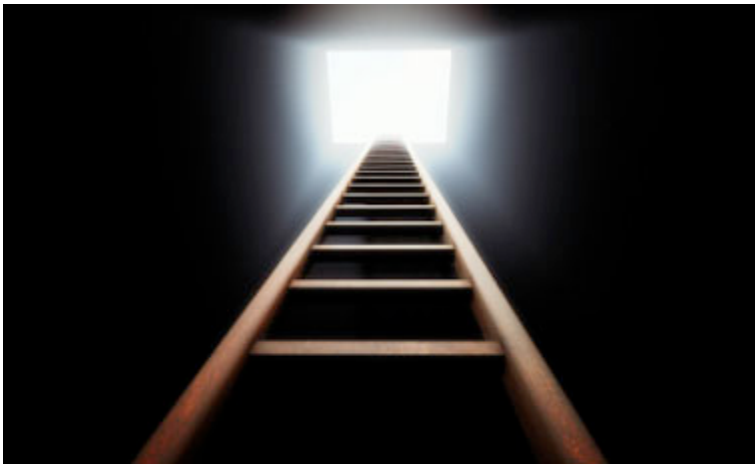


# An Agile Approach to Release Management



Written by [Steve Berczuk](#), [Robert Cowham](#), [Brad Appleton](#)

Monday, 26 May 2008



Teams practicing Agile Software Development value working software over other artifacts. A feature from the release plan is not complete until you can demonstrate it to your customer, ideally in a shippable state. Agile teams strive to have a working system ("potentially shippable") ready at the end of each iteration. Thus Release Management should be easy for an ideal agile team, as agile teams, in theory, are ready to release at regular intervals, and the release management aspect is the customer saying "ship it!."

Agile teams work under the same constraints as other software development teams, having to address issues of maintenance and support, the need for external artifacts like documentation and training, and

the reality that not every customer can accept change at the rate that an agile team can (in theory) deliver it. To attain the goal of a shippable product at the end of every iteration, an agile team must manage the flow of change from a customer, and maintain high discipline and good engineering practices.

In the paragraphs below we will discuss how release management works in an agile project, and explain how even non-agile teams can benefit from applying aspects of an agile approach so that you can deliver value to your customers in a more predictable fashion.

## Agile Principles

Agile Project Management Practices enable you to manage schedule risk in a project. In a sense, many are simply good practice:

- Plan to work in time boxed iterations of 2-4 weeks
- Maintain a backlog of feature requests in prioritized order, and revisit the priority at each iteration boundary
- At the start of an iteration select the highest value items from the backlog. Do detailed planning for these items.
- Integrate Often
- Verify Often
- Deliver and review against a plan.

These practices affect how you think about release management because release management is a planning activity and any planning activity has uncertainty which increases the further away you are from the present. The goal of having features complete or not complete at the end of an iteration allows you to determine have a clear idea of what will be available to ship. The prioritization of the backlog allows you to compensate for problems by having the most important features developed first, so that you can still meet a release date should that matter for market or regulatory reasons. The "always shippable" goal means that you can, if need be, accelerate your release schedule.

An agile approach enables better release planning by combining planning discipline, which helps you to focus on the highest value work, and engineering (including SCM) discipline, which helps you to identify and fix problems early, giving you more predictability. Agile practices make shipping a release a decision that the product owners can make without worrying if the team will meet a date far in the future. The role of the agile team is to enable allow the product owner to make the decision on short notice if need be.

## Realities

The IT Process Institute's report [Change Configuration and Release Performance Study](#) (full report costs \$1,695 but the summary is available for free if you register) says that:

- Release trumps change - rigorous release build, testing and rollback practices have broad impact on individual

performance measures and overall performance. Change tracking and change oversight practices are necessary but not sufficient to achieve performance improvement on their own.

- Process discipline matters - there are no change, configuration and release silver bullets. Building a process-focused culture and monitoring and responding to process exceptions predicts top-levels of performance more than the many of the industry recognized best practices in these areas.

This is very much in line with an agile approach to software development.

While the focus of much of the literature on agile methods is about single codeline development, 'Release Management' often refers not merely to managing a single release, but to the overall discipline and methods of managing multiple releases at the same time. There are all sorts of valid business reasons why you might need to support multiple releases. Maintaining multiple codelines doesn't always mean you're not being agile - *particularly if each one of those is generating (or preventing the loss of) additional revenue.*

While it may seem that having only a single stream of development is very restrictive, in practice it can work surprisingly well. That generally means having to deal with a change or a fix where you first have to determine which releases need it, and then how to inject it into (propagate to) possibly multiple release streams. For these cases, agile teams can use a traditional Release Line approach, with an agile twist.

An agile process also makes it easier to manage the disruptions to day-to-day development caused by bug fixes. The key to managing multiple release streams in an agile environment is to prioritize all the work for a team together. The product owner needs to decide how important a "bug fix" is compared to a feature. Agile planning techniques make the cost obvious. All teams, not just agile ones, work best in an environment where change is managed. When additional work is introduced mid-iteration, the performance of the team can suffer. The iteration approach of agile methods gives the product owner a choice when an issue presents itself: "do I fix this immediately, and forego feature work, or do I wait until the next iteration (2-4 weeks hence)."

Laura Wingerd discusses some of the basic "rules of the road" for "channeling the flow of change in software development collaboration" in [chapter 7 of her book](#), Practical Perforce. where she discusses the "flow of change rules" and the "tofu scale". The Tofu scale and change-flow rules/protocol are concerned with the relationships between codeline policies across the entire branching structure when it comes to making decisions about stability -vs- speed: One codeline's policy might make a certain tradeoff, but it is the presence of *multiple* codelines and how they work together, and how their policies define the overall flow of change across codelines, that is key to release management across multiple releases+codelines. These change-flow rules (which also relate to the *Mainline* pattern) advise us on exactly how and when to do this. The change-flow rules are also well aligned with Lean principles ideas of minimizing the form of waste known as "inventory" or "work-in-progress" - functionality that has been developed but not yet released.

We have had experiences where applying an agile approach helped manage the flow of bug fixes. One team commented that their agile method involved a 2 weekly release cycle and they didn't need to rush bug fixes out - it had always been OK to wait for the next release. A key factor in the success of their development method was the trust that the rest of the organization had in the process - work got done (and released), and if a feature was scheduled to go into a release it almost invariably did.

While agile teams pride themselves on their ability to be responsive, being agile does not mean being chaotic and undisciplined. Change has its cost, and agile methods provide ways of making the cost of change explicit. An agile project works best when there is some sort of rhythm for release cycles. Notice that we mentioned that a codeline is "potentially shippable." Whether or not to ship is a business decision. (Which is how it should be!) Even though the codeline is always supposed to be "potentially shippable" all throughout an iteration, the decision to ship (or not) occurs at the end of an iteration. This is important, and emphasizes that the rumors that agile processes are chaotic are not true. You need to plan your releases and iterations to align or you risk hurting the efficiency of the team.

### **Agile Release Management Enablers**

They key enabling patterns for an agile project are

- Private Build
- Integration Build
- Release Build
- Continuous Integration (with automated tests)

- Release Line
- Release-prep Codeline
- Active Development Line

A *Private Build* enables all team members to test any changes they make with some degree of confidence that they will not break the codeline. As part of the private build appropriate tests are run that give some degree of confidence that the code works. Because of the Private Build pattern team members can commit code often and thus have only small integration issues.

Once changes are committed an Integration Build serves as a gatekeeper for more exhaustive verification. The integration build is run on a build server that may be using some sort of *Continuous Integration* tool such as Anthill, Cruise Control or Team City. Since the exhaustive testing is run asynchronously from the development team, tests that you expect will pass will not delay commits. An agile team often has a discipline that a failed Integration Build is an "all Hands" even, thus ensuring that integration problems are addressed quickly.

The combination of small commits, and frequent integration means that the cost of change is small, enabling rapid development.

A *Release Build* and a *Release-Line* may seem like odd beasts to mention in an agile context. Release builds typically have a few extra requirements that aren't strictly necessary for *all* Integration Builds (see our our October 2003 article on "[Build Management for an Agile Team](#)"). And Release Lines are structured to have policies that limit change because you want to be extra cautious with released software. On the other hand, having a Release Line frees the *Active Development Line* to move forward and risk mistakes. Likewise, moving "almost ready to ship" code to a *Release Prep Codeline* means that new feature work can move forward, while still enabling the team to address integration issues that may come up in final testing. While an ideal agile team will not use a Release Prep Codeline, it is a very useful pattern in practice.

Also the agile mindset is to think about what is possible, and decide whether or not to actually do something based on cost and relative value. Not to attempt improvements because the changes seem unattainable is an obstacle to improving your release process

Most any team can benefit from incorporating some of what we call Agile SCM Practices into their release management plan.

### **Common Questions about Agile Releases**

Like many processes that work, an agile approach to release management raises some questions. Here are some common ones, along with some answers.

What does "deliver at the end of each iteration really mean? Being able to deploy or deliver working software at the end of each iteration is an agile principle. It is also a principle that many projects, agile and not, seem to treat as unreasonable. Let's talk about how you can think about this practice in a way that makes practical sense, and also address some common concerns people have about this idea.

What if my customer can't accept changes every 2 or 4 weeks? The fact that you have "Working Software" does not mean that you need to actually deploy it into production. The benefit of having software in a working state is that you can make the "release" decision at each iteration, as business needs dictate.

### **Documentation, Integration Testing and the Meaning of Done**

Many organizations worry that they can't have a fully shippable release each iteration. There is documentation. One thing your team needs to decide on when adopting an agile process is what it means to be "done" with a feature for an iteration. When you are using a non-agile process, "done" may mean that you have a complete distribution package. This is an excellent goal for an agile team, and it is attainable, but starting out with this as your done threshold may end up frustrating you. Agile practices are an excellent tool for risk management, so consider what the riskiest parts of your application are: Is it documentation, or is it the software?

A perfectly valid definition of complete software is:

- All visible features complete and stable
- All in progress features implemented in a way that does not break existing functionality and which can be hidden if needed.

There are also ways that you can get closer to a completed documentation set. While certain things, like final screen shots, may not be able to be done until a product is "finished" there is much that you can do, such as the basics of online help, overview documentation, etc. Part of the challenge of adopting an agile approach is adopting an agile attitude. The usual question one asks is to emphasize what is impossible. An agile attitude is to ask what is possible.

If your product lifecycle permits, you can dedicate one iteration to completing the artifacts necessary for shipping the application and performing release stabilization if needed. A stabilization iteration is a short iteration focused on final integration testing, documentation, and the like. While not strictly part of an agile process (since you should be doing all of these things as you go), a stabilization iteration is a useful tool when you are transitioning to agile or when you have a demand for extensive supporting activities that need to be done after the bulk of the feature work is done. You would do your stabilization work based off of a Release Prep Codeline.

Another common concern is how to address a need for complicated manual integration testing. An agile approach is also pragmatic. An agile approach also highlights potential problems with your architecture and development process. Teams also have a need for complicated (and sometimes) manual integration testing because their architecture is complicated and they have not developed their code in a way that enables automated testing. Strive to have as much automated integration testing as possible during your iteration. Then use a stabilization iteration approach to do any remaining required testing.

The ITIL framework in its new V3 release now covers release management as part of the Service Transition book. One of the factors for many organizations is that the process of releasing requires handing over to operations, and integration with existing live systems and services. In such circumstances, documentation and ongoing maintainability are very important. A related issue is that in many organizations you are not starting to develop a totally new application. You are instead enhancing an existing application and delivering some extra functionality. How many project teams spend the first part of their development trying to understand the existing system and its interfaces and dependencies, before being able to reliably change it. And then the project team disbands, a new one comes along and the whole wasteful process is repeated. So in these situations the appropriate level of documentation is vital. One advantage of development methods which use lots of automation around testing (unit tests, automated system and acceptance tests) is that as long as the tests are maintained, they provide a very real source of documentation and also a safety net for future changes - being able to prove you have added functionality without breaking existing functionality.

## Summary

Agility is about delivering value predictably. Taking an agile approach to your Release Planning process will help you meet the goals in your release plan and allow you to be more confident in your commitments.

---

**Brad Appleton** is an enterprise SCM/ALM solution architect for a Fortune 100 technology company. He is co-author of **Software Configuration Management Patterns: Effective Teamwork, Practical Integration**, the "Agile SCM" column in CMCrossroads.com's CM Journal, and a former section editor for The C++ Report. Since 1987, Brad has extensive experience using, developing, and supporting SCM environments for teams of all shapes and sizes. 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](mailto:brad@bradapp.net) [brad@bradapp.net](mailto:brad@bradapp.net) [brad@bradapp.net](mailto:brad@bradapp.net)

**Robert Cowham** has been in software development for over 20 years in roles ranging from programming to project management. He continues his involvement in development projects but spends most of his time on SCM Consultancy and Training. He is the Chair of the Configuration Management Specialist Group of the British Computer Society, has a BSc in Computer Science from Edinburgh University and is a Chartered Engineer (CEng MBCS CITP). You can contact him at [rc@vaccaperna.co.uk](mailto:rc@vaccaperna.co.uk) [rc@vaccaperna.co.uk](mailto:rc@vaccaperna.co.uk) [rc@vaccaperna.co.uk](mailto:rc@vaccaperna.co.uk) [rc@vaccaperna.co.uk](mailto:rc@vaccaperna.co.uk)

**Steve Berczuk** is a Technical Lead for an Agile Software Development consulting company. He has been developing 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** 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](mailto:steve@berczuk.com) [steve@berczuk.com](mailto:steve@berczuk.com) [steve@berczuk.com](mailto:steve@berczuk.com) [steve@berczuk.com](mailto:steve@berczuk.com) .



## Trackback(0)

 [TrackBack URI for this entry](#)

## Comments (1)




 [Subscribe to this comment's feed](#)


...

written by [PM Hut](#) , May 28, 2008

An excellent and comprehensive article. I like the Agile Principles section a lot.

Votes: +0

 vote up  vote down  report abuse



[Close Window](#)