Chapter 23. Adaptive Software Development

Collaboration is difficult, especially when it involves other people.

—Ken Orr (Cutter Consortium Summit 2001)

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 to assist in marketing a mainframe RAD tool. Sam and I worked with prospects on pilot projects—one-month projects with one-week iterations—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 more than 100 projects using these practices, and in June 1994, we published an article on our experiences (Bayer and Highsmith 1994). During the early to mid-1990s, I also worked with software companies that were using similar techniques on very large projects, while Sam continued to evolve the practices in his work.

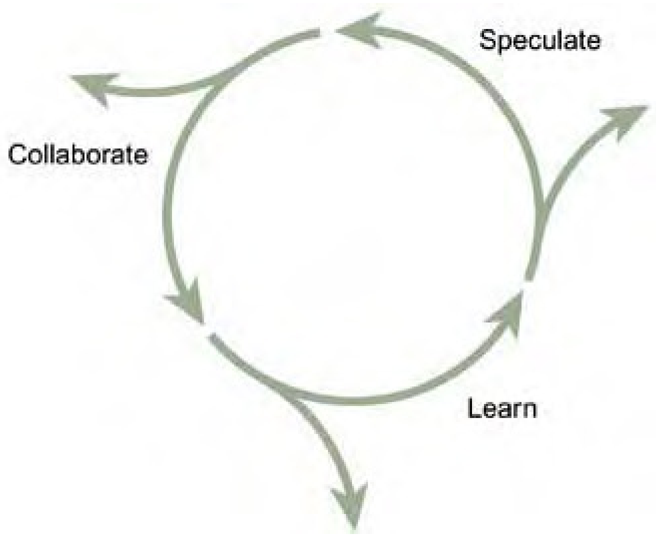
In the mid-1990s, my interest in complex adaptive systems began to add a conceptual background to the team aspects of the practices and was the catalyst for the name change from RADical Software Development to Adaptive Software Development (Highsmith 1997). ASD has been used by companies from New Zealand to Canada for a wide range of project and product types. Interestingly, I became aware of XP just a month prior to the publication of *Adaptive Software Development* (Highsmith 2000), when Kent and I exchanged emails. *Extreme Programming Explained* (Beck 2000) and *Adaptive Software Development* were published within a couple of months of each other.

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 important, we need people, teams, and organizations that are adaptable and Agile. Agile practices alone are not nearly enough; they depend on competent individuals who are nimble and thoughtful. We have become so enamored of precise planning, we forget that products evolve from a little planning and a lot of learning as we proceed.

For all the many books on requirements engineering, the best way to determine how a product should evolve is to use it. Iteration—building, trying, succeeding, failing, rebuilding—governs successful product development, particularly in extremely competitive environments. "Good enough" requirements need to be followed by quick delivery and use, and then followed with evolutionary changes to the requirements and the product based on that use. Although a number of modern software development life cycles have adopted an iterative approach, they still miss the mark in dealing with the messiness of complex environments. Despite the fact that development is iterative, many people's fundamental assumptions are still deterministic—they think of short waterfall life cycles strung together.

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 (see Figure 23.1). 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.

Figure 23.1. The Speculate-Collaborate-Learn Life Cycle



A Change-Oriented Life Cycle[1]

[1] Some material in this chapter has been adapted from the article "Retiring Lifecycle Dinosaurs" (Highsmith 2000a).

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:

1. Mission focused
2. Feature based
3. Iterative
4. Time-boxed
5. Risk driven
6. Change tolerant

For many projects, the requirements may be fuzzy in the beginning, but the overall mission that guides the team is well articulated. (Jens Coldeway's insurance project discussed in Chapter 11 is a good example of this.) Mission statements act as guides that encourage exploration in the beginning but narrow in focus over the course of a project. 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. Mission statements (and the discussions leading to those statements) provide direction and criteria for making critical project tradeoff decisions.

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. While documents (for example, a data model) may be defined as deliverables, they are always secondary to a software feature that provides direct results to a customer. (A customer-oriented document such as a user manual is also a feature.) Features may evolve over several iterations as customers provide feedback.

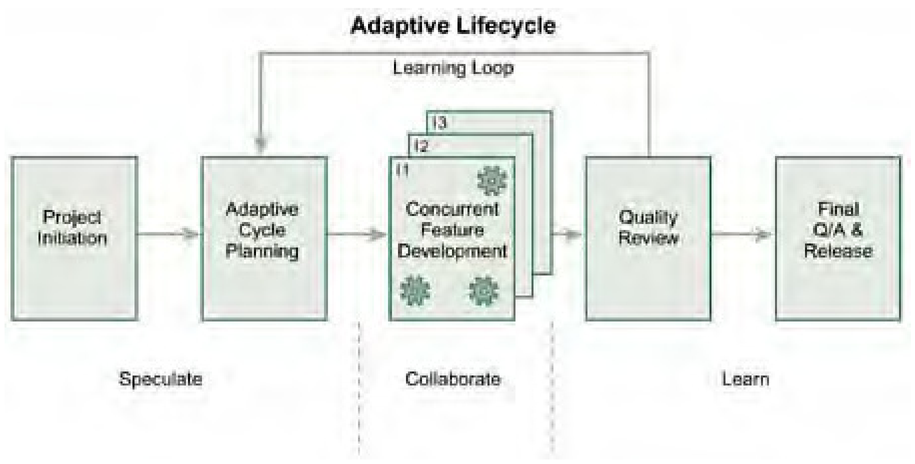
The practice of time-boxing, or setting fixed delivery times for iterations and projects, has been abused by many who use time deadlines incorrectly. Time deadlines used to bludgeon staff into long hours or cutting corners on quality are a form of tyranny; they undermine a collaborative environment. 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.

The Basic ASD Life Cycle

Figure 23.2 shows an expansion of the ASD life cycle phases into specific practices.

Figure 23.2. The Adaptive Life Cycle Phases



Speculate: Initiation and Planning

There are five general steps in "speculating," although the word "steps" is somewhat of a misnomer, as each step may be revised several times during the initiation and planning phase. First, project initiation involves setting the project's mission and objectives, understanding constraints, establishing the project organization, identifying and outlining requirements, making initial size and scope estimates, and identifying key project risks. Because speed is usually a major consideration in using ASD, much of the project initiation data should be gathered in a preliminary JAD session. Initiation can be completed in a concentrated two- to five-day effort for a small- to medium-sized project or take two or three weeks for larger projects. During the JAD sessions, requirements are gathered in enough detail to identify features and establish a skeletal object, data, or other architectural model.

Next, the time-box for the entire project is established based on the scope, feature set requirements, estimates, and resource availability that result from project initiation work. Speculating doesn't abandon estimating, it just means accepting that estimates are tenuous.

The third step is to decide on the number of iterations and assign a time-box to each one. For a small- to medium-sized application, iterations usually vary from four to eight weeks. Some projects work best with two-week iterations, while others might require more than eight weeks (although this is rare). The overall project size and the degree of uncertainty are two factors that determine individual iteration lengths.

After establishing the number of iterations and a schedule for each, the team members develop a theme or objective for each of the iterations. Just as it is important to establish an overall project objective, each iteration should have its own theme (this is similar to the Sprint Goal in Scrum). Each iteration delivers a demonstrable set of features to a customer review process, making the product visible to the customer. Within the iterations, "builds" deliver working features to a daily (or more frequent) integration process, making the product visible to the development team. Testing is an ongoing, integral part of feature development—not an activity tacked on at the end.

Developers and customers assign features to each iteration. The most important criterion for feature assignment is that every iteration must deliver a visible, tangible set of features to the customer. In the assignment process, customers decide on feature prioritization, using feature estimates, risks, and dependency information supplied by the development team. A spreadsheet is an effective tool for feature-based iteration planning. Experience has shown that this type of planning—done as a team rather than by the project manager—provides better understanding of the project than a traditional task-based approach. Feature-based planning reflects the uniqueness of each project.

Collaborate: Concurrent Feature Development

While the technical team delivers working software, project managers facilitate collaboration and concurrent development activities. For projects involving distributed teams, varying alliance partners, and broad-based knowledge, how people interact and how they manage interdependencies are vital issues. For smaller projects in which team members work in physical proximity, collaboration can consist of informal hallway chats and whiteboard scribbling. Larger projects, however, require additional practices, collaboration tools, and project manager interaction.

Collaboration, an act of shared creation, is fostered by trust and respect. Shared creation encompasses the development team, customers, outside consultants, and vendors. Teams must collaborate on technical problems, business requirements, and rapid decision making.

Learn: Quality Review

Learning becomes increasingly difficult in environments in which the "get it right the first time" mantra dominates and development progresses in a linear, waterfall fashion. If people are continually compelled to get it right, they won't experiment and learn. In waterfall development, each phase completion discourages backtracking because there shouldn't be any mistakes. Learning from mistakes and experimentation requires that team members share partially completed code and artifacts early, in order to find mistakes, learn from them, and reduce the total amount of rework by finding small problems before they become large ones. Teams must learn to differentiate between shoddy work and half-done work.

There are four general categories of things to learn about at the end of each development iteration:

- Result quality from the customer's perspective
- Result quality from a technical perspective
- The functioning of the delivery team and the practices team members are utilizing
- The project's status

Getting feedback from the customers is the first priority in Adaptive projects. ASD's recommended practice for this is a customer focus group. Derived from the concept of marketing focus groups, these sessions are designed to explore a working model of the application and record customer change requests. They are facilitated sessions, similar to JAD sessions, but rather than generating requirements or defining project plans, customer focus groups are designed to review the application itself. Customers relate best to working software, not documents or diagrams.

The second review area is technical quality. A standard practice for technical quality assessment is periodic technical reviews; pair programming accomplishes a similar result. Although code reviews or pair programming should be continuous, other reviews, such as an overall technical architecture review, may be conducted weekly or at the end of an iteration.

The third feedback area is for the team to monitor its own performance. This might be called the people and process review. End-of-iteration mini-retrospectives help determine what's not working, what the team needs to do more of, and what the team needs to do less of. Retrospectives encourage teams to learn about themselves and how they work together.

The fourth category of review is project status. This leads into a replanning effort for the next iteration. The basic status review questions are: Where is the project? Where is it versus the plans? Where should it be? Determining a project's status is different in a feature-based approach. In a waterfall life cycle, completed deliverables mark the end of each major phase (a complete requirements document, for example, marks the end of the specification phase). In a feature-based approach, completed features—working software—mark the end of each iteration.

The last of the status questions is particularly important: Where "should" the project be? Since the plans are understood to be speculative, measurement against them is insufficient to establish progress. The project team and customers need to continually ask, "What have we learned so far, and does it change our perspective on where we need to go?"

Leadership-Collaboration Management

Many companies are steeped in a tradition of optimization, efficiency, predictability, control, rigor, and process improvement. The emerging Information Age economy requires adaptability, speed, collaboration, improvisation, flexibility, innovation, and suppleness. "Successful firms in fiercely competitive and unpredictably shifting industries pursue a competing on the edge strategy," write Shona Brown and Kathleen Eisenhardt. "The goal of this strategy is not efficiency or optimality in the usual sense. Rather, the goal is flexibility—that is, adaptation to current change and evolution over time, resilience in the face of setbacks, and the ability to locate the constantly changing sources of advantage. Ultimately, it means engaging in continual reinvention" (Brown and Eisenhardt 1998).

What we name things is important. Traditional management has been characterized by the term Command-Control. It is reminiscent of the traditions of the military, Fredrick Taylor's scientific management, and William Whyte's "Organizational Man" of the 1950s. While new terms have spewed forth from the pages of the *Harvard Business Review* and management books—empowerment, participative management, learning organization, human-centered management, and the like—none have encompassed the breadth, or depth, necessary for managing modern organizations. One alternative to Command-Control management is Dee Hock's "chaordic" style of management (Hock 1999), which I've called Leadership-Collaboration in ASD. It applies to both project and organizational management.

Today, we need leaders more than commanders. "Commanders know the objective; leaders grasp the direction. Commanders dictate; leaders influence. Controllers demand; collaborators facilitate. Controllers micro-manage; collaborators macro-manage. Managers who embrace the Leadership-Collaboration model understand their primary role is to set direction, to provide guidance, and to facilitate connecting people and teams" (Highsmith 2000). Leaders understand that occasionally they need to command, but that's not their predominant style. Leaders provide direction and create environments where talented people can be innovative, creative, and make effective decisions. The Leadership-Collaboration model encompasses the basic philosophical assertion that in turbulent environments, "adaptation" is more important than "optimization."

However, becoming adaptive is not easy. The shift from optimization to adaptation—for an organization or a project team—requires a profound cultural shift. First, we must let go of our need to be tidy and orderly. Of course there are places where orderliness is necessary, but in general, the business world's pace of change precludes it. We need practices such as risk management and configuration control, but to assemble hundreds of practices under the roof of orderliness and statistically controlled rigor ignores the real world.

My guideline for "barely sufficient" rigor in complex projects is "employ slightly less than just enough." Why? Complex problems—those characterized by high speed, high change, and uncertainty—are solved by creativity, innovation, good problem solving, and effective decision making. All of these capabilities suffer from an emphasis on rigorous, optimizing processes. To stabilize a process (make it repeatable), we endeavor to restrict inputs, transfer only the necessary information between processes (for efficiency), and strive to make the transformation as algorithmic as possible. But complex problems in today's organizations require the interaction of many people, diverse information, out-of-the-box thinking, quick reaction, and, yes, rigorous activity at times. Balancing at the edge of chaos provides just enough rigor to keep from plunging into chaos, but not enough to quell creative juices. Chaos is easy—just do it. Stability is easy—just follow the steps. Balancing is hard—it requires enormous managerial and leadership talent.

There is a subtle difference between adapting and changing. Adapting is outwardly focused; it depends on a free flow of information from the environment (one's market) and realistic decisions based upon information and organizational values (goals). Many changes in organizations are in response to internal politics rather than external information. Adaptive organizations learn from the outside in rather from the inside out.

Adaptation needs to happen at all organizational levels, which means free-flowing information and distributed decision making. It means that the traditional hierarchical management structure must share power with the horizontal, networked team structure. Hierarchical power is stabilizing; networked power is more adapting. Too much of one type of power breeds stagnation, while too much of the other breeds chaos. Sharing power is another fundamental characteristic of adap table organizations.

In organizations, power defines who gets to make decisions. Over time, the pattern of decisions an organization makes determines success or failure. Control-oriented managers hoard decision making. New-age managers empower others to make decisions, often diffusing power so much that little gets done. Adaptive managers understand that making good decisions is more important than prescribing activities; further more, they understand when to make decisions and when to distribute them. It is a constant, delicate balancing act that depends on understanding patterns that work rather than processes that prescribe.

Optimizing cultures tend to see the world as black or white. Adaptive cultures, on the other hand, recognize gray. They understand that planning is tenuous and control nearly impossible. Adaptive cultures understand that success evolves from trying a succession of different alternatives and learning from success and failure. Adaptive cultures understand that learning about what we don't know is often more important than doing things we already know how to do. The traditional, linear, Newtonian cause-and-effect perspective no longer adequately models the nature of our world. The concepts of complex systems, and the Leadership-Collaboration ideas about organizational and project management derived from them, provide a better sense of our economic environment.

ASD's Contributions

My hope for ASD is that it contributes to the discussion about the perspective, values, principles, and practices of Agile Software Development and, more critically, how project teams and organizations need to respond to our change-driven economy.