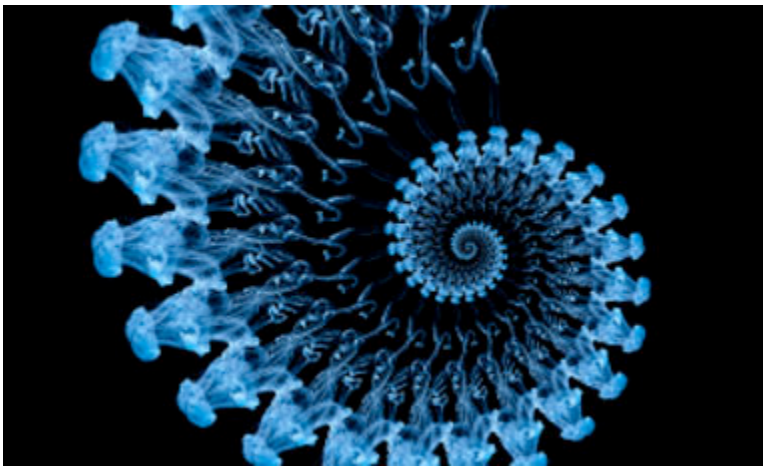
Agile Journal: Iterations

WRITTEN BY STEVE BERCZUK

SUNDAY, 05 OCTOBER 2008



Iteration is at the heart of agile development practices. In an agile project you do something, measure your progress, and then use the feedback from the measurement to figure out what to do next. This cycle allows you to follow the Agile Manifesto value *Responding to change over following a plan* by providing for points in time where you can measure your progress at the project level. Whether your approach to agile is project-focused like Scrum or development-focused, like extreme programming, iteration is what drives an agile project.

Agile methods use engineering practices such as Unit Testing and Continuous Integration to provide for feedback cycles close to the code. The Iteration allows for a feedback at the macro level, giving the stakeholders the ability to view progress in short regular feedback cycles.

While iteration is key to agile development working within iterations can be challenging for customers and developers alike.

Concepts
In an agile project the *team* is group of people delivering the application code. This includes developers, testers, designers, etc. The *product owner*, sometimes referred to as the customer, is the person specifying the functionality that the team will deliver and the priorities. The product owner specifies *what* will be built and the team decides *how* to build it.

The product owner maintains the *product backlog,* a prioritized list of everything that may be in the product someday. The portion of the product backlog that the product owner assigns to the iteration is the *iteration backlog*.

Incremental development means building parts of a system, for example working on the interface to the database. *Iteration* means starting with a rough solution that works, then improving on it as you go. Jeff Patton explains the difference between incremental and iterative development quite concisely in his article The Neglected Practice of Iteration. In practice, teams combine iteration and incremental development, but it iterating on an end-to-end solution is what lets you validate requirements.

Iteration Overview
In an agile project, an iteration is a fixed period of time during which a team implements a set of features, which result in a shippable increment of the product or project. This period of time is referred to as a Sprint in Scrum. XP refers to a *weekly cycle*. Regardless of the name, the key features of an agile iteration are:

- It is time boxed. There is a fixed start and end.
- The amount of work that is planned to be completed during the iteration should not change during the iteration.
- It starts with a planning activity.
- It ends with a review activity.
- At the end there is a shippable product or project increment that can be refined in future iterations.

The last point highlights the difference between iterative and incremental development: each iteration helps the project or application take shape so that the product owner can validate the state of the project with the current goals of the project.

The basics of iterations are that team will:

- Commit to a list of items.
- Work on the list.
- Review what was done.

To iterate successfully an agile team must follow certain activities.

Iteration Activities
During an iteration, the key activities are:

- Estimating and Planning

- Execution
- Review

The iteration starts with the product owner assigning items off of the product backlog to the iteration. The team then meets and plans how they can complete the backlog items by the end of the iteration. If the team is not confident that they can complete the planned work within the iteration, the team should raise their concerns with the product owner and revise the backlog for that iteration.

It is important to have a realistic iteration backlog so that the team and the product owner have common expectations. While it is not unreasonable to have work in queue in case the team finishes their work early, *planning* to do more work than can be done leads to the team and the product owner not taking the plan seriously rather than a commitment. Over-planning also makes it more difficult to improve your estimation process.

At the end of the iteration the team reviews their work with the product owner. The review should be quick and informal, and form a basis for a conversation between the team and the product owner about how the iteration went, and what should happen in the next iteration. The review is an opportunity for all of the stakeholders to look back on the iteration as a whole and understand how to do better.

Issues with Iterations (and Solutions)
While the concepts around iteration are simple, teams and product owners often have some concerns about how well a time-boxed development cycle will fit into their process. Typical concerns are around the areas of setting priorities, scoping work to fit into an iteration, and the practicality of defining a fixed set of work when there are support issues to address. It's possible to adhere to constraints of an iteration and address these issues.

Prioritizing
Iterations force product owners to decide what the most important work items are. In many organizations features are organized into large buckets like "priority one" and "priority two." The reality is that people can generally work on one item at a time, and even if you have all "priority one" tasks, they will be worked on in some order.

By allocating work to an iteration, the product owner has the power to define the order explicitly. Prioritizing individual items can be difficult because a product backlog can be very large. From a practical point of view a product owner need only define strict priorities for work that needs to be done in the next couple of iterations. Coarser buckets are fine for items beyond that.

Another common challenge is that a product owner may not feel like she has all of the information to make a decision that one feature is more important than another when both need to be in the end product. When using iterations, the cost of working on a given feature is smaller than with a non-iterative project; you can decide on one priority and change your mind the next iteration.

With an agile project you move in a direction based on current knowledge, and since iterations are short, the decision does not matter as much as it would should you be planning for a 3-6 month release cycle. The key to assigning work to an iteration is to acknowledge that the difference between 2 items may be quite arbitrary. And you can always reprioritize when the next iteration starts.

Customer Support
While agile teams strive to reduce defects most organizations have to support users while doing new development. Product owners can be frustrated by the restriction that the iteration backlog should not be changed once an iteration starts, as this may lead to newly discovered urgent issues would being put off, reducing the ability of the team to deliver business value.

The "fixed work" rule of iterations is not meant to set up a wall between the team and the product owner. Rather, it forces the product owner to address the priority of any new work and its impact to the original commitments. Adding work to the backlog mid-iteration means that it will be more difficult for the team to deliver the work that it already committed to, and makes it difficult for the team to manage expectations with the product owner.

If you are in a situation where the same team is doing support work and new development, there are a few strategies that are true to the spirit of iteration, but still allow for changes:

- Think about the priority. Is the problem something that users need a fix delivered before the end of the current iteration, or can it wait? If an item won't be released or deployed before the end of the iteration, it probably should not be added to the iteration mid-stream. When something is truly urgent, remove something else from the iteration backlog to make room.
- Allocate time for fixes based on past history. The purpose of a fixed iteration backlog is to manage expectations. You can allow a fixed amount of capacity to address urgent issues. Everyone needs to be aware of how much effort is spent on new issues, and raise concerns when the amount of new work exceeds the planned capacity. And since there is a fixed capacity for new work the product owner still needs to prioritize what that capacity is used for.

- If Planning for urgent issues does not appeal to your organization, at least track the work as added work in a burn down chart. The impact of the added work on the deliverable end date will then be visible early, and having data on the impact of new work to share at an iteration review will the team and the product owner move towards one of the planning approaches.
- If there are significant issues that need to be addressed that will have a serious impact on the iteration plan, then acknowledge this by stopping the iteration and plan a new iteration to allow for the new work. If the added work is truly more important than new development, acknowledge it.

There is nothing inherent in an agile iteration that makes the team less responsive to the needs of the business. Rather, iterations help the business to understand the impact of change more quickly than other approaches

Scoping Work
Development teams often wrestle with the defining tasks that can be done in a short iteration when they are used to thinking of longer projects. Customers have difficulty thinking of meaningful "features" that can be completed in an iteration. Some insist that it is impossible to break work down in a way that allows for anything to be done within the length of an iteration cycle.

Thinking about features that fit into an iteration requires a change of mindset. Agile projects start with the idea that change is inevitable, and agile teams have engineering practices that make change quick and easy. Also, the benefits of agile are that a product owner gets to see something that works (somewhat) while it is still in progress.

Scoping work to fit into iterations is one of the harder things for teams to do, but it is also where you get the most value of iterative development. By iterating you validate project is going in the right direction while you have the ability to make corrections. Iterative development also allows you to decide that what a feature is usable before everything you thought you needed was implemented, thus allowing you to save effort.

With some thought and creativity you can break down almost any feature into something that shows functionality with enough fidelity so that a product owner can decide if the feature is what she really wanted, of if additional requirements are really necessary.

When defining stories or features:

- Try to focus on the smallest non-trivial end-to-end slice through an application. For example, "Look up flight schedules between 2 cities" rather than "book a flight." This will also help you develop tests, as smaller features are easier to test completely.
- Develop in vertical rather than horizontal slices. Don't implement stories by application tier. (This is also a key part of being "iterative" rather than "incremental"). While some architectural design will help an application's coherence, starting with frameworks can lead to build in functionality that you will not need, as well as make it really difficult for the product owner to see progress. It is easier to demonstrate a User Interface than a Database API.

Time Box Boundaries
Iterations are different from the usual long-ish planning cycles teams and product owners are used to.

Some teams will want to extend the iteration a day of two to finish the work that is "almost done." The product owner may want the team to finish a task before starting a new iteration. Avoid this. Even though it is difficult to admit that work is not complete, having leaky iteration boundaries opens the door to schedules slipping without feedback, and does not give you the chance to adjust.

Agile methods acknowledge that there is uncertainty, and that Big Planning is rarely accurate; some customers sometimes take the idea of "*Responding to change over following a plan*" to an extreme, meaning that there need not be a framework for planning. While the "big plan" can change from iteration to iteration, the iteration boundaries provide an essential framework for keeping a project on track.

From a morale point of view, end of iteration reviews are a focal point for the team and the product owners, allowing for a forum where the team can acknowledge small slips and work with the product owner to make adjustments. The longer you put off a review session, the higher the cost of admitting a problem.

(Wrapping Up &) Getting Started
While there is a lot behind an iteration, having your team work effectively with iterations is, well, an iterative process. To be successful the an iteration needs to have a reasonable length, start with a good set of stories, and the amount of work that the team signs up for should match well with the capacity.

When you start off with iterations you may plan for more or less work than fits, or you may find that the stories you are working on are too broad to make reasonable progress on, or that you have not planned for a realistic amount of support work. Working with iterations means that you can take what you learned and strive to do better next time.

Iterations are not an obstacle to having the customer set product priorities. Iterations help both the product owner and the team focus on the most important work, while allowing for adaptation for the unexpected.

If you are new to agile, you may be tempted to adapt the iteration process before really trying it. Agile teams believe in continuous improvement, but it is best to start from an understanding of how the "model" works before making changes.

About the Author
Steve Berczuk is a consultant and developer who works with Agile teams. He has over 20 years experience developing applications, 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 and a Certified ScrumMaster. 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.

Comments