Agile Development in Government:

Myths, Monsters, and Fables

September 2016 ed.

By David Carney, Suzanne Miller, and Mary Ann Lapham

Software Engineering Institute | Carnegie Mellon University

Foreword

This little volume is a somewhat tongue-in-cheek reflection on certain attitudes toward Agile software development now current in the government workplace. In an effort to bring these attitudes into relief, we have consciously avoided the standard trappings of a Software Engineering Institute "technical report" or any of the other standard report types that we usually use to capture our research. Instead, we offer some humorous glimpses of these attitudes and their basis in fact.

But let it be clear at the outset: the authors are not apostles preaching for the Agile approach. We have studied it, have observed its strengths, and are well aware of its weaknesses. Our goal is merely to cast some light on some inaccuracies and misunderstandings about Agile in a light-hearted manner.

The title derives in part from various websites that use the metaphor of a "monster" when speaking of Agile. For instance, on one website discussing Agile code design,[1] the writer notes that: "Like a monster, Agile software methodologies scared the hell out of us." Another, a site called "Fighting the Monster," describes the trials and tribulations the author endured while pleading that Agile could safely be used in programs involving legacy databases.[2]

But a far more significant reason for our title is that there are many untrue assertions commonly being made about the Agile approach, and we feel that these should be corrected. We term these inaccuracies "Myths," and wish, with this little volume, to puncture them. Again: we are not trying to push Agile on the unwilling. But we are seeking to set some records straight. One compelling reason is that there is a common realization throughout the Department of Defense (DoD) that present-day acquisition practice is deeply, perhaps fatally flawed.[3] Numerous attempts at reform have had little success, and a report by J. Ronald Fox, written at the request of the Center of Military History, offers damning evidence of how consistently these reform attempts have failed.[4] Published in 2011, the report studies numerous acquisition reform attempts over several decades, and concludes that there is little hope of solving the chronic problems if the usual attempts at reform are tried once again. And his conclusion is really no surprise: there is widespread agreement within the DoD community that "something *different* needs to be done."

Well, the Agile approach seems to be something that fits that bill pretty well. No, it is not a panacea; like any new practice, it takes thoughtful adoption planning to decide when it should be used. And its usefulness is highly specific – there are many DoD projects for which Agile would be a poor fit. But we believe that, for any program that wants to give it a try, it might be a perfectly reasonable course of action; that is the essence of our message. Unfortunately, however, we have, on several occasions in the recent past, observed programs that were specifically prevented from doing so, and the justification was consistently one or more of the "myths" we describe herein. Hence this little document.

We note, in describing our list of Agile myths, that this ground is well traveled: there is a very large number of documents floating around on the Web that list and debunk Agile myths. A quick Google search on the string "Agile myths," for instance, will produce literally dozens of sites that list particular Agile myths and then describe, often in excellent detail, why they are false. (Interestingly, no one agrees on how many myths should be punctured; the lists from just the first page of the Google search vary from five to twelve myths, with seven the most popular number.) So why are we adding to this already crowded space?

Mostly because these myths persist, no matter how eloquently they have been shown to be false, no matter how much data has been produced to set the record straight. And the many, many technical reports that have been written with the same purpose of beating down these myths have not achieved their goals, at least not in the manner that their authors (ourselves included) have intended.

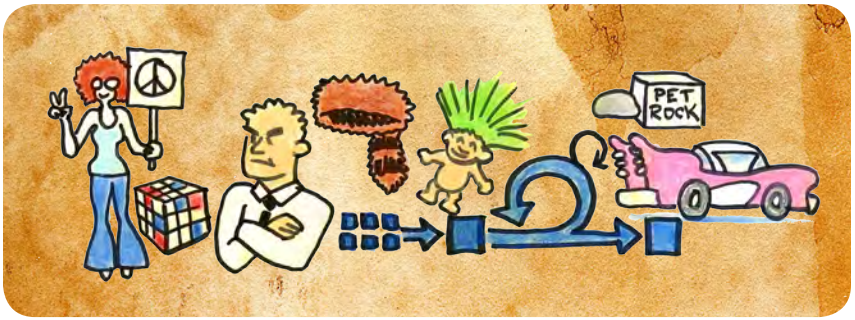
Thus this little paper, admittedly written in search of comic relief. We are hoping that by bringing a rather jocular approach to the fore, we might also bring a few smiles to our readers, and perhaps even persuade some folks to ponder the words a bit more thoughtfully.

And finally, and purely to add an additional touch of humor, we append one of Aesop's "Fables" to each discussion. Why? His fables, although seemingly light, folkish tales, are often jewels of wisdom. And as Aesop himself reminds us, in the fable of the Fisherman:

To do the right thing at the right time is a great art.

PS: like many of the other mythbusters, we chose the number seven. Why? Well just think of all the nifty things of which there are seven—Seas, Continents, Dwarves, Hills of Rome, Wonders of the World…..

Myth #1: Agile is a Fad—If I Wait Long Enough, It Will Go Away



This myth[5] is the perhaps the most widespread, whether concerning Agile or any other unfamiliar practice. The myth is that just about *any* novel software development approach, especially if it does not appear to bear sufficient conformance to present acquisition norms, is somehow a passing fad that will soon be replaced by the next "latest thing." This viewpoint is supported by a number of such fads in DoD history that really did appear with great fanfare only to evaporate very soon afterwards. During the mid-1980s, Ada was touted as the salvation of DoD programs, and would bring untold efficiencies to software programs. After a few short years, Ada quietly checked out of the Pentagon.[6] Several years later, CASE tools and integrated CASE environments, notably such efforts as "I-CASE," were begun with trumpets and hurrahs, limped along for several years, and were then sent to oblivion—usually at great cost—and with little to show except ongoing frustration from end-users.

Thus there is ample evidence that fads in software do exist, that they are costly, and that such fads should be avoided. But not everything that is new is a fad. After all, software, and the ideal methods for its development, is barely a few decades old, and the field is still obviously in a state of maturation. What makes the maturation process so hard is that, in large organizations such as the DoD, and given such urgent need for aid and succor, it is inevitable that *any* new thing that comes down the pike will immediately be seen by many as having

5 See http://www.versionone.com/pdf/AgileMyths_BetterSoftware.pdf for another valuable discussion of this issue.

6 But she did leave a forwarding address: Ada has not disappeared. On the contrary, its present status will surprise anyone who believes that "Ada is dead." It is a widely used language in many domains that need hard real-time capabilities. A glance at http://www.seas.gwu.edu/~mfeldman/ada-project-summary.html will demonstrate quite convincingly that Ada is not at all dead, but is a *very* lively lady indeed, despite the failure of its adoption in DoD acquisition.

messianic powers to suddenly bring acquisition practice to Nirvana. So any new engineering approach (e.g., Agile practice) will be first hailed as a silver bullet, and then the reaction will soon occur that labels it a fad.

So, although everyone agrees that "something different needs to be done," the belief that no new development practices can be trusted almost inevitably comes to the conclusion that "…yes, but not this new thing, not Agile, because it's just a fad." Aiding and abetting this contradictory belief is the natural resistance to change found in any large organization, and the fear of disruption will typically result in accusations of faddishness, whether warranted or not.[7]

If we consider this dilemma, it becomes obvious that we need to figure out how to tell the real from the false, how to discern whether New Approach X will truly bring substantial value to DoD programs, or whether New Approach X, while shiny and glistening at first glance, will turn out to be yet another waste of our fast-diminishing resources. We need the kind of discerning wisdom that Shakespeare had in mind in "The Merchant of Venice," when the Prince of Morocco says:

Men that hazard all do it in hope of fair advantages: A golden mind stoops not to shows of dross.

We need to learn to tell the gold from the dross, because as we noted above, everyone knows that "something different really, really needs to be done" to get acquisition to a healthy state. But we're presently moribund, afraid that we'll wander down the path of another Ada. So how can we solve this problem?

Common sense suggests that one thing that is needed is evidence: tangible indications of genuine success; believable indications that others within the wider community are using the new approach not as a faddish practice but in a real, hard, engineering manner. And, in fact, there is such evidence that Agile practice is proving itself a reliable, useful, and valuable approach in the right

7 During the 1990s, a very highly placed executive, in a very public memo, was caustically distrustful of the "latest new thing" that was appearing: the notion of a "service" as a software approach was being born, and the attendant concept of a "service-oriented architecture" was soon moving quickly through the software engineering community. Said the executive disdainfully: "We're definitely NOT going to get involved with 'services' here – they're just the latest New Thing." Now, two decades later, whether or not there are any service-based systems in that guy's organization is unclear, but there seem to be a whole lot of other people all over the world doing a whole lot of service-oriented stuff…

context; it is demonstrating that the accusations of "faddishness" are proving false. Organizations as diverse as PayPal, Philips Respironics, Intel Corporation, AFLAC, Nissan North America, and the Social Security Administration are corporate members of the Agile Alliance[8], an organization whose mission is to "support those who explore and apply Agile values, principles, and practices to make the software profession productive, humane, and sustainable."[9]

The Agile Alliance organizes annual practitioner conferences for sharing knowledge of techniques and experience, and a number of educational, technical, and diversity initiatives.

The Project Management Institute has also launched a PMI-Agile Certified Practitioner (PMI-ACPSM) professional certification.[10]

Thus, we revisit the myth that started this discussion: "Agile is a fad—if I wait long enough, it will go away." Our response is: Bosh! That statement is untrue: Agile is a useful software development methodology that can be useful for many projects of many types. To repeat: we are not advocating for Agile in and of itself. There are many, many projects for which Agile would be inappropriate. But there also many, many projects where it would indeed be the most appropriate method to use. And we have observed directors of such projects shun using Agile for the very reason we describe, the belief that it is nothing more than a fad.

To that, we reply: the longer DoD waits to make use of proven new approaches such as Agile software development, the more DoD will condemn itself to prolonging the present state of failing acquisition practice. The myth about Agile's faddishness has been punctured by the facts, and it's time to realize that.

Our Fable: Hercules and the Waggoner

A Waggoner was once driving a heavy load along a very muddy way. At last he came to a part of the road where the wheels sank halfway into the mire, and the more the horses pulled, the deeper sank the wheels. So the Waggoner threw down his whip, and knelt down and prayed to Hercules the Strong. "O Hercules, help me in this my hour of distress," said he. But Hercules appeared to him, and said: "Tut, man, don't sprawl there. Get up and put your shoulder to the wheel."

The gods help them that help themselves.

Myth #2: Agile Teams Don't Document Anything

This myth arises from a basic premise that has been part of the Agile community from its very inception. The seminal Agile concepts were embedded in the Agile Manifesto, which set forth the basic principles of the approach in 2001. Introducing the manifesto on behalf of the Agile Alliance, Jim Highsmith commented that the Agile movement was not opposed to methodology; in his comments, he directly addresses the issue of documentation:

> The Agile movement is not anti-methodology, in fact, many of us want to restore credibility to the word methodology. We want to restore a balance. We embrace modeling, but not in order to file some diagram in a dusty corporate repository. **We embrace documentation, but not hundreds of pages of never-maintained and rarely-used tomes**.... [Emphasis ours]

Thus he established the essential Agile view of documentation: provide just enough documentation, but don't provide documentation just for its own sake.

As can easily be imagined, this philosophy found little resonance in the government community, where exhaustive documentation is a way of life. Numerous assertions were made, as a result, that Agile teams ignored documentation; the following describes one well-known attack:

In a letter to IEEE Computer, Steven Rakitin expressed cynicism about agile development, calling an article supporting agile software development "yet another attempt to undermine the discipline of software engineering" and translating "Working software over comprehensive documentation" as "We want to spend all our time coding. Remember, real programmers don't write documentation."[11]

But as a rebuttal to this attack, Agile developers asserted that they should write documentation if that's the best way to achieve the relevant goals, but that there are often better ways to achieve those goals than writing static documentation. Scott Ambler states that documentation should be

"Just Barely Good Enough" (JBGE), that too much or comprehensive documentation would usually cause waste, and developers rarely trust detailed documentation because it's usually out of sync with codes, while too little documentation may also cause problems for maintenance, communication, learning and knowledge sharing.[12]

An IBM website provides a highly useful discussion of this question, particularly with regard to large projects.[13] First, they candidly describe the difficulty of documentation on large Agile projects:

The goal on agile projects is to keep documentation as simple as possible, relying on roadmaps, overviews and concepts rather than enterprise-focused details. But what happens when using an agile approach on more complex projects? For example, what if the team that writes the software is different from the team that must maintain it? ….

In such circumstances, they admit the possibility that Agile might come up short. But then, they state:

As agile teams scale and fit into larger enterprise environments, the agilest must devise more mature, but equally agile, documentation strategies.

Thereafter, they propose a useful series of appropriate Agile practices to ensure that "just enough" documentation is produced no matter how large the project. Their summary:

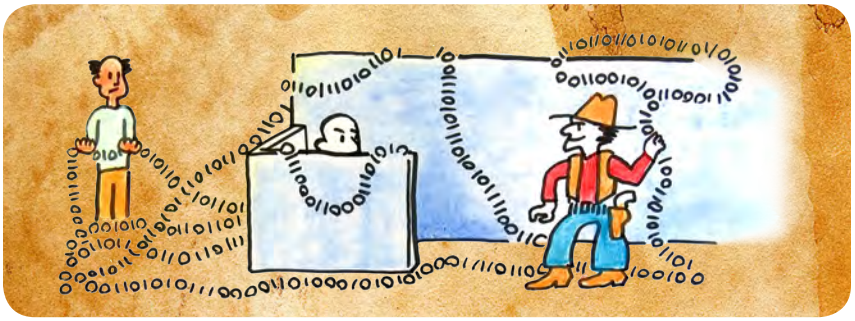
Providing documentation that suits different purposes is possible if you take an agile approach to how you scale. Determine documentation needs as you progress and include the different documentation builds in your iterations and you will provide the right documentation solution for all stakeholders, including maintenance and audits.

Our Fable: The Lioness

A controversy prevailed among the beasts of the field as to which of the animals deserved the most credit for producing the greatest number of whelps at a birth. They rushed clamorously into the presence of the Lioness and demanded of her the settlement of the dispute. "And you," they said, "how many sons have you at a birth?" The Lioness laughed at them, and said: "Why! I have only one; but that one is a thoroughbred Lion."

The value is in the worth, not in the number.

Myth #3: Agile Is Cowboy Programming



This myth is based on the well-known phenomenon of the "Code Cowboy," a software force of nature who cannot be stopped. He or she is almost always a great programmer who can do work two or three times faster than anyone else. The problem is, at least half of that speed comes by cutting corners. The Code Cowboy feels that checking code into source control takes too long, storing configuration data outside of the code itself takes too long, communicating with anyone else takes too long... you get the idea.

The Code Cowboy's code is a spaghetti code mess, because he or she was working so quickly that the needed refactoring never happened. Chances are, seven pages' worth of core functionality looks like the "don't do this" example of a programming textbook, but it magically works. The Code Cowboy definitely does not play well with others. And if you put two Code Cowboys on the same project, it is guaranteed to fail, as they trample on each other's changes and shoot each other in the foot.

If we examine the facts, it is obvious that a good deal of the myth that "Agile is cowboy programming" is related to speed: both the Cowboy and the Agile programmer get results fast. But behind this simple façade, the two could not be more unlike. For the Cowboy, fast is good, period. For the Agile programmer, the goal is not speed in itself but *immediacy*: working side by side with testers and end users, the Agile programmer can, from the very inception of a program, make clear his initial understanding of the required software. By this means, misunderstandings can be corrected at the earliest possible moment; this process of immediate sharing between coders, testers, and users continues throughout a project. In the same manner, a related element of the myth that

is easily debunked is related to teamwork: as noted above, the Cowboy is the antithesis of a team player. By contrast, Agile software development is utterly dependent on teamwork, and a kind of teamwork in which everyone is a peer. Any description of any Agile approach is filled with descriptions of how the participants constantly interact, how *teams* are fluid, how *teams* produce results.

The description of the Code Cowboy above also notes his resistance to standard configuration control practice as a waste of time; equating Cowboys with Agile implicitly accuses both practices as sharing that opinion. A glance at the literature of Agile will reveal how false that assertion is: there is a wealth of books, papers, documents, and websites that provide a clear indication of how important this topic is for the Agile community.

We shall later (in Myth #7) describe an IBM approach to CM in large organizations and programs. In addition, CMCrossRoads, a CM solutions provider, has a similar website with equally useful information on CM practices in Agile development.[14] Websites such as Evocean,[15] AgileConnection,[16] and many others provide comparable (and useful) information. And an article in "The Register," a UK trade journal, titled "Config management: Enemy of agile approach or the reason it works?," makes this telling argument:

> CM is sometimes also seen as the enemy of agile development—but what is the point of deploying a new system quickly if, for instance, it backs out an important emergency fix for a compliance issue—and your business is promptly shut down by the regulators? The secret is "just enough" CM to let the business operate reliably. Good CM is actually fundamental to an agile approach… and should be built into the process—the secret is having "just enough" CM to let the business operate reliably….[17]

And finally, we have already dismissed, in the previous myth, the assertion that the Agile practitioner, like the Code Cowboy, sees no need for documentation. Agile implies less documentation, to be sure. But all the documentation that is really necessary.

Thus on every facet of the claim that Agile equates to Cowboys, the facts are clear—the two are light-years apart, and there is ample evidence to show that. Equating them is simply continuing an unfortunate myth.

Our Fable: The Wolf in Sheep's Clothing

A Wolf found great difficulty in getting at the sheep owing to the vigilance of the shepherd and his dogs. But one day it found the skin of a Sheep that had been flayed and thrown aside, so it put it on over its own pelt and strolled down among the Sheep. The young Lamb that belonged to the Sheep whose skin the Wolf was wearing, thought the Wolf was his mother, and began to follow the Wolf in the Sheep's clothing. So, leading the Lamb a little apart, he soon made a meal of her, and for some time he succeeded in deceiving the sheep, and enjoying hearty meals.

Appearances are deceptive.

Myth #4: Agile Works Only In Co-Located Environments

This myth got an early start, way back when the first descriptions of Agile began to be circulated. People soon had visions of little clusters of people, all looking at a screen together, smiling broadly, pointing out this or that to each other, and so forth.

Well, this image is not all that far wrong, in the sense that such clusters of team members do often occur in Agile projects. The myth part, however, is that that kind of cooperation is the *only* way that an Agile project can unfold—nothing could be more untrue.

First off, let's start with some common sense: for any project—whether it be Agile, or Waterfall, or Spiral—the project is almost always better off if its participants are co-located. Frequent human interaction is a necessary element of Agile, no question, but is no less necessary in Waterfall (e.g., when the designers are trying to make sense of a bunch of haywire or contradictory requirements), or Spiral (e.g., when making a decision concerning some project artifact about how much detail is enough). *All* of those activities, in *all* of those projects, become more straightforward when all of the players are sitting around the same table.

And the lack of co-location can definitely be a serious impediment if a project is poorly managed. For a humorous anecdote from our experience, the authors of this report several years ago were members of a "red team." The team had been tasked by the Pentagon to investigate why a project that had been underway for only ten months was already eight months behind schedule. The project was

a traditional Waterfall project, building a familiar type of system. Hence, we dismissed the notion that unprecedented technology or some wild new design complexity was the problem.

Upon questioning the management team, it became clear that they were having trouble finding qualified programmers. The system was to be written in Ada, and good Ada programmers were, even then, hard to find. "So what do you plan to do about that?" we asked. The reply was that they had found an excellent group of Ada programmers in Boston. (We were currently sitting in California.) "Won't that be expensive to pay their relocation costs?" we asked. The response was that the new programmers would stay in Boston, and the project would simply be a distributed one. "Won't that cause problems?" said we. No, they said, we do it all the time and it works like a charm.

OK, we shrugged, and went on to other topics. When we arrived at testing, they told us it had recently become a difficult issue. "Why?" we asked. *{And here it comes…}* "Well," the Program Manager said, "our lead test guy was working out fine, but they've moved his office into the building across the street, so we can never get hold of him…." *{And then we lost it.}*

So sure, co-location is great. But lots of distributed projects succeed, and succeed quite well. The key as with so many things is good management (which our California example emphatically did not exhibit).

Enough of anecdotes—let's look at some data. VersionOne, an Agile tools and services company, makes an annual survey of the state of Agile-based companies and projects that is well respected; among the data it seeks is information on co-located vs. distributed projects. In 2012, its findings were that "…35% of respondents worked in a company that had distributed software teams. However, those who answered 'yes' had, on average, nine distributed teams."[18]

In the following year, the survey reported that "The number of respondents who have distributed software teams practicing Agile has more than doubled in one year. 76% had distributed software teams in 2013 compared to only 35% in 2012."[19]

Given the ongoing growth of interest in Agile that we see within the community, there is every reason to expect that the 2014 survey will show still another increase in distributed Agile development.

And for another tangible example, one of the Agile projects that has been delivering value to U.S. Air Force customers for more than 10 years, Patriot Excalibur, has been operating with a distributed team for several years—its developers are spread across four different locations in the United States.[20] Access to specialized expertise makes the geographic distribution worthwhile, but the projects managers admit that they have to be mindful of the need to minimize barriers to team communication on an ongoing basis.

So back to our myth. Sure, co-location is probably a preferable way to do things, whether Agile or anything else. But it's not the *only* way, and just as for Spiral or Waterfall projects, a distributed Agile project is more than feasible. And the data shows that it's not just feasible, it's becoming quite frequent. As is true for a project of any type, distribution just calls for careful management, and intelligent awareness of the things that need to be a tad different when some team members are a few thousand miles away. But the evidence shows that it's very, very doable.

Our Fable: The Belly and the Members

The other members of the Body rebelled against the Belly, and said, "Why should we be perpetually engaged in administering to your wants, while you do nothing but take your rest, and enjoy yourself in luxury and self-indulgence?' The members carried out their resolve and refused their assistance to the Belly. The whole Body quickly became debilitated, and the hands, feet, mouth, and eyes, when it was too late, repented of their folly.

Cooperation for the common good means that, even when they are so far apart, the belly, the left hand, and the right hand will work together in harmony.

Myth #5: Agile Is Just Spiral Renamed (Or Iterative, Or Waterfall…)

Old Bill Shakespeare was pretty smart. Among the gazillions of great lines he wrote, one that just might be his most famous line happens when poor Juliet tells Romeo:

What's in a name? That which we call a rose by any other name would smell as sweet.[21]

And, by any measure, he was right (although we're all probably pretty happy that the name we use for that pretty red flower above isn't "flumpwart").

In any case, this myth brings about a two-part problem. First, when it is bandied about within a project, it muddies the water about what the project is actually doing. And second, in a more general sense, it breeds misunderstanding within the software community about what Agile is, what Spiral is, and how these two methodologies differ. (We'll use the Spiral case here, but the same concepts apply if comparing Agile to iterative or Waterfall approaches.)

First, let's start with substance. Agile, in the form of it various method instantiations (e.g., Scrum), is very well documented in multiple media, from Wikipedia to books to You Tube. There is copious information about what, how, why, and just about any other question about what Agile is, what it is not, and different ways to enact the Agile tenets and principles. Spiral software development has an equal amount of specific, detailed information concerning its structure and use. So to assert that one is the other reveals either gross ignorance or willful mischief. And claiming that one is the other demeans both methodologies.

Second, let's consider motivation. Aside from rank ignorance, someone who perpetrates this myth must have some purpose for so doing. One such purpose could be inertia: if management decides that the company will start adopting Agile practices, for instance, this will mean disruption. And a move to Agile definitely means disruption. It means that people will be compelled to stop doing the nice, comfortable things they've been doing for years. Being in a rut isn't all that bad, some people think, precisely because the amount of thinking that they presently have to do is quite small.

A very similar motivation for propagating this myth is fear. Moving a company to a very different development paradigm always brings with it the risk that the existing personnel profile might change, and perhaps radically. People with very limited knowledge of Agile development may well have all sorts of half-baked notions. "Agile means they'll get rid of coders and double the number of testers—and I'm a coder!" "Agile means they'll get rid of all of the testers because the coders do their own testing—and I'm a tester!" And so forth.

There are several different bad results that occur as a result of this myth. One is that people believe it, and so continue with what they've been doing for years, but start calling it Agile. (And then, getting the same poor results they've been getting for years, start trashing Agile practice in pained and loud cries.) Another is they actually do read up on Agile, start out trying to use it, but never make the mental adjustment needed. So just like they used to, in whatever was their old methodology, they start delaying the testing for several weeks ("So that the coders can give the test guys enough to really run some good tests"). They start telling the Product Owner that he should stay away for a few months ("Until they have something good enough to show you"). And, inevitably, they have the same rotten results that the other guys had, and just like them, start Agile-bashing to high heaven.

The bottom line is understanding. Agile is Agile, and Spiral is Spiral. They may not be so different that we would then add "…and never the twain shall meet." But different they certainly are, and for software practitioners of every stripe, if they use those names, they should know what they mean.

Our Fable: The Monkey and the Dolphin

A Sailor, bound on a long voyage, took with him a Monkey to amuse him while shipboard. As he sailed off the coast of Greece, a violent tempest arose in which the ship was wrecked and he, his Monkey, and all the crew were obliged to swim for their lives. A Dolphin saw the Monkey contending with the waves, and supposing him to be a man (whom he is always said to befriend), came and placed himself under him, to convey him on his back in safety to the shore. When the Dolphin arrived with his burden in sight of land not far from Athens, he asked the Monkey if he were an Athenian. The latter replied that he was, and that he was descended from one of the most noble families in that city. The Dolphin then inquired if he knew the Piraeus (the famous harbor of Athens). Supposing that a man was meant, the Monkey answered that he knew him very well and that he was an intimate friend. The Dolphin, indignant at these falsehoods, dipped the Monkey under the water and drowned him.

Those who pretend to be what they are not, sooner or later, find themselves in deep water.

Myth #6: Agile Won't Work in DoD or Government Environments

This one is tough, because it's *soooooooooooooo* pervasive. Even though there's ample evidence to the contrary, this is a mantra that is repeated as gospel from one end of Washington to the other. So let's consider the supposed reasons why Agile won't work for either the suits or the brass.

But first, let's not ignore reality. There are conditions in government IT that are wholly unknown to the commercial sector, and that do form barriers to Agile adoption. For instance, the project controls demanded by traditional government PMOs are incompatible with Agile without mindful tailoring. Demanding detailed project plans prior to doing any development is equally foreign to Agile practice. Everyone knows these things, and many elements within the government are actively working to lower and even eliminate those barriers.

And there are other forces in play that can reduce those barriers. One such is the ongoing budget cuts that began in the recent recession, but are still continuing, particularly for IT. A paper by Peter Stevens ("Mastering the Recession with Lean, Agile and Scrum"[22]) proposes the concept that budget cuts could drive Agile adoption. Indeed, Better Buying Power 3.0 , acquisition guidance released in April 2015, makes the case for reducing cycle time, eliminating unnecessary requirements, and streamlining documentation requirements and staff reviews.[23]

But let's also not ignore another reality: Agile has been used—and successfully used—in government programs, barriers notwithstanding. Examples of how Agile has worked in government are documented in a 2013 report published by the American Council for Technology's Industry Advisory Council [ACT13].

The report indicates successful uses of Agile by several important users:

NASA's Jet Propulsion Lab

- U.S. Department of Veterans Affairs
- CIA
- NTIA
- FCC

Next, let's realize there is significant and growing government support for the use of Agile. Looming large in that area is a recommendation by the Office of Management and Budget, and a subsequent report published by the GAO titled "Effective Practices and Federal Challenges in Applying Agile Methods."[24] The report was written for the following purpose:

Federal agencies depend on IT to support their missions and spent at least $76 billion on IT in fiscal year 2011. However, long-standing congressional interest has contributed to the identification of numerous examples of lengthy IT projects that incurred cost overruns and schedule delays while contributing little to mission-related outcomes. To reduce the risk of such problems, the Office of Management and Budget (OMB) recommends modular software delivery consistent with an approach known as Agile, which calls for producing software in small, short increments. Recently, several agencies have applied Agile practices to their software projects.

A further indication of federal support for Agile has been reported by Information Week, an IT website, where a recent post reported that agencies typically adopt Agile to avoid large-scale failures in systems development programs. The Department of Veterans Affairs (VA), an early adopter of Agile in the federal government, moved to Agile in 2009 for a critical new system (the New GI Bill) when the department was failing on much of the rest of its development portfolio. As a result, VA successfully delivered its first new large-scale system in years, and decided to adopt Agile for the development of a number of other critical systems.[25]

And finally, in terms of all of the regulations, policies, directives, mandates, and so forth, evidence of change are indications that high-level policies are incorporating the new practice in regulations, directives, and the like. The DoDI 5000.02, released in January 2015, includes hybrid lifecycle examples that more easily accommodate Agile methods implementation.[26] The GAO has two projects in work at the time of this writing to provide guidance on scheduling Agile projects in government and costing them. In August 2014 the US Digital Service released the TechFAR Handbook for Procuring Digital Services Using Agile Processes.[27] The TechFAR dispels myths that the use of Agile is inconsistent with the statutory requirements held forth in the FAR: "For each stage of the acquisition lifecycle, this document highlights key regulatory provisions and explains how Agile approaches can be effectively and successfully implemented consistent with core values of public procurement, including impartiality, accountability for results, and providing the best value to the taxpayer."[28] Clearly, they would not invest in these kinds of guidance activities if they did not see continued use of Agile in government settings.

So sure, getting an Agile government program started up takes some doing. But it can be done. There are annoying roadblocks. But many have done it, and many more are getting started.

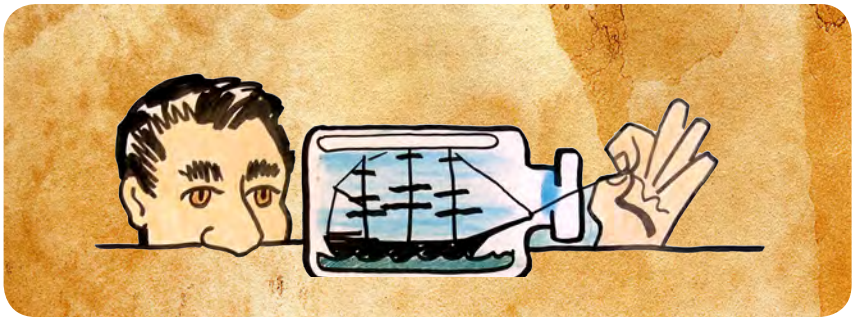
Our Fable: The Mountain in Labor

A Mountain was once greatly agitated. Loud groans and noises were heard, and crowds of people came from all parts to see what was the matter. While they were assembled in anxious expectation of some terrible calamity, out came a Mouse.

Don't make much ado about nothing.

28 In this document, "development" refers to software development from scratch, as well as the configuration and implementation of commercially available off-the-shelf items.

Myth #7: Agile Only Works with Small Projects



This is another myth that dates back to the earliest days. One reason the myth grew was quite natural: the earliest Agile projects did, in fact, tend to be small. Further, with the exception of Rational, the preponderance of the makeup of the original Agile community represented small software organizations. This view was also supported by such authors as Barry Boehm and Rich Turner: in a comparison of different development methods published in 2004,[29] they clearly distinguished Agile from other methods by several characteristics; one of those characteristics was "small number of developers." So historically, at least, the myth has some credence.

But that was then, this is now. A decade and more of experience in performing Agile projects has brought an enormous increase in understanding the many ways in which Agile methods can apply to large projects of many types. On an IBM website, Scott Ambler—a highly regarded author and expert on Agile—states the issue very clearly:

A common misunderstanding about agile software development approaches is that they're only applicable to small, co-located teams. Yes, it's much easier to be successful with small teams, and with co-located teams, and as a result agilists being smart people prefer to work this way. After all, why take on extra risk when you don't need to do so? But, sometimes reality gets in the way and you find yourself in a situation where you need a large team, or you need to distribute your team.... and you would still like to be as agile as possible. The good news is that it's still possible to be agile with a large team, although you will need to go beyond some of the popular "agile in the small" strategies to succeed.[30]

His perspective is echoed by the emergence of several Agile scaling frameworks that are getting more and more press—Dean Leffingwell's Scaled Agile Framework for Enterprises (SAFE), Driving Strategy, Delivering More (DSDM) Agile Project Framework, and Disciplined Agile Delivery (DAD) are among the more well-known approaches that government and commercial users are using for their larger Agile projects. And there is simply too much testimony, from too many highly reliable sources, that Agile can succeed in large projects. For example, Steve Denning, writing for Forbes, makes some telling points that are no less true for government organization than for commercial ones:[31]

> In today's marketplace, [organizations] will need to change their culture or they will die. They need to become Agile….It's true that Agile requires small teams. The reality as Richard Hackman pointed out in his classic book, *Leading Teams: Setting the Stage for Great Performances*, is that all effective teams should be small. A big effective team is an oxymoron. In any event, there are obvious solutions to coping with large projects by dividing the work into a number of relatively independent smaller subprojects then each part can be implemented by an agile team. As explained by Craig Larman and Bas Vodde in their book, *Practices for Scaling Lean & Agile Development: Large, Multisite, and Offshore Product Development with Large-Scale Scrum* (2010).[32]

And there is ample advice on the "how-tos." For example, Ivar Jacobson, one of the creators of the Unified Modeling Language and the Rational Unified Process, offers counsel on large Agile projects; the topics he focuses on include enterprise-scale challenges:[33]

- Multi-site working including off-shore based teams
- Working with outsource partners
- Large project and program-level agility

and enterprise-scale solutions:

- Scalable project team structures
- Risk-focused governance models
- Focus on user experience and business value
- Focus on architecture

Organizations such as Lego, TomTom, Accenture, Intel, John Deere [SAFe][34], Panera, Franklin Templeton Investments, Barclays[35], Cardiff University & Napp Pharmaceuticals[36], have demonstrated successful use of Agile at the large scale on large projects.

Finally, since one stated goal in this little book was humor, no one has taken a better light-hearted approach to this question than Donna Fitzgerald of Gartner:[37]

Question: When is a project too large for Agile project management?

Answer: Since the largest project we know about is the creation of the universe, which we have on good authority only took a single one week sprint (6 days), we think Agile Project and Program Management can scale beyond what most IT projects will ever encounter.

All in all, therefore, we think it's time to put this myth to bed. Large projects can and do use Agile methods. To assert that they cannot is, as was true in all of the other assertions in this book, to perpetuate a myth.

Our Fable:

A Boy put his hand into a pitcher full of filberts.[38] He grasped as many as he could possibly hold, but when he tried to pull out his hand, he was prevented from doing so by the neck of the pitcher. Unwilling to lose his filberts, and yet unable to withdraw his hand, he burst into tears and bitterly lamented his disappointment. A bystander said to him, "Be satisfied with half the quantity, and you will readily draw out your hand."

You may have as large a meal as you wish—but do not attempt too much at once.

38 Another name for hazelnut.

1 http://www.planetgeek.ch/2011/07/08/presentation-agile-code-design-how-to-keep-your-code-flexible/

2 http://gojko.net/2007/11/20/fighting-the-monster/

3 http://www.history.army.mil/catalog/pubs/51/51-3.html

4 Ibid.

5 See http://www.versionone.com/pdf/AgileMyths_BetterSoftware.pdf for another valuable discussion of this issue.

6 But she did leave a forwarding address: Ada has not disappeared. On the contrary, its present status will surprise anyone who believes that "Ada is dead." It is a widely used language in many domains that need hard real-time capabilities. A glance at http://www.seas.gwu.edu/~mfeldman/ada-project-summary.html will demonstrate quite convincingly that Ada is not at all dead, but is a very lively lady indeed, despite the failure of its adoption in DoD acquisition.

7 During the 1990s, a very highly-placed executive, in a very public memo, was caustically distrustful of the "latest new thing" that was appearing: the notion of a "service" as a software approach was being born, and the attendant concept of a "service-oriented architecture" was soon moving quickly through the software engineering community. Said the executive disdainfully: "We're definitely NOT going to get involved with 'services' here—they're just the latest New Thing." Now, two decades later, whether or not there are any service-based systems in that guy's organization is unclear, but there seem to be a whole lot of other people all over the world doing a whole lot of service-oriented stuff…

8 https://www.agilealliance.org/organizations/

9 https://www.agilealliance.org/the-alliance/

10 http://www.pmi.org/certification/agile-management-acp.aspx

11 http://en.wikipedia.org/wiki/Agile_software_development

12 Ibid.

13 http://www.ibm.com/developerworks/rational/agile/agile-documentation-oxymoron/

14 http://www.cmcrossroads.com/article/agile-software-configuration-management-communications-and-documentation

15 http://www.evocean.com/

16 http://www.agileconnection.com

17 http://www.theregister.co.uk/2013/08/22/configuration_management_principles/

18 http://www.versionone.com/pdf/7th-Annual-State-of-Agile-Development-Survey.pdf

19 http://www.versionone.com/pdf/2013-state-of-agile-survey.pdf

20 http://www.dtic.mil/ndia/2011agile/NDIAAgileProcessinDoD.pdf

21 *Romeo and Juliet*, Act II, scene 2.

22 http://www.scrum-breakfast.com/2009/01/mastering-recession-with-lean-agile-and.html

23 http://www.acq.osd.mil/fo/docs/betterBuyingPower3.0(9Apr15).pdf

24 GAO-12-681: Published: Jul 27, 2012. Publicly Released: Jul 27, 2012.

25 http://www.informationweek.com/applications/why-feds-are-embracing-agile/d/d-id/1111116?

26 http://www.dtic.mil/whs/directives/corres/pdf/500002p.pdf

27 https://playbook.cio.gov/techfar/

28 In this document, "development" refers to software development from scratch, as well as the configuration and implementation of commercially available off-the-shelf items.

29 Boehm, B.; R. Turner (2004). Balancing Agility and Discipline: A Guide for the Perplexed. Boston, MA: Addison-Wesley.

30 https://www.ibm.com/developerworks/community/blogs/ambler/entry/large_agile_teams?lang=en

31 http://www.forbes.com/sites/stevedenning/2012/04/17/the-case-against-agile-ten-perennial-management-objections/

32 Ibid.

33 http://www.ivarjacobson.com/enterprise_scale_agile_software_development/

34 http://www.scaledagileframework.com/case-studies/

35 http://www.scottambler.com/page_case_studies.html

36 https://www.dsdm.org/resources/case-studies?topic=All&topic=All

37 http://blogs.gartner.com/donna_fitzgerald/2010/04/20/when-is-a-project-too-large-for-agile-project-management/

38 Another name for hazelnut.

For more information:

For those wanting more information on the SEI's work in adoption of Agile methods in government settings, especially the Department of Defense, please visit our website: http://resources.sei.cmu.edu/library/ and search on "Agile." The left navigation panel will provide pointers to our research papers, blogs, and podcast series.

To provide feedback on this booklet or our other work products, please contact one of the SEI principal investigators for this research area:

Eileen Wrubel (eow@sei.cmu.edu) or
Suzanne Miller (smg@sei.cmu.edu).