified installation of code was then handled by a third party.

Exploit Multiple Forms of Communication

In the case of the distributed team where the migration control expert and system owners were located at USSTRATCOM and the IT development team was located at the DoD contractor headquarters, multiple forms of communication were very useful. E-mails were used for communication and also as another form of documentation. The team used weekly videoconference meetings and quarterly face-to-face meetings to ensure that all members were kept current and to increase team unity. Also, all team members were always available by phone. In addition, they used Instant Messaging and accessed shared web pages to discuss requirements.

The migration control expert was the main liaison between the two groups. He had excellent communication and facilitation skills. In addition, he had a technical background, so he understood the technical issues well enough to discuss them with the development team and could present their concerns to the system owners.

Provide Access to the Internet to Research Solutions

All teams mentioned the importance of having access to the Internet to research technology problems and solutions. This is difficult to achieve in the closed IT environment often found at military installations. This is one of the major reasons the distributed teams worked better and were very effective.

"The success of our agile development effort was increased due to the developers having access to the Internet to research technical solutions. This is often not possible in most DoD IT development projects. Thus, our distributed environment helped overcome this constraint."

Address Classified Environment Challenges

IT managers identified the following major challenges in a classified government development environment that may impact the success of an agile development deployment:

- Personal computer workstation configurations

- Network availability

- Availability to connect to data sources

- NIPRNET (Non-Secure Internet Protocol Router Network) and SIPRNET (Secure Internet Protocol Router Network) versions

- Cross-domain security issues

- Infrastructure of web services architecture.

Although there were not any readily available solutions to these challenges, it is valuable to address them at the start of the project.

Conclusion

The major purpose of this research is to present best practices for implementation and management of agile systems development methods on software projects supporting DoD. We focus primarily on eXtreme Programming and Scrum methodologies, and identify 29 suggested best practices. These were developed through a combination of a comprehensive literature review and interviews with both DoD and commercial IT development professionals. The best practices are further divided into those that should be employed at the initial adoption of agile methodologies, those that should be inserted in the next phase, and those that should be included once the process is more mature. This gradual, phased approach is shown to be the most effective in migrating to the use of agile methods.

This report is aimed at facilitating the adoption of agile development methods by defense-related organizations. This is important, in that the use of agile methods will assist these groups as they help DoD transform its information architecture into a more modern, more adaptable, and more service-oriented entity. This transformation is critical to the future success of DoD, particularly with regards to its command and control systems. Agile methods can and should be at the core of that transformation, and increased understanding and appreciation of agile methods will facilitate and foster their increased use within DoD.

We strove to maintain focus on the operational level, with the intent of providing specific and practical actionable information to help IT managers and analysts prepare their organization for the transition. This research is important because it will help managers successfully introduce and implement the principles and practices of agile

methods into the staunchly traditional software development environment of the U.S. military. This report also presents best practices for managing agile software development teams, discusses strategies to fine-tune the agile practices, and addresses some of the unique challenges when working in a DoD environment.

We find that most organizations implement agile methods in a somewhat different manner and that à la carte adoption is the norm. Many organizations also customize the agile guidelines to fit their specific needs and environment, so future adopters need not worry too much about having to conform to rigid requirements.

Two factors in particular stood out as critical for successful implementation and use of agile methodologies:

- A culture that is ready and willing to embrace change

- An IT infrastructure in place to support the transition

Giving priority attention to such people-related factors as staffing, culture, values, communications, and expectations management is also critical to successful software development using agile methods.

This research is a start in addressing the full-scale adoption of agile development methods within DoD. There is still much work to be done and many opinions to be swayed, but the future is bright. More and more larger projects are utilizing some variant of agile methods, and more and more of them are finding success in that effort.

Agile itself means receptive, reactive, alert, and quick to respond—all outstanding traits for developing a service-oriented, customer-focused approach to application development. Agile methods are finding greater and greater acceptance within DoD, and the use of agile methodologies will allow DoD to compete at Internet speeds, as they must be able to do. Agile methods are the future of software development within DoD, and this research provides a preliminary glimpse of that future.

References

Adkins, M. 2008. Interview, January.

Adkins, M., and R. Baldwin. 2008. Accenture, February.

Alleman, G., M. Henderson, and R. Seggelke. 2003. Making Agile Development Work in a Government Contracting Environment—Measuring Velocity with Earned Value, Agile Development Conference (ADC '03), 114–119.

Ambler, S. 2001. What Is(n't) Agile Modeling. The Official Agile Modeling (AM) Site, www.agilemodeling.com/essays/whatIsAM.htm.

Ambler, S. 2007a. Agile Adoption Rate Survey: March 2007, www.ambysoft.com/surveys/agile-March2007.html (last retrieved March 18, 2008). *Dr. Dobb's Journal,* Jan. 3, 2008, www.ddj.com/architect/205207998.

Ambler, S. 2007b. Requirements Envisioning: An Agile Best Practice, www.agilemodeling.com/essays/initialRequirementsModeling.htm. Retrieved March 18, 2008.

Ambler, S. 2008. Scaling Test-Driven Development, *Dr. Dobb's Journal*, January 3, www.ddj.com/architect/205207998. Retrieved March 18, 2008.

Armitage, J. 2004. Are Agile Methods Good for Design? *Interaction* 11(1): 14–23.

Armstrong, D. 2007. Interview, USSTRATCOM.

Astels, D., G. Miller, and M. Novak. 2002. *A Practical Guide to eXtreme Programming*. Upper Saddle River, NJ: Prentice Hall.

Auer, K., and R. Miller. 2002. *eXtreme Programming Applied*. Boston: Addison-Wesley.

Beck, K. 1999. Embracing Change with eXtreme Programming. *IEEE Computer* 32(10): 70–77.

Beck, K. 2000. *eXtreme Programming Explained*. Boston: Addison-Wesley.

Beck, K. 2004. *eXtreme Programming Explained: Embrace Change*. 2nd ed. Boston: Addison-Wesley.

Beck, K., M. Beedle, A. Bennekum, A. van Cockburn, W. Cunningham, M. Fowler, J. Grenning, J. Highsmith, A. Hunt, R. Jeffries, J. Kern, B. Marick, R. C. Martin, S. Mellor, K. Schwaber, J. Sutherland, and D. Thomas. 2001. Manifesto for Agile Software Development, Snowbird, UT: Agile Alliance. Available online at www.agilemanifesto.org (retrieved October 2005).

Benefield, G. 2008. Rolling Out Agile in a Large Enterprise. Proceedings of the 41st Hawaii International Conference on System Sciences, January.

Biffl, S., A. Aurum, B. Boehm, H. Erdogmus, and P. Grünbacher, eds. 2005. *Value-Based Software Engineering*. Berlin: Springer-Verlag.

Boehm, B. 1988. A Spiral Model of Software Development and Enhancement. *IEEE Computer* 21(5): 61–72.

Boehm, B. 2002. Get Ready for Agile Methods with Care. *IEEE Computer* 35(1): 64–69.

Boehm, B. 2003. Value-Based Software Engineering. *Software Engineering Notes* 28(2): 1–12.

Bradbury, D. 2007. Scrum Down to Get a Project Moving. *Computer Weekly,* 00104787, 9/4/2007.

Cao, L., K. Mohan, P. Xu, and B. Ramesh. 2004. How Extreme Does Programming Have to Be? Adapting XP Practices to Large-Scale Projects. In R. H. Sprague, ed. Proceedings of the 37th Hawaii International Conference on System Sciences. Los Alamitos: IEEE Computer Society Press.

Cockburn, A. 2002. *Agile Software Development.* Reading, MA: Addison-Wesley.

Cockburn, A., and J. Highsmith. 2001. Agile Software Development: The People Factor. *IEEE Computer* 34(11): 131–133.

Cockburn, A., and L. Williams. 2000. The Costs and Benefits of Pair Programming. In Proceedings of eXtreme Programming and Flexible Processes in Software Engineering (XP2000). Cagliari, Italy.

Elssamadisy, A., and G. Schalliol. 2002. Recognizing and Responding to Bad Smells in eXtreme Programming. In J. Magee and M. Young, eds. Proceedings of the 24th International Conference on Software Engineering. New York: ACM Press.

Fruhling, A., and P. McDonald. 2008. A Case Study: Introducing eXtreme Programming in a Command and Control System for the US. Proceedings of the 41st Hawaii International Conference on System Sciences. January.

Fruhling, A., L. Steinhauser, and G. Hoff. 2007. Designing and Evaluating Collaborative Processes for Requirements Elicitation and Validation. Proceedings of the 40th Hawaii International Conference on System Sciences. January.

Fruhling, A., and D. Zhang. 2007. An Empirical Study Examining the Usage and Perceived Importance of XP Practices. Proceedings of the Americas Conference on Information Systems.

Glass, R. 2004. Matching Methodology to Problem Domain. *Communications of the ACM* 47(5): 19–21.

Grimes, J. G. 2006. Which Emerging Technology Will Have The Biggest Impact on Your Organization in the Future? *Signal,* AFCEA's International Journal. September.

Highsmith J., and A. Cockburn. 2001. Agile Software Development: the Business of Innovation. *Computer* 34(9): 120–122.

Iivari, J., R. Hirschheim, and H. K. Klein. 2001. A Dynamic Framework for Classifying Information Systems Development Methodologies and Approaches. *Journal of Management Information Systems* 17(3): 179–218.

Jakobson C. 2007. The Magic Potion for Code Warriors! Maintaining CMMI Level 5 Certification with Scrum. J. Sutherland, ed. Aarhus, Denmark: Agile.

Keefer, G. 2005. Pair programming: an alternative to reviews and inspections? *Cutter IT Journal* 18(1): 14–19.

Kehler, Robert C., Lt General, USAF. Address to the Armed Forces Communications and Electronics Association. Washington, D.C., 16 June 2006. Transcript: www.stratcom.mil/Spch&test/CD_AFCEA_16Jun06.html.

Martin, A., R. Biddle, and J. Noble. 2003. How do XP, SCRUM and ASD Build The Right Software? Position paper for the workshop "Are Agile Methodologies Really Different?" OOPSLA.

Martin, R. 2000. eXtreme Programming Development Through Dialog. *IEEE Software* 17(4): 12–13.

Martin, R. 2003. *Agile Software Development: Principles, Patterns, Practice.* Upper Saddle River, NJ: Pearson Education Inc.

McMahon, J. 2003. Five Lessons from Transitioning to eXtreme Programming. *Control Engineering* 50(3): 59–65.

Muller, M. M., and F. Padberg. 2003. On the Economic Evaluation of XP Projects. In P. Inverardi, ed. Proceedings of the Ninth European Software Engineering Conference held jointly with 10th ACM SIGSOFT International Symposium on Foundations of Software Engineering. New York: ACM Press.

Net-Centric Checklist, Version 2.1.3 2004. Retrieved June 3, 2006 from www.defenselink.mil/ nii/org/cio/doc/NetCentric_Checklist_v2-1-3_May12.doc.

Paulk, M. 2001. Extreme Programming from a CMM Perspective. *IEEE Software* 18(6): 19–26.

Poole, C., T. Murphy, J. Huisman, and A. Higgins. 2001. Extreme Maintenance. In G. Canfora and A. Amschler Andrews, eds. Proceedings of the 17th IEEE International Conference on Software Maintenance (ICSM '01).

Potok, T. E. 1992. Extensions to the Spiral Model to Support Joint Development of Complex Software Systems, Proceedings of the 30th Annual Southeast Regional Conference.

Rising, L., and N. Janoff. 2000. The Scrum Software Development Process for Small Teams. *IEEE Software* 17(4): 26–32.

Royce, W. W. 1970. Managing the Development of Large Software Systems: Concepts and Techniques. In Proceedings of IEEE WESTCON (Los Angeles, CA).

Schatz, B., and I. Abdelshafi. 2005. Primavera Gets Agile: A Successful Transition to Agile. *IEEE Software* 22(3): 26–42.

Schmidt, B. 2008. E-mail correspondence.

Schmidt, B. 2007. Interview, Farm Credit Services of America, October.

Schwaber, K., and M. Beedle. 2001. *Agile Software Development with Scrum.* Englewood Cliffs, NJ: Prentice Hall.

Shore, J. 2004. Continuous Design. *IEEE Software* 21(1): 20–22.

Smith, S., and S. Stoecklin. 2001. What We Can Learn from eXtreme Programming. *The Journal of Computing in Small Colleges* 17(2): 144–151.

Stephens, M., and D. Rosenberg. 2004. The irony of eXtreme programming. *Dr. Dobbs Journal* 29(5): 44–47.

Sutherland, J. 2004. Agile Development: Lessons Learned from the First Scrum. Cutter Agile Project Management Advisory Service. Executive Update, Vol. 5, No. 20.

Sutherland, J. 2005. Future of Scrum: Parallel Pipelining of Sprints in Complex Projects. Agile 2005 Conference, Denver, CO.

Sutherland, J., C. Jakobsen, and K. Johnson. 2008. Scrum and CMMI Level 5: The Magic Potion for Code Warriors, Proceedings of the 41st Hawaii International Conference on System Sciences. January.

Sutherland, J., A. Viktorov, J. Blount, and N. Puntikov. 2007. Distributed Scrum: Agile Project Management with Outsourced Development Teams, Proceedings of the 40th Hawaii Conference on Systems Sciences.

U.S. Strategic Command (USSTRATCOM). US Strategic Command History. www.stratcom.mil/about-ch.html.

Watson, R. T., G. G. Kelly, R. D. Galliers, and J. C. Brancheau. 1997. Key Issues in Information Systems Management: an International Perspective. *Journal of Management Information Systems* 13(4): 91–115.

ABOUT THE AUTHORS

Ann L. Fruhling is an Associate Professor at the Peter Kiewit Institute, College of Information Science and Technology, the University of Nebraska–Omaha (UNO). She teaches core courses for the Management Information Systems graduate program. In 2007, she received the UNO Alumni Outstanding Teaching Award. Dr. Fruhling is a member of the Association for Information Systems (AIS) and serves on the Executive Board of the AIS IT in Healthcare Special Interest Group. In addition, she is a research scholar for Northrop Grumman on the C2SES project located at the U.S. Strategic Command.

Her research areas include agile system development, implementation and management strategies, e-health user interface usability studies, and system design strategies for medical emergency response systems.

Fruhling has published several research articles in the areas of agile system development, emergency response systems, and user interface usability. Her research studies have appeared in publications including *Journal of Management Information Systems, Communications of the Association for Information Systems, Journal of Computer Information Systems, International Journal of Electronic Health Care, International Journal of Cooperative Information Systems*, and *Journal of Electronic Commerce Research*. She also has book chapters in *Value Based Software Engineering, Patient-Centered E-Health*, and *Advances in Management Information Systems* (forthcoming), as well as numerous conference papers.

Since 2002, Fruhling has been the chief principal investigator for a distributed video diagnostics and consultation system for public health laboratories called STATPack™. STATPack™ is used in cases of emergencies and biosecurity-related threats. To date, STATPack™ emergency response systems have been implemented in three states and 48 public health, water, food, and veterinary laboratories. Fruhling's research in emergency response systems has been funded by the Centers for Disease Control and Prevention, the Association of Public Health Laboratories, Nebraska Health and Human Services, the National Aeronautics and Space Administration, and a Nebraska Research Initiative.

Fruhling holds a Ph.D. in management information systems from the University of Nebraska at Lincoln, an MBA from the University of Nebraska at Omaha, and a B.S. in business administration from Colorado State University.

Alvin E. Tarrell is currently a Ph.D. student in information technology at the Peter Kiewit Institute at the University of Nebraska–Omaha (UNO-PKI). His research interests include data visualization, visual computing, knowledge management, and software development methodologies.

In addition, Mr. Tarrell is an operational analyst for Unisys Corporation, supporting the Operations Directorate at the U.S. Strategic Command near Omaha, Nebraska. He serves as software test director and lead systems analyst for the Nuclear Planning and Execution System, a decision support system providing nuclear command and control support to the highest levels of the U.S. government and military.

Prior to this, Tarrell enjoyed a 20-year career in the U.S. Navy, gaining expertise in nuclear reactor operations, strategic weapons operation and maintenance, and nuclear weapons command and control. His final assignment was as chief of strategic operations for the National Airborne Operations Center, an airborne command center supporting the president and secretary of defense in times of national emergency.

Tarrell holds master's degrees in ocean engineering/operational oceanography from the Massachusetts Institute of Technology (MIT) and Woods Hole Oceanographic Institute (WHOI) joint program, a master's degree in civil/environmental engineering from MIT, and a bachelor's degree in chemical engineering from the University of Nebraska–Lincoln. He also expects to complete a master's program in management information systems at UNO-PKI in the summer of 2008.

KEY CONTACT INFORMATION

To contact the authors:

Ann L. Fruhling
Associate Professor
Information Systems and Quantitative Analysis
Peter Kiewit Institute
College of Information Science and Technology
University of Nebraska at Omaha
1110 S. 67th Street
PKI 174 G
Omaha, NE 68182-0500
(402) 554-4968

e-mail: afruhling@unomaha.edu
To learn more about STATPack™, visit: www.statpack.org.

Alvin E. Tarrell
Ph.D. Student, Information Technology
Peter Kiewit Institute
University of Nebraska at Omaha
1110 S. 67th Street
Omaha, NE 68182-0500
(402) 850-0581

e-mail: atarrell@alum.mit.edu

REPORTS from
The IBM Center for The Business of Government

For a full listing of IBM Center publications,
visit the Center's website at *www.businessofgovernment.org*.

Recent reports available on the website include:

Collaboration: Networks and Partnerships

From Forest Fires to Hurricane Katrina: Case Studies of Incident Command Systems by Donald P. Moynihan
A Manager's Guide to Resolving Conflicts in Collaborative Networks by Rosemary O'Leary and Lisa Blomgren Bingham

Contracting

Success Factors for Implementing Shared Services in Government by Timothy J. Burns and Kathryn G. Yeaton

E-Government/Technology

The Blogging Revolution: Government in the Age of Web 2.0 by David C. Wyld
Bridging the Digital Divide for Hard-to-Reach Groups by Heike Boeltzig and Doria Pilling

Human Capital Management

Designing and Implementing Performance-Oriented Payband Systems by James R. Thompson
Managing for Better Performance: Enhancing Federal Performance Management Practices by Howard Risher and Charles H. Fay
Seven Steps of Effective Workforce Planning by Ann Cotten

Innovation

Transforming Government Through Collaborative Innovation by Satish Nambisan

Managing for Performance and Results

Engaging Citizens in Measuring and Reporting Community Conditions: A Manager's Guide by Alfred T. Ho
Strategic Use of Analytics in Government by Thomas H. Davenport and Sirkka L. Jarvenpaa

Missions and Programs

Delivery of Benefits in an Emergency: Lessons from Hurricane Katrina by Thomas H. Stanton

Organizational Transformation

Improving Service Delivery in Government with Lean Six Sigma by John Maleyeff

Presidential Transition

Strengthening Homeland Security: Reforming Planning and Resource Allocation by Cindy Williams

About the IBM Center for The Business of Government

The IBM Center for The Business of Government connects public management research with practice. Since 1998, we have helped public sector executives improve the effectiveness of government with practical ideas and original thinking. We sponsor independent research by top minds in academe and the nonprofit sector, and we create opportunities for dialogue on a broad range of public management topics.

The Center is one of the ways that IBM seeks to advance knowledge on how to improve public sector effectiveness. The IBM Center focuses on the future of the operation and management of the public sector.

About IBM Global Business Services

With consultants and professional staff in more than 160 countries globally, IBM Global Business Services is the world's largest consulting services organization. IBM Global Business Services provides clients with business process and industry expertise, a deep understanding of technology solutions that address specific industry issues, and the ability to design, build and run those solutions in a way that delivers bottom-line business value. For more information visit www.ibm.com.

For additional information, contact:
Jonathan D. Breul
Executive Director
IBM Center for The Business of Government
1301 K Street, NW
Fourth Floor, West Tower
Washington, DC 20005
(202) 515-4504, fax: (202) 515-4375

e-mail: businessofgovernment@us.ibm.com
website: www.businessofgovernment.org