Agile Software Development: The Business of Innovation

Jim Highsmith, Cutter Consortium
Alistair Cockburn, Humans and Technology

The rise and fall of the dot-com-driven Internet economy shouldn't distract us from seeing that the business environment continues to change at a dramatically increasing pace. To thrive in this turbulent environment, we must confront the business need for relentless innovation and forge the future workforce culture. Agile software development approaches such as Extreme Programming, Crystal methods, Lean Development, Scrum, Adaptive Software Development (ASD), and others view change from a perspective that mirrors today's turbulent business and technology environment.

THE PROBLEM

In a recent study of more than 200 software development projects, QSM Associates' Michael Mah reported that the researchers couldn't find nearly half of the projects' original plans to measure against. Why? Conforming to plan was no longer the primary goal; instead, satisfying customers—at the time of delivery, not at project initiation—took precedence. In many projects we review, major changes in the requirements, scope, and technology that are outside the development team's control often occur within a project's life span.

Accepting that Barry Boehm's life cycle cost differentials theory—the cost of change grows through the software's development life cycle—remains valid, the question today is not how to stop change early in a project but how to better handle inevitable changes throughout its life cycle.

Agile development combines creative teamwork with an intense focus on effectiveness and maneuverability.

Traditional approaches assumed that if we just tried hard enough, we could anticipate the complete set of requirements early and reduce cost by eliminating change. Today, eliminating change early means being unresponsive to business conditions—in other words, business failure.

Similarly, traditional process management—by continuous measurement, error identification, and process refinements—strove to drive variations out of processes. This approach assumes that variations are the result of errors. Today, while process problems certainly cause some errors, external environmental changes cause critical variations. Because we cannot eliminate these changes, driving down the cost of responding to them is the only viable strategy. Rather than eliminating rework, the new strategy is to reduce its cost.

However, in not just accommodating change, but embracing it, we also must be careful to retain quality. Expectations have grown over the years. The market demands and expects innovative, high-quality software that meets its needs—and soon.

THE AGILE RESPONSE

Agile methods are a response to this expectation. Their strategy is to reduce the cost of change throughout a project. Extreme Programming (XP), for example, calls for the software development team to

- produce the first delivery in weeks, to achieve an early win and rapid feedback;
- invent simple solutions, so there is less to change and making those changes is easier;
- improve design quality continually, making the next story less costly to implement; and
- test constantly, for earlier, less expensive, defect detection.

Agile software development stresses quality in design. These methods are sometimes confused with ad hoc or cowboy coding because the design is done on an ongoing basis, in smaller chunks, as opposed to all at once and up front. Each agile method addresses quality in certain ways. For example, Dynamic Systems Development Methodology (DSDM) calls for a series of prototypes to attack unstable or unknown areas: new technology, new business rules, and user interface design. Scrum uses intense 15-minute daily meetings and comprehensive iteration reviews at the end of each 30-day iteration.

Basic principles

Agile methods stress two concepts: the unforgiving honesty of working code and the effectiveness of people working together with goodwill.

Working code tells the developers and sponsors what they really have in front of them—as opposed to promises as to what they *will* have in front of them. The working code can be shipped, modified, or scrapped, but it is always *real*.

Using people effectively achieves maneuverability, speed, and cost savings. People can transfer ideas faster by talking face to face than by writing and reading documents. A few designers sitting together can produce a better design than each could produce alone. When developers talk with customers and sponsors, they can iron out difficulties, adjust priorities, and examine alternate paths forward in ways not possible when they are not working together.

Agile Software Manifesto

In recognition of these ideas, in February 2001, we joined 15 other people representing XP, Scrum, DSDM, ASD, Crystal, Feature-Driven Development, pragmatic programming, and others sympathetic to the need for alternative software development methods in signing the Manifesto for Agile Software Development. We wrote:

> We are uncovering better ways of developing software by doing it and helping others do it. Through this work we have come to value
>
> - *individuals and interactions* over processes and tools,
> - *working software* over comprehensive documentation,
> - *customer collaboration* over contract negotiation,
> - *responding to change* over following a plan.
>
> That is, while there is value in the items on the right, we value the items on the left more.

Processes, tools, documentation, contracts, and plans are useful. But when push comes to shove—and it usually does—something must give, and we need to be clear about what stays and what gives.

Relying on interactions between individuals facilitates sharing information and changing the process quickly when it needs changing. Using working software allows us to measure how fast we actually produce results and provides quick feedback. Frequent interaction between individuals compensates for minimizing documentation.

In a complex adaptive system, decentralized, independent individuals interact to create innovative, emergent results.

Customer collaboration means that all players—the sponsor, customer, user, and developer—are on the same team. Merging their different experiences and expertise with goodwill allows the combined group to change directions quickly so they can produce more appropriate results and less expensive designs. Contracts or project charters with the customers are necessary, but without collaboration, they are insufficient.

Working through producing a plan drives the team members to think through their project and its contingencies. The plan itself usually goes out of date within just a few days. Afterward, rather than focusing on the outdated plan, it is important to deal with the changing realities.

GENERATIVE RULES

One aspect of agile development is often missed or glossed over: a world view that organizations are complex adaptive systems. A complex adaptive system is one in which decentralized, independent individuals interact in self-organizing ways, guided by a set of simple, generative rules, to create innovative, emergent results. XP's 12 practices, for example, were never intended to be all-inclusive rules; instead, they are generative rules that interact in concert when a team of individuals practices them.

Most methodologies provide inclusive rules—all the things you could possibly do under all situations. Agile methods offer generative rules—a minimum set of things you must do under all situations to generate appropriate practices for special situations. Teams that follow inclusive rules depend on someone else to name in advance the practices and conditions for every situation. This obviously breaks down quickly. A team that follows generative rules depends on individuals and their creativity to find ways to solve problems as they arise. Creativity, not voluminous written rules, is the only way to manage complex software development problems and diverse situations.

AGILE PRACTICES

A team isn't agile if the feedback loop with customers and management is six months. Agile approaches recommend short iterations in the two- to six-week range during which the team makes constant trade-off decisions and adjusts to new information. XP and Scrum have more directed cycles—two to three weeks for XP, 30 days for Scrum; other methods such as Crystal and ASD tolerate more variation.

Feature planning and dynamic prioritization

Agile approaches combine these short iterative cycles with feature planning and dynamic prioritization. XP uses story cards; Scrum uses the term "backlog"; ASD and Feature-Driven Development refer to features. The key point is that agile approaches plan features, not tasks, as the first priority because features are what customers understand.

Dynamic prioritization means that at the end of an iteration, the customer can reprioritize the features desired in the next cycle, discarding originally planned features and adding new ones. Scrum explicitly states that priorities can only change at the end of an iteration, not during one. DSDM uses "MoSCoW" rules for features—Must have, Should have, Could have, Want to have sometime. XP's priority scheme is binary—in this cycle, or not.

Feedback and change

Because they are most applicable to turbulent, high-change environments, agile approaches recommend a variety of practices for constant feedback on technical decisions, customer requirements, and management constraints. XP advocates pair programming for feedback, and DSDM features short-cycle user prototyping. Crystal and ASD advocate end-of-iteration process and team reviews. ASD and Scrum use end-of-iteration reviews with customer focus groups.

Agile practices encourage change rather than discourage it. In turbulent business situations, a methodology's change tolerance must be geared to the change rate of a specific environment, not some internal view of how much change is acceptable. For example, changes to feature priorities and requirements are handled within the context of a team and the customer partners unless the changes violate the broad scope, schedule, and cost constraints set by the purchasing customer (or management).

Focus on teamwork

Team proximity and intense interaction between team members are hallmarks of all agile methods. XP is noted for pair programming, although the practice has been around for years under other names. Crystal, Scrum, and ASD advocate close collaboration practices including barrier-free collocated teams. Lean Development stresses team interaction.

Using agile development methods requires close customer partnerships. If the customers, either internal department representatives or marketing product managers, don't have a good sense of direction and wander around in strange patterns, agile developers will follow them (with occasional admonitions, of course). Poor customers result in poor systems.

In 1995, Steven L. Goldman, Roger N. Nagel, and Kenneth Preiss, the authors of *Agile Competitors and Virtual Organizations* (Van Nostrand Reinhold, New York), offered this definition of agility:

> Agility is dynamic, context-specific, aggressively change embracing, and growth-oriented. It is not about improving efficiency, cutting costs, or battening down the business hatches to ride out fearsome competitive "storms." It is about succeeding and about winning: about succeeding in emerging competitive arenas, and about winning profits, market share, and customers in the very center of the competitive storms many companies now fear.

This book was about manufacturing, but the definition of agility applies equally to today's software development environment.

Agility, ultimately, is about creating and responding to change. What is new about agile methods is not the practices they use, but their recognition of people as the primary drivers of project success, coupled with an intense focus on effectiveness and maneuverability. This yields a new combination of values and principles that define an "agile" world view.

Agile software development addresses two pressures that characterize today's business and technology world: the need for dynamic, innovative approaches and the desire to build workplaces that aren't described in Dilbert cartoons. ✷

Jim Highsmith *is director of Cutter Consortium's e-Project Management Practice. Contact him at jimh@adaptivesd.com.*

Alistair Cockburn *is a Consulting Fellow at Humans and Technology. Contact him at arc@acm.org.*

Editor: Barry Boehm, Computer Science Department, University of Southern California, Los Angeles, CA 90089; boehm@sunset.usc.edu