Chapter 3
Agility

Agility does not come in a can. One size does not fit all. There are no five common steps to achievement.
—Rick Dove, Response Ability

If turbulence and turmoil define the problem, then agility is key to the solution.

We software developers and high-tech managers often look at ourselves as being in the forefront of innovation. Agile approaches appear, from inside our profession, to be the cutting edge. But if we look across the aisle into the realm of manufacturing, we might be considered on the trailing edge rather than the cutting edge. Witness, for example, the rise of lean manufacturing practices ignited by the Japanese automobile manufacturers and well documented in *The Machine That Changed the World: The Story of Lean Production* (Womack, Jones, and Roos 1990).

Looking back nearly ten years, the foundation of what it means to be agile was described in *Agile Competitors and Virtual Organizations,* by Steven Goldman, Roger Nagel, and Kenneth Preiss—published in 1995! Even though this book addresses manufacturing (the foreword is written by former Chrysler chairman Lee Iacocca), the agility issues it addresses are the same today as they were then.

Bob Charette, originator of Lean Development, stresses that the original concepts behind lean production in Japan have been somewhat blurred in the North American and European implementations, which focus on streamlining and cost control. The Japanese origins of lean production stressed the human and philosophical aspects of the approach. The translation from the Japanese concept to the word "lean" tends to leave out this human-centric flavor, partially, Bob says, because the main driver in U.S.

businesses has been cost reduction. The Japanese word is better translated as "humanware." Whereas lean production and other lean derivatives have focused on reducing staff, the original Japanese concept has more of the Agile concept of working effectively with people.

The definition of agility offered in *Agile Competitors* remains as valid today for software development as it was ten years ago for manufacturing: "Agility . . . is a comprehensive response to the business challenges of profiting from rapidly changing, continually fragmenting, global markets for high-quality, high-performance, customer-configured goods and services." *Agile Competitors* considers the pursuit of agility to be a strategic issue for survival in today's market. Look at the following list, compiled by the authors, of forces that threaten companies:

- Market fragmentation
- Production to order in arbitrary lot sizes
- Information capacity to treat masses of customers as individuals
- Shrinking product lifetimes
- Convergence of physical products and services
- Global production networks
- Simultaneous intercompany cooperation and competition
- Distribution infrastructures for mass customization
- Corporate reorganization frenzy
- Pressure to internalize prevailing social values (Goldman, Nagel, and Preiss 1995)

Would a list drawn up today differ? Not by much. The Information Age economy has increased the pressures on these issues, but they are still critical to business success. And according to *Agile Competitors,* agility is *the* critical characteristic, the overarching skill required—of both corporations and individuals—to address these issues. If software development organizations—internal IT groups, systems integration consultants, and software product companies—are to fulfill their vision of helping business thrive in this Information Age, they must determine how to transform this vision of agility into reality.

Agility

To become Agile, most organizations will need to change their perspective. Peter Drucker, often characterized as the father of modern management theory, wrote an extensive article titled "Management's New Paradigms," which he introduces by saying:

> *Most of our assumptions about business, about technology and organizations are at least 50 years old. They have outlived their time. As a result, we are preaching, teaching, and practicing policies that are increasingly at odds with reality and therefore counterproductive (Drucker 1998).*

Agility isn't a one-shot deal that can be checked off the organizational initiative list. Agility is a way of life, a constantly emerging and changing response to business turbulence. Critics may counter, "Agility is merely waiting for bad things to happen, then responding. It is a fancy name for lack of planning and ad hoc-ism." But Agile organizations still plan; they just understand the limits of planning. Although the defining issue of agility involves creating and responding to change, there are three other components that help define agility: nimbleness and improvisation, conformance to actual, and balancing flexibility and structure.

Creating and Responding to Change

Is agility merely a matter of reacting to stimuli, like an amoeba, or is it something more? My preferred definition of agility has two aspects:

> *Agility is the ability to both* create *and* respond *to change in order to profit in a turbulent business environment.*

Agility is not merely reaction, but also action. First and foremost, Agile organizations create change, change that causes intense pressure on competitors. Creating change requires innovation, the ability to create new knowledge that provides business value. Second, Agile organizations have an ability to react, to respond quickly and effectively to both anticipated and unanticipated changes in the business environment.

Agility means being responsive or flexible within a defined context. Part of the innovative, or proactive, piece involves creating the right context—an adaptable business organization or an adaptable technology architecture, for example. Innovation is understanding that defined context and anticipating when it needs to be altered. The problem with many traditional software development and project management approaches is that they have too narrowly defined the context; that is, they've planned projects to a great level of task detail, leaving very little room for agility to actually happen.

"I think Agile is both being able to respond quickly (which means you have to know that you have to respond quickly—not an easy thing to do) as well as being able to anticipate when the rules or principles are changing or, more importantly, can be changed," says Bob Charette. "Organizations that do this are often called 'lucky'—in the right place at the right time. Some undoubtedly are, while a few have created the necessary infrastructure to exploit the change."

Part of the cost of agility is mistakes, which are caused by having to make decisions in an environment of uncertainty. If we could wait to make product development decisions, they might be better ones; however, the delay could well obviate the need for the decision at all, because aggressive competitors may get their product to market while we dither. "This fact leads to a sort of organizational uncertainty principle: The faster your decision-making cycle, the less assurance you can have that you're making the best possible decision," says David Freedman (2000).

Nimbleness and Improvisation

In our volatile economy, companies need to enhance their "exploration" skills at every level of the organization. Good explorers are Agile explorers—they know how to juggle and improvise. Indiana Jones was a good explorer, somehow living through every outlandish adventure. Agility means quickness, lightness, and nimbleness—the ability to act rapidly, the ability to do the minimum necessary to get a job done, and the ability to adapt to changing conditions. Agility also requires innovation and creativity—the ability to envision new products and new ways of doing business. In particular, IT organizations have not done an adequate job of balancing the needs of exploration and optimization.

The actors in the movie *Crouching Tiger, Hidden Dragon* display incredible agility—running lightly along the tiniest tree branches and showing extraordinary dexterity in sword fighting. Beginning martial arts students are clumsy, not Agile. They become skilled, and Agile, from long hours of training and effective mentoring. Sometimes their drills are repetitive and prescriptive, but only as part of learning.

Agility also requires discipline and skill. A skilled software designer can be more Agile than a beginner because he or she has a better sense of quality. Beginners, with little sense of what is good and what is not, can just revolve in circles—lots of change but not much progress.

"I view the issue as one of almost 'disciplined messiness,'" remarked Bob in an email exchange. "You need to have a very good discipline in place to be able to respond in turbulent times, yet simultaneously know when to be 'undisciplined.' I view anticipation to be actively seeking situations where the generally accepted guiding rules or principles no longer apply, or where shortcuts are the least risky approach to take to gaining some objective. To be able to understand when the rules don't apply, you need to completely understand when they do." For example, Picasso had to become an accomplished fine arts painter to get to a point where he was able to move past that perception of "good art" and create abstract painting. He had to be skilled before he could be Agile.

Agile individuals can improvise, they know the rules and boundaries, but they also know when the problem at hand has moved into uncharted areas. They know how to extend their knowledge into unforeseen realms, to experiment, and to learn. When critical things need to get done, call on the great improvisers.

Improvisation makes great jazz bands. From a few key structural rules, jazz bands improvise extensively. Having a solid foundation enables their tremendous flexibility without allowing the music to degenerate into chaos. The proponents of business process reengineering and software engineering methodologies probably blanch at the thought that improvisation, rather than carefully articulated processes, is key to success. Yet in today's turbulent environment, staff members with good balancing, judging, and improvisational skills are truly invaluable.

Conformance to Actual

"OK," muse the critics, "but how do I *control* development projects in this environment?" There are two answers to this constant question. First, control focuses on boundaries and simple rules rather than prescriptive, detailed procedures and processes. The second aspect of control, though simple in concept, is completely foreign to many managers.

> *Agile projects are* not *controlled by conformance to plan but by conformance to business value.*

"The major problem with planning," say Shona Brown and Kathleen Eisenhardt (1998), "is that plans are virtually always wrong." If we accept the notion of constant change and turbulence, then plans are still useful as guides, but not as control mechanisms. In high-change environments, plans—and the traditional contracts that are derived from plans—are worse than useless as control mechanisms because they tend to punish correct actions. If we deliver a working software product, with features our customers accept, at a high level of quality, within acceptable cost constraints, during the specified time-box, then we have delivered business value. Developers don't get to say, "This is valuable." Customers do. Every three to six weeks, customers tell developers that acceptable business value has been delivered—or not. If it hasn't, the project is cancelled or corrective action is taken. If it has, the project continues for another iterative cycle.

We may look at the plan and say, "Well, because we learned this and this and this during the iteration, we only got 23 of the 28 planned features done." That is useful information for planning our next iteration, but not for control. When we started the project three months ago, we planned only 18 features for this last cycle, and half the features we did deliver were different than the ones in the original plan. We accept that talented people, who are internally motivated, who must work in a volatile environment, who understand the product vision, will do the best they can do. If they don't conform to the plan, the plan was wrong. If they delivered business value, then whether the plan was "right" or not is immaterial. If they conformed to the plan and the customers aren't happy with the delivered business value, it doesn't matter that they conformed to the plan.

An entire generation of project managers has been taught, by leading project management authorities, to succeed at project management by conforming to carefully laid out, detailed task plans. Conformance to plan means locking ourselves into a outdated, often irrelevant plan that some project manager cooked up in great haste (which, of course, precluded talking to developers) 18 months ago, when the world was different. Conformance to plan may get a project manager brownie points with the tightly wound project management authorities, but it won't deliver business value in volatile, high-speed environments.

An exploration perspective contrasts with that of optimization. Take, for example, the use of schedule deadlines. While a schedule may appear to be a schedule, each perspective utilizes dates as a control mechanism in different ways. Optimization cultures use dates to predict and control—they view schedule as achievable, along with the other objectives, and see deviations from the plan as poor performance. Exploration cultures, however, view dates much differently. They basically see dates as a vehicle for managing uncertainty and thereby helping to make necessary scope, schedule, and cost tradeoffs. Exploration cultures, to be sure, use dates as performance targets, but the primary focus is bounding the uncertainty, not predicting dates.

Balancing Flexibility and Structure

It would be very easy to either "plan everything" or "plan nothing," but the really interesting question is how to juggle the two, how to define the context narrowly enough to get something meaningful done, but not so narrowly that we fail to learn and adapt as we go along. The fundamental issue remains one's primary perspective: to anticipate or to depend on resilience and responsiveness. Aaron Wildavsky, a social scientist, writes about this issue and offers an example of the difference between Silicon Valley and Boston. Basically, he believes, the reason Silicon Valley is the primary technology center of the world is that it depends on its resilience to be able to respond to rapid change, whereas the Boston crowd leans more heavily toward anticipation (Postrel 1998). Does that mean no one in Boston ever responds to change and no one in Silicon Valley ever plans? It's not so much

either/or as it is a fundamental, underlying philosophy within which situations are juggled accordingly.

Being Agile means trusting in one's ability to respond more than trusting in one's ability to plan. Good explorers both anticipate when the rules of the game have changed (or are about to change)—that is, they define the context—and operate flexibly within a given rule set. Obviously, if the rule set itself is too prescriptive, it leaves no room for agility. Without some rule set, however, agility can become rudderless reaction.

"Agile" Studies

In Geoffrey Moore's rendition of the technology adoption life cycle, the buying cycle progresses from technology enthusiasts to visionaries to pragmatists to conservatives to skeptics. As he says, "The visionary strategy is to adopt the new technology as a means of capturing a dramatic advantage over competitors who do not adopt it" (Moore 2000). In the main, Agile approaches are still in the technology enthusiast and visionary domains—they haven't penetrated into the mainstream pragmatists' marketplace. Visionaries buy ideas; pragmatists buy proof. Will Agile approaches be the methodology equivalent of Clayton Christensen's disruptive technology (Christensen 1997), or will they become another July 4th rocket, bursting upon the scene, only to fizzle out and return to earth? Realistically, we don't know yet.

Although much of the literature on ASDEs remains anecdotal, there have been several academic studies that have pointed to the efficacy of ASDEs. Laurie Williams, an assistant professor at North Carolina State University, wrote her doctoral dissertation on her study of pair programming. Two other studies, both done at Harvard Business School, provide keen insight into the issues surrounding Agile development, although they are not about Agile development per se.

Product Development in Internet Time

"Now there is proof that the evolutionary approach to software development results in a speedier process and *higher-quality* [emphasis added]

products." This is the tag line from an article in the Winter 2001 issue of the *MIT Sloan Management Review*. The article, "Product-Development Practices that Work: How Internet Companies Build Software," was written by Alan MacCormack, a professor of technology and operations management at Harvard Business School. Much of the material written on Agile methods, iterative development, and other such practices is based on practical experience about what has worked, but MacCormack provides us with explicit research results.

MacCormack and his Harvard Business School colleague Marco Iansiti have been investigating processes that work best in complex, uncertain environments. The question that the research project addressed was, "Does a more evolutionary development process result in better performance?" The study included 20 projects from 17 companies and a panel of experts to evaluate relative "performance" factors between products.

MacCormack writes, "The most striking result to emerge from the research concerned the importance of getting a low-functionality version of the product into customer's hands at the earliest opportunity. The differences in performance are dramatic. That one parameter explains more than one-third of the variation in product quality across the sample—a remarkable result." These are pretty bold statements from an academic researcher. As he points out in the article, there are so many variables that impact performance that finding one that has such a striking impact is rare in research circles.

MacCormack points to four development practices that spell success:

1. An early release of the evolving product design to customers
2. Daily incorporation of new software code and rapid feedback on design changes
3. A team with broad-based experience of shipping multiple projects
4. Major investments in the design of the product architecture

Now, to those who have been practicing Agile techniques for years, these statements may sound ho-hum. However, to those who are just embarking into this arena of exploratory approaches, or to those who are trying to "sell" these approaches to their management, these research findings are significant.

The study found that releasing early versions of products not only increased performance as MacCormack states above, but that the uncertainty

associated with Internet software development "dictates short micro-projects—down to the level of individual features." This finding supports both short, iterative cycles and driving projects by features rather than tasks, as Agilists recommend.

MacCormack's study shows that daily incorporation of new software code and rapid feedback on design changes have a *positive* impact on quality (admitting that there are obviously many factors here that determine quality) and that the quicker the feedback (as in hours, not days), the higher the quality. One of the reasons for this finding may be that if a project has very short feedback cycles, the team is led into continuous, automated testing. "None of the projects with extremely long feedback times (more than 40 hours) had a quality level above the mean," writes MacCormack. On the issue of quality versus feedback time, the study found the highest-quality projects cluster around 2- to 12-hour feedback cycles, with only a couple of data points in the 20- to 40-hour range that were above the mean in terms of quality.

As with any research effort, and when thinking in general about different forms of evolutionary development, understanding the relevant problem domain is critical. MacCormack studied companies operating in complex, uncertain environments—Netscape, Microsoft, Yahoo, and others. To the extent that an organization (or a development project) operates in a slower-paced, more predictable market, some form of evolutionary development may be less critical.

"Heavy" Agile Projects

Isn't "heavy Agile" an oxymoron? I was sitting through Alistair Cockburn's tutorial on designing methodologies at the Software Development 2001 conference when something clicked. I realized that the transition from the label "light" to "Agile" had a secondary benefit; namely, we could now more easily differentiate between small, single-team projects that needed to be Agile due to requirements uncertainty (or other factors) and larger, distributed-team projects that needed to be Agile but also needed additional ceremony (documentation, formality, tools) because of their size.

MacCormack and Iansiti's work at Harvard Business School focused on Internet and software companies whose high-risk profiles are obvious, but

what about larger IT projects? Two other Harvard professors, Rob Austin and Richard Nolan, have been investigating just this issue with respect to major enterprise projects, particularly very large enterprise resource planning (ERP) systems. The title of an in-depth report by Austin indicates their emerging conclusions: "Surviving Enterprise Systems: Adaptive Strategies for Managing Your Largest IT Investments" (Austin 2001).

"As the twenty-first century dawns, we are finally learning to obtain value from these very large IT projects," Austin writes. "The old project approaches do not work in this new space. . . . New and better analogies are based on activities like adaptive software development or new venture investment."

Austin's report first documents several high-profile horror stories. One company, which obviously wanted to remain unnamed, was a major manufacturing company whose ERP installation plan fell apart after spending $30 million, getting board approval for spending $175 million, and then quickly finding out (from the software vendor and the systems integrator) that the price tag would be more like $300 million. After getting board approval, the ERP vendor surprised the company's IT executives by informing them that the "off-the-shelf" package would meet only about 35 percent of their stated requirements.

Dell Computer backed out of an ERP project after spending more than two years and $200 million. The folks at Dell couldn't make the software work (for them); they considered the system too monolithic, and they then opted for a best-of-breed approach.

The survey data in Austin's report came from participants in the Harvard Business School's summer executive education program from the three years 1998 to 2000. This particular program attracts senior IT and general managers from more than 80 companies worldwide. The survey found that up to 88 percent of the respondents (depending upon the year) considered their ERP projects to be large or very large, several responding that they were the largest projects that their companies had ever undertaken.

Besides size, an overwhelming percentage of respondents considered the projects to have considerable technical, organizational, and business risk. The categorization of risk was interesting. Technical risk was defined as the risk of the software's failure to meet business requirements. Organizational risk was defined as the risk that the organization would be unable to make the required changes in order to effectively "use" the software, and

business risk was the risk that implementing the system would actually hurt the company rather than help it. While technical risk concerns declined from 1998 to 2000, concerns over organizational and business risks remained very high. However, in 2000, the survey indicated that companies were starting to achieve the long-anticipated benefits. This led to the second set of study questions: "What are the characteristics of successful projects, and how do they differ from failures?"

Nolan and Austin concluded that there were three dysfunctional elements—in terms of flawed assumptions—in large front-end-loaded projects:

- The first flawed assumption is that it is actually possible to plan such a large project well enough that success is primarily determined by degree of conformance to a plan.
- The second flawed assumption embedded in planning-intensive approaches is that it is possible to protect against late changes to a large system project.
- The third flawed assumption is that it even makes sense to lock in big project decisions early.

"Building a huge new enterprise system is, in many ways, more like building an entirely new venture than it is like managing a traditional IT project," says Austin. Therefore, he and Nolan recommend a staging model, much like venture capitalists use to fund new ventures: spend some money, demand tangible results, spend additional money. Although this "staging" may cost a little extra, it helps companies manage the risk exposure with cost expenditures—much like exploration drilling. Staging can also produce incremental return on investment (ROI). Of the participants surveyed in the 2000 executive education group, 75 percent indicated they used some type of staging strategy. Even more interesting, companies in the planning stage estimated they would need 4 stages; companies in the midst of implementation estimated they would need 7 stages; while companies who had completed implementation said they had taken an average of 12 stages. Tektronix, one of the two in-depth Harvard Business School case studies (the other was Cisco), reported 25 stages, or waves, as it called them.

In the report summary, Austin points to four characteristics of these more successful projects (which I have paraphrased):

- They are all iterative.
- They all rely on fast cycles and insist on frequent delivery.
- They get functionality in some form into business-user hands very early in the project.
- They are preceded by little or no traditional ROI-style analysis of the project as a monolithic whole.

So, in the end, from a strategic perspective, Agile approaches are not about levels of documentation or not using UML or the discipline of pair programming. In the end, Agile approaches are about delivering working products—packaged software, embedded software in a wide range of products, and internal IT products—in environments characterized by high levels of uncertainty and risk. Whether your firm is a dotcom rushing to market or a traditional company slogging through a $50 million ERP system implementation, agility is the key to success.

Agile Software Development Ecosystems

If agility can be characterized as creating and responding to change, nimbleness and improvisation, conformance to actual, and balancing flexibility and structure, then ASDEs should help organizations exhibit these traits. They do so in several ways.

First, ASDEs focus on the set of problems described in Chapter 1—problems characterized by uncertainty and change—which require an exploratory approach. As the degree of uncertainty becomes greater, the probability that an Agile approach will succeed (over a Rigorous approach) increases dramatically, until at some level an Agile approach becomes the only type with a reasonable chance of success. As the level of uncertainty

increases dramatically, it becomes unlikely that any methodology can help. No matter how good an exploration geologist you are, no matter how lucky, the inherent risk of exploring uncharted terrain remains. ASDEs improve the odds, but they don't guarantee success.

Second, ASDEs are intensely customer driven, which is both a characteristic and a risk factor. That is, no customer involvement, no Agile approach. It's basically that simple. But ASDEs advocate something even scarier—actually putting the customer in control. If we drive development from the perspective of value first and traditional scope, schedule, and cost second, then customers must be actively involved.

Third, ASDEs focus on people—both in terms of individual skills and in terms of collaboration, conversations, and communications that enhance flexibility and innovation. As Alistair says, "Software development is a cooperative game." Most traditional "methodologies" place 80 percent of their emphasis on processes, activities, tools, roles, and documents. Agile approaches place 80 percent of their emphasis on ecosystems—people, personalities, collaboration, conversations, and relationships.

Fourth, ASDEs are not about a laundry list of things that development teams should do, they are about the practical things a development team actually needs to do—based on practice. The stories about what methodology manuals contain versus what development teams actually do are legendary. Furthermore, needs change from project to project and from iteration to iteration within a project. Alistair uses the word "embellishment," which means that many methodologies get "embellished" with artifacts, tasks, roles, techniques, and more that creep into the methodology because someone thought they really should be there, even though they themselves could never find the time or inclination to actually do them.

By articulating what ASDEs are and what they are not, people can decide which, if any, of the premises, principles, and practices to use. "Never push Lean Development beyond its limits," is the twelfth principle of Bob's Lean Development. This principle can be extended to all Agile approaches. Every approach to a problem carries the risk of loss. If we don't understand the domain in which some methodology or practice is applicable, if we don't understand the risks associated with a practice, if we don't understand the business risk and opportunity, then we may apply the wrong practice to a particular problem.

One last characteristic of ASDEs—they may be inefficient. Production drilling for oil is about efficiency and cost control. Exploration drilling is about risk management, improvisation, intuition, and occasional lucky guesses. Exploring is often messy, full of fits and starts and rework. Companies that want to explore—to innovate—need to allow for some inefficiency.