Agile Where Agile Fears To Tread!

Thomas M Cagley Jr.
Tom Cagley & Associates LLC
tcagley@tomcagley.com

Bio

Tom Cagley is president of Tom Cagley & Associates. He actively seeks out process problems across the development environment and facilitates problem-solving. Mr. Cagley has over 20 years of experience in the software industry. He has held technical and managerial positions in different industries as a leader in software methods and metrics, quality assurance and systems analysis. Mr. Cagley is a frequent speaker at metrics, quality and project management conferences. His areas of expertise encompass management experience in methods and metrics, quality integration, quality assurance and the application of the SEI's CMMI® to achieve process improvements. Mr. Cagley is the past president of the International Function Point Users Group. He also is an active podcaster, hosting and editing the Software Process and Measurement Cast (www.spamcast.net) and blogger (www.tcagley.wordpress.com).

Abstract

Over the years managers have told many stories about why Agile cannot work and software development using Agile can't be tested. These are excuses to avoid change. They were not valid then and they are not valid now. Taking a less prescriptive definition has allowed teams to create hybrids of lean and Agile techniques to address real-world testing work that spans the entirety of an organization's priorities. Join Tom Cagley as he shares five key scenarios that are cited in which Agile "don't" work. Tom will explore each fallacy and provide insights into practical solutions. He will also explain why testing of all types is often the key to making Agile work, providing a palette of techniques to enable rather than avoid adaptive techniques. You do not want to miss this opportunity to hear how Tom's years of experience in Agile has led to very decisive solutions to address your challenges.

1. Agile Where Agile Fears to Tread

In 1516, a few weeks before the modern project management and new team-based program management approaches, a Bavarian nobleman enacted the Reinheitsgebot[i] (German Beer Purity Law). The law defined the ingredients in beer. Water, hops, and barley was the one true set of ingredients. There are those that want to set and enforce similar purity laws for modern methods and frameworks. For some, Agile is Scrum, Extreme Programming, SAFe, or Kanban. The argument is that you need to pick one but never combine approaches to address your specific culture or scenario. Combining frameworks is a bridge too far. Requiring framework or technique purity is a false requirement, and is often used to eschew less command and control concepts and techniques in certain situations These are five scenarios which are often used as examples where Agile is difficult or daren't go:

1. Mainframe Development and Enhancements
2. Maintenance
3. Commercial Off The Shelf Packages (COTS)
4. Software as a Service (SaaS)
5. Outsourced (Contract) Work

Mainframe: The reasons generally quoted for why Agile can't be used revolve around three macro issues. The first is that large mainframe projects are hard to break up into a component that is deliverable in a short iteration. The second is that much of what goes on in this environment happens behind the scenes. Lots of work occurs in batch processes or in file transfers that have no user interfaces, so its difficult to involve users or the business. Third, when not doing large projects, most teams complete many small enhancements and fixes.

Maintenance: The rationale quoted for why maintenance doesn't fit Agile is typically two-fold. The first is a problem with the difficulty in planning many projects of varied sizes that are typically put into the maintenance bucket. The second and potentially more problematic issue is the emergence of significant production issues that require priority.

COTS: The major issue of fitting Agile into a COTS project is transparency. The firm that builds (or owns the software) has its own process and will deliver what they deliver when they deliver it. A related problem is that most testing is a black box (hard to know what is happening with the code) negating the power of test-driven development.

SaaS: This type of software has concerns similar to COTS with the complication of the software running in the cloud. Why does it matter 'where' the software lives?

Outsourced Work: Issues in this scenario include lack of transparency, contract structure, delivery cadence and potentially testing/acceptance processes.

In each of the five scenarios, the reasons quoted for why Agile can't work have been used over the years to avoid change. They lacked validity then, and they are not valid now. Rather than asking why adaptive approaches won't work, a better question is how can we address the 12 principles in the Manifesto[ii] using a mixture of frameworks and techniques to deliver value. Agile is currently the buzzword *du jour* in IT departments across the world. As the frenzy has grown, the definition of the word has become - fuzzier. The looser definition has allowed teams to create hybrids techniques to address real-world work that spans the entirety of an organization's priorities.

2. Mainframe Projects

Change is never easy when there is fear, the end state is unclear, or when there is a threat to what made you successful in the first place. Doing something different is hard (Karten, 2009), even now when we are well into the Agile age with any number of second and third generation transformations going on. However, many people don't use modern approaches because of the mistaken assumption that frameworks are all or nothing. The first of the five major categories viewed as no-go zones for adaptive and small-scale iterative approaches are mainframe projects.

Tom Henricksen, MyITCareerCoach.com, stated in correspondence[iii], "I have seen resistance to using Agile around mainframe projects. The main issue that comes up is the coordination and communication." In large project scenarios, another issue is that it is hard to break the project up into components that are deliverable in short iterations. There are several solutions to that are useful get started injecting agility into these type of large projects:

1. Adopt Scrum techniques even if you are doing waterfall. Daily standup meetings and retrospectives are examples of techniques that help communication, coordination and process improvement.

2. Focus on flow (Tendon & Muller, 2015), develop a Kanban board that mirrors how work progresses and then track your work. When you SEE bottlenecks hold a retrospective and solve the problem. This step will set up an iterative inspect and adapt loop. Life will start to improve.

3. Where multiple teams participate in group planning events (for example SAFe's PI) provide powerful platforms for coordination. Synchronizing planning also gets everyone on the same schedule.

4. Pair or mini-mob using coders and testers to interpret the requirements and to define tests before coding (test first).

5. Consider borrowing the concept of an architectural runway[iv] from SAFe that provides the architectural, network and technical direction, and components for the development teams. The runway should stay just ahead of the team's(or teams') needs so that they can quickly react. What does quickly mean here? Might be good to define that with data.

6. Use the classic three amigos technique to break the work down into smaller pieces. The piece of work should be small enough to meet a solid definition of done in the iteration (cadence) you are using. The work needs to be potentially implementable. In terms of iteration cadence, I am a big fan of shorter iterations but there is no law that an iteration has to be two weeks or 2 days. Having spent a considerable part of my early career working with folks that wrote macro assembler, COBOL, IMS, DB2 and more I have never seen a piece of work that wasn't refinable into smaller chunks, coded, tested, and put under configuration control before working on the next story. Empathize the importance of adopting 1-week sprints or 2-week sprints. Too short and too long end up hindering teams.

There is no reason why Agile practices can't apply to large mainframe projects. Adopting some techniques might require changing organizational structures to remove boundaries or might require building new infrastructure, but the use of techniques is still possible.

3. Maintenance

The second category that tends to draw consternation from those resistant to Agile is maintenance. There are two issues raised to explain why this type of work doesn't fit well. The first is the difficulty in planning the many projects of varied sizes put into a single bucket. Note – most organizations aggregate small

enhancements and maintenance into the same budget. The second and potentially more problematic issue occurs by expediting emergency production issues. These two primary issues boil down to predictability which, if pushed, could be further reduced into a discussion of coordination.

One of the first team transformations I participated in involved two teams supporting the implementation of the PeopleSoft HR modules. The two teams had slightly different focuses, but could and did help each other out. The backlog was a mixture of enhancements, upgrades, support tickets and software defects on external customizations and reports. The flow was somewhat chaotic. The organization was not interested in directing work that was more stable and predictable to one team and the more erratic flow to the other (no one wanted to have defects and support tickets as the only component of their daily to-do list). Each story followed the same pattern of work: definition, prototyping, coding and configuration, testing, release notes, and implementation. The ultimate solution the team adopted based on two months of experiments was a Kanban heavy form of Scrumban (Reddy, 2015). As part of the transformation, Product Owners began working closely with the team and were able to review and perform user acceptance testing while the work was progressing rather than waiting until the end of the sprint. The solution included the following "ingredients":

1. Adoption of Kanban: The teams defined their process and developed work in progress (WIP) limits for each step. (Burrows, 2014) This allowed both teams to visually identify bottlenecks and the impact of "unplanned" work entering the workflow. The agreement to only pull work when there was room for it to move was one of the hardest concepts to implement. However, after the teams had implemented it, the ability to visualize the flow of work provided a critical tool to identify and try out process improvements.

2. Coordinated Planning: Both teams planned together. Many pieces of work required software code and/or configuration changes that were highly related. When the teams discovered these items, they combined them to minimize contention between teams.

3. Adoption of multi-level planning. The whole team developed a quarterly product roadmap (as a form of release planning) for enhancements. The roadmap left the team with enough capacity to meet their service level agreement for support tickets and most defects. The second level of very tactical planning occurred formally every two days with the whole team rearranging the ready queue in light of the current brush fires (high priority defects).

4. Bi-weekly retrospectives and daily stand-ups. The teams retained two classic Scrum ceremonies out of the basic Scrum playbook. The teams used formal introspection to review process changes and to define the next tweak. They used daily stand-ups to identify bottlenecks and adjust the flow of work.

5. Slack. The team began its journey with a plan for 100% capacity utilization. Any unanticipated impact caused them to miss commitments (a story for a different time). The two teams freed one FTE as slack to catch emergencies or to swarm to a problem if someone else got in trouble the team rotated the "reliever" (they were in a baseball town) to get everyone exposure and training.

6. Runways. We will return to this topic when we address commercial off-the-shelf (COTS) directly. However, the vendor's release plans act as guidance on what is in the package, what the team will need to add, and what the team will need to configure. Knowing what is coming reduces rework and is useful for prioritizing the backlog. This runway or guidance approach is useful for any COTS package that provides a heads up on what is coming.

7. Breaking work down into smaller pieces[1] (Steve Tendon and Daniel Dorian, 2018). They adopted the user story concept of disaggregation into stories that are demonstrable and potentially

[1] Consider Steve Tendon and Daniel Dorians MOVE concept where the smallest outcome is pursued iteratively.

implementable. As the team got into the habit of focusing on flow, there was a natural tendency to make stories smaller. One of their “learnings” from the first retrospective was that large stories tended to get stuck when complications arose.

The team, with a little coaching, was able to make their process more Agile by combining lean, SAFe and other ideas. This was not possible until they began to shift their mindset. Doing the work and the organization had to decide to change how they performed and managed the work. Sergio Brigido[v] said it best in his response on LinkedIn about why maintenance and other scenarios are difficult; “Not that I believe that it´s not possible to have a system maintenance (at least in respect to functional changes, on mainframe or different) operating under an adaptive mindset, but what I see is that we are very far to really have the necessary organizational mindset change to achieve that desired state.” The process of adopting Agile (in a broad sense) begins by establishing a goal then adopting the proper mindset. Process changes combined with feedback bolster and reinforce the new mindset.

4. Commercial Off The Shelf Packages

Buying a package to perform a major function in an organization is rarely as easy as buying and implementing an app on your smartphone. Package implementations often include:

5. Integration with other applications
6. Configuration
7. External customizations
8. Conversion of current data
9. Training users
10. Communication and change management activities
11. Business process re-engineering
12. Buying and installing the package
13. Hardware and network changes

Purchasing Microsoft Word and installing it on your computer might not be as complicated as this list suggests; however, installing packages like PeopleSoft, SAP or Workday will require most, if not more, than this list of activities. Implementing Commercial Off-The-Shelf (COTS) software for anything substantial is a complicated endeavor (albeit less complicated than building the whole thing yourself). The most common reasons given for why an Agile approach to these efforts won’t work are three-fold:

1. Coordination amongst all of the teams and organizations
2. Transparency between the organization and software vendor(s)
3. Implementations for packages are typically are either comprised of a big bang or a few biggish banglets.

Coordination:

COTS implementations for major packages are large projects and involve many different teams. Many of the same solutions that we noted for mainframe projects are germane to this scenario.

1. Adopt a Scrum of Scrums (SoS). SoS's foster focused cross-team coordination by getting a small set of representatives together on periodic basis for micro-planning and coordinating the work between Agile teams. (Dalton, 2019)

2. Implement visual management using a Kanban board and other information radiators. Make sure everyone can see the progress toward the goal. This will help involvement and will support self-organization.

3. Follow a common cadence of joint planning at a strategic and tactical level. Consider the 90-day product increment adopted by SAFe combined with everyone participating in team level planning at the same time (every 1 to 3 weeks based on a common sprint/iteration cadence).

4. Leverage the package to seed the project's architectural runway. The COTS package will have a required footprint including code structure, components, and infrastructure; use the package to seed the organization's architectural runway. The subtitle of the movie Dr. Strangelove put it best, "How I Learned to Stop Worrying and Love The Bomb." When you buy a major package you often have just bought the structure for a big piece of your business.

Transparency:

Transparency might be the hardest problem to solve. Each organization will have its own methodology for implementation. The level of transparency spelled out in the contract defines how the organizations will interact. If you are going to use Agile to coordinate and communicate it must be in the contract. Building many coordination steps into the contract before signing will solve much of the transparency problem. Another recommendation is to review the timing for when the COTS vendor releases draft release notes. The earlier and more in-depth the draft release notes, the easier it will be for teams to integrate the flow of work into standard iteration planning and execution.

Implementation:

It is difficult to implement a package in production a little bit at a time. Most organizations put packages into production all at once or in large chunks. An Agile approach often cannot impact that trajectory. However, much of the activity (integrations, conversions, configurations, coding and more) can use best practices identified earlier, such as small pieces of work completed in short iterations that deliver production-ready code.

Coordination, transparency, and implementation make implementing and maintaining major COTS packages complicated. No methodology will make these efforts easy, but the structure and techniques that improve communication and generate faster feedback will improve coordination, transparency and reduce the risk of the typical big bang implementation.

5. SaaS

Software as a service (SaaS) is one of the most significant trends of the early 21st Century. SaaS is a scenario in which a person or organization access a piece of software hosted outside of the company via the internet. You use the software as long as you want, you don't need the infrastructure to host the software and you don't have to worry about dealing with the code. The organization using these types of solutions needs to configure the software, manage the data, code and integrate organization-specific solutions outside of the package and manage internal uses. Significant distributed cloud solutions rarely are "plug and play". Even straightforward applications, such as Slack, often need administration and support inside the organization. Large packages, such as Workday, need teams to support, configure and administer the implementation. Some organizations using distributed cloud solutions to meet their

information technology needs find it difficult to leverage approaches like Scrum; however, with the correct mindset and a flexible approach, Agile is useful in this environment.

Teams and organizations cite several problematic scenarios for why Agile is not a good fit for SaaS. We will focus on on-going release scenarios after the initial implementation of the package.

1. The package is a black box, which obscures the activity until the users receive a release. Most SaaS providers do their best to provide release notes to organizations to help them know both what is coming and what has arrived (note the definition of 'best' varies). The issue is that the release is not always available in test environments before they are releases nor do organizations always have the option of whether or not to accept the change.

   Suggestions for addressing this issue: Ensure that your vendor provides good release notes and a solid test environment that is available before release dates. Use the release notes to drive user stories and the acceptance tests for the user stories. Planning for this type of release needs to incorporate the internal changes needed. Write and prioritize user stories for the internal work. Use automated testing tools that incorporate both the package and integrated functionality if possible, to incorporate continuous testing.

2. Configuration is not coding. Functionality is often controlled by setting which change the behavior of screens or whether specific functions are visible or available. Occasionally teams make the argument that because they are not coding they can just wing changes (that's not Agile). Rather they can make changes on the fly without a user story, prioritization by Product Owners or even testing. I call this the big boom approach. There are several flaws in the logic. First, doing work that isn't prioritized by the Product Owner takes time away from delivering value. At best adding extra work into an iteration of any size requires stretching the boundary, or at worst abandoning committed functionality. Daniel Vacanti's Actionable Agile Metrics for Predictability (Vacanti, 2015) lays out the negative impact of adding expedited work to a process.

   Suggestions for addressing this issue: Treat configuration like any other type of change you would make in an Agile project. The means generating user stories, prioritizing with the Product Owner(s), testing the change (this means you need a test region with test data that mirrors production). Develop tests before you make the change. This is a form of test-first development that will allow you to understand whether you have completed their task. Using test first thinking will also help to ensure that the user story is well-formed and thought out.

3. SaaS vendors and internal cadences don't match (Rothman, 2016). I recently spent time with a team supporting the package solution delivered on a weekly basis and two major releases during the year. Deliveries of software happened like clockwork. The internal team used a Scrum-y version of Scrumban with a two-week cadence. At first, the team had difficulty determining how the solution providers schedule could integrate with their schedule.

   Suggestions for addressing this issue: Forget for a moment the techniques and ceremonies from the frameworks or methods you are using and consider the principles of Agile. The techniques you are using should deliver value by making how you work more adaptable. Review the approaches you are using and tailor them to meet your environment. Use the principles from the Manifesto to test the approach. For example, in this case, a sub-team reviewed the changes that the vendor anticipated in each major release (a variant of the Three Amigos approach). The sub-team crafted user stories and incorporated those stories into the larger backlog. For the weekly release, a team member playing a Business Analyst (BA) role swept the weekly release notes and crafted user stories of any changes that required testing or where optional changes in the portion of the solution they used. The team reviewed the new user stories for incorporation into the backlog at the standup meeting every Monday.

Many of the solutions to address using Agile in a SaaS environment are similar to those teams using a commercial off-the-shelf package (COTS) environment. Most of the suggestions require a mindset that approaches adaptable approaches by asking how the principles in the Manifesto can address those issues, rather than immediately jumping to why these approaches don't fit. Using any approach in this type of environment feels different than for systems of engagement developed and maintained within the organization. Different does not mean impossible but does require both conscious decisions about how a team or organization will work and an adaptable mindset.

6. Contracts and Outsourcing

The final category of reputed places Agile fears to tread is the land of contractual relationships. Outsourced work is the primary denizen of the land of contracts. In all but the simplest scenarios, asking another firm to do work for you generates a formal relationship. In most cases, the legal document is a proxy for intimacy and trust that occurs inside a single organization. Fences make good neighbors to keep relationships orderly; in the business world, formal agreements take the place of fences to establish boundaries. Boundaries constrain the free flow of information and ideas, which is antithetical to Agile principles. There are four core reasons organizations and practitioners see issues of high-trust approaches in low trust environments. None of these issues is insurmountable (Atkinson & Benefield, The Curse of the change control mechanism, 2011) if negotiated and written down upfront. Some of the issues are:

1. Transparency – Agreements establish boundaries and obligations between the parties. Obligations are often tied to payments and in some cases performance incentives and penalties. All of the parties in the contractual arrangement have a variety of goals that include delivery of an output (generally software), maximization of profit by the outsourcer and minimization of expense by the outsourcee. All goals in a contract, taken together, generate a behavioral equilibrium so that everyone gets what they need. Unfortunately, not all behaviors are useful which makes sharing information less likely. A classic example is a scenario where someone is over budget or behind schedule but thinks they can fix the problem. In this case, they feel incented to avoid the pain of exposure until they know they can't fix the problem. Highly structured agreements exacerbate this issue.

   Solution: Before signing the contract build in the participation required by Agile for all parties. The Product Owner must be from the outsourcer, the outsourcer needs to take part in all planning activities, involve the Product Owner (or proxy) in the daily stand-ups, and they should have access to the backlog management tool. After signing: involve the outsourcer in the demonstration and the team is adhering to their definition of done.

2. Access – Interactions between team members and the business are often constrained. Often point people assigned for each role. These people act as a funnel for all conversations and decisions. This is often done as a configuration management technique. Only the identified point people can accept inputs or changes to the scope of work or tweaks to the requirements. Access is a specialized form of transparency. Agile principles and techniques explicitly assume that the team(s) have access and interact with the business daily (sometimes stated as continuously). Also, there is an expectation for the Product Owner (from the business) to manage and prioritize the backlog.

   Solution: Before signing, explicitly build Agile roles into the document. For example, build the Product Owner role (part of the business) as part of the team. Identify the need for intimate, daily (or close) communication with the team. In the contract make it OK for team members to talk with the business without a chaperone. If at all possible, embed the Product Owner and/or subject matter experts with the team (full time if at all possible). After signing, this issue is more difficult to solve post. One possible solution to generate the conversation between the team and the business is to embed the Product Owner and/or point persons with the team at least on a part-time basis. This fix generates conversation by forcing access. In the longer-term embedding

helps to foster trust.

3. Delivery Cadence – Agile techniques expect that the team will deliver “potentially” implementable functionality on a periodic basis. The periodic delivery gets value to the client sooner and provides a feedback mechanism. Many agreements only recognize value by the final outcome. The contact views anything else as a distraction that might take the focus away from the real payout.

   Solution: Before signing, build in delivery on cadence. After signing, deliver each iteration and map the stories (or work units) delivered to the ultimate goal. By linking each delivery to the ultimate goal in the contract everyone involved will stay focused on the agreed-upon obligations.

4. Trust – Signed agreements are tools to explicitly define the obligations of all parties in lieu of trust or a handshake. The goal is to enforce trust because the penalties are painful. Agile assumes that the parties have a common goal. The common goal creates an environment in which everyone involved has a clear understanding of how the team and business will behave.

   Solution: Before and after signing, establish a common goal. Recognize that each party might have other goals that conflict. State the conflicts and made them subservient to the common outcome. If team members believe that there are ulterior motives, trust will be elusive. Use embedding, joint planning, and retrospectives to make a contractual relationship more Agile.

Contracts are tools to create fences (Atkinson & Benefield, 2013). Most Agile principles and techniques get rid of fences between the business and the team or between the outsourcee and the outsourcer. Legal agreements can explicitly prohibit interaction or they can create scenarios where participants are afraid to interact. Once signed negotiation to embed any techniques is painful. Adopt techniques such as joint planning, attending demonstrations and stand-up and sitting with the team (even if it is just occasionally) to help build trust and inculcate principles.

7. Conclusion

Agile is a mindset first and a set of techniques second. Saying that adaptive approaches won’t work is less about technique but about a fear of a mindset or being satisfied with the status quo. Rather than saying what can’t work it is better to question how can we address the 12 principles using a mixture of frameworks and techniques to deliver value. The Manifesto establishes goals that provide a solution path. In the end, the reason Agile has become important not only in software but most business domains is that it helps people deliver value. Who wouldn’t want a value-based mindset?

References

Atkinson, S., & Benefield, G. (2011). The Curse of the change control mechanism*, Society for Computers & Law Publication*.

Atkinson, S., & Benefield, G. (2013). Software Development: Why the Traditional Contract Model Is Not Fit for Purpose. *Hawaii International Conference On System Science.* IEEE.

Burrows, M. (2014). *Kanban from the Inside.* Sequim: Blue Hole Press.

Dalton, J. (2019). *Great Big Agile.* New York: Apress.

Karten, N. (2009). *Changing How You Manage and Communicate Change, Focusing on the human side of change.* Cambridgeshire: IT Governance Publishing.

Reddy, A. (2015). *The Scrumban [R]Evolution: Getting the Most Out of Agile, Scrum, and Lean Kanban.* Hoboken: Addison-Wesley Professional.

Rothman, J. (2016). *Agile and Lean Program Management.* Practical Ink.

Steve Tendon and Daniel Dorian, T. Y. (2018). *Tame Your Work Flow.* Amsterdam: LeanPub.

Tendon, S., & Muller, W. (2015). *Hyper-Productive Knowledge Work Performance.* Plantation: J. Ross Publishing.

Vacanti, D. S. (2015). *Actionable Agile Metrics for Predictability.* Amsterdam: LeanPub.

---

i Reinheitsgebot, Wikipedia, accessed April 3, 2018, https://en.wikipedia.org/wiki/Reinheitsgebot

[ii] Agile Manifesto, Manifesto for Agile Software Development, accessed April 3, 2018, https://agilemanifesto.org/

[iii] T Henricksen (personal communication and LinkedIn, February 14, 2018).

[iv] Architectural Runway, accessed August 16, 2019, https://www.scaledagileframework.com/architectural-runway/

[v] S. Brigido (personal communication, LinkedIn, February 17, 2018)