Agile Software Acquisition Guidebook

Best practices & lessons learned from the FY18 NDAA
Section 873/874 Agile Pilot Program

Office of the Under Secretary of Defense
For Acquisition and Sustainment

Version 1.0

27 February 2020
Foreword

The Department of Defense has recognized the need to transform our acquisition system to improve deployment of software-intensive combat capabilities and supporting systems. We need to develop, deliver, and evolve our defense business systems and weapon systems more efficiently and at the speed of relevance to maintain our competitive edge against our adversaries and competitors. This requires adopting and adapting modern software development methods and best practices.

Addressing this need, we are well underway in defining and coordinating an Adaptive Acquisition Framework which streamlines and enables the DoD 5000 series acquisition policy. This new framework will allow program managers to tailor their acquisition strategy to the unique characteristics of the capabilities being acquired – including, for the first time, a brand new software pathway. These Department policies, guidance, and process changes have been informed by the Agile Software Pilot Program which was congressionally directed in the FY18 and FY19 National Defense Authorization Acts. In FY18, we planned and initiated fifteen Agile software development pilot programs of both short duration (Sec 874 one year or less) and longer duration (Sec 873 up to five years). The participating programs reflected a cross-section of both defense business systems and software-intensive weapons systems.

This fall, the shorter duration Section 874 Agile pilot programs were completed. In our Agile Pilot Program Completion Report to the Congressional Defense Committees, we shared our DoD experience in transitioning from largely traditional waterfall software development approaches to more modern iterative and Agile software development processes. This guidebook, intended for our software acquisition workforce and supporting ecosystem members, supplements that recent pilot experience material, shares Agile and iterative development lessons learned, and provides helpful white papers that provide advice/guidance on topics that reflect typical software development organizational challenges during transition to more modern practices. My strategy is to provide tools, like this guidebook, that support the more efficient planning of current and future software acquisition opportunities while providing insight into proven approaches addressing individual program challenges.

The core mission of the office of the Deputy Assistance Secretary of Defense, Acquisition Enablers is to Empower, Analyze, and Innovate. I encourage you to share your Agile or iterative software development and delivery experience with my team, so we can share them with other aspiring Agile software program teams. You may have heard that “All of us together are smarter than any one alone”. By your sharing of program experiences, which supports our ability to share lessons learned distilled through analysis, we will all collectively benefit from being better prepared to bring innovation to our processes and deliver products to the warfighter. Together, we are enabling and empowering our software professionals to change the way we do business and deliver against the goals of our National Defense Strategy.

Stacy A. Cummings
Principal Deputy Assistant Secretary of Defense,
Acquisition Enablers
# Table of Contents

1 Purpose .................................................................................................................. 1

2 Scope ....................................................................................................................... 1

3 Foundational Agile Principles .................................................................................. 1
   3.1 Common Agile Frameworks .................................................................................. 3
   3.2 Agile Connection to DevSecOps ......................................................................... 4
   3.3 Agile Connection to Lean .................................................................................. 5

4 Defining the Capability Need .................................................................................. 6
   4.1 Epics and User Stories ...................................................................................... 6
   4.2 Tracking Progress ............................................................................................. 7

5 User Engagement .................................................................................................... 8
   5.1 Identifying Users .............................................................................................. 8
   5.2 Users and the Product Owner ........................................................................... 9

6 Program Roadmap .................................................................................................. 9
   6.1 Minimum Viable Product and Deployment Cadence ........................................ 11
   6.2 Minimum Viable Capability Release (MVCR) .................................................. 13
   6.3 Product Roadmap and Graphic ....................................................................... 14
   6.4 Capability Management .................................................................................... 16
   6.5 Schedule Interdependencies ............................................................................ 18

7 The Software Development / Deployment Infrastructure ...................................... 19
   7.1 Intro to Cloud .................................................................................................. 19
   7.2 Platform/Infrastructure Technical Decisions .................................................... 20
      7.2.1 On-Premises ........................................................................................... 20
      7.2.2 Infrastructure as a Service (IaaS) ............................................................. 21
      7.2.3 Platform as a Service (PaaS) ................................................................... 21
      7.2.4 Software as a Service (SaaS) ................................................................... 22

8 Risk Management .................................................................................................... 23

9 Metrics ...................................................................................................................... 26

10 Sustainment Strategy .............................................................................................. 29

11 Contracting ............................................................................................................. 30
   11.1 Acquisition Strategy ....................................................................................... 31
   11.2 Contracting Strategy ....................................................................................... 32
11.3 Intellectual Property Strategy ........................................................................................................ 33

12 Cost and Funding .............................................................................................................................. 34
  12.1 Cost and Pricing Considerations .................................................................................................. 35
  12.2 Observations from Practice ......................................................................................................... 36
  12.3 Existing Policies and Guidance Related to Costing ..................................................................... 37

13 Program Office Staffing and Resourcing Profile ........................................................................... 37

Appendix A – Additional Guidance (White Papers) ................................................................................. 39
Appendix B – Key Agile Terms ............................................................................................................... 41
Appendix C – Case Study: Kobayashi Maru .......................................................................................... 43
1 Purpose

The DoD is working with Government and industry leaders to modify how DoD plans, acquires and delivers software products and services. Traditionally DoD projects used a waterfall delivery approach or a “hybrid” approach that did not fully adopt Agile principles. However, the increasing complexity of systems, decentralization of tools and technologies, and rapid pace of change pose challenges in delivering meaningful, secure, and modern capabilities that meet user expectations on time and within budget. Addressing these challenges requires a significant change in DoD's approach to planning, acquiring, and delivering capability to the warfighter.

Transitioning from a waterfall to an Agile approach represents a true paradigm shift. It impacts all levels of DoD including the management and execution of programs, projects, and enabling technologies. The guide provides our software acquisition workforce and supporting ecosystem members (e.g., Program Managers and their teams) with an understanding of how the delivery of capability using Agile approaches differs from traditional waterfall approaches and identifies key focus areas that merit special attention when developing strategies for acquiring systems using Agile methodologies.

The Office of the Secretary of Defense (OSD) is responsible for this document, and to provide guidance to the Agile pilot programs directed under Sections 873 and 874 of the FY2018 National Defense Authorization Act. Several white papers have already been written addressing the needs and findings encountered by the pilot programs. This document is intended as an over-arching guidebook covering major topics that DoD programs will need to think through as they contemplate changing their software acquisition strategies to adopt Agile practices, with pointers to the white papers which provide much more detail for interested readers.

The guide is organized to provide foundational Agile knowledge elements first, followed by guidance on contracting strategies for Agile solutions so that readers understand the principles gained in the foundational sections prior to considering a contracting strategy. Appendix C provides a short vignette describing a program which restructured around a new acquisition strategy consistent with the technical principles laid out in this document.

2 Scope

This guidebook provides Program Managers with information on developing acquisition strategies for Agile software development. This guide will also support all other members of the program team by providing an understanding of Agile practices. While this guidebook offers actionable information, it focuses primarily on the principles and good practices of an Agile software development approach through the lens of an acquisition strategy. This guidebook references several white papers that provide actionable guidance in specific areas related to the acquisition strategy (e.g., contracting; roadmap and MVP development; acquisition strategy; and metrics).

3 Foundational Agile Principles

Agile, as previously noted, represents a true paradigm shift in the way work is planned, executed and structured. The structure impacts organizational governance, roles and responsibilities, team constructions, and other areas. To improve the likelihood of success in Agile software development and acquisition, Program Managers should understand the mindset, values, and principles that Agile espouses.
Agile is a Mindset

The Agile mindset begins with four value statements (see Figure 1) to emphasize the factors that contribute directly to quickly delivering value to users while avoiding work that may not directly add user value.

Agile is Described by Four Values

The four Agile values are:

1. **Individuals and interactions** over processes and tools
2. **Working software** over comprehensive documentation
3. **Customer collaboration** over contract negotiation
4. **Responding to change** over following a plan.

Because the pace of change is increasing, DoD needs to remain open to developing and delivering capability incrementally. The traditional approach to product planning and delivery has focused on a predictive approach, whereby planning, requirements, and design activities are locked in as soon as possible and contracts are developed to support the initial design. The complex nature of projects, combined with the desire for a faster pace, has led to dynamics that sometimes become a point of friction between the Government and the contractor. Often, the friction is due to change requests that impact the scope of work and level of effort. Given this dynamic, if the program team is not confident in the requirements, or if the project duration is longer than a year, programs are recommended to consider incremental and iterative implementations using Agile approaches.

Agile accepts the nature of large, complex systems and acknowledges that upfront, predictive planning often leads down the wrong path and gets it wrong. The incremental deliveries characteristic of Agile allows users to receive value (i.e., benefit as defined by the user) faster and, in return, provide feedback faster to the development team, thus enabling the Agile team to apply what has been learned for the next incremental delivery. The smaller increments of work translate to smaller batches of effort that reduce complexity and initial investment cost and result in improved overall quality of delivered work.

Agile Contains 12 Principles

The Agile Manifesto also identifies 12 Agile principles:

---

1. **Individuals and interactions** over processes and tools
2. **Working software** over comprehensive documentation
3. **Customer collaboration** over contract negotiation
4. **Responding to change** over following a plan.

Because the pace of change is increasing, DoD needs to remain open to developing and delivering capability incrementally. The traditional approach to product planning and delivery has focused on a predictive approach, whereby planning, requirements, and design activities are locked in as soon as possible and contracts are developed to support the initial design. The complex nature of projects, combined with the desire for a faster pace, has led to dynamics that sometimes become a point of friction between the Government and the contractor. Often, the friction is due to change requests that impact the scope of work and level of effort. Given this dynamic, if the program team is not confident in the requirements, or if the project duration is longer than a year, programs are recommended to consider incremental and iterative implementations using Agile approaches.

Agile accepts the nature of large, complex systems and acknowledges that upfront, predictive planning often leads down the wrong path and gets it wrong. The incremental deliveries characteristic of Agile allows users to receive value (i.e., benefit as defined by the user) faster and, in return, provide feedback faster to the development team, thus enabling the Agile team to apply what has been learned for the next incremental delivery. The smaller increments of work translate to smaller batches of effort that reduce complexity and initial investment cost and result in improved overall quality of delivered work.

Agile Contains 12 Principles

The Agile Manifesto also identifies 12 Agile principles:

---

1. **Individuals and interactions** over processes and tools
2. **Working software** over comprehensive documentation
3. **Customer collaboration** over contract negotiation
4. **Responding to change** over following a plan.

---

1. Source: https://agilemanifesto.org
2. Source: https://agilemanifesto.org
1. Our highest priority is to satisfy the customer through early and continuous delivery of valuable software.

2. Welcome changing requirements, even late in development. Agile processes harness change for the customer's competitive advantage.

3. Deliver working software frequently, from a couple of weeks to a couple of months, with a preference to the shorter timescale.

4. Business people and developers must work together daily throughout the project.

5. Build projects around motivated individuals. Give them the environment and support they need and trust them to get the job done.

6. The most efficient and effective method of conveying information to and within a development team is face-to-face conversation.

7. Working software is the primary measure of progress.

8. Agile processes promote sustainable development. The sponsors, developers, and users should be able to maintain a constant pace indefinitely.

9. Continuous attention to technical excellence and good design enhances agility.

10. Simplicity—the art of maximizing the amount of work not done—is essential.

11. The best architectures, requirements, and designs emerge from self-organizing teams.

12. At regular intervals, the team reflects on how to become more effective, then tunes and adjusts.

Many of the Agile principles are traceable to Lean principles, which were popularized by Toyota in manufacturing. From the concepts and principles defined in the Agile Manifesto arose a series of Agile practices, frameworks and methodologies used today in software development.

### 3.1 Common Agile Frameworks

There are many Agile frameworks, each offering methodologies that apply variations of iterative development and continuous feedback. The most popular Agile frameworks include:

- **Scrum** – A lightweight, simple framework for teams to collaborate incrementally and iteratively on delivering value to the customer.

- **Kanban** – A framework that enables visualization of the flow of work and allows the team to monitor work in queue, work in progress, and the overall flow of work from inception through completion.³

- **Scrum of Scrums** – A scaled version of Scrum, in which multiple Scrum teams work together on a large project or program.

- **Extreme Programming (XP)** – An Agile framework that focuses significantly on engineering and development practices to bring value to the customer by producing higher quality software.⁴ Practices such as pair programming have been popularized by XP.⁵

---
³ The Kanban board concept originated from the 1978 book by Taiichi Ohno called “Toyota Production System – Beyond Large-Scale Production”, which described LEAN, Lean manufacturing and Kanban.

⁴ Source: [https://www.agilealliance.org/glossary/xp/](https://www.agilealliance.org/glossary/xp/)

⁵ Source: [http://www.extremeprogramming.org/](http://www.extremeprogramming.org/)
Additional Agile frameworks include:

- Large-Scale Scrum (LeSS)
- Dynamic Systems Development Method (DSDM)
- Disciplined Agile Delivery (DaD)
- Nexus Framework by Scrum.org
- Scaled Agile Framework (SAFe).

Each of these frameworks has different strengths and weaknesses that may be appropriate for different goals and contexts. For this reason, this Guidebook does not advocate one framework over another but encourages programs to evaluate each for fitness to purpose.

For more detail on these frameworks, reference the “Agile 101” white paper listed in Appendix A.

3.2 Agile Connection to DevSecOps

DevSecOps practices enable the Agile teams to quickly, continuously and consistently deliver value. Its practices apply to all the steps required to deliver work from initial request through operation. As Agile development practices have matured over time and development teams have become more efficient producing capabilities, the team has realized the limitations of delivering solutions without early and continuous involvement from everyone involved in delivering value to the user. This includes the requirements, design, development, testing, security, engineering and operations teams. The early and continuous involvement of the engineering and operations members is critical for the implementation of the necessary infrastructure to deliver a high-quality solution into operation on a regular and repeatable basis.

Traditional timelines for delivering supporting infrastructure could take months or years in some cases, as teams are required to produce various architecture and specification plans prior to receiving an environment. The back-and-forth handoffs between traditionally siloed organizational entities consume valuable time. Context is lost in the handoffs as well. Direct face-to-face communication and collaboration early and often reduces the time between information exchanges and builds a trust relationship to allow the team to move forward toward a common goal of delivering value to the user.

DevSecOps practices also encourage automation of repetitive practices through scripting and code-based execution. Common practices like environment deployments can be automated to reduce inconsistencies across environments. The multiple environments necessary for the successful development and operation of a system throughout its lifecycle (e.g., development; staging; production environments) suffer from inconsistencies due to differences in applied patches, updates and applications and these inconsistencies build up over time. It is important to maintain consistent environments so that as code moves from one environment to the next it is supported by the proper environment. Using DevSecOps tools and techniques to automate and test the deployment and configuration of steps involved in deployments and environments will help reduce such inconsistencies.

Testing practices can be time consuming and labor intensive. The need to repeat tests continuously due to the Agile practice of continuous delivery of capabilities places a strain on the testing teams the can lead to delayed release and can result in inconsistent application of manual tests over time. A tenet of DevSecOps is leveraging automation to relieve strain of repetitive tests and also can result in improved quality of products over time.
Monolithic architectures and bloated code are another contributor to the slow pace of change. Architectures developed to promote large-batch releases do not align with loosely coupled, modular architectures that support incremental and distributed changes by multiple teams. As a result, relatively small changes can require significant collaboration among multiple teams, which increases product life cycle costs and overall risk of ownership. In response to the architecture challenges, DevSecOps practices promote modular architectures to provide the technical space needed to support continual improvement of product code and design. DevSecOps practices promote planning-in time for refactoring and improving the code as new technologies and solutions become available. This practice helps reduce technical debt and fosters innovation. Building-in time during sprints for refactoring and learning also reduces team burnout from the repetitive pressures of continuous delivery demands.

DevSecOps adoption encourages product teams to devote attention to the end-to-end flow of value delivery, or the value stream. Applying such a transformational change in the way that technology is managed requires collaboration among all the teams involved in the delivery of value to the user. The steps involved to plan, develop and deliver capability (or value) to the user is considered the development value stream, and requires collaboration among everyone involved in the value delivery chain (e.g., requirements, engineering, design, development, testing, security, deployment, and operations functions) to work toward a common goal of continuously delivering value to the user.

DevSecOps practices promote Lean principles of flow of work through implementation of practices that enable continuous integration and continuous delivery (CI/CD) of capability via use of a delivery pipeline. The delivery pipeline encompasses all steps and tools necessary to support CI/CD activities. DevSecOps CI/CD practices encourage automation to enable consistent and repeatable practices within the delivery pipeline.

Additionally, DevSecOps practices align well with use of cloud-based services to take advantage of service-based applications, platforms and infrastructure that enable teams to quickly respond to infrastructure and scaling needs. Through implementation of good practices such as modular design, automation, end-to-end flow, continuous integration, and continuous delivery, DevSecOps practices enable Agile teams to deliver value to the user on a continuous cadence.

For more detail on DevSecOps, refer to “DevSecOps Best Practices” white paper listed in Appendix A.

3.3 Agile Connection to Lean

Agile and DevSecOps practices are based on Lean principles. Lean principles provide teams with the context behind why Agile practices, roles and supporting processes are defined as they are, and more importantly, helps teams understand better how to tailor Agile practices while maintaining the desired Lean outcomes. Gaining a clear understanding of the underlying Lean principles helps to prevent Agile-in-name-only practices and anti-patterns that result in less than desirable outcomes. The lackluster results lead organizations to conclude that Agile does not work. Avoiding this pattern requires that organizations understand the Lean principles outlined here:

- **Deliver value quickly and continually** - Manage the flow and batch size of work in order to increase the pace of feedback and opportunities for learning. To be able to deliver value quickly, the organization must have leadership support, training and staff capability, and a clear understanding of Agile and DevSecOps practices.
- **Build-in quality at all levels of work** – To accomplish this, there must be accountability at all levels of work. The Agile team must avoid single point quality checks in favor of quality checks at
all levels of work, which involve everyone on the team from developers to testers, security, and operations team members. Standardizing and automating as much as possible to avoid issues resulting from inconsistent application of testing and quality measures will help reduce human error (e.g., environments will remain consistent if applied consistently).

- **Eliminate waste** – Reduce the urge to “gold plate” or overbuild based on assumptions of desired user value. Eliminating waste also encourages delivering capability in small batches to reduce complexity and possible rework resulting from issues that naturally rise from complexity and adopting an MVP design to begin to gain valuable feedback for incremental improvements.

- **Make decisions at the last responsible moment** – This does not mean that the team should procrastinate. Rather, the team should avoid locking down a design early in the product life cycle and instead learn and adjust as needed by taking advantage of the information gained in the Agile continuous delivery process. With each release, the team has an opportunity to learn and adjust.

- **Take a scientific approach** – This includes making data-driven decisions based on testing specific hypotheses and iterating based on the findings. Examples include application of the MVP concept to enable testing of smaller pieces of functionality, experimentation, and increased collaboration with users through application of user-centered design concepts to test usability before implementing.

- **Create a learning culture based on respect** – Promote continuous learning, improvement, and innovation, and encourage a blameless culture that allows teams and individuals to take calculated risks and learn openly.

## 4 Defining the Capability Need

Agile approaches to software avoid the need for very detailed upfront, predictive requirements capture. That is, they dispense with the idea that through sufficiently rigorous analysis, all of a system’s requirements can be determined and specified upfront. In contrast, Agile approaches begin with a high-level capture of business and technical needs that provides enough information to define the software solution space, while also considering associated quality needs (such as security). This activity aims to define the value proposition of the capabilities to be developed, and the expected impact to the operational mission.

An initial description of a capability need should:

- Aim for a length and complexity of documentation that best matches to the size, scope, and complexity of the operational need;

- Be focused on operational needs, key features and interfaces, not detailed system specifications;

- Convey high-level priorities, timelines, and operational constraints.

Agile principles acknowledge the “cone of uncertainty” and stress collaboration and communication between product owners and developers throughout the program life. Agile practices are open to requirement changes in support of customer satisfaction and needs. The approach acknowledges that identifying all requirements (i.e. tasks, features, user stories) upfront isn’t realistic and it also restricts the team from building the best solution. Therefore, focusing on capabilities provides the right level of definition for program planning and development priority. These capabilities should be displayed on the product roadmap. (More information on roadmaps is found in Section Error! Reference source not found.)
4.1 Epics and User Stories

In an Agile program, capabilities are documented by the product owner (with support from the end-user/warfighter) as epics. An epic communicates a high-level need. For example, an epic may be the ability for a customer service representative to create a new customer profile. This capability / epic can be managed on a roadmap in order to help convey where the priority for this capability falls in relation to other capabilities. The epic provides a high-level description of why the end-user/warfighter needs the ability to create a new customer profile, clarifying the “why, where and how” without going deep into implementation detail.

<table>
<thead>
<tr>
<th>Epic</th>
<th>A large body of work to be completed during development. Depending on the Agile framework, the Epic can be too large to complete within a sprint. Epics are further decomposed into smaller features and user stories. Epics may express business functionality or identify constraints placed on the product or system.</th>
</tr>
</thead>
<tbody>
<tr>
<td>User Story</td>
<td>The smallest unit of requirements written from a user’s perspective of how they will use the software. User stories are defined and prioritized by the Product Owner via backlogs. User stories that cannot be completed within a single sprint should be divided into smaller elements. Each user story should have clear acceptance criteria.</td>
</tr>
</tbody>
</table>

When it is time for the epic to be worked, the development team works with the product owner to decompose the epic into smaller units (user stories) to be developed. For example, the customer profile can have a user story that the customer representative, while on the phone with a new customer, needs the ability to input basic customer definition details which includes capturing the customer’s first name, middle initial and last name. The middle initial can be further clarified as an optional field since not all customers have a middle name.

The user story also includes capturing the acceptance criteria that will be used during testing. Acceptance criteria answer the question of when the user story has been successfully implemented and can be considered “done.” The acceptance criteria are applied during test and verification of the user stories. User stories identify the product owner who owns the user story and is responsible for reviewing and accepting (or rejecting) the work.

The acceptance criteria are used during testing activities that take place during the development sprint cycles. The same criteria are used when end-user / warfighter (user acceptance) testing is conducted. The acceptance criteria are also reviewed during the sprint and/or iteration demonstrations, during which the product owner and end-user/warfighter see all the work accomplished and have the responsibility to communicate whether it is approved and should be noted as “complete”.

Note that during sprint and/or iteration demonstration ceremonies, the product owner may define new requirements and/or changes to priorities. They also see and/or have used the solution allowing them to clarify design preferences and identify options they might not have realized before.

4.2 Tracking Progress

The product owner is responsible for the product backlog. The product backlog identifies what work / capability is in progress and what is expected to be worked. The sprint backlog identifies the user stories that are currently being worked, as well as user stories that are available for the next one to three sprints. The development team manages the sprint backlog.

The user stories identified in the current sprint backlog are noted as planned/in-progress. Only once the work is complete, demonstrated, and has received product owner approval, can the user story be updated/noted as complete.
Once all the user stories associated to a product backlog item have been noted as complete, the product capability can also be noted as complete. Updates to the product roadmap can be made based on the product backlog status.

For more detail on the Capability Needs Statement document and associated Agile strategies for managing and decomposing capabilities to be developed / acquired, reference the “Agile Acquisition Strategy” white paper listed in Appendix A.

5 User Engagement

User commitment and involvement is essential to building and delivering a successful Agile-based solution. Under an Agile paradigm, the continuous delivery model requires that users become involved earlier and more often to align with the cadence of Agile work. If the Agile team is to deliver iteratively and incrementally, then the users should be involved in the continuous planning, review and testing processes in alignment with the Agile team’s delivery cadence. Specifically, users will have a role to play in:

- **Capability and feature identification and prioritization** – Users will work with the product owner to identify and prioritize the overall capability needs and a roadmap that lays out a coherent approach to delivering the major capabilities over time. For each capability, users will shape software features and determine prioritization.

- **Define a minimum viable product and roadmap** – Users will work with the product owner to define a minimal version of the software that can be shared with the end user to start collecting feedback (i.e. Minimum Viable Product (MVP)). Users will also provide input to the product owner to develop a product roadmap that maps how the product will evolve and be delivered over time.

- **End user-focused testing** – Users will work with the product owner to formulate test plans and document the “definition-of-done” for features and capabilities. That information will provide the development team with additional context and clarity regarding user expectations for the defined feature or capability.

- **Deployment and rollout decisions** – Users should specify what sites will receive new features/capabilities and the sequencing of rollout locations. Users should also document the decision-making authorities for ‘go/no-go’ decisions.

5.1 Identifying Users

At the most basic level, the key Agile team roles are the program manager, the development team, and users. Within the user role, each program may have a unique set of specific functional roles. The list below identifies some of the typical functional roles examined by pilot DoD programs involved in cooperative software development programs. Depending on the context of a given program, some of these roles may be played by the same individual.

- **Operational sponsor**: the sponsor represents operational needs at the highest level.
- **Capability sponsor**: owner of the capability and responsible for providing input into prioritization of features in the overall product roadmap.
- **Program manager**: responsible for contracting for, acquiring, delivering, and deploying the software capability.
- **Requirements manager**: responsible for development and maintenance of user requirements in the initial capability needs definition and in greater detail as the program evolves.

- **Operational users**: the intended end users of the software system. They work directly with the software development team during the software development cycle.

- **Other impacted stakeholders**: others directly or indirectly impacted by the product.

Defining users can be tricky and not always straight-forward. Understanding the size of the user community can help shape the user engagement strategy. If the primary, or core, user community is too large to engage with every user, it may be beneficial to identify proxy users to represent the interests of the larger community of core users during development. To gain feedback from the larger community, the program could perform a limited operational release to a subset of the core user community. Alternatively, the program could deploy an alpha or beta release to early adopters during a period of parallel uptime.

### 5.2 Users and the Product Owner

The product owner serves as the primary liaison and representative of the user community by reflecting user requirements and priorities daily as a primary member of the Agile team. In this role, the product owner ensures that the product backlog reflects the noted requirements and priorities. As noted earlier, the product owner is the primary lead for backlog and roadmap grooming and updates, and as such, must have a good understanding of the solution domain.

Given the significant responsibilities of the product owner to represent the user community and serve as the daily bridge between users and the technology teams, the product owner should have the ability to dedicate a significant portion of time to the program. The product owner should align the vision with the execution of work by ensuring that the work done and the priorities set reflect the vision. The product owner should also understand the important role of collaboration as they are a key collaborator across the entire team. Last, but not least, the product owner must understand Agile values and principles, as must the entire team, including the program leadership.

The actions and decisions of the product owner significantly contribute to the solution’s success, or lack thereof. If the product owner cannot allocate the time to regularly groom and prioritize the backlog, then there is a risk the program will not focus their activities appropriately, leaving higher value work unfinished. The team will also lose the connection to what is truly high priority work, making the nature of work performed susceptible to pet features and bloat. To avoid such outcomes, good product ownership requires that the product owner understand and commit to Agile values and principles, particularly the role of iteration and experimentation; commit to delivering outcomes that align with the larger vision; and possess an intuitive, curious and collaborative nature.

### 6 Program Roadmap

Corporations communicate to industry and employees their strategic goals and plans for the next planning year. Internal, organizations and divisions align to the corporate strategic goals and their programs and projects follow suit. To support delivering on the strategic goals, leadership requires an understanding of the products being developed to understand the plan and track progress. The program roadmap is a high-level representation of a program’s direction. It provides a visual highlighting program milestone(s) (delivery of strategic capability), and linkage to any corporate initiatives and/or external dependencies in the coming
year. Note, program roadmap timelines are typically selected based on the level of change an organization is driven by but in general they show 12 to 18 months of planning.

When developing a program roadmap, it helps to review the program vision, goals and strategies to help focus input. Items that are external but are strategically significant for a program along with items that can impact program progress and delivery should be noted. Risks, roadblocks, external dependencies and/or events that need to be highlighted at the leadership/program level are also appropriate to display for awareness.

The program roadmap and product roadmap complement each other. The product roadmap is discussed in later sections but release data from the product roadmaps is incorporated into the program roadmap. The program roadmap visually depicts how individual products align with each other and communicates planned delivery of strategic capabilities. The program roadmap shows the timeline of releases for each product.

An Agile release is made up of work completed in multiple iterations (sprints). It is recommended that release cycles follow Agile principles and are kept short and frequent. Agile releases are driven by the product owner (in conjunction with end-users) identifying that enough “value” has been delivered (by the sprint process) and capability is ready to be deployed to production. The product roadmap provides product specific details while the program roadmap identifies the linkage between products and major milestones in relationship to organizational strategy.

Figure 4 shows the program roadmap for the new generation force support. Three products are identified on the roadmap (aircraft, autonomous rover, action rifle). In the upper right corner, the external milestone “Public launch of New Gen” is identified to highlight all products will be part of the launch and leadership is tracking that date. Go Live milestones for the autonomous rover and action rifle are also highlighted for leadership awareness/tracking.

Agile programs break work down into the smallest pieces for development to help minimize risk, provide ease of management and delivery. Roadmaps are defined at a higher-level to provide direction and allow development teams to clarify design and details when time appropriate. Roadmaps are also recommended to not look out past 12 to 18 months because the uncertainty increases, and change becomes more certain. But in the Government space there are often larger and more complex system solutions with large budgets, critical delivery dates, and end-user commitments that stretch the timeline recommendations.
Programs such as the Veterans Affairs (VA) and DoD's health care solution programs are examples of complex programs. These program solutions are built using many development teams with multiple contractors, external suppliers, and they often have long lead times for developing hardware and major subsystems. For these types of solutions, longer planning and investment horizons are typically requested and consequently, longer-term roadmaps are built to support these needs. Based on recommendations, the most detail is provided for the near-term delivery functions, moving to broader time intervals and less specificity the further out the planning horizon goes.

![Figure 3. Long Term, Complex Solution Program Roadmap](image_url)

Figure 5 illustrates a solution roadmap that covers about three years. The first year is planned in quarters (e.g., Q1, Q2, etc.), which may or may not align with program planning boundaries. The second year is planned in six-month increments (e.g., H1 and H2). Anything beyond that is scheduled in years (e.g., Y3 and Y4). As the time horizon extends, the level of granularity and certainty is significantly reduced. This roadmap also uses color coding to highlight capabilities/work to be delivered based on business drivers. For example, programs can use color coding on program or product maps to highlight capabilities that are focused on usability, technology, and financial.

### 6.1 Minimum Viable Product and Deployment Cadence

Agile practices are built around frequent, small batch delivery of working functionality into the hands of end-users to gain fast feedback with short learning cycles that enable the development team to hone the solution. These quick iterations mold the end-product to better meet the evolving needs of the end user. Agile practices use the Minimum Viable Product (MVP) concept to refer to the first delivery (to an environment where it can be accessed by the end-user) of meaningful capability of a product. The MVP is a collection of features that provide just enough capability to accomplish minimum functionality for the targeted end-user so that this early user of the system can provide feedback to influence future development iterations. To successfully achieve this approach, the MVP should be a manageable piece of work that provides the ability to use and assess how the targeted system solution performs in a production setting and give that fast feedback to the system development team to be incorporated into future iterations.

---


7. [https://www.scaledagileframework.com/roadmap/](https://www.scaledagileframework.com/roadmap/)
It is important that the MVP capability is evaluated in their native environment. The MVP can be placed into a production environment and possibly limited by number of users and/or locations so that the end-user can experience the capability to provide feedback. Often in the Government space, MVP’s are delivered to staging environments that mimic production and provide the end-user a place to experience the capability to provide the development team with needed feedback.

Deployment cadence for the MVP and iterations that follow can be supported by a DevSecOps pipeline to enhance reliability and speed. A DevSecOps pipeline and the supporting tools can provide the ability to “shift left” security and operational tasks that traditionally came after development and often impacted schedules. The ability to automate testing and improve the speed to perform testing supports the desire to deliver and deploy often and on a repeatable timeline.

End-users’ active participation is required to define the MVP, as well as to clarify and make requirement decisions, and provide feedback for future iterations. The end-user is also responsible for approving that the delivered MVP met the requirements. The approval process typically takes place after the demonstration of capability is performed.

The layers of the pyramid in Figure 6\(^8\) identify some aspects of a software solution.

- **Functional** – The software solves a problem.
- **Reliable** – It is available when the end-user needs it.
- **Usable** – It is easy to learn, use and remember.
- **Empathically Designed** – The end-user has a positive reaction

The pyramid on the left (in Figure 6) describes an MVP deliverable that is focused mainly on functionality. This is not recommended because focusing only on limited functions, the end-user doesn’t have enough of the solution experience. To gain meaningful feedback from the end-user the MVP needs to present a

\(^8\) [https://dvmobile.squarespace.com/dvmobile-blog/tag/Lean](https://dvmobile.squarespace.com/dvmobile-blog/tag/Lean)
reliable, functioning solution with enough design usability for the end-user to react. In contrast, the pyramid on the right highlights the need for the MVP to be a vertical slice of the pyramid. Delivering minimum capability of infrastructure, functionality, and design so that the end-user can evaluate the experience and provide helpful feedback. The delivered MVP (slice) should allow the end-user to experience a piece of the proposed solution in a state that provides learning.

The Product Owner/Product Manager (collectively referred to as the product management team, for larger systems solutions) and development team(s) should work together to identify and agree upon the MVP. The product management team is responsible to ensure the MVP provides end-user value and the development team identifies whether the scope of the MVP is feasible to achieve within the selected delivery period. When defining an MVP keep in mind the following:

- **Select and Engage the User Community/End-User** – The product roadmap may impact multiple end-user groups and the MVP may select a subset that will be the focus of the first deployment. Make sure those end-users are identified, agree to their role, and understand what is expected of them and are engaged from the start.

- **Think Simple/Bare Bones** – Keeping the identified user community/end-user in mind, select the simplest possible, most bare bones capability to focus on delivering. Typically, this should be an end-to-end scenario or thread of functionality that is demonstrable and allows the system/product to perform in the production or near-production environment, providing end-user capability, and subsequently providing the development team with valuable user feedback to incorporate into future iterations.

For more detail on MVP, reference the “Agile Roadmap and MVP” white paper listed in Appendix A.

### 6.2 Minimum Viable Capability Release (MVCR)

Delivery of new capability to production may require additional work by individuals with a different skillset and focus from the scrum team. Depending on the type of solution being delivered (i.e. health care, weapon system, transportation system) additional rigor may be required prior to deploying the software into operational use. This work traditionally requires, for example, development of materials to support training, authorization to operate, and security testing. Agile practices strive to pull these activities into the sprints and minimize sequential and time-consuming bottlenecks before delivery. DevSecOps pipelines support the ability to automate testing and pull in security validation, allowing for continuous integration and continuous delivery, but there are still situations where additional work is required before solutions can be deployed to production. These external efforts are focused on packaging the solution.

While Agile practices strive to “shift left” test activities, due to the nature of some systems and specifically weapon systems, additional productization and packaging actions are required before software and hardware is deployed. Weapon systems require rigorous hardware and software verification and validation before deploying to the end-user. They also must follow policy dictating the need for realistic system developmental and operational test and evaluation (T&E) along with training, doctrine, and tactics, techniques and procedures (TTPs).
### 6.3 Product Roadmap and Graphic

A roadmap is defined as “any plan or guide to show how something is arranged or can be accomplished”\(^9\). There are multiple formats and types of roadmaps that development teams and organizations use. The format and detail within the roadmap reflect the audience needs. As mentioned earlier, the program roadmap is generated to provide top-level information and shows multiple products. The focus of the product roadmap is on one product. The feature sets identified on the product roadmap provide focus and align the development team(s) with the product management team on the priority of the feature sets to be delivered first. The product roadmap should be reviewed frequently (frequency being driven by the level of change and the emerging system solution). This review helps ensure the necessary alignment of the teams with the priorities of the product management team. This section provides examples of different product roadmaps and describes how they can be used.

![Product Roadmap Diagram](image)

**Figure 5 “Now Next Later” Product Roadmap Example**

The Now-Next-Later product roadmap is a visualization focused on planned capability/functions of the product. This model is useful for communicating with program stakeholders of all types, “since it’s easy to understand that you are working on the stuff in the ‘now’ part, the ‘next’ part is what is coming up soon and the work to be done in the ‘later’ part is further away in time and still has to be prioritized.”\(^10\) This model allows stakeholders to see the progression of capability to be delivered and timing is noted in very general

---

\(^9\) [https://www.dictionary.com/browse/roadmap](https://www.dictionary.com/browse/roadmap)

terms of the immediate (now) and future (next and later). What this model lacks, is a gauge for timing (months/quarters) of capability delivery.

A product roadmap can provide a rolling calendar-based view of key feature sets to be delivered. Time periods often depict the near term as 10-12 weeks or first quarter through the coming 12-18 months.

The product roadmap seen in Figure 8, illustrates a goal-oriented product roadmap mapping roughly a year of work. The goal row identifies the end-user objective to be met (the “what and why” of the delivery). The roadmap shows that the MVP is set for the March timeframe, with the goal to deploy an online product catalog capability. The goals across all releases should provide enough information for team members to understand priority and focus of a release. The features row summarizes, using simple language terminology, the “how” of the delivery. The features associated to the goal are identified at a high enough level to allow developers and product owners to design and generate user stories to clarify/refine the features. The features identified for the MVP focus on product definition and the algorithms needed to calculate product prices. The metrics/milestone row provides a description of the full end-user value, the acceptance criteria for the features (definition of done). For the MVP, the roadmap identifies moving from a paper to electronic catalog is a milestone along with the metric of 500 products being established in the online product catalog.
The product roadmap displayed in figure 9 provides a view into the product development team structure. This roadmap highlights development teams responsible for the features noted in each of the three “swim lanes” that make up the product. Using the online product catalog example, figure 9 shows the web team is responsible to deploy the online product catalog, the second swim lane is for the mobile device team (i.e. product catalog app for the phone) followed by the marketing team swim lane. All three teams need to work in parallel to deploy a complete solution.

No matter the type or format of product roadmap, the product owner should routinely update/review the product roadmap. Establishing a cadence for product roadmap review and sharing the information with team members and stakeholders supports open communication and ensuring everyone has the same information.

Incorporating the Agile principles into the daily work processes and changing the corporate culture to embrace Agile principles is required to effectively deploy use of the product roadmap. The time consuming, lengthy process of developing detailed program schedules is no longer needed. Focus is on delivering capability to the end-user and product roadmaps communicates priority of needs and delivery. It is imperative that the product owner prepares and is responsible for the product roadmap and establishes frequent reviews, on a predictable cadence, while working along with development teams. The product roadmap needs to be at a high enough level of abstraction to allow teams to elaborate and decompose the high-level capabilities and features to the emerging system design.

The lower-level user stories and tasks should not be referenced on the product roadmap. A product roadmap neither assigns work to specific teams nor dictates explicit design. The Agile product roadmap provides a prioritized, value-driven view of work to be performed.

The functionally oriented product roadmap helps the team set a pace for work and provides just enough information to plan work for future sprints. The roadmap depicts the capabilities or features to be developed over time while avoiding overly prescriptive plans. Because Agile program roadmaps are defined at a higher level of abstraction, they promote/allow the teams to collaborate on lower-level requirements and design details.

The product roadmap supports alignment of the development team with the product team to ensure a common understanding of delivery, priorities, and key events to enable teams to better plan together for the critical events. In an Agile environment, the roadmap drives definition of feasible key feature sets (or capabilities) and supports prioritization and timing of work to be accomplished.

For more detail on Agile roadmaps, refer to “Agile Roadmap and MVP” white paper listed in Appendix A.

6.4 Capability Management

Traditional program requirement definition focuses on identifying, defining, and documenting requirements fully in the beginning of the program followed by development of the system against those requirements. This method assumes the business can identify and document all requirements upfront. These requirements are noted in contracts and to change the requirements (during the software development/program lifecycle) typically requires a complex and slow change control process. History has shown that programs developed in this manner run into issues at deployment. Translation from requirement to solution often don’t match original intentions, business changes have occurred, and the solution ends up not meeting the end-user expectations.
Agile principles acknowledge the “cone of uncertainty” and stress collaboration and communication between product owners and developers throughout the program life. As mentioned in earlier sections, product roadmaps provide a prioritized map of capabilities that can be adjusted based on changing needs. The product roadmap guides the team to decompose top priority capabilities into features, demonstrate completed work and allow for the product owner/end-user feedback to guide future work. The development methodology is open to requirement changes in support of customer satisfaction and needs. The methodology acknowledges that identifying all requirements (i.e. tasks, features) upfront isn’t realistic.

When capability is selected to be worked on, the development team(s) have responsibility to break the capability into further detail for development. User stories are defined through interviewing and/or observing end-users to help clarify needs. The user stories are then divided into development sprints based on scrum team capacity and overall management. Storing of the detailed requirements/user stories is referred to as backlog management. Strategies for managing backlogs are:

- **Product Vision/Roadmap** – Prior to managing the product backlog, awareness/review of the product roadmap and vision is a good practice. If adjustments have been made to the vision or roadmap, that can translate to priority changes and / or new requirements.
- **Dependencies and Risk** – During prioritization identification/review of dependencies is key for program management. If external teams/systems are identified as dependencies, they tend to also be identified as a risk. Communication, coordination and management of the dependencies and risks is necessary.
- **Open Communication** – The team should understand the priority and be able to view the backlog. Clarifying and working on user stories, while typically owned by the product owner, including developers and providing awareness to the team is beneficial to all.
- **Keep it manageable** – Focus on near term needs. Don’t overload the backlog with as many user stories and requirements the team can think of. Use a tool that fits the needs and provides the ability for traceability.
- **Planning** – The backlog should contain requirements and user stories to allow planning for the next two or three sprints. This provides opportunities to identify and pull in stretch goals and program manage dependencies and risks.

After each sprint, a working product is available and demonstrated to the product owner/end-users/stakeholders so that they can visualize progress and can identify any necessary adjustments that may need to be incorporated in the next sprint. Agile practices anticipate that changes will occur, and the product owner is expected to ensure the backlog reflects any adjustments needed. The development team can react/adjust between sprints and releases based on product owner/end-user feedback.

A couple of challenges and benefits Agile programs should keep in mind are:

- **Non-Functional Requirements** – developers and end-users are often focused on design (what they can see) and can neglect security and performance requirements. This can be mitigated by ensuring that teams have participation from all practical areas (e.g. operations, security, test). To realize "shift-left", the ability for continuous integration (CI), continuous delivery (CD), and DevSecOps capability, these requirements should be thought of at the beginning of programs and designed and built in. Pushing off these requirements to later releases makes it more difficult and is not recommended.
- **Limited or no documentation** – teams that are used to working with traditional programs are accustomed to receiving detailed documentation upfront. As previously noted, this documentation can become out of date quickly and can detract attention from delivering value to the end-user.
Most Agile programs use a tool to define and store their product and sprint backlogs. If all requirement data is stored in the tool, any stakeholder can view the current status of a requirement at any given time. Information is organized to understand what is currently being worked, what has been delivered and what is next. The benefit to the program is always being able to view updated and current data. The tools also allow traceability from the requirement to test scripts, test results, and deployment. While this is a benefit, team members/stakeholders need to accept and work with the new tools to gain the benefits.

6.5 Schedule Interdependencies

Existing traditional / legacy processes and procedures that have not changed to support “shift left” principles, may impact an Agile program’s ability to stay on schedule. There are existing Government systems that were originally developed using traditional software methods and are still being maintained using traditional sustainment methods. These systems typically have longer development and release schedules that are planned out months / years in advance and are not as flexible to incorporate change.

New Agile programs often need to interface and/or have dependencies with the traditional systems. These dependencies require work not only for the new system but also the legacy system. Traditional processes and procedures that software systems may need to follow to achieve security and authorization to deploy may also impact system agility. These dependencies can derail an Agile release schedule.

Agile teams should use their program and product roadmaps to identify possible dependencies early. During sprint planning and work refinement activities, dependencies and risks should be identified, tracked and monitored at the appropriate levels. It is imperative to synchronize agile and traditional teams when dependencies are identified.

Some items that could possibly impact program schedules include:

- Business process change / reengineering
- Database / data structure changes
- System interface requirements
- External test and security processes
- Certification /Authority to Operate (ATO) processes

Traditional programs typically rely on documentation that most Agile programs do not produce so working through this change is important. While the traditional document may not be produced the data can be found using Agile tools. The benefit for using the tools, compared to the use of traditional documentation, is it provides a current snapshot of work completed and designs.

Identification of traditional program needs upfront and developing a plan for working together will help minimize impact. Traditional programs may have business processes that they must follow. One example, traditional program review gates that require executive level coordination and review that could prevent or impact delivery dates. It is recommended to keep all stakeholders involved and part of Agile efforts. If daily interaction is not an option with executive stakeholders, inviting them and having their participation in development demonstrations is important. This allows the executives to see the working solution and ask questions.
7 The Software Development / Deployment Infrastructure

Before development can commence, there are many decisions that need to be made regarding the underlying infrastructure that both a deployment pipeline and a deployed system will exist on. These decisions include

- architectural choices regarding how to organize the software being built;
- the development environment and tool suite that implements the “pipeline” that moves software from the development team through to operations; and
- choices about whether to host, build, and provision infrastructure on premises, in the cloud, or utilizing a hybrid approach that gives the flexibility of both to cater to specific business or mission requirements.

7.1 Intro to Cloud

Understanding what the cloud is, how it can be used, and what its benefits are is crucial to implementing a successful and efficient software pipeline. The cloud is a network of computers which are accessed over the internet, that provide various types of services. These services may provide functionality for end users to consume, or may also provide services that enable capability within an automated deployment pipeline. The main advantages of the cloud are the scalability and reliability offered for the infrastructure which is utilized. Infrastructure created can be easily customized to ensure that there is never downtime through auto-scaling due to resource demand, as well as accessibility across various availability zones to ensure redundancy.

Any service that may be hosted on-premises is available in the cloud, whether it be a custom service, or a service provided by a specific cloud provider. Both the cloud infrastructure, and the software development platform that run on it, operate on a “pay for what is used” model. It is important to recognize that cloud computing costs are often unpredictable and billing can be extremely complicated. If choosing to use the cloud, it’s highly recommended that personnel with the appropriate architecture certification or highly competent understanding of the costs of specific cloud resources, for the chosen cloud provider, are chosen. Similar specialized expertise is needed to address software architectural decisions that will be best for specific mission goals. Choosing commercially available services offered by cloud providers and then implementing custom solutions will be much more cost effective and will also allow end users to begin consuming cloud resources at a faster rate due to ease of setup.

Important key introductory cloud takeaways:

- Programs should explore the cost-effectiveness of using enterprise cloud services, and/or subscriptions to commercial cloud services, as opposed to creating entirely custom solutions.
- Cloud resources and storage are effectively infinite, only limited by program financial constraints.
- The speed of the cloud is only as fast as the internet connection.
- Programs should understand the capabilities and the APIs between vendor-based solutions, and be aware of the possibilities for, and risks of, vendor lock-in.
- There is no “cookie-cutter” cloud solution that works for all projects.
- Focus on utilizing cloud services/infrastructure that bring a true benefit to the program. While this can be difficult to determine for most programs at first, consider what data collection can be useful for monitoring whether expected benefits are found over time.
7.2 Platform/Infrastructure Technical Decisions

It is important to understand that there is no one size fits all model for correct infrastructure choices. Architectural decisions must be made based on what infrastructure and services will best support requirements and Service Level Agreements (SLA’s)/nonfunctional requirements. There are five paths that can be chosen: on-premises, Infrastructure as a Service (IaaS), Platform as a Service (PaaS), Software as a Service (SaaS), and a hybrid\(^{11}\), each of which have their advantages and disadvantages. The following should be considered when selecting infrastructure solutions:

- Cost
- Security
- Agility and Scalability
- Reliability
- Deployment

![Figure 8 Infrastructure responsibilities](image_url)

7.2.1 On-Premises

The traditional on-premises solution has been favored in the DoD until recently due to the peace of mind of complete ownership of infrastructure, full control of locally stored data, and most importantly the former lack of cloud solutions that provide classified environments.\(^{12}\) Although the DoD is now aggressively moving to the cloud, the use of some form of on-premises solutions (hybrid) are still being utilized, examples include:


testing environments for embedded systems, disconnected operations environments, etc. Before implementing an on-premise solution, the following should be kept in mind:

- **Cost**: Implementing an entirely on-premises solution may be cheaper in the long run, but there will be considerable upfront costs for hardware, employee effort and potential technical debt due to infrastructure misconfiguration that must be addressed.

- **Security**: The product owner(s) will be in full control of not only the physical security of the infrastructure, but of the locally stored data as well. This can be a risk however if the proper expertise is not available to create and maintain a proper security posture.

- **Agility and Scalability**: Since the product owner(s) have control over the physical infrastructure, upgrades can be monitored and implemented as much as budget allows. However this requires an extensive amount of planning for system downtime and maintenance. Additionally, specific configuration of physical systems could potentially lead to infrastructure that can’t be repurposed.

- **Reliability**: Proper infrastructure can be selected and configured to provide enough redundancy to meet SLA’s, but can be costly to implement when 99.9% uptime is required.

- **Deployment**: Automation can be used in coordination with on-premises infrastructure to deliver/deploy code.

### 7.2.2 Infrastructure as a Service (IaaS)

IaaS allows for the complete purchasing and customization of a virtual datacenter. It is by far the most flexible of the three service options. However, it still requires an immense amount of architectural knowledge (primarily to avoid overpaying for choosing an incorrect or inefficient infrastructure), and system administration costs since users are still responsible for managing every aspect of the provided infrastructure. Before implementing an IaaS solution, the following should be kept in mind:

- **Cost**: All major IaaS providers offer subscription-based fees that allow for a rough forecasting of costs. However, the need to purchase licensing for services hosted on the provided infrastructure may considerably increase overall implementation cost.

- **Security**: Offers complete access control and security of data storage. The proper configuration of access to data offered by API’s and storage entities must be ensured. Employees responsible for this configuration should have the appropriate certification(s) for the IaaS provider that is being used.

- **Agility and Scalability**: Resources can be configured to provide automatic load balancing and scaling to handle increased demand. This takes advantage of containers/microservices which requires an additional skillset.

- **Reliability**: Resources can be distributed over different geographic regions and/or cloud availability zones to provide redundancy.

- **Deployment**: Automated tools can be used in combination with PaaS and SaaS tools to deploy/deliver code to the provided infrastructure.

### 7.2.3 Platform as a Service (PaaS)

PaaS solutions remove the responsibility of creating, configuring and managing the underlying infrastructure that is necessary for delivering value into an environment. Additionally, they offer built in automation that cuts out much of the code that is necessary to automate deployment and rollback. There are some limitations to PaaS solutions, however. Many services may be limited to specific frameworks,
limiting development to certain languages, dependencies, and versions, as well as a lack of development of custom dependencies. Before implementing a PaaS solution, the following should be kept in mind:

- **Cost:** Taking advantage of a PaaS can help to drastically reduce labor costs due to the immense amount of automation built into the services.
- **Security:** Although the underlying infrastructure is provided in a PaaS solution, it is still the user’s responsibility to ensure that the versions of software chosen through a PaaS do not have inherited vulnerabilities.
- **Agility and Scalability:** Autoscaling and availability across geographic regions are features often provided by an underlying IaaS solution.
- **Reliability:** Users can often be at the mercy of the cloud provider for availability options.
- **Deployment:** Built-in deployment/delivery automation allows for developers to quickly deliver value without having to distract attention from feature development.

### 7.2.4 Software as a Service (SaaS)

SaaS solutions offer extreme ease of use of an entire software application or service over an internet connection through a web browser. There is no responsibility for, or access to, the underlying infrastructure that the applications are hosted on. This extends even to managing the administrative functions of the application itself, which eliminates a layer of responsibility and allows anyone using the SaaS to be a true end-user. Before implementing a SaaS solution, the following should be kept in mind:

- **Cost:** SaaS solutions are either free to use, for a restricted version functionality-wise, or most commonly offered through a monthly or annual subscription fee.
- **Security:** The SaaS provider is responsible for securing the entire stack that provides the application or service. However, the SaaS provider is not responsible for securing any customer data or proper user access control. Data loss prevention techniques should be implemented to ensure that user data cannot be exfiltrated. Proper data access policies should be defined.
- **Agility and Scalability:** The SaaS provider is responsible for ensuring the agility and scalability of the services provided.
- **Reliability:** The SaaS provider is responsible for ensuring the reliability of the service provided.
- **Deployment:** End-users will not be deploying code for a paid-for SaaS.

The most common use of these above options is to create a hybrid solution that may utilize some combination of on-premises, IaaS, PaaS, and SaaS. It is recommended that an assessment be conducted to determine:

- Available stakeholder expertise
- Current, project relevant, usable infrastructure
- Services of the deployment pipeline already in-place either on-premises, or IaaS/PaaS/SaaS
Evaluating these categories will allow an organization to assess which hybrid combination is the best fit for their business requirements and mission. Certain programs may prefer to own the infrastructure which houses all application data with an on-premises solution but take advantage of the ease of setup and automation provided by certain PaaS and SaaS services in the cloud. Each hybrid solution will vary based on current stakeholder expertise and number of stakeholders available.

For more detail on DevSecOps and associated technical decisions, reference the “DevSecOps Best Practices” white paper listed in Appendix A.

8 Risk Management

Identification of risks for Agile programs is like traditional program risk management but Agile programs often remove the request for formal risk management documentation. Removing the formal documentation does not mean risk management practices are not needed. While Agile methodologies lend themselves to daily action and review of team risks, the program manager needs to identify the process and method that will be used for managing program level risks. Risks that require external team participation or executive/enterprise level decision making need to be identified and managed outside the daily scrums. In some situations where external processes dictate additional levels of risk reporting, it is recommended that the program manager take on this responsibility allowing the development team(s) to focus on delivery of capability.

The cost of a risk mitigation within an Agile framework must be equal to or less than the potential impact of the risks if left unmanaged. Therefore, developing a risk management framework for an Agile project that is “too heavy” or resource intensive can distract and become, in Lean terms, wasteful. The Agile program manager must strike a balance in executing a risk management framework that is light enough to add value to the delivery of product and capabilities and not impede or reduce team throughput. The largest inherent risk mitigant in an Agile framework is the frequent delivery of a product or capability.

As shown on the left side in Figure 10\textsuperscript{14} a traditional waterfall development lifecycle builds the entire product with a single delivery at the end, thereby increasing the risk of either delivering the wrong product or not successfully delivering a product. End-user feedback is generally not received until the full solution is developed, and the end-user receives no value potentially for years. On the right side, an Agile development lifecycle provides continuous incremental delivery and provides opportunities for user feedback, adjustments, and realignment within each delivery cycle.\textsuperscript{15} This iterative process provides incremental value in weeks or months, further enabling the fast feedback and learning cycles and thus reducing the risk of missing the target of product outcome.

During Agile program planning ceremonies (e.g. program increment planning, sprint planning) risks that have been identified during the ceremony are presented and discussed as a team. Using agile planning methods such as planning poker or sticky notes, the team can identify the potential impact and/or importance of the risk.

Planning poker can use custom or the Fibonacci scoring. A risk is presented, the team votes based on the probability or impact of the risk to the program. The voting identifies the risk score. Note, if a team member’s individual score for a risk deviates from the rest of the group by two or more places, the scoring and risk can be discussed. Discussion is focused on why the outlier was scored as such. Clarification and discussion by the group should be open and free. Voting is repeated in an effort to reach consensus and once consensus is achieved the risk is assigned a weight.

\textbf{Fibonacci & Planning Poker}

The Fibonacci sequence is numbering 1, 2, 3, 5, 8, 13, 21 and so on, with each number being the sum of the preceding two numbers. Some Agile teams use numbers 1,2,3,5,8 and 13 in their poker planning exercises.

Planning Poker is commonly used for user story estimation of work and is also used for risk impact assessment.

\textsuperscript{14}https://www.maine.gov/oit/project_management/agile.html
\textsuperscript{15}https://www.maine.gov/oit/project_management/agile.html
An additional way to identify program risks is to hand out sticky notes to team members so that they can write down concerns/risks that they have. The team members can then place the risk(s) on a white board for team review. The white board can be sectioned into low, medium and high (figure 11) to allow team members to communicate potential risk impact to the program. Online tools can be used in place of a physical white board and sticky notes for teams that are geographically dispersed. As team members place their risks on the board, they explain their reasoning for the risk.

**Figure 11 Risk Whiteboard Identification/Tracking**

<table>
<thead>
<tr>
<th>Likelihood</th>
<th>Negligible</th>
<th>Minor</th>
<th>Moderate</th>
<th>Significant</th>
<th>Severe</th>
</tr>
</thead>
<tbody>
<tr>
<td>Very Likely</td>
<td>Low Med</td>
<td>Medium</td>
<td>Med Hi</td>
<td>High</td>
<td>High</td>
</tr>
<tr>
<td>Likely</td>
<td>Low</td>
<td>Low Med</td>
<td>Medium</td>
<td>Med Hi</td>
<td>High</td>
</tr>
<tr>
<td>Possible</td>
<td>Low</td>
<td>Low Med</td>
<td>Medium</td>
<td>Med Hi</td>
<td>Med Hi</td>
</tr>
<tr>
<td>Unlikely</td>
<td>Low</td>
<td>Low Med</td>
<td>Low Med</td>
<td>Medium</td>
<td>Med Hi</td>
</tr>
<tr>
<td>Very Unlikely</td>
<td>Low</td>
<td>Low</td>
<td>Low Med</td>
<td>Medium</td>
<td>Medium</td>
</tr>
</tbody>
</table>

**Figure 12 Traditional Risk Matrix**

- **Probability:** the chance that the risk will occur
- **Impact:** the estimate of the potential loss/disruption the risk will have if realized
placement based on their assumption of probability and impact the risk has to the program. The concept is simple and can be adjusted to suit program needs such as having probability and impact estimates identified on the white board (or chart) and sticky note placement takes probability and impact into account. An additional option is to have team members note in the left and right corners of the sticky notes their probability and impact estimate. The more traditional risk matrix shown in figure 12\(^{16}\) can be incorporated and the team can place their sticky notes accordingly on top of it. Using any of these variations and after initial placement on the board, team members are given time to review and adjust placement of the risks. If adjustments are made the team member communicates reasoning behind the adjustment and this continues until the team agrees to all risk placement.

Once product/program risks have been identified they should be tracked and managed at the appropriate level. Risks that require development team action can be identified and assigned as items in the sprint backlog or as actions in a Kanban. Tools can also be used to manage risks that require external team involvement and need program management to monitor and resolve.

## 9 Metrics

Measurements are key to providing the visibility and transparency of information necessary to determine whether a program is moving in the right direction. Programs must have visibility into the work being performed and the results of the work. The increased visibility allows Programs to make decisions backed by data. Therefore, Programs should consider a set of metrics that help provide the necessary information to make necessary decisions.

Programs should collect metrics in several categories:

- **Process Metrics** – Agile process metrics measure process performance, or how well planning, execution, and delivery activities are performing. These flow-based metrics can be analyzed to uncover flow-related issues such as bottlenecks in the value delivery process.
  
  o Sample metrics in this category include story points; velocity; velocity variance; velocity predictability; story completion rate; sprint burndown; release burnup; and cumulative flow

- **Quality Metrics** – Agile quality metrics measure the quality of work being delivered. One of the benefits of Agile, if executed correctly, is that the quality of the product being delivered improves over time due to adherence to Lean-Agile and DevSecOps practices such as ensuring that development, testing, security and operations activities are integrated as much as possible into one delivery value stream. The increased collaboration and “shifting-left” of key quality-related activities such as testing contribute to improved product quality over time.
  
  o Sample metrics in this category include recidivism; first-time pass rate; defect count; test coverage; number of blockers

- **Product Metrics** – Agile product metrics measure the value that the product delivers in terms of user acceptance and alignment to desired outcomes (measured by value). As a result, some of the metrics might not be generated from a tool. Rather, the metrics might reflect a combination of tool-supported data, survey responses, and communication with the business users to determine whether the product is delivering the desired value.

Sample metrics in this category include delivered features or capabilities; delivered value point; and level of user satisfaction

DevSecOps Metrics – DevSecOps metrics relate to measurements that provide insight into the organization’s delivery pipeline. They also include metrics that provide a high-level assessment of the organization’s ability to integrate, deliver, monitor, and restore products. DevSecOps focuses on integration of all activities related to planning, development, testing, security, integration, deployment, and operations. In order to deliver on a regular and continuous basis, the organization must adhere to a set of principles that align with continuous planning, continuous integration, continuous delivery, and continuous monitoring. Organizations that actively measure the efficiencies of their delivery pipeline have greater insight into inefficiencies, and therefore will be in a better position to improve the pipeline for consistent and fast delivery.

Key DevSecOps metrics include mean time to restore (MTTR); deployment frequency; lead time; and change fail rate

Cost Metrics – Cost metrics provide insight into the organization’s planned and actual costs. Planning and measuring cost are just as critical for Agile projects as for traditional projects. An advantage of Agile development is that the frequent deliveries of software provide actual data soon after delivery of capability. This information can and should be used to estimate the costs of future delivery efforts. Near-term software delivery efforts can be estimated by leveraging process, quality, and product metrics as well as product backlog information. Estimates for near-term deliveries will have a greater level of fidelity than estimates for later deliveries. Longer term delivery efforts should rely on higher level approaches and establish a connection between near-term work and longer-term work, which is usually planned in a roadmap of capabilities or features to be delivered over time. Some connection between work scope and effort will be needed across the detailed metrics and overall capability plan. Metrics related to estimating the costs of software acquisition and development as well as any enterprise services and hardware needed. There may be additional program costs outside of these areas that should also be estimated and included in an overall program cost. Additionally, metrics must include costs for both Government and contractor efforts.

Sample costing metrics include Total Cost Estimate; Agile Team Cost; Total Hardware, Software, Cloud, and Licensing Costs; Total Program Management Costs; Allocation of Development Costs; Percentage of Resources by Function; Software Licensing Fees; Computing Costs (including cloud services); Bandwidth Costs; Storage Costs; Other Costs Associated with the Program; and Burn Rate;

Value Assessment Metrics – Related to evaluating the impact of the work.

Sample value assessment metrics include mission effectiveness; cost savings; workload reduction; manpower reduction; equipment footprint reduction; number and significance of field reports; and user satisfaction. (Additional measures and formulas related to cost savings and value delivery are also included in the Agile Metrics Guide published by OUSD(A&S))

Programs should consider the categories as a starting point and identify and tailor the metrics within each category for the unique circumstances of the program. Additionally, programs should identify stakeholders for each metric. The metrics should provide the program with usable information to support decision-making. Given that each metric takes time and resources to frame, collect, observe, analyze, and report,
the program should consider and weigh the cost and benefit of each metric. The key to a successful metrics strategy is to identify the right set of metrics that provide the information necessary to act, while minimizing the collection and analysis burden on the team. Ultimately, the program should seek to automate metrics collection as much as possible to reduce the overhead of data collection and align with Lean-Agile principles of focusing on work that directly produces value for the user.

*For more detail on Agile metrics, reference the “Agile Metrics Guide” white paper listed in Appendix A. The Guide explains each metric’s purpose, benefits, and challenges. It also provides context for each metric and offers some examples to clarify specific points related to visualization of data as it pertains to the metric. Additionally, the Guide introduces the importance of the concept of flow in Lean-Agile and DevSecOps practices.*
10 Sustainment Strategy

In the words of the recent Defense Innovation Board study, “software is never done.” Software components often need to continue to adapt and evolve in response to changing mission needs, cyber threats, and other dynamic drivers across their entire lifecycle. As a result, there is often a continual pressure to add and adapt capabilities, not just maintain function.

It is important to include software sustainment personnel and expertise in upfront planning so that their equities can be addressed from the beginning in a way that supports this need for continuous engineering. Sustainment concerns extend beyond personnel responsibilities and need to be evaluated upfront. E.g. if the government does not plan on taking delivery of source code from the beginning, it can be substantially more expensive to negotiate delivery downstream. Including sustainment equities in upfront discussions also allows the sustainment organization to develop or plan for access to any specialized engineering expertise that will be necessary for sustaining the code.

Based on experiences from the pilots, sustainment planning should consider how to address activities such as the following:

- **Architect and design for modularity and understandability of code.** Almost any software system will be modified continually over its lifetime. Care must be taken upfront to develop an architectural strategy that will allow those changes to be done effectively and efficiently, allowing the software to be designed in such a way that it is well-organized and modular. A suitable architecture makes it more likely that many types of changes to the code can be done as cleanly and in as localized a fashion as possible, reducing the risk of unintended consequences.

- **Plan on a system for identifying and managing technical debt.** A universal feature of software systems is that architecture will degrade over time unless care and effort is spent to maintain it. Software systems tend to accumulate “technical debt,” instances in which short-term benefits like on-time delivery are prioritized over long-term benefits like maintaining a clean architecture. (The term represents that these instances represent “debt” that must be paid back to the system in order to avoid continually paying for them, in the sense of additional effort or increased likelihood of introducing defects for every change.) Programs should plan for some effort being spent periodically on paying down technical debt so that the ability to maintain and adapt the system is kept high. To accomplish this, some of the Agile pilots have begun to incorporate automated scans as part of their development pipeline, using tools that analyze the complexity and modularity of code, to maintain awareness of the quality of the software design over time. The application of proper tools can ensure that the government has insight and understanding of code quality from this perspective.

- **Ensure that the government has access to the source code and all supporting engineering artifacts.** Software cannot be sustained without access to the source code, which means that if the government does not negotiate delivery of or access to source code within appropriate Intellectual Property constraints, sustainment will not be able to be done by any personnel other than the developer’s. It is also important to note that the source code alone will be insufficient because the source code may not be able to be compiled into working software without access to additional source code libraries, specialized development environments and tools, etc. Additional specialized equipment such as test
harnesses or Modeling & Simulation tools may also be necessary to sustain the software cost-effectively. DoD Agile programs have dealt with these constraints in a variety of ways, some by ensuring that delivery of appropriate engineering resources were negotiated upfront as part of the contract, others by working to develop engineering environments and artifacts that would be supplied to contractors as Government Furnished Equipment (GFE), and others by adopting a policy of contracting for software engineering expertise (rather than a set of system requirements) that would be applied to software code which the government owns from the beginning.

The incorporation of Commercial Off-The-Shelf (COTS) software requires some additional considerations: For any COTS products purchased and/or incorporated into a system it is recommended to configure the solution rather than customize. Configuration of a solution will allow the Government to upgrade with COTS enhancements with minimal work. Customization adds time and expense to a COTS solution and impacts flexibility in the future. The benefit of purchasing a COTS solution is to inherent the expertise of an industry standard or capability. To modify the COTS solution to “match existing processes” tends to remove the benefits of the COTS solution. When custom work is being requested, the requirements should be reviewed to ensure that the benefits of customization outweigh the cost, future expense and impacts it will have on the organization. It is also important to understand any custom work contractors and/or Government employees are developing for the COTS solution. The program must have specificity around which version of a COTS product is being purchased / used and what has been done to configure that software in the developers’ environment, and must have access to all customized code (whether new or modified) and configuration items. The program must also consider what licenses or other requirements the government will need to purchase to be able to use that product.

The CTAAP Contracting Guidebook has additional information on resources that should be considered for delivery and how this would be specified in contract language.

- **Account for the sustainment of engineering artifacts as well as the software system itself.** Above we discussed the types of additional engineering artifacts and environments, in addition to just the source code, that may be required for the government to be able to sustain software systems. It is important to note that these elements are in some cases systems of their own and need to be maintained and kept up to date as well. That is, having access to or possession of these resources may require access to expertise and effort that come with associated costs as well. In addition, managing the different versions of these artifacts and their interrelations requires having sufficient expertise and tooling available for Configuration Management.

For more detail on sustainment strategies and how they may relate to the larger acquisition strategy for an Agile program, refer to “Acquisition Strategy Template and Guidance” white paper listed in Appendix A.

## 11 Contracting

This section provides an understanding of how contracting for Agile development solutions differs from traditional waterfall approaches and identifies key focus areas that merit special attention when working with programs to develop software solutions using Agile methodologies. Although Agile can be used for delivery of software, hardware, and other types of solutions, the guidance centers mostly on software delivery.
In order to successfully transform contracting practices to align with Agile practices, Contracting Officers and Specialists should have a basic understanding of Agile software development practices and work closely with the program office and the Agile team throughout the Agile project lifecycle. Additionally, the Program/Project Manager must engage Contracting Officers and Specialists early and often to enable development of contracting strategies that align with the Agile project vision and objectives.

### 11.1 Acquisition Strategy

One of the key differences between an Agile project and traditional waterfall project is that Agile principles align with emergent and adaptive planning and design principles, whereas waterfall projects are focused on predictive planning and upfront design readiness. Agile teams expect to learn through continuous experimentation and continuous delivery. The Agile teams anticipate the need for change or modification of requirements and design as they learn from the fast feedback loops. For these reasons, the acquisition of services over specific product deliverables becomes an important consideration in Agile development efforts. Traditional IT acquisition programs contract to deliver a product capability based on a defined set of “complete” requirements. A services contract is based on the consistent delivery of contractor labor hours vs. a defined product. By using a services-based contract, in an Agile environment, the government can acquire the time and expertise of a contractor team while specifying the exact skillsets needed to perform the work. Service contracts can be used to acquire support for a variety of activities that will enable the Government to implement Agile development. Service contracts enable the Government to contract for development teams by describing desired outcomes while enabling the ability for continuous reprioritization of work without requiring contract modifications.

The procedures in FAR Part 8.4 (Federal Supply Schedules), FAR Part 12 (Commercial Items), and FAR Part 13.5 (Simplified Acquisition Procedures for Commercial Items) can be applied to quickly execute contracts for services and microservices as a program evolves. FAR Part 8.4 and FAR Part 12 contracts are limited to Fixed-Price or Time and Materials (T&M) contract types. FAR Part 13.5 authorizes commercial item purchases up to $7M.

FAR Part 16.5 procedures may be used to award task orders against existing contracts to acquire platform and integration subscriptions, as well microservices and services. Examples include Government-wide Acquisition Contracts (GWACs) (e.g., NASA SEWP V, NIH NITAAC, GSA Alliant), Multi-Agency Contracts (MACs) (e.g., DISA Encore III, DoD IAC), and agency-specific Indefinite Delivery Indefinite Quantity (IDIQ) contracts (e.g., Air Force NETCENTS-2 and SBEAS, Navy NGEN and NGEN-R, and Army CHESS). When considering FAR Part 16 solutions, Program Managers need to carefully consider the Office of Management and Budget (OMB) Spend Under Management Memorandum (SUM) and Best in Class (BIC) solutions.

Another option is to consider is establishing a new multiple-award IDIQ contract (leveraging FAR Part 15) or Blanket Purchase Agreement (BPA) (leveraging FAR Part 8.405-3) at the program, portfolio, enterprise, or Program Executive Office (PEO) level to contract necessary skillsets. A portfolio-, enterprise-, or PEO-
level contract would enable many programs to leverage the contract and the benefits of its streamlined processes, allowing them to focus their strategy and energy on task orders and calls. New enterprise-level IDIQ contracts or BPAs require lead time to establish, but once in place they can offer streamlined procedures for an organization to execute orders or BPA calls as part of a modular contracting strategy.

For Agile development projects that include a prototyping element, “Other Transaction” agreements offer opportunities to structure flexible business arrangements to advance new technologies or to evaluate technical feasibility of new or existing technology in a Government environment. The OSD Other Transactions Guide24 and the Contracting Cone25 provide additional details guiding execution of Other Transaction agreements. The Contracting Cone (https://aaf.dau.edu/contracting-cone/) outlines a variety of FAR and non-FAR contract strategies.

It is recommended that programs determine the competition strategy and whether the approach will include sole source, limited competition, or full and open competition. Depending on the maturity of the requirements, consider whether the program should develop a prototype or model first in a limited capacity prior to embarking on a larger effort. If applying such an approach, consider the Challenge Based Acquisition Approach as one strategic option.

Also, consider whether existing contract vehicles could be used to support the Agile software development process. If existing contract vehicles do not support the effort or have limitations, the Contracting Officer should use other contract vehicles.

11.2 Contracting Strategy

Although there is no single recommended contracting strategy for an Agile software development effort, establishing a flexible, modular contract approach is essential to success. Modular contracting is the preferred approach to acquire software development in accordance with 41 U.S.C. §2308 – Modular Contracting for Information Technology and implemented in FAR Part 39.103. Modular contracting reduces program risk by promoting management of smaller contracts rather than one monolithic contract; incentivizing contractor performance (by structuring contracts so that the Government is not required to procure additional increments); and enabling the Government to continuously innovate and take advantage of the latest technology. Agile best practices recommend contracts with a short (one year or less) initial period of performance (PoP) and options for additional support. Shorter PoPs and options do not commit the Government to years of service by a particular contractor if the contractor’s performance does not meet expectations.

If using a modular construct, multiple service contracts for development may be executed to develop discrete capabilities that fit into the overall solution managed by the Government so that an acquisition of a system may be divided into several smaller acquisition increments that:

1. Are easier to manage individually than would be possible with one comprehensive acquisition
2. Address complex IT objectives incrementally in order to increase the likelihood of achieving workable systems or solutions for attainment of those objectives
3. Provide for delivery, implementation, and testing of workable systems or solutions in discrete increments, each of which comprises a system or solution that does not depend on any subsequent increment in order to perform its principal functions

24 Source: OSD Other Transactions Guide https://aaf.dau.edu/ot-guide/
25 Source: Contracting Cone https://aaf.dau.edu/contracting-cone/ot/
4. Provide an opportunity for subsequent increments to take advantage of any evolution in technology or needs that occur during implementation and use of the earlier increments

5. Reduce risk of potential adverse consequences for the overall project by isolating and avoiding custom-designed system components.

When planning a modular contracting strategy, the collection of modular contracts should be expected to change and evolve throughout the Agile development lifecycle, especially as scaling occurs and more development activities are added. Working closely with the program to understand the operational and programmatic priorities, constraints, and considerations involved in Agile software development is important in order design an appropriate contract strategy.26

Regarding contract pricing structures and type, contractors are not limited by either, as acquisitions using Agile software development are no different than selecting a structure for any other contract. In a fixed-price contract, the line items may be structured by desired capabilities, increments, release cycles, and/or groups of resources/teams. The Government may also use optional line items to account for additional work or resources if needed (See https://techfarhub.cio.gov/handbook/pricing/#structured for more information).

While contracting fundamentals remain the same, some of the greatest challenges that a Contracting Officer or Specialist will face include:

- Maintaining alignment with the cadence of the work being performed by the Agile program, which requires that the Contracting team work closely with the Agile program to ensure that the contracting activities align with the Agile work cadence.
- Supporting the Agile program to deliver the right value by continuously reprioritization and delivery of work. This requires the ability to clearly communicate the needs of the warfighter while avoiding locking down requirements upfront, which will work against the intent of Agile delivery of work. Agile requirements should be developed at a level that explains the vision, intent and high level requirements in terms of capabilities or desired features, but should leave enough room for the Agile team to refine the requirement details throughout the project life cycle and as the team learns more from the fast feedback received as a result of the continuous delivery of capabilities.

If pursuing a service contract to acquire the services of Agile development teams, cost-type contracts are not allowed if using FAR 8.4 (Federal Supply Schedule) or FAR Part 12 (Commercial Items). Instead, Fixed-Price or T&M contracts should be used to acquire services using FAR Part 8.4 or Part 12. Cost-type contracts may be considered if using other contract strategies, such as IDIQ or MAC awards using FAR Part 16.5 or when labor hours and mix are not well understood.

11.3 Intellectual Property Strategy

Per DoD Instruction 5010.44 – Intellectual Property (IP) Acquisition and Licensing, acquiring and licensing IP is vital for maintaining functionality, sustainability, ongoing improvement, and affordability of weapon and information systems. The Government should acquire IP while also considering the rights and fair treatment of IP owners. [1]

Due to the unique nature of every solution, the IP strategy will vary. However, the DoD should specifically consider the following when shaping the IP strategy:

26 Source: AIDA (https://aida.mitre.org/agile/agile-contract-preparation/)
• DoD should only pay for the IP that is needed.
• DoD should only pay once for the same IP.
• Consider differed ordering/contract options to acquire IP at a later date.
• Consider having the bidders propose an IP strategy as part of source selection evaluation.
• Specifically, consider the program’s desired access to code, data, licensing, and knowledge to affordably operate, maintain, modernize, and sustain the system over the life of the solution.
• Consider whether there will be a need for specially negotiated licenses to acquire customized IP deliverables.
• Determine whether it is clear how the solution will be delivered and whether the source code and data will accessible by the Government.
• Determine whether the IP strategy considers what would happen to the data rights/access in the event of early termination of the contract by either party. A lack of clear guidance could put the Government at risk of loss of access to vital data required to maintain the solution moving forward.
• Consider specific tools required to provide authorized access, retention, integration, sharing, transferring, and conversion of IP deliverables.

For more detail on contracting for Agile programs, reference the “Contracting Considerations” white paper listed in Appendix A.

12 Cost and Funding

Estimating costs in an Agile environment requires a more iterative, integrated and collaborative approach than traditional programs. Initial estimates are at a higher level and are refined as the program is further defined. Near term releases have more definition in Agile projects due to the iterative and incremental nature of Agile work. As a result, more traditional approaches to cost estimation can be applied to near term releases, while more analogous estimation techniques must be applied for future releases that are less defined. The program should use the product backlog and roadmap to assess the level of effort for each Agile requirement and gain a better understanding of the amount of effort required for each release.

In order to successfully apply an iterative approach to costing, the cost estimators must be more integrated with the Agile team, including the requirements, design, architecture and development team members. Once the costs for the program have been ‘fixed,’ the cost analysis activities focus on assessing how much work can be accomplished within the funds available.

Two approaches for costing Agile efforts include costing by complexity level and costing by team capacity.

The TechFAR Hub\textsuperscript{27} describes how government agencies have effectively used Agile acquisition in the context of Fixed Price contracts, T&M or Labor-hour contracts, or hybrids to allow the agency to choose among the various pricing structures (e.g. using IDIQ), and shows how the work may be priced out in these scenarios.

In a fixed-price contract, the line items may be structured by incremental deliveries (release cycles) with the unit of measure being the increment or release – each containing multiple sprints. The Government may

\textsuperscript{27} https://techfarhub.cio.gov/handbook/pricing/
also use optional line items to account for additional work if needed. The TechFAR Hub provides examples of how cost estimation and pricing intersect under a fixed-price contract, for example:

- In the case where a new task order or new contract is being put into place, and there is no pre-existing team data that would help determine the likely sprint velocity / capacity, a 100-day baselining period is suggested to provide enough time to collect information that could assist with providing more predictable estimates. At the end of the 100-day period the Government and contractor can mutually agree upon the team’s velocity based on the collected telemetry.
- A second case is one in which a sprint capacity has already been defined and agreed upon by the Government, whether through historical knowledge of a previous effort or through the solicitation requirements established in a competition. The sprint capacity allows the Government or contractor to predict the work that can be completed per sprint given the estimated capacity. The goal of each sprint deliverable is to have working code that has undergone some process of testing and quality assurance. However, each sprint is not expected to be released into a production environment on operationalized. Estimation methods for the sprint can rely on a Small, Medium, Large scale for each capability, feature or story (see complexity-level costing in Section 12.1).

### 12.1 Cost and Pricing Considerations

The size of the Agile team should not change significantly over the course of the work because Agile frameworks encourage fixed resource levels and time while allowing flexibility of requirements. Teams will add an additional development team (scrum) if a new work stream is identified and makes sense.

![Figure 13 - Fixed and Estimated Variables for Waterfall and Agile Projects](image)

The consistent Agile team size allows the team to accurately predict their capacity over time. Agile team size varies between 7 to 12 people depending on the specific Agile framework being used. Work with the Program to understand the number of Agile teams, the size of each team and the team makeup in terms of functional roles. This information can be used to extrapolate the team’s burn rate over the duration of the effort.

Alternatively, the Program can apply a Complexity Level pricing model, which requires that the Program assign a relative size or complexity level (e.g., x-small; small; medium; large; x-large) to the items in the
Backlog based on level of effort (LOE). The Government negotiates an upfront standard price for each complexity level. Future stories defined in the Backlog would be sized by complexity level and priced according to the predetermined price for the associated complexity level.

Based on this information, cost can be applied in one of two ways:

- **Apply cost based on team size and duration of contract**: Applying cost based on fixed resources (based on resources over time, estimated against a set of high-level capabilities to be developed over time). 28
- **Or apply Complexity Level costing**: If applying cost based on LOE, then contractually define complexity level categories, each with a fixed cost (i.e., very low = $1,500, low = $3,000, medium = $7,000, high = $10,000, very high = $15,000) and level set Agile stories to complexity levels. This approach allows the Government to apply LOE costing while maintaining flexibility to prioritize work based on changes in need over time. Allow the teams to define new Agile work, as long as they assign it to a complexity level. The team can determine what to trade out of similar size. 29

### 12.2 Observations from Practice

While the pilots are using somewhat different approaches to estimating costs, and not all have moved beyond traditional cost estimation approaches, there are several that are taking a generally similar approach that is consistent with Agile principles. Key aspects of this approach include:

- **Costs for the (DevSecOps) infrastructure are separated from costs for developing the applications.** The costs for both are built up in very different ways, and costs for the infrastructure are generally less stable than costs for the application development teams. Some of the Agile pilots have explicitly stated that the costs for the infrastructure are a basic cost of doing business and that, if budget cuts were to come, their infrastructure costs would be kept stable and they would only vary the number of teams being used to develop the applications on top of it.

- **Estimation against high-level capabilities only – no build-up from detailed requirements.** Since there are incomplete detailed requirements upfront, and Agile methods emphasize flexible requirements, cost estimates are not built up from a detailed, bottom-up analysis of software size, complexity, or interfaces. “Traditional” cost models such as COCOMO, once an industry norm, are particularly avoided due to their reliance on Software Lines of Code (SLOC). Costs may be estimated for high-level capabilities, usually by comparison based on an assessment of which are similar to each other, larger or smaller.

- **Capacity-based planning** – Rather than build up from a detailed analysis of requirements, Programs estimate the team’s capacity to deliver work using information collected related to delivery lead times – or the time it takes the Agile team to deliver a feature or capability to a user. This assumes that the Agile team has stood up a pipeline that can measure the time from user need to completion of work. Over time, the Agile team can use the collected information to determine their capacity and ultimately predict more accurate time estimates for delivering work to the user. As the demand for work begins to exceed the team’s capacity, the Program can stand up a second Agile team to increase throughput to meet user demand.

---

28 Source: DAU – Capacity Team-Based Contract Approach ([https://media.dau.edu/media/Capacity+Team-Based++Contract+Approach/1_e97qx4ko](https://media.dau.edu/media/Capacity+Team-Based++Contract+Approach/1_e97qx4ko))

29 Source: DAU – Complexity-Level Contracting Approach ([https://media.dau.edu/media/Complexity+Level+Contracting+Approach/1_noifyk52](https://media.dau.edu/media/Complexity+Level+Contracting+Approach/1_noifyk52))
• **Learn-as-you-go** – Working on software is the best way to learn what is really needed, programs understand that estimates will have greater accuracy and fidelity as they collect actuals about how the teams perform. Because Agile development is iterative, programs have many opportunities to collect data about whether the planned commitments for each increment were achieved and develop a more accurate view of the program’s work capacity over time. Also, in an Agile environment, specific capabilities may be merged or take on a higher or lower priority, meaning that teams can be reassigned as the program better understands where the capacity is needed. This re-emphasizes the usefulness of planning based on capacity versus detailed upfront analyses of capability.

### 12.3 Existing Policies and Guidance Related to Costing

Cost estimates for ACAT I programs must be conducted in accordance with the policies and procedures in DoDI 5000.73. The DoDI 5000.73 also provides a timeline for cost analysis activities and activities required to obtain a full funding certification memorandum. The DoDI 5000.73 cites the DoDM 5000.04-M-1, Cost and Software Data Reporting (CSDR) Manual, which services as the primary requirements document for the development, implementation, and operation of the contractor cost data reporting (CCDR) and software resources data reporting (SRDR) systems, collectively referred to as the CSDR system.

### 13 Program Office Staffing and Resourcing Profile

One emphasis shared by Agile and DevSecOps approaches is the need for teams that combine personnel from multiple areas of expertise to work together in tight iterations, rather than having a sequential hand-off of work from one stakeholder to the next. For example, rather than considering testing expertise as something that is applied when the developers are done with their job, teams should have a test lead and sufficient testing expertise to develop test scripts in an ongoing manner. The team composition may vary from program to program, e.g. depending on the specific types of test and accreditation that are required. A program’s Agile approach may also play a role – some of the Agile pilots reported good experiences using Pair Programming to get multiple technical perspectives involved and aligned within sprints. All these decisions (personnel, organization, methodology) will have resourcing implications for the way that Program Offices and program personnel are staffed and organized.

In analyzing the workload and organization needed to execute an Agile program, based on manpower and functional competency requirements, the following factors should be kept in mind:

- **Agile program execution requires expertise from multiple stakeholders being brought together on a regular basis to work through issues as close to real time as possible – not as a series of sequential handoffs.**

There were many success stories on this topic from the Agile pilots. It was often the case that the set of competencies that was important to bring into the team varied from program to program, depending on specific needs and context. For example, in the case of a program that was just getting started, positive results were obtained when the contract specialists were integrated with the program team. These stakeholders were in another office but were involved as part of the project team, and this provided an

---

environment for the contract specialists and acquisition team to understand and learn Agile practices and methodology fast. In another case, an Agile pilot program created a Joint Program Office that combined and co-located acquisition and functional stakeholders so that decisions about functionality could be made quickly and efficiently. Conversely, when stakeholders did not work with the program and remained in separate “silos,” difficulties and extra work were created because those stakeholders were more likely to stay with traditional practices that were not compatible with Agile methodologies.

Integrating other acquisition stakeholders into the team can mean having them use the same work practices as the acquisition team. For example, one Agile pilot found it helpful to get buy-in from other stakeholders (in this case, contract specialists) by assigning user stories to the contract specialists and having them report back on progress in the same forum as other team members.

It can be especially valuable to have stakeholders who traditionally get involved on the back end of the process (e.g. for ATO certification, testing) participate in Agile ceremonies so that requirements can be identified early, testers understand the work performed, etc.

- **Breaking down barriers between the government and contractor teams (as well as across teams from different contractors) contributes to effective execution.**

A contributing factor to Agile success is the ability to have government personnel working closely together with contractors in order to gauge progress, provide real-time decision making, better direct resources, and adapt plans (e.g. re-prioritizing requirements) based on working software. This can represent a break from traditional practice. Making this level of collaboration work requires culture change – the government and contractors are one-team working together to deliver value. Sufficient expertise and decision-making authority on the government side is needed to stay on schedule, assess progress and understand the value of the work being done.

- **Access to Agile coaching is important for programs that are transforming from waterfall to Agile.**

Some of the Agile pilot programs reported a progression from requiring access to a full-time Agile coach when they began their transformation, to an as-needed consultant as their work progressed and the government team became more familiar with what was needed. Engaging with a consultant also suited the programs’ needs in later phases as it turned out that they needed help with more limited topics but typically in more detail.

Agile pilot programs have also reported good results from investments in training to ensure that acquisition personnel in the PMO are familiar with software methodologies and Agile approaches. Multiple programs have accomplished this by sending their personnel to training boot camps, dojos, and similar offerings. Higher-level courses have been useful for acquainting other stakeholder groups (e.g. leadership, contract specialists) with basic Agile tenets and working approaches.
Appendix A – Additional Guidance (White Papers)

The following white papers support and enhance the information presented in this guidebook:

Acquisition Strategy Template

- The Acquisition Strategy document encapsulates the decisions made by a program regarding how the acquisition will be structured and executed in order to achieve the desired capabilities. The document summarizes these decisions for a number of reasons: so that they can be analyzed, reviewed, and vetted by oversight; to communicate to key stakeholders what they can expect regarding the program; and to document the choices made so that other programs can better understand options and best practices. The Agile Software Acquisition document contains key sections and guidance for each specific to an Agile approach to acquiring software-intensive capabilities. It emphasizes key practices that are central to effective Agile approaches, including iterative and incremental development, user engagement, and “shifting left” of multiple stakeholder responsibilities.
- https://www.dau.edu/cop/it/DAU%20Sponsored%20Documents/WP-ASD-v1.2.1.docx

Contracting Considerations for Agile Solutions

- This paper identifies key focus areas that merit special attention when contracting for and developing capabilities using Agile methodologies. Specifically, it offers actionable guidance and sample work statement language that programs could use as a starting point and tailor to meet their unique needs. The guide:
  - Explains key differences between waterfall and Agile projects to increase understanding and context
  - Presents frequently asked questions and responses
  - Suggests considerations for selecting contract strategies
  - Discusses the contractual impacts of an Agile software development approach on the following areas, and provides considerations, guidance, and sample language for each of the following areas: Requirements; Minimum viable product (MVP); Roadmap; Agile framework and methodology; Acquisition lifecycle considerations; Metrics; Roles and responsibilities; Training; DevSecOps; Tools and Technology
  - Contains references to Agile resources such as work statement templates

Agile 101 – An Agile Primer

- This paper provides an overview of Agile values, principles, concepts, vocabulary, terms and roles to provide an understanding of the breadth of Agile and how it differs from traditional project management practices. This document also touches on Agile ties to Lean and DevSecOps practices.
- https://www.dau.edu/cop/it/DAU%20Sponsored%20Documents/Agile%20101%20v1.0.pdf

Agile Roadmap and Minimum Viable Product (MVP) Guide

- This paper provides Pilot teams, Product Owners/Product Managers (PMs), Project/Program Managers and key stakeholders with a better understanding of what a product roadmap and MVP are and strategies for defining each for their Pilots and/or Agile initiatives.
The Agile Metrics Guide

- This paper provides Agile development, testing, operations and program management teams with a set of Agile metrics organized by category: process; quality; product; cost; value; and DevSecOps.
- [https://www.dau.edu/cop/it/DAU%20Sponsored%20Documents/Agile%20Metrics%20v1.1%2020191122.pdf](https://www.dau.edu/cop/it/DAU%20Sponsored%20Documents/Agile%20Metrics%20v1.1%2020191122.pdf)

DevSecOps Best Practices Guide

- This paper provides Pilot teams, Product Owners/Product Managers (PMs), Project/Program Managers and key stakeholders with a better understanding of how to successfully address both the cultural and technical challenges that are presented when implementing DevSecOps. Fundamental concepts include necessary organizational cultural changes; Infrastructure as Code (IaC); Continuous Integration (CI); Continuous Delivery/Deployment (CD); automated testing, and containerization.
- [https://www.dau.edu/cop/it/DAU%20Sponsored%20Documents/DevSecOps_Whitepaper_v1.0.pdf](https://www.dau.edu/cop/it/DAU%20Sponsored%20Documents/DevSecOps_Whitepaper_v1.0.pdf)
Appendix B – Key Agile Terms

Terminology may vary slightly across frameworks. However, important Agile terms noted in this guide include:

- **Users** – Those who will ultimately use the software solution. Users convey operational concepts and requirements/needs, participate in continuous testing activities, and provide feedback on developed capabilities. It is critical for the development team to have a clear understanding who the end-users are, to ensure that the team focuses on meeting and exceeding their needs. A core Agile tenet is active user involvement throughout development.

- **Epic** – A large body of work to be completed during development. Epics are further decomposed into smaller features and user stories. Epics may express business functionality or identify constraints placed on the product or system.

- **User Story** – The smallest unit of requirements written from the users’ perspective on how they will use the software. The Product Owner defines and prioritizes user stories via backlogs. User stories that cannot be completed within a single sprint should be divided into smaller elements. Each user story should have clear acceptance criteria.

- **Story Point** – A unit of measure to determine the size or amount of work a development team needs in order to complete a user story. More complex user stories will require more story points to complete development. Most Agile teams work together with the Product Owner to estimate the story points for each user story in the Product Backlog. Usually, story points are measured by numbers (for example, a story point of 1 is the smallest unit and all other work is assessed relative to that size), but other sizing techniques are also used. Story points are unique to each development team.

- **Definition of Done** – A shared and understood definition of the activities that must be completed for a given user story to be considered complete. The definition of done is used to provide clarity regarding the expected quality of work to meet the users’ needs.

- **Backlog** – A ledger of Agile requirements written mostly in the form of user stories and epics. Types of backlogs include the Product Backlog, Increment or Release Backlog, and Sprint Backlog. Each type of Backlog contains the requirements for the given program, product, increment, release, or sprint, respectively. The Product Owner, working with stakeholders, regularly grooms the backlog to ensure the work is clearly defined, in priority order, and is accompanied by reasonable estimates. The higher priority work is defined in the greatest detail and will be completed first, whereas work planned for future releases will be lower on the backlogs and be described with less fidelity.

- **Sprint** – A short cycle of work (notionally two to four weeks in duration) that focuses on completing a defined subset of project deliverables or usable functionality. Each sprint includes planning, designing, developing, integrating, testing, and demonstrating working software to the Product Owner, users, and other stakeholders.

- **Release** – The core element of the program structure, guiding how frequently the program delivers capabilities to the end users. The length of time between releases depends upon operational, acquisition, and technical factors that the developers should discuss with stakeholders across the user and acquisition organizations. As a general guideline, most releases should take less than six months (as championed by US Chief Information Officer (US CIO), the Government

31 Depending on the specific Agile framework, some backlogs can contain features and themes, which are additional levels of abstraction of user requirements.
Accountability Office (GAO), and the Federal IT Acquisition Reform Act (FITARA)). Shorter release cycles have several benefits, the most important being that the program deploys useful capability to the end-user faster.

- **Velocity** – The measure of the amount of work completed in a given sprint for a given Agile team. Velocity is measured by summing the total number of story points completed by the team. A team’s velocity over multiple sprints is a key metric to track a team’s performance and aids in planning and scheduling future work. Velocity is only applicable to a particular team and cannot be transferred as an estimation tool for another team, because each team measures level of effort, size and complexity of work differently in terms of story points.

- **DevSecOps** – All work related to ensuring the ability to continuously integrate and continuously deliver working code. DevSecOps encapsulates multiple areas (i.e., configuration management, automation, development, testing, security, integration, deployment, and operations). It encourages the concept of “shifting left” – where functional teams work together as early and often as possible – to reduce handoffs and include all functional areas in planning as early as possible.  

- **Test-Driven Development (TDD)** – A practice that involves developing test cases and test scripts before developing the functioning code. The software is then developed and improved upon until it passes the test.

- **Vision Statement** – A short description of the desired end-state of the solution to be developed.

- **Minimum Viable Product (MVP)** – An Agile concept that refers to the first version of a product that provides value, data, and learning for the users.

- **Product Roadmap** – A high-level strategic plan to guide organizational vision and align Product Owner and stakeholder expectations for future development. The roadmap, like the backlog, should be revisited and updated regularly.

- **Kanban** – A framework used by Agile teams to make work visible. The most commonly recognized tool within the Kanban framework is the Kanban Board, which is used to track the flow of work and make work visible by showing work in queue and work in progress. The Kanban Board can also show blocked work and work dependencies.

---

32 See the Agile 101 – Agile Primer paper referenced included in Appendix A and published by OUSD(A&S) for more information on “shifting left”
Appendix C – Case Study: Kobayashi Maru

The guidance and experiences described in this document were compiled to support the “Agile Acquisition Strategy Template” for software acquisition. The template was designed based on general industry and government experience with Agile software engineering, as well as specific experiences from the DoD’s Agile pathfinding programs (stipulated by 2018 NDAA Sections 873 and 874).

To illustrate the types of outcomes expected from use of the template, this section relates experiences with one early adopter of the Agile Acquisition Strategy Template, Space C2 (aka “Kobayashi Maru”). Space C2’s efforts have been instrumental in improving the template for practical use.

Background on the Program. The GAO describes space command and control (C2) as “the ability for military commanders to make timely, strategic decisions; take tactical actions to meet mission goals; and counter threats to U.S. space assets.”

Due to the urgency of the capability gap, Space C2 was allowed to begin without an explicit acquisition strategy being in place. However, the Acquisition Decision Memorandum did specify that an acquisition strategy had to be documented and approved using the Agile Acquisition Strategy template.

Because multiple prior programs aimed at improving C2 capabilities in space all ended significantly over budget and schedule and with key capabilities going undelivered, GAO was tasked with a review of the program in FY18. The acquisition strategy was one area of focus of this review.

Experiences with the Agile Acquisition Strategy Template and Guidance. In their review of the program, GAO examined the Agile Acquisition Strategy template. The GAO concluded that the template covered the key topics found by their prior work to be important contributors to success in new product development efforts.

As a user of the Agile Acquisition Strategy template, Space C2 was atypical in already having a mature strategy when filling out the template. However, this was valuable for the case study in that it allowed the program to test whether all the information they found relevant in preparing for execution were addressed somewhere in the template and associated guidance.

The Space C2 Acquisition Strategy conformed to the guidance contained in this document. The program’s strategy was organized around a program roadmap that guided planned work across multiple product lines. For each product line, the roadmap laid out when work would begin, when an executable system would be available to users, and when operational acceptance was expected. The program budget reflected an approach based around level-of-effort planning, which supported the roadmap.

Space C2’s Agile Acquisition Strategy, as documented, was able to support review and discussion with key decision makers. The Strategy was approved by the USAF Service Acquisition Executive, Dr. Will Roper. The feedback from Dr. Roper was that the strategy was well-written and clearly conveyed the program’s plans.

The Agile Acquisition Strategy template and guidance is focused on strategies and documenting strategic-level information rather than tactical details. The Kobayashi Maru example is a good example for why such a focus is needed – even for a relatively high-level strategy, some of the content became outdated within the time required to staff the document. Agile approaches emphasize learning and adaptation across timely

---

iterations, and as a result they are incompatible with a long and detailed strategy document which would require constant update and revision. While this will be a constant tension in almost any approach to documenting an acquisition strategy, Colonel Jennifer Krolikowski, the PM of Kobayashi Maru, has said that the template “gives me the flexibility to operate as I need to keep things running.”

Operational Impact of Executing the Strategy. As a strategy document, the Agile Acquisition Strategy is useful insofar as it supports effective downstream execution. Space C2 is still in early phases of the program. However, several noteworthy features can already be observed in comparison to the predecessor project, Joint Space Operations Center Mission System (JMS). As a traditional waterfall program, JMS worked to establish requirements upfront and then implement them – as a result, it has been said that the program went for multiple years without having discussions with users. Space C2 in contrast has almost daily collaborations.

Early progress may indicate that the results will be different as well. The GAO notes that when development ended on JMS, the program was “almost 3 years behind schedule and $139 million (42 percent) over budget.” Nine out of 12 planned capabilities were deemed unsuitable for operational use. Space C2, in contrast, is already delivering initial working capabilities. Two applications have been developed and fielded within six months (this timeframe includes operational acceptance testing). Additional applications are waiting only on certification to field, to reach operations.

Early fielding of applications is allowing the value of the operational software to be quantified and reported. For example, one app is saving each individual about 4 hours per shift, through the automation of tasks. There are already about 600 users, with more expected as coalition partners are given accounts.