Crossroads News
A Monthly Publication for Software and CM Professionals

"Agile" Change Management - from first principles to best practices

Brad Appleton, Steve Konieczka, Steve Berczuk – August 2003

This month we will discuss what "agile" change management is, what it means, and how to do it. First, we will describe what we mean by agile change management, and give some examples of the relevant parts of some agile methods. Then we will discuss how to make change management "agile", and some of the principles and techniques behind it.

What is Change Management?

> "Change management is a general term encompassing controlling and tracking change. Change management may be very low ceremony (developers handle physical cards) or high ceremony (change is allowed only through formal change control boards), but should be set to the appropriate level for the given development domain and culture." -- *Grady Booch, posted 08-July-2003 on the extreme-programming mailing list*

To some people, "change management" refers to a delivered product configuration and its physical and functional contents. Controlling changes is then largely about ensuring that the delivered configuration conforms to its physical and functional specifications, and that it is indeed what you said it was when you agreed to produce and deliver it.

Others regard change management as more of a project management function to manage the scope of work that is performed to provide a product or service. Controlling changes then becomes largely about managing expectations to realistically match the amount of work that can be accomplished within the agreed upon scope and schedule, and the means by which scope and schedule changes are accepted.

The reality is that change management encompasses both of these perspectives: the project's scope and the product's scope. What's more, the project and the product may each have different customers whose needs and expectations aren't always in alignment. Successfully managing and tracking the expectations and requirements of the project's organization and sponsors, as well as the product's purchasers and consumers (end-users) hinges upon successful cooperation between the project management and configuration management functions.

What is "Agile" Change Management?

So what's so "agile" about change management as proposed by the agile methods "du jour"? The "agile" in "agile change management" can be interpreted in a couple of ways:

- Change management that successfully meets the needs of an "agile" project
- An approach to change management that is "agile" in its own right

These are not mutually exclusive: an agile approach to change management certainly helps to meet the needs of an agile project. So there is a broad area of overlap between the two. Furthermore, the argument can (and perhaps should) be made that so-called "agile" change management is really just plain old sound, and solid change management practices. You know, the things that we are supposed to have been doing all along, but either got lost in all the noise, or perhaps just fell upon deaf ears.

Whether we regard agile change management as new, or yet another case of "what's old is new", an agile "spin" has since been "spun": it is a focus on close collaboration via informal face-to-face communication, and on keeping processes and tools lean and simple. Let's take a look at some relevant portions of two of the more commonly used agile methods today: eXtreme programming (XP) and Scrum.

Agile Change Management in XP

At the beginning of each iteration (which lasts typically between 1-3 weeks), XP conducts *The Planning Game*: the project manager, developers, and customers get together in a room and look at the existing back-log of to-be-implemented requests (called *User Stories* and captured on plain old index cards). The customers may add or remove requests from the stack. Developer's converse with the customers and attach estimates to each request. (See *[1]* for more details.)

Then the customer decides which requests they want implemented during the next iteration. They do this based upon the iteration length, the estimates for each request, and their stated priorities about which requests are the most important. The customer may completely change priorities from one iteration to the next. This is okay as long as development and project management agree the result can be successfully implemented within the scheduled period.

Once customers decide the requests to implement for an iteration, developers sign up for "stories" to implement and may work only on stories for that iteration. Developers dialogue with customers to elaborate the details of a story. To capture these requirements, customers write acceptance tests for each story on the back of its corresponding index card.

When a request has been implemented, integrated and tested, the corresponding story card is placed upon a *big visible chart* that shows it as completed. The chart also conveys the number of passed versus still-to-be-passed test cases.

During an iteration, customers may reprioritize unimplemented stories, and remove or replace stories provided that the overall estimated effort remains within the planned iteration end-date. Developers are permitted and trusted to make refactoring changes, or code clarity improvements (or coding standard violation corrections), and to fix any problems encountered, provided it doesn't impact the schedule of their current "story."

There is a brief *daily stand-up meeting* where each team member quickly gives the status of what they have been working on since the previous day's meeting. If an issue threatens on-time completion of their task, the issue is raised to the customer or project manager, either "on the spot" (with the *onsite customer*) or in the daily stand-up meeting. The customer then decides the course of action to take, based on the verbally communicated risk and impact, and the possible set of alternatives)=.

At the end of the iteration (and for each build during the iteration, when feasible) all the completed (and preferably automated) customer acceptance tests are executed to ensure that the built result conforms to what the customers specified in the tests.

Agile Change Management in Scrum

What Scrum does for change management looks very similar to what XP does (see *[2]*). A large part of this is because some of the project management practices in XP were borrowed directly from Scrum. In fact the *daily stand-up meeting* in XP is what Scrum calls a *scrum* and is how the method got its name:

> A "*scrum*" is a rugby term that refers to when all the players on a team quickly huddle together to decide what they are going to do for the next play. They have to do this very quickly because the play is "live" and the clock is running during the huddled *scrum*.

Scrum divides development up into iterations called *sprints*. Each *sprint* is approximately 30 days and delivers an executable *product increment* of agreed upon functionality from the *product backlog*. Scrum maintains two formal backlogs of requests: a *product backlog* and a *sprint backlog*:

- The *product backlog* is "an evolving, prioritized queue of business and technical functionality that needs to be developed into a system"*[2]*. Only the *product owner* (the person empowered to represent customer priorities) may prioritize the backlog.
- The *sprint backlog* is the list of all requests to work in the current iteration (sprint).

The content of the *sprint backlog* is agreed upon by the *product owner* at the beginning of planning for the sprint (very similar to the planning game in XP). During the sprint, there is a *daily scrum meeting* in which each team member must answer the following three questions:

- What have you done since the last scrum?
- What will you do between now and the next scrum?
- What got in your way of doing work?

Issues that arise which may impact a task are raised during the daily scrum and might be added to the sprint backlog or the product backlog based upon the customer's decision. At the end of a sprint, a *sprint review* is held to present the *product increment* to the customers for evaluation to ensure it has met the agreed upon *sprint goals* and satisfied the specified requirements for each request in the completed sprint backlog.

What's so "Agile" about that?

The first step toward agile change management is a change in mind-set! Those of us who think of change control as preventing changes to an agreed upon “baseline” of project/product scope must change our mind-set to embrace change as a natural and expected part of development. The focus needs to shift from preventing change, to managing expectations and tracking changes. Instead of saying "No!" to a change request, the "agile" answer should simply inform the customer of the associated risk and impact upon the currently agreed upon cost, schedule, scope and quality, and then let the customer make an informed decision. This is what the fourth “value” of *the Agile Manifesto [3]* implies when it espouses "*Responding to change over* following a plan."

If agility is defined as “the ability to both create and respond to change in order to profit in a turbulent business environment”*[4]*, then agile change management must help us do these two things:

1. Be more responsive to requests for change
2. More quickly and easily implement those changes

As mentioned in an earlier column *[5]*, responding quickly and effectively to change is easiest to do when we can minimize the following:

- The cost of knowledge-transfer between individuals
- The amount of knowledge that must be captured in intermediate artifacts
- The duration of time between making a project decision, and exploring its results to learn the consequences of implementing that decision

Agile methods enable projects to be more responsive to requests for change by:

a) Using short and frequent iterations to minimize the time between specifying a requirement and seeing it implemented so that adjustments to functionality and priorities can be recognized sooner rather than later

b) Requiring developers and customers to communicate and work together daily (co-located together if possible) so that change-control decisions can be made and communicated quickly and communicated face-to-face with minimal waiting and documentation

c) Allowing and accommodating (even encouraging) changes in scope and priorities via a highly collaborative agreement process that informs the customer of impact and risk, and puts the customer in control of the scope and priorities

d) Authorizing and empowering team members to correct problems with the code’s behavior and structure without having to suffer delays waiting for formal change proposal, review, and authorization of such corrective changes

These four things together are what help transform traditional contract negotiation into customer collaboration, in accordance with the third “value” of *the Agile Manifesto*.

Agile methods enable projects to more quickly and easily implement changes by:

a) Working in short but complete cycles on the smallest possible "quanta" of work with tight feedback loops (e.g., short iterations, test-driven development, pair programming, continuous integration, etc. *[1], [2]* and *[4]*)

b) Mandating simple design, and emergent design methods (like refactoring *[6]*) to make the code as simple and easy as possible to change

c) Minimizing the number and size of non-code artifacts that must be produced or updated in order to implement a change

d) Working only on the features scheduled for the current iteration

Continuous Customer Collaboration: The Key to Responsiveness

Perhaps the single most influential practices that agile methods provide for change management are the use of:

1. Incremental and iterative development to quickly deliver very small but very frequent executable milestones and obtain customer feedback on the results
2. Close customer collaboration to promote daily interaction and face-to-face communication between customers and the development team

The first of these two practices has the most influence upon the time and effort required to effect a requested change; the latter of the two has the most influence upon the time and effort required to respond to a requested change.

Since much of change control is about decision-making and prioritization of requested functionality, we would like to remove or reduce anything that slows down the process of obtaining customer agreement to approve, prioritize, and authorize a requested change. If the customers are always available to converse with most anytime, then we don't need to spend any time waiting for customers to convene and approve requests. Such approval can instead transpire in a matter of minutes or hours rather than days or weeks.

Agility mandates that customer input and decisions can be obtained within at most one or two days, and preferably within a matter of minutes or hours. If customers are not readily accessible, then a suitable customer representative (a *product manager/champion*) must be available, and must be empowered to make decisions on behalf of the customers.

Getting "One Voice" from the Customer: The Product Champion's Burden

Getting a group of customer's to agree upon functionality and priorities can be very difficult! Competing business interests, differing customer markets, and personal and political agendas can make it seem futile to attempt to facilitate alignment on a vision and a shared set of goals and priorities. Furthermore, issues of broad or diverse customer bases make timely and effective customer communication and collaboration even more difficult.

The role of the *Product Champion* (or *Product Manager*) is to effectively overcome these obstacles to quickly and accurately convey a common and collective set of needs and priorities that is representative of the product's installed (and prospective) customer base. Jim Highsmith writes the following about the role of the product manager (in a 14-July-2003 posting to the agileprojectmanagement Yahoo group):

> The primary functions [of the product manager] are to facilitate, nudge, and mentor the customers in carrying out their partnership responsibilities. For internal IT organizations, the product manager works with various user departments, making sure the right person is available when the development team needs them. The Product Manager is also responsible for the participatory

decision-making process on the customer side, just like the project manager is on the development side. Neither one's primary style is to make all the decisions, but they have the power if need be.

For product companies, the product manager works with others within the company to convey customer requirements, priorities, etc. to the development team. How and when the product group involves the real outside customer is a fairly complex issue depending on the market, product stage, etc.

The danger in any [customer] proxy is that they may come to believe that they *are* the customer and not just a proxy--their job is more like a facilitator's job--most of the time…. In the past, IT organizations have gotten in trouble because the "analyst", who usually comes from IT, has implicitly taken on the product manager's role--trying not only to gather requirements, but also to referee between different customer groups. The customer groups always then had someone else to blame--**IT**. The Product Manager (who might co-ordinate 10, 15, or more customer groups in a big IT project) has to come from the customer side and has to have some authority delegated from those customers.

Participatory Decision-Making: The Key to Facilitating Agreement

Effective facilitation practices and techniques for obtaining "one voice" from the customer are the "bane" of the product manager's existence. One of the first and most important practices for effectively facilitating customer group decisions is to get the group itself to *Decide How to Decide [7]*. The values, objectives, and criteria for making a decision must be both shared and explicit! A simple consensus approach is best, when it works. If it doesn't, some successful strategies for driving customers to reach consensus on iteration and release content are as follows:

Normative Voting: Each customer group gets the same, fixed, number of total "points" (or "votes") to distribute among the set of candidate requests to be prioritized. They can allocate more points to the requests that they deem most important (but no "partial point distributions" are allowed). So I could put all my 100 "votes" on one request if I wished; or I could give 10 points each to my top-ten requests; or I might give 20 points to each of my top three, then 10 points to each of my next three, and split the remaining 10 points across my next 2-3 most desired requests. At the end, the number of points cast by all customer groups is tallied for each request, and then the result is sorted in descending order from highest-to-lowest number of points.

Weighted Normative Voting: This is very similar to normative voting, but not every customer group gets the same number of "points" for voting. Instead, they are assigned a number of points that is directly proportional to the amount of funding (or business revenue, or investment capital, etc.) they provide to the project. This essentially assigns a "weight factor" to each customer group based on their perceived contribution to the project's livelihood. This is also sometimes likened to "Congressional Voting" where, unlike the U.S. Senate where each Senator gets the same number of votes, instead each "representative" is given a number of votes indicative of the amount of the customer-base (or market share) that they control.

Even Effort Distribution: Rather than assigning votes and tallying totals to see which requests "make the cut" for the next iteration, *even effort distribution* takes the total number of *effort units* (e.g., staff days, or staff hours) that are available to be scheduled for the next iteration, and distributes the resulting total effort units equally between all the customer groups. Each customer group may then select as many requests from the backlog as they wish, provided the total estimated effort for their selected requests does not exceed the number of effort units allocated to them for that iteration. So If I'm allotted 10 staff days of effort, I might select one request estimated at 5 staff days, another estimated at 3 staff days, and one more estimated at 2 staff days.

Weighted Effort Distribution: Instead of evenly distributing effort units among all the customer groups, each customer group is allocated a number of effort units that is directly proportional to the amount

of funding (or business revenue, or investment capital, etc.) they provide to the project. The "weight factor" is determined the same way as with the normative voting approach, but it applies to the amount of effort allocated to select requests, instead of voting on their relative importance (selection instead of election).

Both the "effort distribution" and "normative voting" approaches provide an incentive for cooperation between the participating customer groups. Disparate customers may "pool" their votes or effort allocations for mutually desirable requests - especially those that require significant effort. More cooperation and commonality among the selected requests results in a greater volume of requests being implemented that align with each customer group's wants and needs.

The "voting" approaches tend to take less time to reach consensus; they also run the risk of requests that are important to a minority not being implemented in a timely fashion. The "effort distribution" approaches tend to involve more "wheeling and dealing" between customers to forge alliances for their most desired requests. This can be time consuming, but can also ensure that each group has at least some of its "top picks" implemented in the near future.

As previously mentioned, the customer base itself should be responsible for deciding which of the above approaches (or other approaches, including simple consensus) should be used. Decision making criteria should include the types and amounts of the impacts and risks, as well as the associated business urgency and value. Don't forget that the longer it takes to convene a customer meeting to communicate, prioritize, and select the requests to implement, the less responsive your change management process will be (and the less agile the project and team can be).

There must be a balance between the extent to which the product manager must seek and facilitate agreement from customers, and the extent to which the product manager is entrusted and empowered to make "on the spot" decisions in the interests of the customer base. A good rule of thumb is to gather the customer representatives together for the request prioritization and planning meetings at the beginning of each iteration, and let the product manager make the decisions for any issues that arise in the during an iteration. When working with the development team to elaborate the details of a request, the product manager is the preferred person to work with for the sake of consistency and conceptual integrity. However, it can also work well to have 1-3 customer representatives do this on a request-by-request basis.

A Word about Fixed Scope/Price Contracts

Conducting "agile" change control for fixed scope/price contracts can be particularly tricky. One approach that has worked well for ObjectMentor is to use the time normally spent conducting thorough analysis and estimation to instead conduct a few trial iterations that implement some of the higher-priority features. Then use the resulting data from those iterations to estimate (and hence bid) on the overall contract.

It is also vitally important that the contract specifically outline the process and mechanism for proposing and accepting changes to the scope of the contract (and to the process and mechanism itself, not to mention the rest of the contract). See the results of a recent Workshop on Agile Contracts (held May 2003 in Genoa Italy) at http://www.poppendieck.com/agilecontractsworkshop1.

What about Change Tracking?

So far we've mentioned very little about change tracking and traceability. Tracking and reporting the status and content of requests and changes across many artifacts and activities can be extremely unwieldy. We often require sophisticated tools to assist us in this complex and tedious endeavor. Agile approaches strive to minimize the number and size of non-code artifacts that are created. Fewer artifacts and documents mean fewer items to tracked, and less effort and complexity to track them.

In the case of XP, its proponents say index cards are preferred in favor of tools. Their physical limitations force developers and customers to keep things short and simple, and physically associate a request, its

corresponding requirements, and its tests with the same physical artifact (in XP, the customer acceptance tests are the detailed requirements). Scrum and other agile methods appear to be less insistent upon the use of index cards, and less resistant to the use of tools (such as spreadsheets or change/request tracking systems).

Most experienced change and configuration managers would gasp at the seeming low-tech inefficiency of index cards as means to track and report conformance and status. At the same time, it is hard to combat the efficacy of index cards to engage customers in a more participative and collaborative dialogue for eliciting requirements, and drawing simple diagrams – apparently, the physicality and tactile experience of the cards is simply more inviting and lets the customer do the writing instead of relying upon a single input-controlling "scribe." (Some have suggested "pair writing" could be done between a developer and a customer in the same fashion that pair programming is done during implementation.)

Even if index cards are used as the medium for initial capture of requests and requirements, many (if not most) change and configuration managers will want to subsequently transfer this information into a spreadsheet or a tracking system for fast and easy reporting, querying, searching, sorting, as well as for real-time dissemination across the project's organization and stakeholder-sites (not to mention affording more efficient and reliable storage, record retention, archival, and retrieval/recovery). Keeping "stories" in a tool also lets you apply a simple workflow see how the work is progressing.

Several "agilists" would argue that such a tool "rails against" the mandate of simplicity. To be certain, many have gone overboard with defining and enforcing process through a tool - we highly recommend against that since it often results in drastically increased administration overhead, drastically decreased face-to-face communication, and hence very low agility. On the other hand, many have had bad experiences with tools used to enforce too much process. We believe this, combined with bad experiences from misapplied CMM level-climbing attempts, is responsible for much of the agile community's backlash against the use of more sophisticated but useful tools and processes.

The relentless focus on keeping things as simple as possible, and on face-to-face interaction over face-to-machine interaction still provides sound guidelines and important reminders when adopting processes and tools. With the "right" amount of process using a simple and "smart" tool, agile projects will find increased productivity and better coordination. The bottom line is really to do what you know works for you, and keep it as simple as possible, applying the principles of lean development *[8]* every step of the way.

Recap

Change management is concerned with controlling and tracking changes to project and product scope and ensuring conformance to customer expectations. Agile change management is concerned with increasing the ability of the project to be responsive to requests for change and to quickly implement accepted change requests. This requires minimizing: the cost of effective knowledge transfer, the amount of knowledge captured in intermediate artifacts, and the time between making a decision and learning the effects of its result. The key success factors of agile change management are the use of iterative and incremental development with short feedback cycles, and close collaboration with frequent face-to-face interaction between developers and customers.

Sometimes the customer base is diverse and/or dispersed and a product manager role is needed to facilitate agreement from, and make decisions on behalf of the customer base. Participatory decision-making tends to produce the most collaborative results, and normative voting and effort allocation approaches have proved effective in reaching customer consensus to prioritize and plan the requests to implement at the beginning of an iteration. The product manager should be empowered to make decisions about issues that arise during an iteration, but either the product manager or a small sampling of customer reps can elaborate the details of a particular request to be implemented.

Index cards can be an effective means of engaging the customer during requirements capture, and simple tools (and processes) are an effective means of tracking, coordinating, and reporting visible progress of requests and changes against expected functionality and content. Don't be fooled by the allure of sophisticated processes and tools; and don't overcompensate by discarding simple but effective tools and techniques. Look for a balance of utility and simplicity that is both effective and efficient in meeting your change management needs. And keep an eye out for opportunities to eliminate redundant or unused elements of your processes, tools, and artifacts after each iteration.

Next month we will address some questions and concerns raised by readers who gave us feedback on previous articles, and attempt to clarify some common facts and fallacies regarding "agility" and agile methods.

References

[1] **Planning Extreme Programming**, by Kent Beck and Martin Fowler; Addison-Wesley, October 2000

[2] **Agile Software Development with Scrum**, by Ken Schwaber and Mike Beedle; Addison-Wesley, 2002

[3] *The Agile Manifesto*, see www.AgileManifesto.org

[4] **Agile Software Development Ecosystems**, by James Highsmith; Addison-Wesley, March 2002

[5] *Agile Configuration Management Environments*, by Brad Appleton; CM Crossroads Newsletter, April 2003 (Vol. 2 No. 4)

[6] **Refactoring: Improving the Design of Existing Code**, by Martin Fowler; Addison-Wesley, July 1999

[7] **Requirements by Collaboration: Workshops for Defining Needs**, by Ellen Gottesdiener; Addison-Wesley, April 2002

[8] **Lean Software Development: An Agile Toolkit**, by Mary Poppendieck and Tom Poppendieck; Addison-Wesley, June 2003

Acknowledgements

Thanks to the participants of the Yahoo groups for extremeprogramming, industrialxp, leandevelopment, and agileprojectmanagement. Particular thanks go out to the following individuals:

- Kent Beck, www.threeriversinstitute.com
- Ron Jeffries, www.xprogramming.com
- Robert Martin, www.objectmentor.com
- Josh Kerievsky, www.industriallogic.com
- Diana Larsen, www.industriallogic.com
- Mary Poppendieck, www.poppendieck.com
- James Highsmith, www.jimhighsmith.com
- Alistair Cockburn, http://alistair.cockburn.us
- Amy Schwab, www.projectcommunity.com
- C. Keith Ray, http://homepage.mac.com/keithray/blog
- Glen B. Alleman, www.niwotridge.com
- Frank Patrick, www.focusedperformance.com
- Stephen Gordon, http://sf.asu.eduT

I heartily recommend visiting each of the above websites for a wealth of insights and resources on agility in particular, as well as on successful software development and project & team collaboration in general.

Editor Biographies

Brad Appleton is co-author of H**Software Configuration Management Patterns: Effective Teamwork, Practical Integration**H. He has been a software developer since 1987 and has extensive experience using, developing, and supporting SCM environments for teams of all shapes and sizes. In addition to SCM, Brad is well versed in agile development, and cofounded the Chicago Agile Development and Chicago Patterns Groups. He holds an M.S. in Software Engineering and a B.S. in Computer Science and Mathematics. You can reach Brad by email at *brad@bradapp.net*

Steve Berczuk is an Independent consultant who has been developing object-oriented software applications since 1989, often as part of geographically distributed teams. In addition to developing software he helps teams use Software Configuration Management effectively in their development process. Steve is co-author of the book **Software Configuration Management Patterns: Effective Teamwork, Practical Integration**. He has an M.S. in Operations Research from Stanford University and an S.B. in Electrical Engineering from MIT. You can contact him at *steve@berczuk.com*. His web site is www.berczuk.com

Steve Konieczka is President and Chief Operating Officer of SCM Labs, a leading Software Configuration Management solutions provider. An IT consultant for 14 years, Steve understands the challenges IT organizations face in change management. He has helped shape companies' methodologies for creating and implementing effective SCM solutions for local and national clients. Steve is a member of Young Entrepreneurs Organization and serves on the board of the Association for Configuration and Data Management (ACDM). He holds a Bachelor of Science in Computer Information Systems from Colorado State University. You can reach Steve at *steve@scmlabs.com*