TIP Publication Date
July 16, 2014

The Agile Manager

As Quickly as Possible, but no Faster

Agile Software Development is now thirteen years old. It started with the 2001 Agile Manifesto[1], although the principles of incremental software development appeared as early as 1957. You may have heard of the various dialects that joined under the Agile umbrella: Rational Unified Process, Scrum, Crystal Clear, Extreme Programming, Adaptive Software Development, Feature Driven Development, and Dynamic Systems Development Method. One of the reasons Agile methods have endured is because they work. And even though Agile was formulated as a software development methodology, elements work elsewhere in the ecosystem in which software development lives. DevOps[2] is one example, born of the realization that successfully developing software more rapidly than the rest of the enterprise can safely deploy and support is still a failure—repeatedly winning territory that cannot be held by the rest of the army, if you will.

Another arena where Agile works is management. This is an interesting observation because some fervent proponents of Agile dialect(s) insist that their methods are entirely self-directed and need no management. Nevertheless, regardless whether Agile software developers benefit from managers, I have found that managers benefit from Agile. My first clue was reading an article, back when Digital was known as DEC and was the second largest computer company behind IBM, about how they succeeded in building the Alpha microprocessor. Alpha was the first new processor successfully brought to production, by a company other than Intel, in years. DEC pointed out that a crucial factor in their success was "managing managers like a project". The Alpha team laid out every necessary management decision on a timeline, synchronized with the development schedule, and explicitly managed that timeline to ensure the decisions were made in time to support the developers without pause. Truly, Alpha succeeded because managers made decisions "as quickly as possible, but no faster".

[1] *http://agilemanifesto.org*
[2] *http://en.wikipedia.org/wiki/DevOps*

The Agile Manifesto

The Agile Manifesto reads, in its entirety, as follows:

We are uncovering better ways of developing software by doing it and helping others do it. Through this work we have come to value:

• Individuals and interactions	over	Processes and tools
• Working software	over	Comprehensive documentation
• Customer collaboration	over	Contract negotiation
• Responding to change	over	Following a plan

That is, while there is value in the items on the right, we value the items on the left more.

The signatories are: Kent Beck, Mike Beedle, Arie van Bennekum, Alistair Cockburn, Ward Cunningham, Martin Fowler, James Grenning, Jim Highsmith, Andrew Hunt, Ron Jeffries, Jon Kern, Brian Marick, Robert C. Martin, Steve Mellor, Ken Schwaber, Jeff Sutherland, and Dave Thomas.[3]

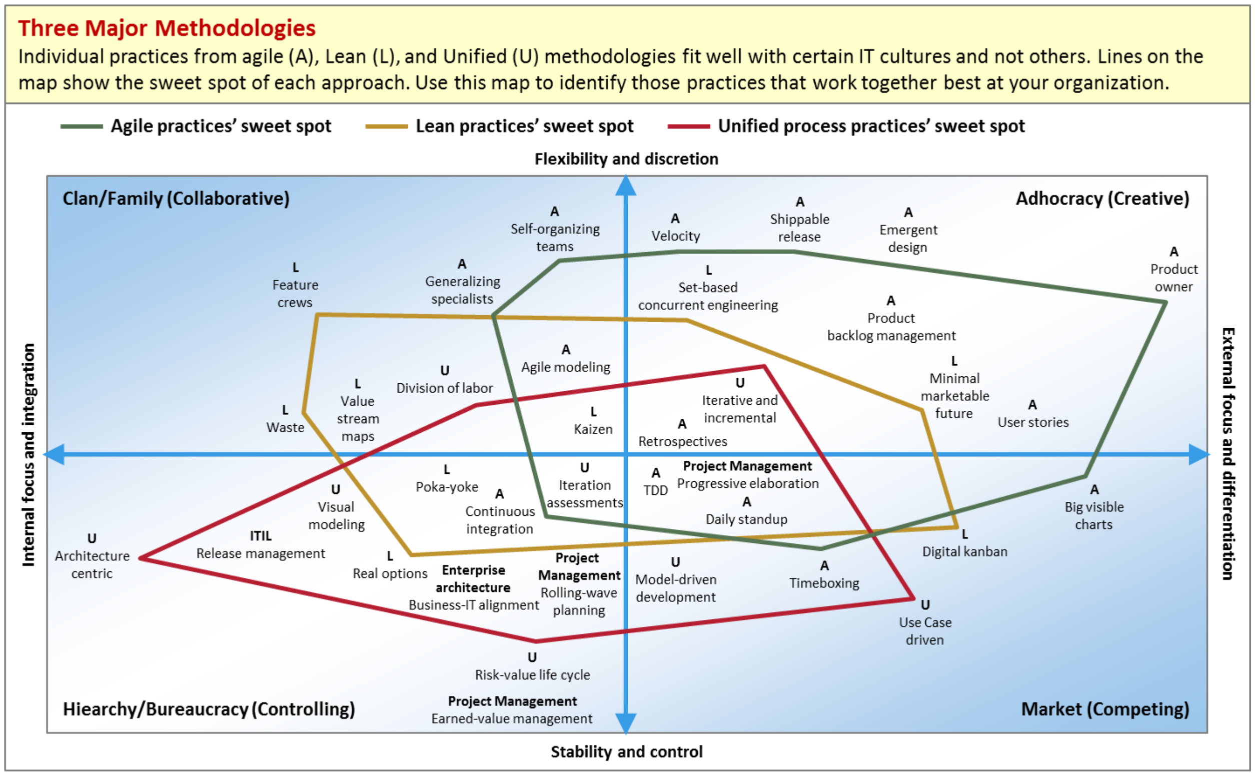
The lesson in the Manifesto is a familiar one to managers, who live with ambiguity on a daily basis: everything is a top priority (sound familiar?), but whenever there is a choice, favor the left-hand value over the right-hand value. And, in fact, successful managers reading this far will already agree with some of the value choices (and undoubtedly be ready to argue with others!).

Managers are familiar with another lesson: keep it simple. The Agile Manifesto was signed by major names in the software development field, many of whom developed their own methods, which are simpatico at the highest level but differ (sometimes contradict) in the details.

[3] © 2001, the above authors. This declaration may be freely copied in any form, but only in its entirety through this notice.

Keeping It Simple

A few years ago, I found a study that clearly articulated what I have found to be true in my own experience: that there are a relatively small number of common factors among various successful Agile dialects. The study results are summarized in the following diagram[4]:


Three Major Methodologies

Individual practices from agile (A), Lean (L), and Unified (U) methodologies fit well with certain IT cultures and not others. Lines on the map show the sweet spot of each approach. Use this map to identify those practices that work together best at your organization.

Agile practices' sweet spot
Lean practices' sweet spot
Unified process practices' sweet spot
Flexibility and discretion
Stability and control
Internal focus and integration
External focus and differentiation
Clan/Family (Collaborative)
Adhocracy (Creative)
Hiearchy/Bureaucracy (Controlling)
Market (Competing)
A Self-organizing teams
A Velocity
A Shippable release
A Emergent design
A Product owner
L Feature crews
A Generalizing specialists
L Set-based concurrent engineering
A Product backlog management
A Agile modeling
U Division of labor
L Value stream maps
L Waste
L Kaizen
U Iterative and incremental
A Retrospectives
L Minimal marketable future
A User stories
L Poka-yoke
U Iteration assessments
A TDD
Project Management Progressive elaboration
A Daily standup
U Visual modeling
A Continuous integration
A Big visible charts
U Architecture centric
ITIL Release management
L Real options
Enterprise architecture Business-IT alignment
Project Management Rolling-wave planning
U Model-driven development
A Timeboxing
L Digital kanban
U Use Case driven
U Risk-value life cycle
Project Management Earned-value management

The sweet spot of practices among these three Agile dialects can be further distilled into four fundamental and key practices:

1. Iterative and Incremental
2. Daily Standups
3. Regular Retrospectives
4. Test-driven Development

Let us examine how these fundamentals can be applied by managers to succeed where others fail.

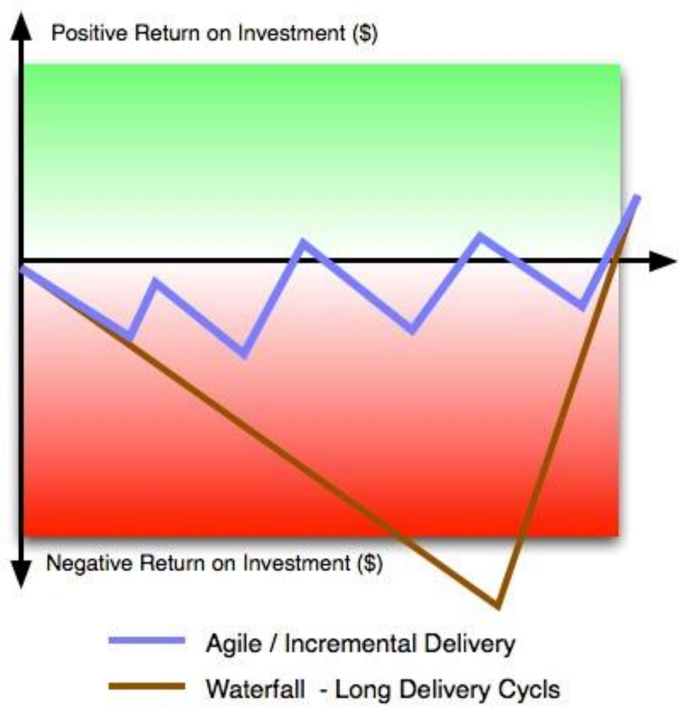
Iterative and Incremental

When I worked at one of the top ten banks, I was responsible for IT due diligence during mergers and acquisitions. When I took on the role, I found the process took three weeks, based on a ten-page questionnaire passed back and forth between the bank and the acquisition. I

[4] *http://www.slideshare.net/MEL1971/pragmatic-development-7912159*

shortened the cycle to three days by splitting the questions into three parts, spread across three half-day sessions. I personally flew out to handle the assessment, and worked through the first batch of questions with the executive team. The purpose of this first session was to determine which areas needed probing, and who at the next level down had the answer. The next half-day session, unsurprisingly, repeated that process; the final session was for verification of what the VPs and Directors said.

So far, this process sounds obvious; these assessment questionnaires commonly are arranged in parts, and typically delivered as an interactive application. What is different is the human interaction that is favored over tools in Agile. I went in person, and actually invested less time overall compared to the previous three-week benchmark. I not only determined the appropriate questions, I worked my way down the management chain to the appropriate people. And in the end, my assessments were higher quality than a questionnaire, since I could diverge from the script to probe as indicated. In short, I *iterated* through three cycles, probing *incrementally* deeper into the appropriate details, resulting in better results in less time. Borrowing a graphic[5] from Agile Software Development illustrates this:



Reading "Investment" as "time" instead of "$", the message is clear: if I do my assessment as a single block of time with a single do-it-all questionnaire and/or single meeting, the longer it takes and the more I have to "invest" of my (and the other folks!) time before getting anything of value back. Instead of trying to do it all at once, I identify smaller blocks that show progress and also permit tweaking attendees and/or questions in the next phase. Overall, total time spent is smaller and value appears sooner.

Also, the discerning reader noticed I said three days, but only accounted for 1.5 days. I used an initial day to research and prepare the "story curve" of my questions. I used the final half-day to gather the management team back together to explain my findings, and I also described how they could readily mitigate their risks, according to the "Regular Retrospectives" practice discussed below. This resulted in additional time and effort savings later in the M&A process.

[5] *http://leanthinkingyoursoftware.blogspot.com/2010/08/how-deep-are-your-pockets.html*

Daily Standups

When I worked at one of the top five healthcare insurance firms, I was asked to manage a transition from one large IT contractor to two others. The switch made good business sense at the senior-most management levels, but created considerable disruption in project schedules and IT commitments due to the sheer size of the deal, training, and skill challenges. My first step was to institute daily meetings with the top executives from each vendor to discuss three things:

1. What was not done yesterday, as planned?
2. What is planned to be done today?
3. What is preventing ("blocking") progress?

Despite my invitation, only one vendor attended; the executive from the other vendor did not see a need to do so. At the end of two weeks, the executive who attended told me that these twenty minutes each morning were the most valuable of his entire day: he heard of issues before his own team told him, and he then had the entire day to work the issues before they grew any bigger. The executive who chose not to attend was mired in contract scope discussions and standardized status reports; his company fell behind until his lieutenant adopted my methods and their performance improved.

In Agile terms, what we implemented at a very senior level favored:

• Individuals and interactions	over	Processes and tools
• Customer collaboration	over	Contract negotiation
• Responding to change	over	Following a plan

Regular Retrospectives

Whether it is called a post mortem, root-cause analysis, or retrospective, this best practice is about stopping long enough to ask:

- What went right?
- What went wrong?
- What could/should be done better next time?

The particular Agile perspective, however, reminds a good manager to do it regularly and often. I do not see this as a particularly profound concept—Weight Watchers knows the same "secret", after all. The difficulty, just like with dieting, is actually doing what you know is right. The key to succeeding at this practice, as Weight Watchers found with dieting, is to incorporate it over time, in small incremental steps, into every one-on-one, every team meeting, every skip-level meeting … and to encourage others to do it with you so you create a supportive network to sustain the practice.

Test-Driven Development

The concept of TDD sounded like cheating to me, the first time I encountered it. It means that a coder should write her test jig first, then repeatedly run her code against it, fixing where it fails until it passes. Back in school "learning to the test" was considered poor practice, if not outright subverting the educational process. But the software development world is not academia, and it turns out that TDD does facilitate the success "on time, on budget, on target".

In the world of management, the "product" being delivered is often a decision—the right decision for the situation, rendered at the right time. When DEC was delivering its Alpha chip, managers needed to be explicitly herded toward decisions in the right order, at the right time, to keep the groundbreaking project on track. At my bank, decisions mired in endless discussion sapped the time of senior management and teams alike, reducing the productivity of both.

Amazon and Google have mastered the art of test-driven decision-making. As a rule, neither company wastes much time in debate; rather, they implement a quick "A/B test" on their site(s), diverting a statistically appropriate number of users to each option, and measure the result. This practice, also called "data-driven decisions", repeatedly tests management decisions against reality, until they result in measurable improvement or progress. This method of management clearly follows the Agile practice of favoring customer collaboration.

Conclusion

Full disclosure, I never even heard of Agile until six years ago. I just developed what I thought of as my "bag of tricks" over years of working with really smart people. Only then did I discover that others had collected the same tricks and called it Agile (for software development). My contribution to Agile was applying these principles to management practices.[6] From that eclectic background, then, I propose four key concepts that you can blend into your management style to improve your effectiveness—*iterative* and *incremental*, *daily standups*, *regular retrospectives*, and *test-driven development*—and justify their effectiveness both from rigorous methodological frameworks and "just doing what works".

This TIP was written by: Glenn Kapetansky – experienced in architecting systems, processes, and organizations that endure. Glenn welcomes comments and discussion on this topic and can be reached at glenn.kapetansky@trexin.com

[6] At approximately the same time as I was refining my Agile Management thoughts, Jim Highsmith was writing his excellent book Agile Project Management. His treatment is far more rigorous than mine, and I highly recommend his work for further reading.