

In-the-SPIN

Newsletter of the Boston@SPIN

Issue 52 April 2003

Editors: Sheila Lynch
Judi Brodman

From the Editor

Greetings Fellow SPINners,

We hope to see you at our annual joint meeting with ASQ on April 30. Our featured speaker that night is Jim Highsmith who will be talking to us about Agile Software Development. You can find more information about this meeting on our website, <http://www.cs.uml.edu/Boston-SPIN/bos-SPIN.html>. His topic is certainly a hot one and an area in which we all have great interest. Jim's article below will just whet your appetite to learn more about this emerging technology.

We are happy to welcome a new Boston SPIN sponsor this month, MKS, Inc. MKS provides award-winning enterprise SCM solutions enabling companies like Northrop Grumman and Park Place Entertainment to better manage and control the complete process of software development. Try an online demo of the MKS Integrity Solution:
<http://www.mks.com/go/spindemo>

In the meantime, please think about running for a position on our SPIN Steering Committee. Elections are coming up at our June meeting. The Steering Committee meets just prior to our meeting each month. Those of us on the Steering Committee all feel that we have benefited, both professionally and socially, by this congenial interaction with our peers. Think about it!!!

Sheila Lynch
In-the-SPIN co-editor



Boston@SPIN Established January 1993
Software Process Improvement Network

IN THIS ISSUE . . .

From the Editor	1
Speaker Spotlight - What is Agile Software Development	1
April Meeting	7
January Synopsis	8
March Synopsis	8
SPIN Information	11
Upcoming Meetings	11

Speaker Spotlight

What is Agile Software Development?

by Jim Highsmith

Jim Highsmith is director of the Cutter Consortium's Agile Project Management Practice, and author of Agile Software Development Ecosystems (2002), and Adaptive Software Development: A Collaborative Approach to Managing Complex Systems (2000), winner of the 2000 Jolt award. He has over 30 years experience as a consultant, software developer, manager, and writer. In the last ten years, he has worked with both IT organizations, industrial product companies, and software companies in the US, Europe, Canada, South Africa, Australia, Japan, India, and New Zealand to help them adapt to the accelerated pace of development in increasingly complex, uncertain environments. Jim will be our featured speaker at the annual ASQ/SPIN dinner meeting on April 30. He can be reached at the following:



jim@jimhighsmith.com
www.jimhighsmith.com
www.cutter.com

The following article is reprinted with Jim's permission. It first appeared in the October 2002 issue of CrossTalk.

Abstract

In the past two years, the ideas of "Agile Software Development," which encompasses such individual methodologies as Crystal Methods, Extreme Programming, Feature-Driven Development, and Adaptive Software Development are being increasingly applied, and causing considerable debate. This article attempts to answer the fundamental question on many people's minds—**What is Agile Software Development?**¹

"Never do anything that is a waste of time—and be prepared to wage long, tedious wars over this principle," declares Michael O'Connor, project manager at Trimble Navigation in Christchurch, New Zealand. This product group at Trimble is typical of the "homegrown" approach to agile methodologies. While interest in agile methodologies has blossomed in the past two years, their roots go back more than a decade. Teams using early versions of Scrum, Dynamic Systems De-

velopment Methodology, and Adaptive Software Development were delivering successful projects in the early to mid 1990s.

This article attempts to answer the question of what constitutes agile software development. Because of the breadth of agile approaches and the people who practice them, it isn't as easy a question to answer as one might expect. I will try to answer the question by first focusing on the "sweet spot" problem domain for agile approaches, then delving into the three dimensions (barely sufficient methodology, collaborative values, and chaordic perspective) of what I refer to as agile "ecosystems," and finally, examining several of these agile "ecosystems."

The Agile Problem Domain: Fitting the Process to the Project

Problems are different. Where is the battlefield CMM? Battlefield commanders plan, but they realize that plans are just a starting place and that probing enemy defenses (creating change) and responding to enemy actions (responding to change) are more important. Battlefield commanders succeed by defeating the enemy (the mission) not conforming to a plan. I can't imagine a battlefield commander saying, "We lost the battle, but by damn, we were successful because we followed our plan to the letter."



Battlefields are messy, turbulent, uncertain, and full of change. Again, could we imagine a battlefield commander saying, "If we just plan this battle long and hard enough and put repeatable processes in place, we can eliminate change early in the battle and not have to deal with it later on?"

A growing number software projects operate in the equivalent of a battle zone—they are extreme projects. This is where agile approaches shine. Project teams operating in this zone are attempting to utilize leading or bleeding edge technologies, respond to erratic requirements changes, and deliver quickly. Projects may have a relatively clear mission, but the specific requirements can be volatile and evolving as customers and development teams alike explore into the unknown. These projects, which I call high exploration-factor projects, don't succumb to rigorous, plan-driven methods.

The critical issues with high exploration factor projects are first identifying those projects; second, managing those projects in a different way; and third, measuring success for those projects differently. Just as winning is the primary measure of success for a battlefield commander, delivering customer value (however the customer defines it) measure success for the agile project manager. Conformance to plan has little meaning in either case. If we want to be agile, we have to reward agility.

What is the critical difference in managing a battle and managing warehouse logistics? The second is a process that can be described by calculations involving material on hand, deliveries, and shipments—managing can be described as a *defined process*, one that involves a relatively high degree of predictability and algorithmic precision. Many manufacturing

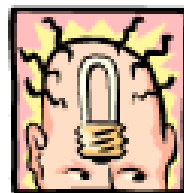
plants operate as defined processes. Battlefields are messy and ever changing. Battle plans are important, but adapting to changing conditions is vital. Battlefields are managed, not by algorithms and predictions, but by constant monitoring of conditions and rapid course alterations—by *empirical processes*. The concepts and assumptions behind defined and empirical processes are fundamentally and irreconcilably different. The practices of agile development—short iterations, continuous testing, self-organizing teams, constant collaboration (daily integration meetings and pair programming for example), and frequent re-planning based on current reality (rather than 6-month old plans) are all geared to the understanding of software development as an empirical process.

On the other hand, the fundamental basis of the CMM and CMMI is a belief in software development as a defined process, one whose tasks can be defined in detail, whose "algorithms" are definable, whose results can be accurately measured, and whose measured variations can be used to refine the processes until they are repeatable within very close tolerances. For projects with any degree of exploration at all, agile developers just don't believe these assumptions are valid. This is a deep, fundamental divide—and not one that can be reconciled to some comfortable middle ground. It is part of having a chaordic perspective on the world as described in the next section.

While agile practices—refactoring, iterative feature-driven cycles, customer focus groups—are applicable to nearly any project, I believe the agile "sweet spot" is this *Exploratory Projects* problem category. The more volatile the requirements and the more experimental the technology, the more agile approaches improve the odds of success.

The Agile Ecosystem: Chaordic, Collaborative and Streamlined

The agile movement covers a broader set of issues than the word *methodology* connotes, so I use the word *ecosystem* to include the three characteristics that define agile development: a "chaordic" perspective, "collaborative" values and principles, and "barely sufficient" methodology. A chaordic (the word, a combination of chaos and order, was coined by Dee Hock, former CEO of Visa International) perspective arises from recognition and acceptance of increasing levels of unpredictability in our turbulent economy. Two concrete ramifications of trying to manage in an unpredictable environment are that while goals are achievable, project details are often unpredictable; and that the foundation of many process-driven approaches, the goal of repeatable processes, is unattainable. In company after company, I've found successful projects that met the customer's vision, but in the end, looked nothing like the original plan. Furthermore, truly repeatable processes would be almost mechanical in nature, and no mechanical process could possibly react to the infinite variety of variations that software projects encounter. The reason many projects attain their goals have little to do with repeatable processes and much to do with the skill and adaptability of the people who are working on the project.



with the skill and adaptability of the people who are working on the project.

Dee Hock, founder and CEO emeritus of Visa International, built Visa using what he calls a “chaordic” management style, balanced mid-way between chaos and order. Hock’s chaordic style is similar to what I call Leadership-Collaboration or adaptive management, which is about creating an environment with the *requisite variety* to meet the challenge of extreme projects, particularly the challenge of high change. Agile managers understand that demanding certainty in the face of uncertainty is dysfunctional. They set goals and constraints, providing boundaries within which creativity and innovation can flourish. They are macromanagers rather than micromanagers.ⁱⁱ

An entrepreneurial, Silicon Valley company has one culture. The Space Shuttle flight avionics team has another. One of the biggest problems in implementing software development methodologies over the years has been the attempted mismatch of culture and methodology. Rather than recognizing the inherent differences between people, project teams, and organizations, we denigrate those who have different cultures by labeling them unprofessional, immature, or undisciplined. Or, conversely, bureaucratic, rigid, and close-minded. For example, you can call a well-oiled XP team a lot of things, but watching them practice test-first development, pair-programming, constant refactoring, and simple design, the last thing you can call them is undisciplined, immature, or unprofessional.

The second characteristic of agile development is “collaborative” values and principles. Agile and rigorous organizations view people, and how to improve their performance, differently. Rigorous methodologies are designed to standardize people to the process, while agile processes are designed to capitalize on each individual’s and each team’s unique



strengths—they adapt the process to the people.ⁱⁱⁱ Agile organizations focus on building individual skills and on fostering a high degree of interaction among team members and the project’s customers. Agilists believe that in today’s complex projects, understanding comes more from face-to-

face interaction than from documentation. Agilists don’t believe that a reliance on heavy processes makes up for lack of skill, talent, and knowledge.

The third aspect of agile ecosystems is the concept of a “barely sufficient” methodology that attempts to answer the question of how much “structure” is enough. To be agile, one must balance flexibility and structure, and “barely sufficient” doesn’t mean insufficient. Bare sufficiency reduces costs through streamlining, but even more importantly, it incorporates the chaordic perspective that creativity and innovation occur in a slightly messy environment, not a primarily structured one. Too many organizations operate on the unspoken assumption that “if a little process is good, then lots of process will be better.”

Concepts drive action and behavior. Software inspections or any other software engineering technique will be implemented differently depending upon



one’s underlying conceptual framework. The underlying conceptual frameworks behind agile development and the CMM are different and will therefore drive organizations to different behaviors. The really important questions are about what kind of core capabilities one’s conceptual foundation leads to. It’s not always an easy question to answer, and organizations will have different answers for different stages in their evolution and for different projects in their portfolio.

An Agile Case Story

Jeff De Luca, project director of Nebulon, an IT consulting firm in Melbourne, Australia, offers an example of an agile methodology’s success using Feature-Driven Development (FDD). De Luca’s project was a complex commercial lending application for a large Singapore bank, utilizing 50 people for 15 months (after a short initialization period). I tracked Jeff down looking for an FDD case story for my book, and subsequently spent several hours on the phone, and exchanged many emails, with Jeff.

FDD arose, in name at least, in the 1997-98 timeframe during this Singapore project. Jeff had been using a streamlined, light-process framework for many years. Peter Coad, brought in to develop the object model for the project, had been advocating very granular, feature-oriented development but hadn’t embedded it in any particular process model. These two threads came together on this project to fashion what was dubbed Feature-Driven Development.

The Singapore lending project had been a colossal failure. Prior to Jeff’s involvement in the project, a large, well-known systems integration firm had spent two years working on the project and finally declared it undoable. Its deliverables: 3,500 pages of use cases, an object model with hundreds of classes, thousands of attributes (but no methods), and, of course, no code. The project—an extensive commercial, corporate, and consumer lending system—incorporated a broad range of lending instruments (from credit cards to large multi-bank corporate loans) and a breadth of lending functions (from prospecting to implementation to back-office monitoring). “The scope was really too big,” said Jeff.

Less than two months into the “new” project, Jeff’s team was producing demonstrable features for the client. The team spent about a month working on the overall object model (the original model and what Jeff refers to as the previous team’s “useless cases” were trashed), and then spent another couple of weeks working on the feature decomposition and short-iteration planning. Finally, to demonstrate the project’s viability to a once-burned and skeptical client, Jeff and his team built a portion of the relationship management application as a proof of concept. From this point on, with about four months elapsed, they staffed to 50 people and delivered approximately 2,000 features in 15 months. The project was completed significantly under budget, and the client, the CEO of the bank, wrote a glowing letter about the success of the project.



As Jeff and I talked, a couple of things struck me about this project. Certainly the FDD process contributed to the pro-

ject's success. When I asked Jeff what made FDD successful, though, his first response was that the overriding assumption behind FDD is that it embraces and accepts software development as a decidedly human activity. The key, Jeff said, is having good people—good domain experts, good developers, good chief programmers. No process makes up for lack of talent and skill.

My guess is that even if the first vendor's staff had used FDD as a process model, they would not have been successful because they just did not have the appropriate level of technical and project management talent. However, had they been using an FDD-like agile process, their inability to complete the project might have surfaced in less than two years. This is a clear example of why working code is the ultimate arbiter of real progress. In the end, thousands of use cases and hundreds of object model elements didn't prove real progress.

A Sampler of Agile Approaches

There are a growing number of agile "methodologies" (or Agile Software Development Ecosystems (ASDEs) as I prefer to label them) and a number of agile "practices" such as Scott Ambler's agile modeling. The core set of these include Lean Development, Adaptive Software Development, Scrum, Extreme Programming, Crystal Methods, Feature-Driven Development, and Dynamic Systems Development Method. Authors of all of these approaches (except LD) participated in writing the Agile Manifesto and so its principles form a common bond among practitioners of these approaches. While individual practices are varied, they fall into six general categories:

Visioning: A good visioning practice helps assure agile projects remain focused on key business values. (Example: ASD's product visioning session)



Project Initiation: A project's overall scope, objectives, constraints, clients, risks, etc. should be briefly documented. (Example: ASD's One-page Project Data Sheet)

Short, iterative, feature-driven, timeboxed development cycles: Exploration should be done in definitive, customer relevant, chunks. (Example: FDD's feature planning)

Constant Feedback: Exploratory processes require constant feedback to stay on track. (Example: Scrum's short daily meetings, XP's Pair Programming)

Customer Involvement: Focusing on business value requires constant interaction between customers and developers. (Example: DSDM's facilitated workshops, ASD's Customer Focus Groups)



Technical Excellence: Creating and maintaining a technically excellent product makes a major contribution to creating business value today and in the future. (Example: XP's refactoring).

Some agile approaches focus more heavily on project management and collaboration practices (ASD, Scrum, DSDM), while others, such as XP, focus on software development practices, although all the approaches touch the six key prac-

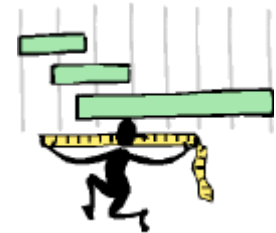
tice areas. The rest of this section delves into four of these approaches, illustrating different aspects of each.

Lean Development (LD)

The most strategy-oriented ASDE is also the least known: Bob Charette's Lean Development, which is derived from the principles of lean production, the restructuring of the Japanese automobile manufacturing industry that occurred in the 1980s. Charette serves as a senior risk advisor to Global 100 CEOs, CFOs, and government officials worldwide on the effectiveness, impacts, rewards, and risks of their information and telecommunications systems. He chaired the IEEE committee for the "IEEE Standard for Software Life Cycle Processes—Risk Management," which was released early in 2001. In LD, Bob extends traditional methodology's view of change as a risk of loss to be controlled with restrictive management practices to a view of change as producing "opportunities" to be pursued using "risk entrepreneurship." LD has been used successfully on a number of large telecommunications projects in Europe.

One-third the time, one-third the budget, one-third the defect rate. These are the goals of Lean Development. While most other ASDEs are tactical in nature, Charette thinks that the major changes required to become Agile must be initiated from the top of the organization. Organizational strategy becomes the context within which Agile processes can operate effectively. Without this strategic piece, Agile development—as all those who have tried to implement ASDEs in organizations can testify—are shunted aside by the organizational forces that seek equilibrium.

LD is the "operational" piece in a three-tiered approach that leads to change-tolerant businesses. It provides a delivery mechanism for implementing "risk entrepreneurship." The key in business, according to Charette, is that the opportunity for competitive advantage comes from being more Agile than competitors in one's market.



LD's risk entrepreneurship enables companies to turn risk into opportunity. Bob defines change tolerance as "the ability of an organization to continue to operate effectively in the face of high market turbulence." A change-tolerant business not only responds to changes in the marketplace, but also causes changes that keep competitors off balance. "Most software systems are not Agile, but fragile," Charette says. "Furthermore, they act as brakes on competitiveness." Every business must deal with change, building a change-tolerant organization that can impose change on competitors.

Charette's work sends several key messages to Agile developers and business stakeholders in IT. First, wide adoption of ASDEs will require strategic selling at senior levels within organizations. Second, the strategic message that will sell ASDEs is the ability to pluck opportunity from fast-moving, high-risk "exploration" situations. And third, proponents of ASDEs must understand, and communicate to their custom-

ers, the risks associated with Agile approaches and, therefore, the situations in which they are and are not appropriate.

LD is as much a management challenge as a set of practices.



Bob comments, “You have to set the bar high enough to force rethinking traditional practices. LD initiatives focus on accelerating the speed of delivering software applications, but not at the expense of higher cost or defect rates. These three objectives need to be achieved concurrently, or it isn’t LD.”

Adaptive Software Development (ASD)

In 1992, I started working on a short-interval, iterative, RAD process that evolved into Adaptive Software Development. The original process, developed in conjunction with colleague Sam Bayer, was used on projects in companies from Wall Street brokerage houses to airlines to telecommunications firms. Over the next several years, Sam and I (together and separately) successfully delivered over 100 projects using these practices. During the early to mid-1990s, I also worked with software companies that were using similar techniques on very large projects.

In the mid-1990s, my interest in complex adaptive systems began to add a conceptual background to the team aspects of the practices and was the catalyst for the name change to Adaptive Software Development. Complexity theory helps us understand unpredictability and that our inability to predict doesn’t imply an inability to make progress. ASD works with change rather than fighting against it. In order to thrive in turbulent environments, we must have practices that embrace and respond to change—practices that are adaptable. Even more importantly, we need people, teams, and organizations that are adaptable and Agile.

The practices of ASD are driven by a belief in continuous adaptation—a different philosophy and a different life cycle—geared to accepting continuous change as the norm. In ASD, the static plan-design-build life cycle is replaced by a dynamic Speculate-Collaborate-Learn life cycle. It is a life cycle dedicated to continuous learning and oriented to change, reevaluation, peering into an uncertain future, and intense collaboration among developers, management, and customers.



A Change-Oriented Life Cycle

A waterfall development life cycle, based on an assumption of a relatively stable business environment, becomes overwhelmed by high change. Planning is one of the most difficult concepts for engineers and managers to reexamine. For those raised on the science of reductionism (reducing everything to its component parts) and the near-religious belief that careful planning followed by rigorous engineering execution produces the desired results (we are in control), the idea that there is no way to “do it right the first time” remains foreign. The word “plan,” when used in most organizations, indicates a reasonably high degree of certainty about the desired result. The implicit and explicit goal of “conformance to plan” restricts a manager’s ability to steer the project in innovative directions.

“Speculate” gives us room to explore, to make clear the realization that we are unsure, to deviate from plans without fear. It doesn’t mean that planning is obsolete, just that planning is acknowledgeably tenuous. It means we have to keep delivery iterations short and encourage iteration. A team that “speculates” doesn’t abandon planning, it acknowledges the reality of uncertainty.



Speculation recognizes the uncertain nature of complex problems and encourages exploration and experimentation. We can finally admit that we don’t know everything.

The second conceptual component of ASD is collaboration. Complex applications are not built, they evolve. Complex applications require that a large volume of information be collected, analyzed, and applied to the problem—a much larger volume than any individual can handle by him- or herself. Although there is always room for improvement, most software developers are reasonably proficient in analysis, programming, testing, and similar skills. But turbulent environments are defined in part by high rates of information flow and diverse knowledge requirements. Building an eCommerce site requires greater diversity of both technology and business knowledge than the typical project of five to ten years ago. In this high-information-flow environment, in which one person or small group can’t possibly “know it all,” collaboration skills (the ability to work jointly to produce results, share knowledge, or make decisions) are paramount.



Once we admit to ourselves that we are fallible, then learning practices—the “Learn” part of the life cycle—become vital for success. We have to test our knowledge constantly, using practices like project retrospectives and customer focus groups. Furthermore, reviews should be done after each iteration rather than waiting until the end of the project.

An ASD life cycle has six basic characteristics: Mission focused, Feature based, Iterative, Time-boxed, Risk driven, and Change tolerant.

For many projects, the requirements may be fuzzy in the beginning, but the overall mission that guides the team is well articulated. A mission provides boundaries rather than a fixed destination. Without a good mission and a constant mission refinement practice, iterative life cycles become oscillating life cycles—swinging back and forth with no progress.

The ASD life cycle focuses on results, not tasks, and the results are identified as application features. Features are the customer functionality that is to be developed during an iteration.

The practice of time-boxing, or setting fixed delivery times for iterations and projects, has been abused by many who use time deadlines incorrectly. Time deadlines used to bludgeon staff into long hours or cutting corners on quality are a form of tyranny; they undermine a collaborative environment. It took several years of managing ASD projects before I realized that time-boxing was minimally about time—it was

really about focusing and forcing hard tradeoff decisions. In an uncertain environment in which change rates are high, there needs to be a periodic forcing function to get work finished.



As in Barry Boehm's spiral development model, the plans for adaptive iterations are driven by analyzing the critical risks. ASD is also change tolerant, not viewing change as a "problem" but seeing the ability to incorporate change as a competitive advantage.

Extreme Programming (XP)

Extreme programming, or XP to most aficionados, was developed by Kent Beck, Ward Cunningham, and Ron Jeffries and has, to date, clearly garnered the most interest of any of the Agile approaches. XP preaches the values of community, simplicity, feedback, and courage, and is defined, in part at least, by its 12 practices—The Planning Game, Small releases, Metaphor, Simple design, Refactoring, Test-First Development, Pair programming, Collective ownership, Continuous integration, 40-hour week, On-site customer, and Coding standards.

There has been so much written about XP's practices that another rehash seems less important than discussing XP's impact on software development. The interest in XP generally comes from the bottom up, from developers and testers tired of burdensome processes, documentation, metrics, and formality. These individuals are not abandoning discipline, but excessive formality which is often mistaken for discipline. They are finding new ways to deliver high-quality

software faster and flexibly. XP, and other agile approaches, are forcing organizations to re-examine whether their processes are adding any value to their organizations. Well over 400 individuals have signed the Agile Manifesto web page (www.agilealliance.com). These individuals reaffirm their



desire to deliver high-quality software without the burdens of bureaucracy.

Other important contributions of XP proponents are their views on reducing the cost of change over software's life and their emphasis on technical excellence through refactoring and test-first development. XP provides a *system* of dynamic practices, whose integrity as a holistic unit has been proven.

Some people think "Extreme" is too extreme, that XP would be more appealing with a more moderate name. I don't think many people would get excited about a book on "Moderate Programming." New markets, new technologies, new ideas aren't forged from moderation, but from radically different ideas and the courage to challenge the status quo. XP has led the way.

Dynamic Systems Development Method (DSDM)

The Dynamic Systems Development Method was developed in the U.K. in the mid-1990s. It is an outgrowth of, and extension to, rapid application development (RAD) practices. DSDM boasts the best-supported training and documentation of any ASDE, at least in Europe. DSDM's nine principles

include active user involvement, frequent delivery, team decision making, integrated testing throughout the project life cycle, and reversible changes in development.

Each of the major phases of the DSDM development process—Functional Model iteration, Design and Build iteration, and Implementation—are themselves iterative processes. DSDM's use of three interacting iterative models and time-boxes can be used to construct very flexible project plans.

The Functional Model iteration is a process of gathering and prototyping functional requirements based on an initial list of prioritized requirements. Nonfunctional requirements are also specified during this phase. The Design and Build iteration refines the prototypes to meet all requirements (functional and nonfunctional) and engineers the software to meet those requirements. One set of business functions (features) may go through both Functional Model and Design and Build iterations during a time-box, and then another set of features goes through the same processes in a second time-box. Implementation deploys the system into a user's environment.

DSDM also addresses other issues common to ASDEs. First, it explicitly states the difference between DSDM and traditional methodologies with respect to flexible requirements. The traditional view, according to the DSDM manual, is that functionality stays relatively fixed (after it is established in the original requirements specifications), while time and resources are allowed to vary. DSDM reverses this viewpoint, allowing the functionality to vary over the life of the project as new things are learned. However, while functionality is allowed to vary, control is maintained by using time-boxes.

DSDM also addresses the issues of documentation, or lack thereof—a constant criticism of ASDEs. Because one of the principles of DSDM relates to the importance of collaboration, it uses prototypes rather than lengthy documents to capture information. DSDM recommends only 15 work products from its five major development phases, and several of these are prototypes. There is an interesting comment in the DSDM white paper on contracting—"The mere presence of a detailed specification may act to the detriment of cooperation between the parties, encouraging both parties to hide behind the specification rather than seeking mutual beneficial solutions."

With respect to work products, DSDM, unlike rigorous methodologies, doesn't offer detailed documentation formats for its 15 defined work products. Instead, the DSDM work product guidelines offer a brief description, a listing of the purposes, and a half-dozen or so quality criteria questions for each work product.



Another area that DSDM focuses on is establishing and managing the proper culture for a project. The manual describes, for example, the different emphasis of project managers and points out how difficult the transition can be for project managers steeped in traditional approaches. A passage from the DSDM manual illustrate this point.

A traditional project manager will normally focus on agreeing a detailed contract with customers about the totality of the system to be delivered along with the costs and time scales. In a DSDM project, the project manager is focused on setting up a collaborative relationship with the customers.

The focal point for a DSDM project manager shifts from the traditional emphasis on tasks and schedules to sustaining progress, getting agreement on requirement priorities, managing customer relationships, and supporting the team culture and motivation.

The Future of Agile Development

There are fundamental shifts driving economies, the structure of products that we build, and the nature of the processes we use to build products. “These changes in products, technologies, firms, and markets are not a passing phenomenon,” according to Carliss Baldwin, Harvard Business School professor and Kim Clark, dean of the Harvard Business School faculty. “These fundamental changes driven by powerful forces deep in the economic system, forces which moreover have been at work for many years. . . . we must be prepared to



dig deep, for the forces that matter are rooted in the very nature of things, and in the processes used to create them.”^{iv}

In the foreword to *Planning Extreme Programming*, Tom DeMarco makes the analogy between military history and software development as each swing from the relative advantages of armor to those of mobility. “In the field of IT,” says DeMarco. “We are just emerging from a time in which armor (process) has been king. And now we are moving into a time when only mobility matters.”^v

Agile development is not defined by a small set of practices and techniques. Agile development defines a strategic capability, a capability to create and respond to change, a capability to balance flexibility and structure, a capability to draw creativity and innovation out of a development team, and a capability to lead organizations through turbulence and uncertainty.

Agile development doesn't abandon structure, but attempts to balance flexibility and structure—trying to figure out that delicate balance between chaos and rigidity. The greater the uncertainty, the faster the pace of change, the lower the probability that success will be found in structure. Plan-driven methodologies have a definite place for some problem domains just as individual practices (configuration management for example) have a definite place in the most agile of software development projects. In a less volatile era, rigorous processes were applicable for a wide range of projects. In an era in which traditional management styles dominated, plan-driven software development approaches fit well.



But as the old Bob Dylan song goes, “Times, they are a-changin.” Volatility and uncertainty increasingly defines today's business, and even today's military environment. Talented technical people want to work in an organization in

which they have more control over how they work and how they interact with peers, customers, and management. Problems are changing, people are changing, ideas are changing. While there are still opportunities for plan-driven style development and management, I believe growth lies in being agile and flexible.

Throughout the last three years, I've used agile methods with project teams in India, Canada, Norway, the U.S., New Zealand, Poland and Australia. These companies are struggling with exploratory projects that run the gamut: an e-commerce infrastructure product, a clinical drug-trial monitoring application, 300,000 lines of embedded C code in a new cell-phone chip, a world-wide financial system product, a myriad of internal IT applications, the complete business system for a dot-com start-up (one that's still in business), and an oil-field geophysical data gathering and analysis system.

These companies want to be more agile: They want to create change for their competitors and respond quickly to market conditions. They plan, but they aren't blinded by those plans. They focus on delivering customer value, not adding up how many processes they have in place. They document, but they don't get lost in piles of paper. They rough out blueprints (models), but they concentrate on creating working software. They focus on individuals and their skills, and on the intense interaction of development team members among themselves and with customers and management. In short, they deliver results in a turbulent, messy, ever-changing, ever-exciting marketplace.



Footnotes:

- ⁱ This article is adapted from Jim Highsmith's book, *Agile Software Development Ecosystems*, Addison Wesley 2002.
- ⁱⁱ For additional information, see James A. Highsmith, *Adaptive Software Development: A Collaborative Approach to Managing Complex Systems*, Dorset House 2000.
- ⁱⁱⁱ For extensive research in this area see, Buckingham, Marcus and Curt Coffman. *First, Break All the Rules*. New York: Simon & Schuster, 1999. *Now, Discover Your Strengths*. New York: Simon & Schuster, 2001.
- ^{iv} Carliss Y. Baldwin and Kim B. Clark, *Design Rules: The Power of Modularity*, The MIT Press, 2000. p. 1-2.
- ^v Beck, Kent and Martin Fowler. *Planning Extreme Programming*, Addison Wesley, 2001.

April Meeting

Annual ASQ-SPIN Dinner Meeting

Featured Speaker – Jim Highsmith

Agile Software Development

In the past several years, a wide range of publications—*Software Development*, *IEEE Software*, *IEEE Computer*, *CIO*

Magazine, Cutter IT Journal, Software Testing and Quality Engineering, and even the *Economist*—have published articles on Agile Software Development. Conferences such as OOPSLA and Software Development have been highlighting *Agile* talks, tutorials, and panels. Is this extreme interest a fad, or not? Although recently publicized, many of these new approaches have been used successfully for nearly a decade. The main players are: Extreme Programming, Scrum, Crystal Methods, Adaptive Software Development, Feature-Driven Development, Lean Development, and Dynamic Systems Development Methodology. Furthermore, scores of organizations have developed their own “lighter” approach to building software.

Agile Ecosystems—which encompass world view, values and principles, social culture and methodology—are hot for two main reasons. First, they better address the problem domain characterized by high-speed, high-change, and high-risk projects. Second, the social culture and principles appeal to many developers and managers. This session will delve into the history, principles, practices, and successes of Agile Software Development.

About the Speaker:

Jim Highsmith is Director, Agile Project Management Practice and Fellow, Business Technology Council at Cutter Consortium; and Member, Software Development Productivity Council, Flashline, Inc. He is the author of *Adaptive Software Development: A Collaborative Approach to Managing Complex Systems*, Dorset House 2000, which won the prestigious Jolt award for product excellence; and, *Agile Software Development Ecosystems*, Addison Wesley 2002.

Jim has 25-plus years experience as a consultant, software developer, manager, and writer. He has published dozens of articles in major industry publications including “The Agile Manifesto,” co-authored with Martin Fowler, in the August 2001 issue of *Software Development* and “Does Agility Work,” in the June 2002 issue of *Software Development*.

In the last dozen years, Jim has worked with IT departments, software companies, and new product development organizations in the U.S., Europe, Canada, South Africa, Australia, Japan, India, and New Zealand to help them adapt to the accelerated pace of development in increasingly complex, uncertain environments.

January Roundtable

Software Process Improvement Forum

Facilitator and Scribe: Judi Brodman, Co-editor of the Boston SPIN's Newsletter, In-the-SPIN; Process Consultant, LOGOS International, Inc.

Because the RT began late, we started discussing Action Planning but then decided to open up the discussion to include issues and items the group wanted to discuss.

Action Planning:

Within the attending group, three people had assessments performed on their organizations.

How do you start Software Process Improvement (SPI)?

First you need to determine where you are. Use an assessment of some type (interviews and checklists against a model). Then, the organization can decide what issues it wants to tackle.

Do you have to be working toward a Level to start SPI?

Organizations can work towards continuous process improvement (no Level). In this case, they work off the priority issues of the organization and develop action plans to solve the problems. In this case they would still have to start with some type of assessment of the organization to determine the problems.

Other SPI Questions asked of the Group:

How long does it take to move up the SEI's SW-CMM levels?



It was recommended that the information be taken off the SEI's web site where they post a maturity profile quarterly. This profile can be downloaded as a .pdf file.

The culture of an organization can impact how quickly or slowly you improve. Items such as non-collated teams can add to the problems – how do you communicate effectively?

How do I introduce SPI and the CMM to a group who wants to cut the Test Group?

It was recommended that you start SPI slowly. Choose your battles carefully. Maybe concentrate on solving the test group problems first and then branch out. Associate with someone who is suffering pain from some problems and start SPI there. Show what not doing the improvements is costing the organization.

Use metrics to determine the root cause of the problems – why are they occurring and where are they occurring.

March Synopsis

Featured Speaker – Howie Dow

Results from Inspecting Test Automation Scripts

Notes by Dolores McCarthy, Quality Manager, Computer Sciences Corporation and Boston SPIN Secretary

This evening's audience was eagerly looking forward to hearing from Howie Dow, an alumni of Carnegie Mellon University, process improvement specialist and change agent, on real life results from inspecting test automation scripts. Howie's energetic and detailed presentation of the process and data from the endeavor was very interesting and informative.

The inspections were performed on a Call Center system using functional test scripts prepared by three test engineers. The goals of the inspections were to produce defect free test scripts and increase the skill level of the team.

The type of inspections performed had some qualities of a Fagan style inspection. The main differences were that some

data was not collected on the amount of time required for preparation and whether everyone had prepared fully, suggestions for fixes were allowed during the inspection, and a manager was allowed to attend, although the manager did diffuse some conflicts among the inspectors.

For purposes of analyzing the results of the inspections, the test engineers' skill levels were rated low, medium, or high against desired and actual levels. However, the ratings were not associated with engineers' names, as the point of an inspection is to focus on the product, not people.



Seventeen test scripts were inspected against the functional requirements by 3-6 inspectors overall. The data collected included the size of the script in LOC, number of defects, location, description, severity, and time to inspect. The inspectors spent 75 hours inspecting 12,949 LOC of test scripts and found 544 defects. Results of the inspections were graphed to reveal a picture of what was happening over time of inspections regarding percent severity of defects, defects per KLOC, and defects found per hour. Also revealing were graphs of the relationships of defect count vs. size (LOC), size vs. defects/KLOC, and defects per KLOC (normalized) by engineer (named A, B, C for anonymity). The graphs showed a need to investigate what was happening with some inconsistent results and outliers. For example, the graphs showed that one engineer's defect rate was decreasing as inspections went on, but others were not. At a later time, it was found that the more successful engineer was methodical, disciplined, and had a process for high-level design, detailed design, and implementation of the test scripts. Others could not describe their method.

Defects found during the inspections were corrected. As a result of the inspections there were no operational defects in the product, it was delivered on time, and the team increased its skill level, as determined subjectively in subsequent projects.

Howie stated there were several conclusions drawn from the inspections. Test scripts can be inspected. Share your results with other testers and developers. Inspections can be a learning tool to increase the skill level of developers. Avoid unrealistic expectations.



The full presentation with graphs and further references on inspections can be found on the Boston SPIN web site <http://www.cs.uml.edu/Boston-SPIN>.

About the speaker:

Howie Dow is a process improvement specialist, change agent and solver of complex problems. His focus is on achieving significant and measurable business results and has specific expertise in improving software development activities.

Previously, disguised as a System-Software Engineer at Enterprise Systems Lab of Hewlett Packard Corporation (formerly Compaq Computer and Digital Equipment), Howie led the definition and implementation of business and technical

process improvements; worked with customers to create appropriate and meaningful test requirements; and has trained and coached software teams in defect reduction strategies, project planning and establishing effective metrics.

Howie has a Master of Software Engineering from Carnegie Mellon University, a Master of Science in Computer Engineering and a BS in Electrical Engineering. He is an ASQ Certified Software Quality Engineer (CSQE) and a practitioner of Six Sigma methods. He has given papers at the International Conference on Software Quality (ICSQ), BOSCON and other conferences.

"You Are What You Measure" Roundtable

Release Criteria: Is This Software Done?

Facilitator and scribe, Ron Torres, Project Manager, Schneider Electric Inc.

This new round table had quite a lively discussion and focused on an article written by our own Johanna Rothman, entitled "Release Criteria: Is your Software Done?" (<http://www.jrothman.com/Papers/releasecriteria.html>). We kicked-off the discussion by posing a simple multiple choice question:

"Release Criteria" is a:

- Mechanism to shift blame from one group to another.
- Documented way to determine a group's importance.
- Item of the development plan that perpetually changes.
- Objective measure of products readiness to ship.

The good news is that half the group passed! We went forward and structured the remainder of the discussion around a general four-step process for establishing release criteria.

Step 1: Analyze and Define

This step begins by examining the question at the very core of a project's existence:

"What is critically important to 'this' project?"

The answer to this depended on your view of the project or the stake you have in its success. In any case, this step must define the success in terms of:

- The problem(s) this project is trying to solve
- How good does the product have to be?
- When do the customers want it?
- Are there particular constraints?
- What do the customer really want?

A lot of the conversation centered on the following example of release criteria:

- All code must compile and build on all platforms
- Zero high priority bugs
- For all open bugs, documentation in release notes and workarounds
- All planned QA tests run, at least X% pass
- Number of open defects decreasing for last three weeks

- Feature Y unit tested by developers, system tested by QA and verified with customer A and B before release
- Ready to release by June 1
- All open defects evaluated by cross-functional team

Step 2: Prioritize and Train

The list above is in no particular order, and that's a big problem. Certainly, every project will have different sensitivities. Some release requirements may not be necessary and at a minimum, each must be weighed against each other to determine a priority order.

One sticky problem the group brought out was to be careful when deploying the draft criteria. Be sensitive to the political landscape and if you need buy-in from those higher up the food chain before you ask for everyone's opinion, you'd better get it.

Step 3: Use and Communicate

Once you have established a baseline for the projects release criteria, be sure to use them at project team meetings and keep them visible. Having the team review their progress against the criteria can really drive home reality. We've all felt this with the release date' requirement ("Oh my look at time!").

Step 4: Evaluate and Document

The final step in the process was to evaluate the success of using the release criteria and be sure to document these in appropriate project artifacts:

- Development Plans
- Team Meeting Minutes
- Test Status Reporting
- Release to Manufacturing

As a group, we came to agree with Johanna Rothman's conclusions: "Use the criteria as you progress through the project and up to the final release. Then you can say, 'Release it!' with pride."



March Book Club

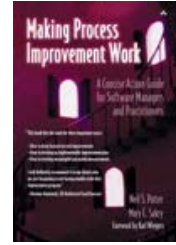
Making Process Improvement Work: A Concise Action Guide for Software Managers and Practitioners

by Neil S. Potter and Mary E. Sakry

Publisher: Addison Wesley Professional, Copyright: 2002, Format: Paper, 192 pp., ISBN: 0-201-77577-8.

The authors of this book, members of the Dallas SPIN, contacted us recently and asked if we'd like to review their recent book. They graciously provided us with several copies of the book – one of which was our raffle prize at the March meeting. Contact one of the SPIN Steering Committee members if you would like to borrow a copy. Or if you would like to purchase it, they have recommended the cheapest way to buy it (35% discount) is via the InformIT web site at http://vig.pearsoned.com/store/product/1,3498_store-1972_isbn-0201775778_type-ALL_editmode-1,00.html.

Software process improvement too often reflects a significant disconnect between theory and practice. This book bridges the gap—offering a straightforward, systematic approach to planning, implementing, and monitoring a process improvement program. Project managers will appreciate the book's concise presentation style and will be able to apply its practical ideas immediately to real-life challenges.



With examples based on the authors' own extensive experience, this book shows how to define goals that directly address the needs of your organization, use improvement models appropriately, and devise a pragmatic action plan. In addition, it reveals valuable strategies for deploying organizational change, and delineates essential metrics for tracking your progress. Appendices provide examples of an action plan, a risk management plan, and a mini-assessment process.

You will learn how to:

- Scope and develop an improvement plan
- Identify and prioritize risks and mitigate anticipated difficulties
- Derive metrics that accurately measure progress toward business goals
- Sell your improvement program in-house
- Initially target practitioners and projects most-open to new approaches and techniques
- Stay focused on goals and problems
- Align the actions of managers and practitioners
- Delay major policy documents and edicts until solutions have been practiced and tested
- Use existing resources to speed deployment
- Incorporate improvement models, such as SEI CMM® and CMMISM, into your improvement program

For those managers who are tired of chronic project difficulties, constant new improvement schemes, and a lack of real progress, this easily digestible volume provides the real-world wisdom you need to realize positive change in your organization.

Who should read this book?

You are probably more than aware of the problems facing your software development organization. The list of problems usually starts with an overwhelming string of commitments and optimistic deadlines. For example, the marketing department has been promised that the product will be shipped by the end of the year. Customers have been told that everything will be delivered on time, and top management has established year-end bonuses based on meeting these dates. Now the programmers are working progressively longer hours, and the system test group is anxiously awaiting the software to begin intensive testing. The technical writers are lost in 300 pull-down menus and cannot get feedback from the programmers. Meanwhile, support engineers are still fixing defects from the previous release and are not optimistic that their lives will improve any time soon.

On top of all this, your group has been signed up to use the new standards and processes developed by corporate engi-

neering. At best, this sounds like just another documentation exercise with little or no positive impact on your group. You have been through numerous improvement programs, each one consuming time, but not providing you with the gains for which you had hoped. The benefits you did see were quickly forgotten in subsequent projects.

Sound familiar? If you have lived in an organization like this for a year or two, you are probably a little tired of the chronic problems, new improvement schemes, and lack of real progress. If you are ready for a straightforward, systematic approach to improvement, read on.

This book is for managers and practitioners. If you are the director of a division, read the book to understand how your group can systematically improve and tie those improvements directly to your business goals. If you are a project or program manager tasked with developing a specific product, use the information to plan, deploy, and track improvements within your team. If you are a process improvement, quality management, or development engineer, apply the techniques in each chapter to coach your team through its improvement journey.

SPIN Information

The Boston SPIN is a forum for the free and open exchange of software process improvement experiences and ideas. Meetings are usually held on third Tuesdays, September - June. Boston SPIN welcomes volunteers and sponsors. There is no charge to attend the meetings. Additional information about the Boston SPIN can be found at our Web home page: <http://www.cs.uml.edu/Boston-SPIN/bos-SPIN.html>.

For more information about our programs and events contact John Britis, Program Chair, jbritis@mitre.org.

Cancelleds (including weather)

Starting at 3pm, we'll notify you via email to the SPIN distribution list, we'll post the notice on the SPIN web page, and we'll send the cancellation announcement to Channel 7 TV and WRKO AM 680.

SPIN Meeting Location

Boston SPIN meetings are held at The MITRE Corporation in Bedford. Please be aware that MITRE has advised us that, due to increased security concerns, you will need a Picture ID for admission to the SPIN meetings. We encourage you to leave all carrying bags, backpacks, and briefcases behind (i.e., in your car). Otherwise, you should be prepared to have these opened and inspected upon arrival.

MITRE's campus is located at 202 Burlington Road (Route 62), Bedford. Directions can be found on our Web site: <http://www.cs.uml.edu/Boston-SPIN/bos-SPIN.html> or on The MITRE Corporation Web site.

Sponsors

The following organizations/individuals support the Boston SPIN:

- The MITRE Corporation, <http://www.mitre.org/>
- Raytheon Company, <http://www.raytheon.com/>
- UMASS – Lowell, <http://www.cs.uml.edu/>
- William George Associates, www.williamgeorgeassociates.com

And please **welcome Our New Sponsor!**

- MKS Inc., <http://www.mks.com>

If your organization is interested in sponsoring Boston SPIN, please contact SPIN Steering Committee member Rick Brenner at rbrenner@chacocanyon.com or ask any Steering Committee member for a brochure.

Email Lists

To receive Boston SPIN specific notices, send an email to:

- Jim Withall, Boston SPIN membership chair, at withall@rcn.com.

Future Programs

We welcome your suggestions for future Boston SPIN programs. We are always looking for interesting speakers. If you'd like to speak at Boston SPIN, please review the criteria specified on the Boston SPIN Web site before sending an abstract to:

- Barb Purchia, Boston SPIN chair, at bpurchia@rational.com.

Newsletter Call for Articles

The *In-the-SPIN Newsletter* is always in need of new and interesting articles dealing with process improvement, software development methodologies, project management and other related subjects that may be of interest to our readership. Please send general correspondence or articles that you would like to have considered for publication to:

- Sheila Lynch, Co-editor of In-the-SPIN, at salynch@mitre.org
- Judi Brodman, Co-editor of In-the-SPIN at brodman@logos-intl.com

Back issues of the *In-the-SPIN Newsletter* can be found on the Boston SPIN Web site: <http://www.cs.uml.edu/Boston-SPIN/bos-SPIN.html>.

Upcoming Meetings

Our remaining 2002—2003 schedule follows.

DATE	SPEAKER	TOPIC
May 20, '03	Linda Northrup, SEI	
June 17, '03	Robin Goldsmith	Non-CMM Software Quality Improvement