Introducing the Agile Risk Management (ARM) Framework

Agile Six Applications, Inc.
Brian Derfer, CTO

Introduction

Agile frameworks like Scrum, XP, SAFe, and Kanban have proven to be very effective at delivering software in a more collaborative, transparent, and predictable fashion than the traditional waterfall framework. However, Agile practices, by themselves, are not sufficient to address the risks that impact most medium-to-large software projects, and in particular are often not well suited to delivering on federal government projects, which often have a structure - such as fixed scope and delivery timelines - that is not easily or naturally accommodated by Agile. To fill the gap between what Agile does well at the team level, and the risk management requirements of medium-to-large projects, we have developed the **Agile Risk Management (ARM) Framework**.

The Need for Better Risk Management

In our experience, there is a noticeable lack of proactive risk management in Agile projects in both the commercial and government sectors. Some would argue that Agile practices are by themselves mechanisms for mitigating risk, and to a certain extent this is true. Techniques such as time-boxed iterations allow many opportunities for course correction, while demos provide regular empirical verification of a team's output. Retrospectives provide teams with a mechanism for addressing team dysfunction and facilitate continuous improvement, and *team ownership* of each sprint's commitment often provides the emotional investment needed to keep a project moving effectively. However these mechanisms, while necessary, are not sufficient to manage risk across even a medium-sized software project. For example, Agile by itself does not provide an effective mechanism for addressing structural problems within a project that are beyond a team's control. Consider the following problems that are fairly typical in Scrum projects of non-trivial size:

- The Product Owner (PO), despite ongoing pleas from the team during retrospectives, fails to prioritize user stories that help the team manage technical risk or technical debt.
- The PO is not capable of leaving a Scrum team alone for an entire iteration without introducing new user stories or acceptance criteria.
- The team does not have enough consistent members to estimate velocity.
- The team's velocity simply isn't fast enough to meet current schedule commitments.
- The architecture that has evolved incrementally is brittle, or does not align with the priorities of the project overall.
- The customer has not provided access to users or important stakeholders.
- The team does not know who all of the stakeholders are who define success for the project.

When the additional constraints imposed by federal projects are considered, Agile rituals by themselves are simply insufficient to manage risk on many projects. Some combination of the following usually happens when risk is not managed well on a project:

- Team velocity and quality of deliverables declines.
- The team must constantly fight fires and deal with emergencies which results in them regularly failing to deliver on sprint commitments.
- The team loses whatever autonomy and "ownership" of the sprint commitment they had, and is simply directed by POs or management to work longer and harder "because we can't move the deadline."
- Release dates are missed and/or promised functionality is not delivered.
- Team morale deteriorates as unhappy POs, customers, and managers bemoan the ongoing low quality if the team's output despite their working longer hours. The project gets a reputation as the "death march that no one wants to be on."
- The team has to scramble late in the project to meet a whole set of security and architectural constraints that had been invisible, or at least not appropriately prioritized.
- The last 20% of a project ends up taking 80% of the time.
- The team gets blindsided by new user stories or acceptance criteria that come out of the woodwork late in the project schedule as previously unengaged stakeholders finally gain interest in the project and add their two cents.

Problem	Impact
• Technical debt is not addressed in a timely manner. • New requirements are injected mid-sprint. • The team does not have enough consistent members to estimate velocity. • The team's velocity is not sufficient to meet schedule commitments. • The architecture has become brittle and/or does not align with project priorities. • The customer has not provided access to users or other important stakeholders. • The team does not know who all of the stakeholders are.	• Team velocity and the quality of deliverables declines. • The team regularly fails to deliver on sprint commitments. • Release dates are missed, and/or promised functionality is not delivered. • The team loses their autonomy. • Team morale suffers. • There is a "balloon payment" of technical and/or feature debt at the end of the project.

One consistent problem that we have encountered is a misunderstanding that embracing an Agile framework somehow removes the need to perform any basic project management all. ***It doesn't***. Agile simply places a higher value on incremental delivery of working code over the creation of project management artifacts and deliverables, and provides flexibility to address facts on the ground rather than rigidly stick to a plan that

may become obsolete. But that does not mean that Agile removes the need to manage projects at all. There is still the need to determine which framework is appropriate for a given project, adequately resource teams, identify and address risks and structural problems within the projects, coordinate the activities of multiple teams on large projects, and maintain an ecosystem that supports the development framework, practices, and rituals. In fact, Agile processes are often better at identifying problems in a project early and often than serving as a solution to the problems per se.

The ARM Framework ©

Based on our years of experience using Agile to successfully deliver across a range of large commercial and federal government projects, we have developed an Agile Risk Management (ARM) Framework. The ARM framework provides a set of tools and practices that projects can use to better manage risk on Agile projects.

Determine which framework is appropriate

Scrum, and other iterative and incremental frameworks such as XP and SAFe, can be very powerful delivery frameworks when implemented correctly and properly supported within the right set of conditions. However, Agile frameworks in general, and any specific Agile framework in particular, may not be suitable or optimal for a given project. If a given framework is not a good fit, it is far better to determine that up-front or as early as possible than to invest a lot of time trying to make it work on a project for which it is simply a bad fit. For example, we assert that the following minimum conditions should be in place for any Scrum project:

- ***The team can be left alone for a sprint cycle (usually 2 weeks) without injection of new backlog items or significant changes to existing user stories***. Sprint commitments are cornerstone to Scrum; in order for the team to succeed in their commitment, the corollary commitment needs to be made to leave the team alone to self-organize to meet its commitment. If your customer demands that the team be able to respond to incoming requests and changes in priorities more frequently than the sprint cycle frequency, then Scrum is probably not the right choice. Kanban (see below) may be a better choice in this circumstance.
- ***There is a critical mass of consistent, dedicated team members***. While it is ideal to have all team members 100% dedicated to a Scrum team, with little or no turnover, this is hardly a realistic expectation. Developers move between jobs, and companies need a certain amount of resource flexibility and fluidity in order to keep employees fully tasked and projects moving forward. However, there does need to be at least a "core" team that is consistently dedicated, otherwise it becomes very difficult to hold team members accountable ("I got called in to fight a fire on that other project, so I couldn't get to that."), and it may be difficult to implement any kind of continuous improvement if velocity or other metrics cannot be consistently measured and compared in an "apples-to-apples" way.
- ***There needs to be support for Scrum throughout the organization***. Senior leaders in particular need to understand both the philosophy of Agile and its practices. There is nothing that will undermine any Agile effort faster than a VP circumventing the

framework and going directly to a team with a hot task that is not in their sprint commitment. This is especially true for a nascent conversion to an Agile framework such as Scrum. If the VP really can't avoid bothering the team through an iteration, see the first minimum condition described above.

- ***The team needs a qualified Scrum Master***. Scrum supplants a lot of external structure (process, documentation, etc.) with an emphasis on real-time interactions among team members. This entails a *tradeoff* in investments - companies need to invest in facilitating healthy teams and interactions - not simply an abandonment of investments in formal process structure. Too often the move to Scrum is undertaken as a one-way embrace of freedom from formal structure and the costs associated with it (time, money, tools, etc.) without fully embracing the investments that need to be made in a culture that emphasizes collaboration, communication, and trust. As a result, developers or project managers are thrust into the Scrum Master role without sufficient training or experience. They may not possess some of the essential personality traits required of a good Scrum Master such as emotional intelligence, the ability to coach and lead informally, and the ability to engage a project at many different levels of detail.
- ***The Scrum Master needs to be dedicated enough to be able to effectively coach the team***. Assuming you have a properly qualified Scrum Master, he or she needs to have time to properly do the job. "Dedicated enough" here is intentionally vague. Teams new to Scrum or have not been practicing healthy Scrum will probably need someone who is more dedicated than a very experienced senior team. A Scrum Master needs to be dedicated enough to gain the team's emotional trust, and be present enough to see and respond to issues across a wide spectrum of causes (project structure, team dynamics, individual psychology, technical risk, etc.). Companies need be very careful about asking people to take on "extra" Scrum Master responsibilities alongside their existing development or PM responsibilities. Unless both the Scrum Master and team are very experienced at Agile, part-time Scrum Masters usually do not work out well.

Similarly, there are minimum conditions that should be in place for other Agile frameworks, such as XP or Kanban, to succeed as well, though it is beyond the scope of this paper to exhaustively enumerate these for each flavor of Agile. The point is that companies and teams should take stock of the entire ecosystem within which their project will operate, and select (or invent) a framework that can thrive in that ecosystem. Picking the right framework, and/or making the necessary changes to the ecosystem to support the framework chosen, is a huge first step in mitigating the risk that your Agile project will fail.

Assign an Architecture Owner

In Agile projects, architecture is not defined within a separate phase of the development lifecycle and handed off to teams, but rather emerges iteratively and incrementally through sprints in response to the evolving needs of the project as these are expressed through the priorities of product backlog. There are tremendous advantages to this

approach, not the least of which is that software ideas can be tried out and killed off relatively cheaply when there is little up-front investment in architecture. One risk, however, is that it provides little structure - in terms of approval ceremonies, documentation artifacts, etc. - for the team and other stakeholders to ensure that the architecture is on track. It may be a challenge for an evolving architecture to address complex requirements, strict security controls, stringent standards requirements, or approval gateways such as those found in many healthcare integration or federal government projects if there is no one held accountable for ensuring that the architecture meets these needs.

To better support complex and/or formal architectural requirements, the ARM Framework includes a new Agile role: Architecture Owner. The Architecture Owner is analogous to the traditional Product Owner role; whereas the Product Owner is responsible for the observable "user-driven" feature set of an application, an Architecture Owner is responsible for the underlying architecture needs of an application or system that may not be apparent or observable to users, or even well-understood by the Product Owner. Product Owners may not be highly technical, and hence may not understand or appreciate such things as design patterns, security standards and vulnerabilities, or complex and multi-faceted architecture requirements of an existing environment or set of processes. Without a deep technical understanding of these issues, Product Owners may not be able to appropriately prioritize work efforts related to them, which can result in products that meet visible users needs well but are not well-architected and/or fail to meet architecture requirements. The Architecture Owner is one mechanism for helping to ensure that the architecture continues to align with complex requirements within a framework that prioritizes individuals and interactions over documentation and formal process.

But aren't Agile teams supposed to self-organize? One frequent objection to identifying an Architecture Owner role is that it undermines the ability of a team to self-organize. And we can't really argue with that, though we would prefer the more neutral term "limit" to the term "undermine." Structure by definition puts certain constraints on freedom. A prioritized backlog constrains a team's ability to self-organize in that teams are not permitted to take on lower priority user stories even though they may strongly feel that doing so would benefit the project. It really comes down to what bits of structure you decide to build into the framework, and what freedom to self-organize you permit within that structure. In our opinion, the advantages of the additional structure that an Architecture Owner brings is worth the constraints on self-organization that come with it.

The Architecture Owner works within or alongside the team to understand the architecture requirements of an application, works with the team to identify toolsets, frameworks, templates, and patterns that support the architecture requirements and best-practices, helps the team identify and prioritize technical debt, and provides guidance through frequent architecture reviews to ensure that architecture requirements and and best-practices are being met. At the end of each sprint, the Architecture Owner and team are available to demonstrate to other technical stakeholders, including client stakeholders, how the application(s) support architecture requirements and best-practices in the sprint review.

The architecture owner role can range from informal (the person generally recognized as the most experienced and capable on the team) to formal (someone officially designated that role and held accountable for it). The level of formality required depends on a number of factors, including:

- The complexity of the project
- The requirements for meeting specific technical standards
- The need for production of formal artifacts and documents
- The level of formality in the project structure itself

In general, small teams working on commercial MVP projects will probably fall more on the informal end of this continuum, while larger teams working on federal government projects will probably fall more on the formal end.

The architecture owner is not a silver bullet, and there is still the need to manage technical risk and technical debt in other ways. For example, the risk meeting provides a forum for communicating technical risk and the product backlog is still the mechanism by which that risk is addressed. However, having an architecture owner may provide a level of oversight that, along with some other practices described here, may help projects avoid brittle architectures, unmet technical requirements, and skyrocketing technical debt.

There is an informative discussion of the Architecture Owner role and how it can be scaled out to large projects at AgileModeling.com:

http://www.Agilemodeling.com/essays/architectureOwner.htm

Establish a Common Project Vision

The prioritization of working software over process and documentation, and the quickness with which Agile frameworks can often deliver it, may cause some teams to inappropriately de-emphasize the importance of ensuring that there is both a coherent vision for the project and that it will well understood by the entire team. Most Agile projects we've encountered either do not have any artifacts at all describing the project vision, or, when something has been created, it has been shared by senior executives and Product Owners, but not with the team. The vision description does not need to be (and in fact in many cases should not be) an elaborate 87-page piece of marketing shelf-ware. Rather, a good vision artifact should be as short as possible, very public, visible, and reviewed regularly. In fact, it doesn't even need to be a document; it needs to be some kind of artifact that the team can refer to easily. A wiki page often makes a great home for a project vision. Regardless of the format, what we call the ***Vision Artifact*** should accomplish the following in the most efficient way possible.

- ***The Vision Artifact should succinctly describe what problem is being solved or what business opportunity is being met.*** Precisely because Agile emphasizes individuals and interactions over process and documentation, teams are usually

operating without a technical specification document. Acceptance criteria within user stories were never meant to replace a technical spec, and trying to make them fill this purpose will quickly lead to an over-worked Product Owner. Hence there are hundreds of technical and design decisions that the team makes each sprint that are supported only by acceptance criteria described at a high level and the team's understanding of the problem(s) they are solving. Having the team internalize the big picture "WHY?" that is driving the project will help them make daily micro-decisions that better support the goals of the project. Of course the team will not always get it right the first time, and getting the working software into the hands of target users is by far the best way of ensuring that your project is on track. But having a basic shared understanding of the users' needs up-front can dramatically reduce the number of iterations required to deliver software that users are happy with.

- ***The vision artifact should succinctly prioritize the technical or architectural principles driving the project.*** Many architectural decisions imply tradeoffs between benefits. For example, a system built with security as the paramount concern may not be as extensible, or may not perform as well, as a system without such constraints. Architects and team members are going to be making these tradeoff decisions on a constant basis. It is better that these tradeoffs are discussed up-front, and some decisions made on what principles will take priority, so that tradeoff decisions made within sprints align with the overall goals of the project.
- ***The Vision Artifact should identify the stakeholders that will determine whether the project is a success or failure.*** This is a critical point that cannot be overlooked. There are usually significant stakeholders beyond the user community that can and will determine whether or not your project will ever see the light of day. These include approval committees, IT stakeholders, contracting officers, subject matter experts (especially in clinical or technical domains), and senior executives who just want to have a say-so. Projects should do everything they can to identify these stakeholders up front and make sure they are included in Agile ceremonies wherever possible. It may never be possible to completely eliminate the risk that a senior executive will suddenly become interested in the project, swoop in two weeks before release, and start demanding changes. Agile is not a panacea for dysfunctional companies. But having a mechanism for identifying stakeholders upfront and providing many

MVP and "Spiking the Process"
Teams and POs may not know up front who all of the important stakeholders are in a project. This is especially true for teams operating in new environment with complex organizational and approval structures, such as DoD or VA. In such cases, one mechanism for ferreting out these stakeholders is to drive a minimum viable product (MVP) through the SDLC process as far as possible as early as possible - a practice we call "Spiking the Process." Driving a software deliverable through all of the approval gates and (if possible) getting it deployed can bring out important stakeholders and their concerns, so that these can be accounted for relatively early in a project.

opportunities to make sure the list correct should help mitigate the risk that new important stakeholders are identified late in the release cycle.

Finally, we have found that it is valuable for the team to demonstrate that they have internalized the project vision. Some examples of quick exercises that the team can do include:

- Have a team member other than the Product Owner go over the vision artifact with the team, or have different team members go over different sections.
- Create and perform a pretend TV commercial about the product being built.
- Construct a competitive game like Jeopardy or Family Feud that requires team members to demonstrate their understanding of the project vision.

Create, Share, and Maintain Roadmaps

Traditional Agile teams frequently fail to deliver fixed scope projects on time. This is not surprising, since Agile traditionally rejects fixed release scope and schedule in favor of fixed sprint time boxes and flexibility around priorities. The common answer to the mismatch between Agile and fixed scope projects is a demand to remove the fixed scope and schedule from the project. However, this is not a very realistic choice for companies who want to use Agile on government contracts where fixed scope and budget are built in to the contractual agreement and are usually inflexible. As someone has probably said somewhere, outright rejection of fixed scope projects can be a “career limiting move.”

For those companies who do want to use Agile on fixed scope projects, the common approach is to make the Product Owner responsible for larger milestones (such as releases), while the team’s obligations, and hence focus, remains tied to sprint increments – namely, what it can commit to in a sprint. The problem here is that if the team’s sprint velocity is not sufficient to complete the work on schedule, there is no clear mechanism for course correcting other than to “break” the Agile model of team ownership of its delivery capacity each sprint, and make teams work harder and longer than they want to. This, in turn, can cause team members to leave the project and company for projects where their buy-in and commitments are respected. In any case, it is not a healthy mechanism for getting a project done on time.

In our experience, one way of addressing this apparent paradox is to ask the PO to create a roadmap of deliverables and ask the team to share ownership of it. The roadmap is reviewed each sprint or every couple of sprints, either during the planning meeting or the backlog grooming meeting, so teams are regularly exposed to the entire schedule, not just to the POs top priorities at any given time. Teams who are solely focused on sprint deliverables every two weeks often fail to adequately “own” the entire release schedule and deliverables, and hence fail to raise concerns about resources, technical risk, user story prioritization, or feature creep on user stories until it is too late. In our experience, a team that has a clear view of the roadmap is less likely to spend time gold-plating features or invest time in exploring alternative implementations for the heck of it. It is more likely to raise alarms about the issues that it has visibility into.

This is not to suggest that roadmaps are fixed and cannot or should not change. One of the foundational drivers of Agile is the recognition that the bigger the time increment that a plan covers, the more likely it is to be wrong. Put another way, sprints are a mechanism to quarantine inaccurate planning to manageable chunks of time. We don't dispute this. We understand that the roadmap created at the beginning of a project will probably look quite a bit different than the roadmap actually followed by the team over the course of a project. In our view, however, the value of the roadmap isn't that it is accurate at the start of a project, but that it provides something against which to measure actual progress and make the necessary micro-adjustments in order to deliver a project on time.

What do we mean when we say that the team should "own" the roadmap along with the PO? It should:

- Understand it and be emotionally invested in delivering on it, and be invested in fixing it when it no longer represents reality, or no longer provides value
- Raise alarms about any issue, including access, resources, processes, or external commitments, that will impede delivery on the roadmap
- Work with the PO to identify user stories or features that pose risk to the roadmap so that these can be prioritized appropriately in the product backlog

Even on projects that support a more flexible delivery model, we believe that creating, maintaining, and regularly reviewing a roadmap may provide benefits to Agile teams. Usually the PO has some idea of how they envision the project unfolding over a timeline larger than the next sprint. Sharing that vision with the team in the form of a high-level roadmap, getting the team's input on it, measuring progress against it, and updating it when appropriate will make sure that the PO and the team stay on the same page about the progress of the project and the risks to delivery.

Having the team buy into and "own" the larger product roadmap only works if the team has a mechanism for addressing its concerns about the schedule, resources, and larger project structure within which it operates, which is described in the following section.

Manage Risk Transparently and Collaboratively

Teams can only be asked to own something if they are given mechanisms to manage it. In the case of a product roadmap or a fixed release schedule, one mechanism for managing a roadmap is a formal risk register that the team contributes to, and is regularly reviewed with the PO and all stakeholders, including the customer. The risk register and risk meeting provide a high degree of transparency into the challenges faced by the team, ensures that each risk is associated with a mitigation strategy and an owner responsible for executing it. It also provides a feedback mechanism for important items to be prioritized in the team's backlog and items which once seemed important to be either cut or deprioritized. Think of Agile as an alarm that gets sounded every couple of weeks about current and potential problems. Without a mechanism for managing that risk, such as a risk register and recurring risk meeting, that alarm may go unheard.

The inclusion of the customer in the risk mitigation meetings is a topic of vigorous debate, but in our view is central to a healthy risk management system. In our experience, when customer stakeholders are “invited in” to see what challenges and risks are faced by the team, can see that risks are being actively managed, and are invited to take part in mitigating the risks, the vendor-customer relationship can often undergo a profound transformation — from one in which the customer is a passive recipient of the vendor’s output (or, too often, its “victim”) into a true collaborative partner which shares in the challenges and solutions. Of course, not every customer will make a good collaborative partner, and there are plenty of dysfunctional vendor-customer relationships where increased transparency may not be a good idea. Most reasonable customers, however, will appreciate being invited in to the risk management process, and can be key contributors to making necessary structural changes on their end to help teams be more efficient. Ideally, customers may also we willing to adjust the priority or scope of deliverables once they have a full understanding of the challenges facing the team. Most important, however, is that by involving customers in seeking solutions, they become invested collaborators in those solutions, and it becomes less easy for them to simply blame the team for not meeting challenges.

At this point, those familiar with Agile might counter that sprint retrospectives are the mechanism for identifying and mitigating risks or challenges to the team. There may be situations on small projects with teams composed of very experienced Agile practitioners where this actually plays out. In our experience, however, retrospectives are not adequate for managing risk for a couple of reasons:

1. Retrospectives are too often focused on what happened during that previous sprint, and do not provide an opportunity for team members and other stakeholders to raise their heads out of the sprint to focus on what’s going on at the project level or roadmap level.
2. Retrospectives do not have all of the right people in the room to address project risk well. Retrospectives are the appropriate forum for team members to identify opportunities for improvement during the last sprint and try little experiments to improve. Retrospective commitments assume the autonomy of the team to try things over which they have direct local control. However, many of the elements that can add risk to a project is outside of team’s control. Hence there needs to be some mechanism for identifying those risks early in a project and mitigating them, and that often requires involvement by POs, other stakeholders, and sometimes even the customer.

Backlog grooming meetings may also be considered a mechanism for managing project risk, especially around areas like technical debt. And actually, this answer is frequently pretty close to right. The product backlog is usually the right tool for *prioritizing* mitigation strategies owned by the team. However, the backlog cannot be the primary mechanism for managing project risk, since it cannot be used to manage the risk mitigation the efforts of those outside the team. The backlog is also usually a deficient mechanism for tracking things like risk severity and likelihood, or communicating high risk items in a visually compelling format.

There are good existing discussions on the web about what makes a good risk register, so we won't repeat them here. Here is one example:

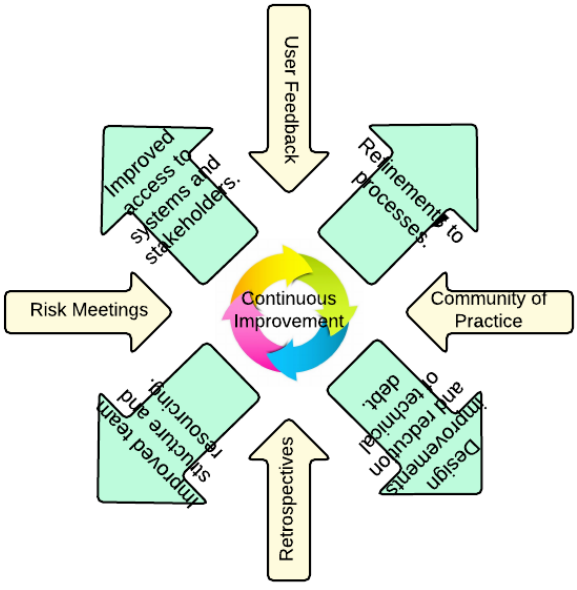
https://michaellant.com/2010/06/04/five-simple-steps-to-Agile-risk-management/

Establish an Agile Community of Practice

An Agile Community of Practice is a regular meeting of Agile practitioners — Scrum Masters, Product Owners, Architecture Owners, and other stakeholders in the framework — whose purpose is to improve the framework itself or its application in a specific context. During Agile Community of Practice meetings, attendees are invited to discuss problems encountered by teams and possible solutions, as well as bring in good ideas that their own teams have tried, or which they have read about or heard about outside of their current project or organization. One of the benefits of Agile is continuous improvement, and the Community of Practice provides a mechanism for sharing good ideas across people, teams and organizations. The Community of Practice is also a great vehicle for enculturating new Scrum Masters or Product Owners into an Agile practice.

A more comprehensive and inclusive process of Continuous Improvement

In traditional Agile approaches, the sprint retrospective provides a valuable mechanism for the team to identify problems, create solutions, and continually improve its effectiveness. While this practice works well at the team level, we have found that there are almost always larger structural issues beyond the team's control that can also benefit from continuous improvement — issues such as project structure and communication, processes for provisioning or gaining access to systems, and ensuring that the team itself is adequately resourced, in terms of personnel, tools, and knowledge. The ARM Framework provides a set of mechanisms that facilitate this continuous improvement on a larger scale. The transparent and collaborative risk meetings, Communities of Practice, along with retrospective meetings and user feedback, provide a rich set of inputs and opportunities to address larger "structural" issues that cut across the team, the company, and the customer. Software projects are complex and somewhat unpredictable by nature, and need to be able to adjust to "facts on the ground." Having a set of practices that support continuous improvement is far more important to most projects than the ability to define and strictly adhere to any given set of process structures or practices up-front.



Conclusion

The ARM Framework, like the Agile frameworks it supports, is not a panacea for curing or mitigating any and all problems that might affect a project. It is important to reflect a bit about what the ARM Framework can and cannot do.

The ARM Framework can:

- Improve the chances of success of an appropriately resourced project with realistic scope and deadlines.
- Identify a project that is likely to fail earlier rather than later so that teams and projects can fail in a planned way instead of a non-planned way.
- Identify issues in a project ecosystem that will are likely to create drag on projects.
- Act as a starting point for thinking about managing risk on Agile projects.

The ARM Framework cannot:

- Change the basic laws governing resources, scope, and quality.
- Fix fundamentally dysfunctional organizations or vendor-customer relationships.
- Cover all challenges a project might face.

The ARM Framework is best thought of a way of channeling resources and information into activities that allow teams to see problems early and act on them efficiently. However, If your organization has, for example, massively underbid a project, the ARM Framework is unlikely to alter the underlying conditions sufficiently to “save” the project.

Agile frameworks have revolutionized the way that software is delivered. However, the need to proactively and systematically manage risk on Agile projects has often been neglected, with the result that, too often – especially on fixed scope projects typical on federal government projects – Agile projects fail to deliver on time with the feature set and quality that users demand. The ARM Framework provides a set of tools that can be used to manage risk on Agile projects and improve the chances that any given project will succeed.

About Agile Six Applications

Agile Six Applications, Inc. was established to serve those who have bravely served our country. We are passionate about our mission *to improve the lives of veterans and their families by delivering world-class software solutions*. Our collaborative and highly transparent Agile development framework invites users and program representatives to participate in the development process, and results in better solutions, delivered more quickly, at a lower overall cost. Our firm was founded in 2014 by former executives from the federal and commercial space (i.e. DefenseWeb & Amazon) in direct response to the formation of the US Digital Services where “America’s most capable problem solvers are striving to make critical services — like Healthcare, student loans, and Veterans’ benefits — as simple as buying a book online.” As such, we actively promote the tenets of the CIO Playbook:

Digital Service Plays

1. Understand what people need
2. Address the whole experience, from start to finish
3. Make it simple and intuitive
4. Build the service using agile and iterative practices
5. Structure budgets and contracts to support delivery
6. Assign one leader and hold that person accountable
7. Bring in experienced teams
8. Choose a modern technology stack
9. Deploy in a flexible hosting environment
10. Automate testing and deployments
11. Manage security and privacy through reusable processes
12. Use data to drive decisions
13. Default to open

Please visit www.agile6.com for more information.

About Brian Derfer

Brian blends nearly 20 years of hands-on experience in the software industry with a background in cognitive anthropology in order to bring a unique perspective to technical problem domains. He has been involved in several start-up ventures as founding partner or early associate, and worked as an architect in the development of key systems within the Defense Health Information Management System (DHIMS) as well as Fortune-100 companies, in vertical markets that include family services, healthcare, education, and social media. His main areas of focus include Agile frameworks, software development best-practices, artificial intelligence, organizational learning, web and mobile development, cloud computing, and test-driven design. Brian holds a Master's Degree in Cognitive Anthropology from UCSD, and is a Certified Scrum Master through the Scrum Alliance. He is named on one US patent. His other interests include soccer, sailing, music, history, and human cognition.

He can be contacted at mailto:brian.derfer@agile6.com