Introducing An Agile Process to an Organization

by Mike Cohn & Doris Ford •

originally published in IEEE Computer on 2003-06-01

In this article I want to discuss introducing an agile process to an organization. Since the publication of Kent Beck's *Extreme Programming Explained*,[1] agile processes have grown increasingly popular. Agile processes allow for changing requirements throughout the development cycle and stress collaboration between software developers and customers and early product delivery.

The "Agile Manifesto" establishes a common framework for these processes: Value individuals and interactions over processes and tools, working software over comprehensive documentation, customer collaboration over contract negotiation, and responding to change over following a plan.[2] The processes most commonly considered agile include Extreme Programming (XP),[1] Lean Development,[3] Crystal,[4] and Scrum.[5-6]

In Scrum, projects progress in a series of month-long iterations called sprints. Development teams meet with the client, or *product owner*, before each sprint to prioritize the work to be done and select the tasks the team can complete in the upcoming sprint.

During the sprint, the team stays on track by holding brief daily meetings. At the end of each sprint, the team delivers a potentially shippable product increment. Scrum is ideally suited for projects with rapidly changing or highly emergent requirements such as Web projects or product development for new markets.

Over the past four years, we have introduced Scrum to seven organizations in four states. Some of the projects were complex and involved distributed teams. Others were straightforward and involved small, co-located teams. However, even the simple projects reached across many departments or functional areas. A failure to sell the agile process change to any one area can negatively impact the project's outcome.

Through trial and error, we have discovered several approaches for successfully introducing an agile process to organizations.

DEVELOPERS

Most developers respond to the proposed introduction of an agile process with the appropriate combination of skepticism, enthusiasm, and cautious optimism. Other developers, however, either resist the change or overzealously jump into the project without enough forethought. Either reaction can cause problems.

Resistance

In general, agile processes value code production more than plan-driven processes do. In a plan-driven process, developers might treat Unified Modeling Language designs and other noncode items as first-class artifacts. In an agile process, however, these items usually exist only to support the coding activity.

While introducing Scrum to various project teams, we invariably found programmers who enjoy producing noncode artifacts far more than they are willing to admit. We also encountered programmers who measure their contribution to a project by the number of meetings they attend. These programmers go beyond “analysis paralysis” and actively seek opportunities to add formalized tasks back into an agile process. One programmer, for example, created a formal document type and attempted to coerce his peers into generating the document for hundreds of cases when it was called for in perhaps a dozen.

In such situations, it is best not to intervene. The other team members can quickly assess the value of these activities and will not adopt them if they do not help the overall development effort.

Micromanagement

A surprising number of developers view using agile processes as an attempt to micromanage. Because approaches like Scrum and XP accelerate project cycles, developers typically interact with their managers more frequently but for shorter periods. In a plan-driven process, a manager might go a full week without talking with a particular developer, but daily contact is the norm in most agile processes.

Developers who view agile processes as micromanagement tend to perceive interactions with

their project managers as being about due dates and missed deadlines. To avoid this problem, project managers should constantly demonstrate their desire to remove obstacles as quickly as possible and not complain if a task takes too long. Managers can be surprised, but should not be judgmental, when told that a task will take longer than originally thought.

Transitioning from a heavyweight process

Some developers we encountered preferred heavyweight plan-driven processes because they believe they looked better on a resume. Because these individuals do not have deeply held convictions about the process's value, they can eventually be swayed by their coworkers' opinions and actions.

A gradual transition from a heavyweight to an agile process can make the change easier on the development team. Some teams, when first introduced to Scrum, are overwhelmed to the point of inaction by the freedom of not having a day-by-day Gantt chart directing their work.

We have helped teams ease into Scrum by defining a set of sprint types:

- prototyping,
- requirements capture,
- analysis and design,
- implementation, and
- stabilization.

We then work with the teams to define the artifacts that will result from each sprint type. Using sprint types introduces just enough formality that the teams can more clearly see their way through the project. As the teams become more adept with the informality of the agile process, they gradually drop the sprint types concept.

Distributed development

Teams using agile processes tend to make decisions more quickly than plan-driven teams, relying on more frequent (and usually informal) communication to support this pace.

A failed attempt to use an agile process in a project with sites 2,000 miles apart taught us that organizations should resist distributed development for at least the first two or three months after initiating an agile process. The companies involved in the merger that initiated the project needed to resolve their political and cultural issues before developers could succeed with a project shared across multiple cities.

Teams using agile processes tend to make decisions more quickly than plan-driven teams.

If distributed teams must be combined, bringing as many people as possible together for the first week or two of the project can increase the likelihood of success. We have successfully used this approach on multiple distributed projects.

The need for top talent

Barry Boehm's principle of top talent, "use better and fewer people,"[7] is central to an agile process. Agile processes strip nonessential activities from projects, leaving developers more time to develop.

Although the difference in productivity between the best and worst programmers on a team may exceed the documented ratio of 10:1,[8] the productivity difference matters most when the programmers are working on tasks essential to software delivery. Productivity differences are irrelevant when the programmers are engaged in nonessential activities.

When fully engaged and comfortable with an agile process, a development team moves very quickly. Too many slow workers either slow the entire team or end up left behind by their faster colleagues.

Overzealous teams

One team we worked with was overly enthusiastic about a move to XP. At the project's onset, team members aggressively began documenting requirements by writing user stories, planning iterations, and pairing up for programming tasks.

Agile processes do not imply making decisions without forethought.

They even moved out of their existing space and into an adjacent abandoned building better suited for XP. Unfortunately, they did all of this so quickly that the rest of the organization had no idea what they were doing, resulting in a number of problems.

Although agile processes promise greater productivity once in place, productivity may decrease during the transition while the team learns new techniques. Not having anticipated this decreased productivity, the team chose to redouble its efforts when it occurred.

Agile processes do not imply making decisions without forethought, a distinction this overzealous team missed in its quest for speed and agility. To this team, Beck's recommendation to "put in what you need when you need it"[1] meant they needed to think only

about an hour ahead. They didn't have the discipline XP requires, and, while paying lip service to XP, they were actually doing nothing more than hacking.[9]

Overzealousness also caused the team to split into two camps: the overzealous team members and a group that knew decisions were being made too quickly and often incorrectly. Much like the tortoise and the hare, these two subteams pursued the project differently. After the project's failure, it took a while to convince both groups to jointly pursue an agile process, albeit one that was "less agile" than the overzealous members initially wanted.

Testers

Writing source code is a primary activity in any development process, but it is especially important in agile processes: "Code is the one artifact that development absolutely cannot live without."[1]

Agile processes do not have separate coding and testing phases; rather, code written during an iteration must be tested and debugged during that iteration. Testers and programmers work more closely earlier in an agile process than in other processes. Thus, testers and other nonprogrammers must be carefully integrated into any agile project in which they participate.

Initially, testers, even more than programmers, tend to view an agile process as micromanagement. Before the adoption of an agile process, many testers (especially those in organizations without a separate and dedicated testing group) do not receive much attention from managers. Test activities are often relegated to a single line item on a project plan.

In plan-driven projects, testing tends to occur without explicit notice from a project manager, so testers are not used to the extra attention they receive on most agile projects. As with programmers, testers will see over time that an agile process is not synonymous with micromanagement.

We have encountered testers who secretly want to be programmers and use a project's early iterations to sneak in some programming. We also have worked with testers who either volunteer or are coerced into writing unit tests for programmers.

Especially during a project's earliest iterations, you should discourage testers from such inappropriate activities. First, if the tester knows enough about programming to program and you need another programmer, hire the tester. Second, writing a unit test for someone else may result in a useful test, but it almost certainly loses some of the white-box advantages inherent in

self-written unit tests.

UPPER MANAGEMENT

Upper management often presents unique challenges to organizations wishing to move toward an agile process. Upper-management concerns generally fall into four categories:

- How can we promise new features to customers?
- How can we track progress?
- How will the agile process impact other groups?
- When does the project end?

Many managers, particularly those at higher levels, are reluctant to surrender the feeling of control that Gantt charts and other plan-driven process artifacts give them. Similarly, they may be comforted by the development group's promise to deliver an exact amount of functionality on a specified date, even if they know the group won't be able to do so.

Customer commitments

Managers who worry that an agile process will mean they can't make product commitments must understand that any past project plan that implied guarantees about delivery date, cost, and functionality was either wrong, heavily padded, or both.

In organizations with a history of incorrect project estimates, it might not be difficult to convince upper management that an agile process is worth trying. If a software group has a record of on-time delivery, however, you must convince upper management that using an agile process could have resulted in projects being completed either sooner or with fewer resources.

Drawing a typical cost, date, and feature triangle usually can persuade upper management that such commitments are an illusion. Once upper management realizes that past commitments were mostly combinations of good luck and padded estimates, they become very interested in any process that promises greater efficiency.

Tracking progress

Plan-driven processes appeal to many upper managers because they facilitate progress tracking. A manager can track a process that results in numerous documents by simply asking if the necessary documents have been produced.

If a Software Test Plan is called for, a first level of tracking can occur when the manager verifies

its existence. A second level of tracking can occur when the manager weighs the document, and a third if the manager reads it.

To persuade upper managers that agile processes allow adequate project tracking, we usually create two or more model status reports based entirely on fictional data about the project an organization is considering for an agile process. The reports depict a fictional two- to four-week project cycle.

A typical status report for a Scrum process project includes a list of key dates, a two- to five-paragraph commentary on the project's state, a burndown chart comparing progress to planned work, key metrics (defect in?ow, percentage of tests passed, and so on) appropriate to the project's current state, and a list of key risks. The upper-management decision makers we have worked with have found this type of status reporting satisfactory.

Impact on other groups

Some upper managers have expressed concern that although an agile process might benefit the development group, it can adversely affect one or more other groups. This concern becomes unhealthy when another group's processes negatively impact the development team's work. For example, one software engineering group wanted to use Scrum, while the product management group that provided all specifications and requirements wanted to continue with a heavyweight waterfall approach. The CEO saw no problem in letting each group pursue its independent strategies. The result was 2,000 pages of detailed product specifications fed into a development process that needed work prioritized in month-long units.

The software engineering group had to guess priorities from the product management group's three- to four-month task lists. Once the software engineering group was accustomed to Scrum, however, they moved through the requirements faster than the other group could write new requirements.

When introducing an agile process into an organization, upper management must understand and agree on how this will impact groups outside the development group. If they don't, and if they can't agree on how to resolve differences of opinion, most efforts will likely fail.

Project completion

Finally, upper management commonly fears that a project will go on forever. Most managers are comfortable with a model in which project budgets are approved and the project remains within the budget confines. They are less comfortable when told that project iterations will persist as

long as the customer or a customer proxy continues to identify high-priority, high-value work. Model status reports can help convince upper management that agile processes allow adequate project tracking.

Wrapping budgeting and strategic-planning activities around the project can address these concerns. For example, we estimated one commercial project would take from nine to 15 months—a fairly imprecise estimate because we didn't know exactly what features the completed system would include. Regardless of how many bells and whistles the client desired, however, we felt reasonably sure that a suitable initial release would be available in nine months. We therefore convinced upper management to fund a nine-month development effort with the agreement that additional funding would be discussed near the end of that period.

HUMAN RESOURCES

You might think that a human resources department would have no interest in one group's transition to an agile development process, but this is not the case. On a project that was transitioning to XP, two programmers approached the HR department with complaints about pair programming—two programmers sharing a keyboard and monitor and writing code in tandem—which, of course, sounded odd and unproductive to HR. Because we hadn't told HR about the process change, we were immediately in the difficult position of having to explain why pair programming made sense.

Another time, a small subset of programmers complained to HR that they didn't like how a project was being managed. The complaints stemmed from their unwillingness to consider or try anything new or different. The programmers were familiar with plan-driven processes, and anything else seemed too chaotic. This was in 1999, before XP had become an intriguing buzzword, before the Agile Alliance[2] had been formed, and when Scrum was only beginning to be documented.

The HR department must be told when a group is transitioning to an agile process. When told, however, the HR department may raise its own concerns, centering on an agile process's imprecise deliverables and dynamic goals. Many HR departments require corrective action plans, which cite specific deliverables and deadlines that can result in the employee's termination if not met.

It is difficult to shoehorn tasks from an XP iteration or Scrum sprint into a deterministic corrective

action plan. However, we have found that by proactively working with HR, we can sufficiently define tasks and deadlines that both satisfy HR and let the project follow an agile process.

How an agile process is introduced into an organization will significantly impact the ultimate success of the process change. Any new process is likely to appeal to some developers, who are excited to be among the first to try it. Similarly, this newness is an obstacle to developers who oppose change.

Agile processes have continued to evolve over the past four years, and approaches that worked in one case have not worked in another. As experience in the introduction of object technology into companies in the late 1980s and early 1990s led to the discovery of best practices in introducing that technology, we expect an understanding to arise over the next few years for agile processes.

References

1. K. Beck, *Extreme Programming Explained*, AddisonWesley, 2000.
2. K. Beck et al., “Manifesto for Agile Software Development,” Feb. 2001; www.agilemanifesto.org.
3. M. Poppendieck and T. Poppendieck, *Lean Software Development*, Addison-Wesley, 2003.
4. A. Cockburn, *Agile Software Development*, Addison-Wesley, 2002.
5. M. Cohn, “The Scrum Development Process”; www.mountaingoatsoftware.com/scrum.
6. K. Schwaber and M. Beedle, *Agile Software Development with Scrum*, Prentice Hall, 2002.
7. B. Boehm, *Software Engineering Economics*, Prentice Hall, 1981.
8. F. Brooks Jr., *The Mythical Man-Month*, AddisonWesley, 1975.
9. B. Boehm, “Get Ready for Agile Methods, with Care,” *Computer*, Jan. 2002, pp. 64-69.
10. A. Cockburn and J. Highsmith, “Agile Software Development: The People Factor,” *Computer*, Nov. 2001, pp. 131-133.
11. M. Cohn, “The Upside of Downsizing,” *Software Test and Quality Eng.*, Jan. 2003, pp. 18-21.
12. Shine Technologies, “Agile Methodologies Survey”; www.shinetech.com/agile_survey_results.jsp, Jan. 2003.

Posted: June 1, 2003

Tagged: scrum teams, product owner, sprinting, user stories, metrics, management, meetings, testing, programming, customers

About the Author

Doris Ford is the president of Precision Projects. Her current research interests include software pro- ject management, project metric development, and tracking methodologies. Ford received an MBA from Regis University. She is a member of the Pro- ject Management Institute. Contact her at dtford@yahoo.com.