What Is Agile Software Development?[1]

Jim Highsmith
Cutter Consortium

In the past two years, the ideas of "agile software development," which encompasses individual methodologies such as Crystal methods, eXtreme Programming, feature-driven development, and adaptive software development, are being increasingly applied and are causing considerable debate. This article attempts to answer the fundamental question on many people's minds: What is agile software development?

"Never do anything that is a waste of time – and be prepared to wage long, tedious wars over this principle," said 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 software development methodologies.

While interest in agile methodologies has blossomed in the past two years, its roots go back more than a decade. Teams using early versions of Scrum, Dynamic Systems Development Methodology (DSDM), and adaptive software development (ASD) were delivering successful projects in the early- to mid-1990s.

This article attempts to answer the question, "What constitutes agile software development?" Because of the breadth of agile approaches and the people who practice them, this is not as easy a question to answer as one might expect. I will try to answer this question by first focusing on the sweet-spot problem domain for agile approaches. Then I will delve into the three dimensions that I refer to as *agile ecosystems*: barely sufficient methodology, collaborative values, and chaordic perspective. Finally, I will examine several of these agile ecosystems.

The Agile Problem Domain: Fitting the Process to the Project

All problems are different and require different strategies. While battlefield commanders plan extensively, they realize that plans are just a beginning; 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 cannot imagine a battlefield commander saying, "We lost the battle, but by golly, we were successful because we followed our plan to the letter." Battlefields are messy, turbulent, uncertain, and full of change. No battlefield commander would say, "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 of 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 attempt to utilize leading or bleeding-edge technologies, respond to erratic requirements changes, and deliver products quickly. Projects may have a relatively clear mission, but the specific requirements can be volatile and evolving as customers and development teams alike explore the unknown. These projects, which I call high-exploration factor projects, do not succumb to rigorous, plan-driven methods.

> *"Projects may have a relatively clear mission, but the specific requirements can be volatile and evolving as customers and development teams alike explore the unknown."*

The critical issues with high-exploration factor projects are as follows: first, identifying them; second, managing them in a different way; and third, measuring their success differently. Just as winning is the primary measure of success for a battlefield commander, delivering customer value (however the customer defines it) measures 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.

There is, however, a critical difference between managing a battle and managing warehouse logistics. Battlefields are managed by constant monitoring of conditions and rapid course alterations – by *empirical processes*. Adapting to changing conditions is vital. Conversely, managing warehouse logistics is a process that can be described by calculations involving materials 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.

The concepts and assumptions behind empirical and defined processes are fundamentally and irreconcilably different. The practices of agile software development – short iterations, continuous testing, self-organizing teams, constant collaboration (daily integration meetings and pair programming for example), and frequent replanning based on current reality (rather than six-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 Capability Maturity Model® (CMM®) and CMM Integration℠ (CMMI℠) is a belief in software development as a defined process. As such, tasks can be defined in detail, *algorithms* can be defined, results can be accurately measured, and 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 do not 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 (meaning a combination of chaos and order as coined by Dee Hock, founder and former CEO of Visa

® Capability Maturity Model and CMM are registered in the U.S. Patent and Trademark Office.
℠ CMM Integration and CMMI are service marks of Carnegie Mellon University.

International) 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 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 have 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 has little to do with repeatable processes and much to do with the skill and adaptability of the people who are working on the project.

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 that provide boundaries within which creativity and innovation can flourish. They are macromanagers rather than micromanagers[2].

While an entrepreneurial Silicon Valley company has one culture, a space shuttle 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, we label them bureaucratic, rigid, and closed-minded. For example, you can call a well-oiled extreme programming team a lot of things, but after 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[3].

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 with today's complex projects, understanding comes more from face-to-face interaction than from documentation. Agilists do not 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 does not 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 to what kind of core capabilities one's conceptual foundation leads. These are not always easy questions 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 information technology 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 De Luca down looking for an FDD case story for my book, and subsequently spent several hours on the phone and exchanged many e-mails with him.

Previously, the Singapore lending project had been a colossal failure. Prior to De Luca'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 included the following: 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 De Luca.

FDD arose, in name at least, in 1997-98 when Nebulon took over the Singapore project. De Luca had been using a streamlined, light-process framework for many years. Peter Coad, who was brought in to develop the object model for the project, had been advocating very granular, feature-oriented development but had not embedded it in any particular process model. These two threads came together on this project to fashion what was dubbed FDD.

Less than two months into the *new* project, De Luca'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 De Luca refers to as the previous team's useless cases were trashed). They 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, De Luca

and his team built a portion of the relationship management application as a proof of concept. From that 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.

While talking with De Luca, a couple of things struck me about this project. Certainly the FDD process contributed to the project's success, but when I asked De Luca what made the FDD successful, his first response was that the overriding assumption behind the FDD is that it embraces and accepts software development as a decidedly human activity. The key, he said, is having good people – good domain experts, good developers, good chief programmers. No process makes up for a 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 a 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 did not 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 [1]. The core set of these includes lean development (LD), ASD, Scrum, eXtreme Programming (XP), Crystal methods, FDD, and DSDM. Authors of all of these approaches (except LD) participated in writing the Agile Software Development Manifesto [2] 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 that agile projects remain focused on key business values (for example, ASD's product visioning session).
- Project initiation. A project's overall scope, objectives, constraints, clients, risks, etc. should be briefly documented (for example, ASD's one-page project data sheet).
- Short, iterative, feature-driven, time-boxed development cycles. Exploration should be done in definitive, customer-relevant chunks (for example, FDD's feature planning).
- Constant feedback. Exploratory processes require constant feedback to stay on track (for example, Scrum's short daily meetings and XP's pair programming).
- Customer involvement. Focusing on business value requires constant interaction between customers and developers (for example, DSDM's facilitated workshops and 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 (for example, XP's refactoring).

Some agile approaches focus more heavily on project management and collaboration practices (ASD, Scrum, and DSDM), while others such as XP focus on software development practices, although all the approaches touch the six key practice areas. The rest of this section delves into four of these approaches, illustrating different aspects of each.

Lean Development

The most strategy-oriented ASDE is also the least known: ITABHI, Inc. President Bob Charette's LD was derived from the principles of lean production used during the restructuring of the Japanese automobile manufacturing industry in the 1980s. In LD, Charette extends traditional methodology's view of change from 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.

The goals of LD are completion in one-third the time, within one-third the budget, and with one-third the defect rate. 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 – is shunted aside by the organizational forces that seek equilibrium.

LD is the *operational* piece in a three-tiered approach[4] 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 the competitors in one's market.

LD's risk entrepreneurship enables companies to turn risk into opportunity. Charette 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," said Charette. "Furthermore, they act as brakes on competitiveness." Every business must deal with change by building a change-tolerant organization that can impose change on competitors.

Charettes's work sends three key messages to agile developers and business stakeholders in information technology. First, the 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 customers 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. Charette said, "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 goals need to be achieved concurrently, or it isn't LD."

Adaptive Software Development

In 1992, I started working on a short interval, iterative, rapid application development process that evolved into ASD. 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. During the next several years, Sam and I (together and separately) successfully delivered more than 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 ASD. Complexity theory helps us understand unpredictability and that our inability to predict does not 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, re-evaluation, 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 re-examine. 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.

Speculation, the first conceptual concept, gives us room to explore, to make clear the realization that we are unsure and to deviate from plans without fear. It does not 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 does not 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 do not 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 is collected, analyzed, and applied to the problem – a much larger volume than any individual can handle by himself 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 e-commerce site requires greater diversity of both technology and business knowledge than the typical project of five to 10 years ago. In this high-information-flow environment, in which one person or small group cannot possibly *know it all*, collaboration skills (the ability to work jointly to produce results, share knowledge, or make decisions) are paramount.

> *"A change-tolerant business not only responds to changes in the marketplace, but also causes changes that keep competitors off balance."*

Once we admit to ourselves that we are fallible, then learning practices – the *learn* part of the life cycle – becomes vital for success. Learning is the third component in the speculate-collaborate-learn life cycle. 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 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 to cut 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 trade-off 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 [3], analyzing the critical risks drives the plans for adaptive iterations. 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, 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, at least in part, 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 that is often mistaken for discipline. They are finding new ways to deliver high-quality software faster and more 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 Software Development Manifesto Web page, available at: <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 during a 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

The DSDM was developed in the United Kingdom in the mid-1990s. It is an outgrowth of, and extension to, rapid application development practices. The DSDM boasts the best-supported training and documentation of any ASDE, at least in Europe.

Each of the major phases of the DSDM development process – functional model iteration, design-and-build iteration, and implementation – are themselves iterative processes. The DSDM's use of three interactive 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.

The DSDM also addresses other issues common to ASDEs. First, it explicitly states the difference between the 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. The 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.

The DSDM also addresses the issues of documentation – or lack thereof – a constant criticism of ASDEs. Because one of the principles of the DSDM relates to the importance of collaboration, it uses prototypes rather than lengthy documents to capture information. The 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. [4]

With respect to work products, the DSDM, unlike rigorous methodologies, does not 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 the 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 illustrates 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. [4]

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. [5]

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. DeMarco says:

> In the field of IT, 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. [6]

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 does not 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, and 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 develop-

ment approaches fit well.

But as Bob Dylan sang, "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, and 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 have used agile methods with project teams in India, Canada, Norway, the United States, New Zealand, Poland, and Australia. Companies in these countries are struggling with exploratory projects that run the gamut, including 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 worldwide financial system product, a myriad of internal IT applications, the complete business system for a dot-com start-up (that is 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 are not blinded by those plans. They focus on delivering customer value, not adding up how many processes they have in place. They document, but they do not 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. They deliver results in a turbulent, messy, ever-changing, ever-exciting marketplace.◆

References

1. Ambler, Scott. Agile Modeling. New York: John Wiley, 2002.
2. AgileAlliance. "Agile Software Development Manifesto." 13 Feb. 2001 <www.agilemanifesto.org>.
3. Boehm, Barry. "A Spiral Model of Software Development Enhancement." IEEE Computer May 1988.
4. DSDM Consortium. Dynamic Systems Development Method. Version 3. United Kingdom <www.dsdm.org>.
5. Baldwin, Carliss Y., and Kim B. Clark. Design Rules – Vol. 1: The Power of Modularity. Cambridge: The MIT Press, 2000.
6. Beck, Kent, and Martin Fowler. Planning eXtreme Programming. Boston: Addison-Wesley, 2001.

Notes

1. This article is adapted from Jim Highsmith's book "Agile Software Development Ecosystems." Addison-Wesley, 2002. (Article quotes and examples taken from this book.)
2. For additional information, see James A. Highsmith. Adaptive Software Development: A Collaborative Approach to Managing Complex Systems. New York: Dorset House, 2000.
3. For extensive research in this area, see Buckingham, Marcus, and Curt Coffman. First, Break All the Rules: What the World's Greatest Managers Do Differently. New York: Simon & Schuster, 1999, and Buckingham, Marcus and Donald O. Clifton. Now, Discover Your Strengths. New York: Simon & Schuster, 2001.
4. The three tiers are Risk Leadership, Risk Entrepreneurship, and Lean Development.

About the Author

Jim Highsmith is director of the Cutter Consortium's Agile Project Management Practice, author of "Agile Software Development Ecosystems (2002)" and "Adaptive Software Development: A Collaborative Approach to Managing Complex Systems (2000)," and winner of the 2000 Jolt Award. He has more than 30 years experience as a consultant, software developer, manager, and writer. In the last 10 years, he has worked with information technology organizations, industrial product companies, and software companies in the United States, 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.

Cutter Consortium
2288 North Coulter Drive
Flagstaff, AZ 86004
Phone: (781) 648-8700
E-mail: jim@jimhighsmith.com

COMING EVENTS

October 14-16
20th Annual Pacific Northwest Software Quality Conference
Portland, OR
www.pnsqc.org

November 3-6
3rd Annual Amplifying Your Effectiveness (AYE) Conference 2002
Phoenix, AZ
www.ayeconference.com

November 4-8
Software Testing Analysis and Review Conference
Anaheim, CA
www.sqe.com/starwest

November 11-14
National Defense Industrial Association
Denver, CO
www.ndia.org

November 18-21
International Conference on Software Process Improvement
Washington, DC
www.software-process-institute.com

February 24-27, 2003
Software Engineering Process Group Conference
Boston, MA
www.sei.cmu.edu/sepg/

April 28-May 1, 2003
Software Technology Conference 2003
Salt Lake City, UT
www.stc-online.org

May 3-10, 2003
International Conference on Software Engineering
Portland, OR
www.icse-conferences.org/2003