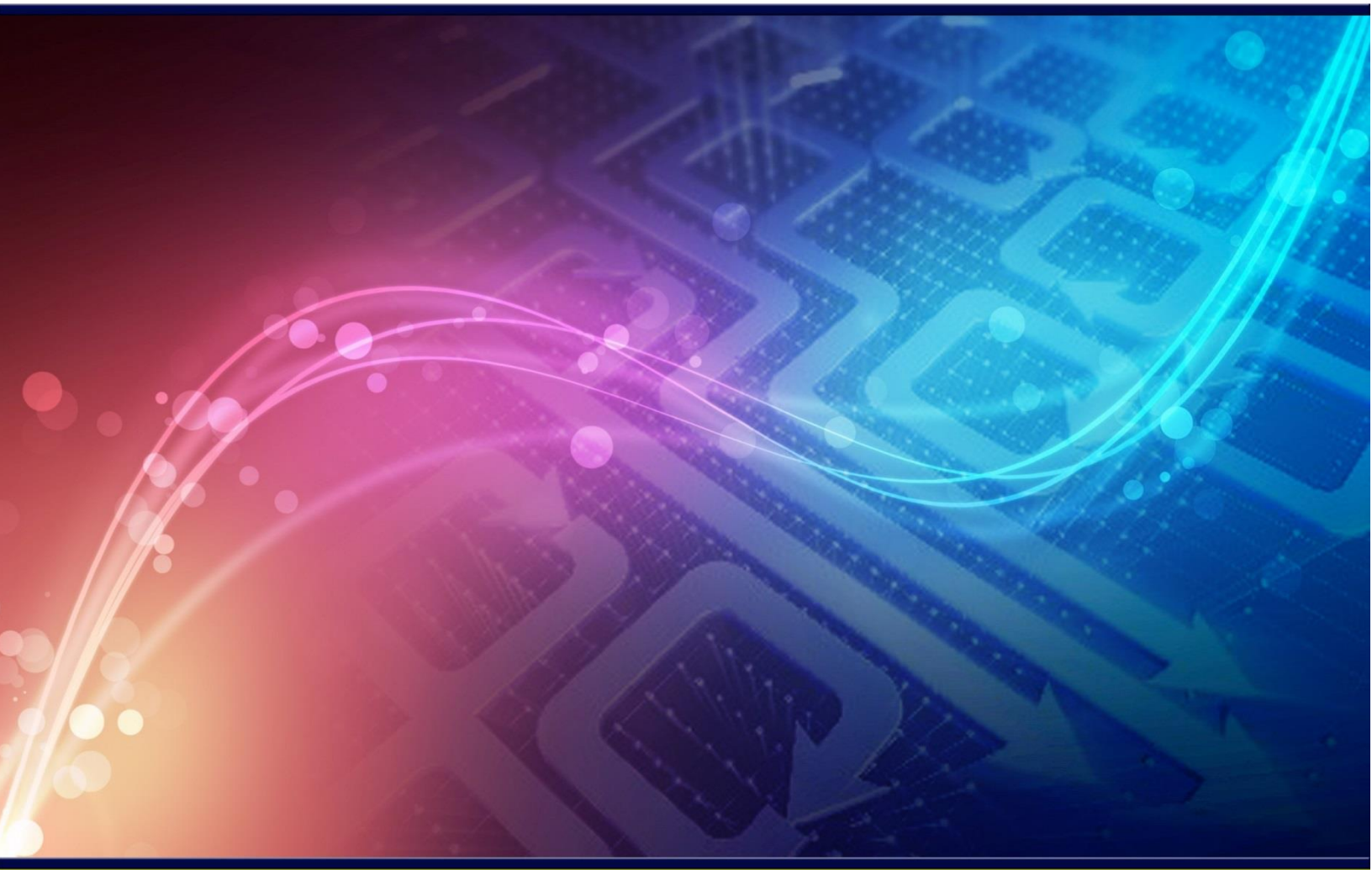
The MITRE Corporation

Defense Agile Acquisition Guide

Tailoring DoD IT Acquisition Program Structures and Processes to Rapidly Deliver Capabilities

March 2014

Pete Modigliani and Su Chang



MITRE

© 2014 The MITRE Corporation. All rights reserved.

Approved for Public Release; Distribution Unlimited. 14-0391

Executive Summary

The Department of Defense (DoD) needs an acquisition framework for information technology (IT) that can keep pace with rapidly changing technologies and operations, including the challenges associated with information assurance. Agile development practices can help the DoD to transform IT acquisition by delivering capabilities faster and responding more effectively to changes in operations, technology, and budgets. This guide provides DoD acquisition professionals with details on how to adopt Agile practices within each element of their programs, thus helping them to succeed in an increasingly complex environment.

Agile has emerged as the leading industry software development methodology, and has seen growing adoption across the DoD and other federal agencies. Agile practices enable the DoD to achieve reforms directed by Congress and DoD Acquisition Executives. DoD Instruction 5000.02 (Dec 2013) heavily emphasizes tailoring program structures and acquisition processes to the program characteristics. Agile development can achieve these objectives through:

- **Focusing on small, frequent capability releases**
- **Valuing working software over comprehensive documentation**
- **Responding rapidly to changes in operations, technology, and budgets**
- **Actively involving users throughout development to ensure high operational value**

Agile practices integrate planning, design, development, and testing into an iterative lifecycle to deliver software at frequent intervals. Developers can demonstrate interim capabilities to users and stakeholders monthly. These frequent iterations effectively measure progress, reduce technical and programmatic risk, and respond to feedback and changes more quickly than traditional methods.

Programs can adopt Agile practices within current policy by tailoring program processes and structure to deliver releases every 6–12 months. The DoD can apply Agile practices to the full range of IT product and service acquisitions. Some practices can even be applied to non-IT acquisitions. Program managers should evaluate the environment, constraints, and objectives to determine the right structure and methods to apply.

Agile requires a set of processes, roles, and culture that will take time to employ. This guide is intended to show how the DoD could tailor the Defense Acquisition Framework to benefit from Agile development best practices. To succeed with an Agile approach, program managers need to work with stakeholders representing the requirements, systems engineering, contracting, cost estimating, and testing communities to design processes around short releases. Acquisition executives must also streamline the decision process by empowering small, dynamic, government-contractor teams. Agile cannot solve all of the DoD's IT acquisition challenges, but offers a set of principles that can help reduce cycle times and risks to deliver IT in a complex environment.

"The US joint force will be smaller and leaner. But its great strength will be that it will be more agile, more flexible, ready to deploy quickly, innovative, and technologically advanced. That is the force for the future." - Secretary Panetta, Defense Security Review, 5 Jan 12

Foreword

Department of Defense (DoD) program managers and executives have struggled for years to tailor the Defense Acquisition Framework to promote delivery of information technology (IT) capabilities in small, frequent releases – the approach that characterizes Agile development. Although broad adoption of Agile methods across the commercial world has spawned countless books, articles, and websites, that literature focuses specifically on the practices and culture of development teams operating within a corporate setting. DoD acquisition professionals increasingly recognize the potential of Agile methods, but don't know *how* to apply Agile within the unique and complex DoD acquisition environment. This guide seeks to adapt proven principles of Agile development specifically to the DoD context.

More and more federal acquisition programs have begun to integrate aspects of Agile development into their strategy. Yet the DoD has not yet accumulated enough experience with Agile approaches to permit rigorous analysis of strategies, methods, and outcomes. Given this lack of well-documented research and of historical examples that other programs could use as models, we sought the views of experts representing diverse acquisition disciplines on how to appropriately and effectively implement Agile practices within current DoD policies. This guide draws on their insights to help program managers better understand Agile fundamentals, how to structure and design a program to enable Agile development, and how to partner with the process owners of various acquisition disciplines to execute Agile processes. It presents options for structuring a program, developing a contract strategy, shaping systems engineering processes, managing requirements, and developing cost estimates for programs with a dynamic scope.

Experience indicates that cultural changes must occur if programs are to implement Agile effectively, and that institutional resistance to these changes can prove especially hard to overcome. However, we believe that with strong leadership, a well-informed program office, and a cohesive and committed government and contractor team, Agile could enable the DoD to deliver IT capabilities faster and more effectively than traditional incremental approaches.

The concepts in this guide will continue to evolve as more DoD programs adopt Agile practices and managers gain additional insight on their successes and failures. We welcome your questions and feedback on the guidebook so that future editions can continue to advance Agile strategies and techniques across DoD. Please contact us at pmodigliani@mitre.org and sjchang@mitre.org.

Pete Modigliani and Su Chang
The MITRE Corporation

Table of Contents

Acknowledgments

This guidebook was created by The MITRE Corporation. We would like to thank the following people who provided valuable subject matter expertise, contributions, and reviews:

- Michelle Casagni
- Tom Fugate
- Ann Chavtur
- Raj Agrawal
- Julia Taylor
- Pete Christensen
- Nancy Markuson
- Dr. Jim Dobbins
- Craig Braunschweiger
- TJ Restivo
- Hugh Goodwin
- Deborah Basilis
- Erin Schultz
- Margaret MacDonald
- Nadine Tronick
- Mike Janiga

I. Introduction

1 Purpose

The DoD needs an acquisition framework for IT that can keep pace with rapidly changing technologies and operations, including the evolving cyber threat. Countless articles and reports have documented the failure of IT programs in the current DoD acquisition environment. While acquisition executives emphasize the need to tailor policies and processes for greater effectiveness, few can do so successfully. This guide provides DoD acquisition professionals with details on how to adopt Agile development practices to improve outcomes in today's complex acquisition environment.

Agile has emerged as the leading industry software development methodology, with growing adoption across the DoD and other federal agencies. Hundreds of books, articles, and websites describe Agile development fundamentals. As illustrated in Figure 1, this document fills the void at the intersection of Agile practices, DoD acquisition policies, and program office operations by digesting the extensive Agile strategies and acquisition processes and provides guidance across all the major acquisition disciplines. Specifically, it aids acquisition professionals within program offices that are exploring a more flexible approach than the traditional defense acquisition framework to apply Agile principles effectively.

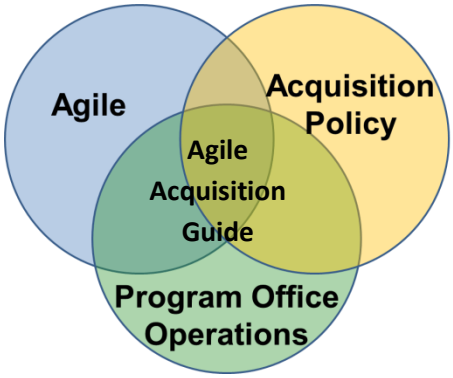


Figure 1 Agile Acquisition Guidebook Venn Diagram

Because there is no single right way to "do Agile," program managers can adopt the Agile practices that best suit their program and environment, New programs can develop an Agile structure and environment from the start, while existing programs can iteratively modify their processes to adopt more Agile practices.

Despite widespread use in industry, Agile is just starting to take root in federal acquisitions. The DoD has already empowered some programs to incorporate many Agile practices, and a growing number of programs continue to chart new paths in this field. This guide is intended to support that growth, as well as to capture the best practices and lessons learned by these early Agile adopters. Broader, more successful Agile execution will take time, trial and error, and shaping of processes, policies, and culture, but with support from oversight and process owners can reach the full potential of Agile development.

This document is structured to provide an overview of Agile development and recommendations for DoD adoption. It examines the major acquisition disciplines to describe potential tailored processes and strategies. The blue boxes presented throughout the guide pose a series of key questions to help acquisition professionals think critically about the Agile processes for a program. The guide also includes dozens of hyperlinks to external sources for further research and information.

2 Agile Development Fundamentals

Agile development emerged in 2001, when 17 leading software developers created the **Agile Manifesto** to design and share better ways to develop software. The values and 12 principles of the Agile Manifesto can be distilled into four core elements:

- **Focusing on small, frequent capability releases**
- **Valuing working software over comprehensive documentation**
- **Responding rapidly to changes in operations, technology, and budgets**
- **Actively involving users throughout development to ensure high operational value**

Agile is built around a culture of small, dynamic, empowered teams actively collaborating with stakeholders throughout product development. Agile development requires team members to follow disciplined processes that require training, guidance, and culture change. While Agile does impose some rigor, the method does not consist of simply following a set of prescribed processes, but is designed to allow dynamic, tailored, and rapidly evolving approaches to support each organization's IT environment.

> ***"You never know less than on the day you begin your new project. Each incremental delivery / review cycle adds knowledge and provides insights that the team could have never known when initial requirements were defined."***
>
> — Steve Elfenbaum , CIO, Schafer Corp.

Each of the many existing Agile methods (e.g., Scrum, Extreme Programming (XP), Kanban, Test Driven Development) has its own unique processes, terms, techniques, and timelines. Because Scrum is the most widely used Agile methodology, this guide uses Scrum terms and processes, but DoD could employ practices from all of these Agile methodologies. This guide addresses the distinct difference between a company employing Agile methods internally and the government-contractor relationship in federal acquisitions. The remainder of this section describes some common Agile terms to provide a lexicon for this guide. Later sections will present more detail on the Agile processes and their application in DoD programs.

The foundational structure of an Agile program is:

Release - Capability delivered to users, composed of multiple sprints
Sprint - Priority capabilities developed, integrated, tested, and demonstrated (aka: iteration)
Daily Scrum - Team synchronization meeting to plan activities and assess progress and impediments

A release comprises a series of sprints. As an example, consider Figure 2, which depicts a six-month release with a series of one-month sprints. During each sprint, a daily scrum meeting takes place.

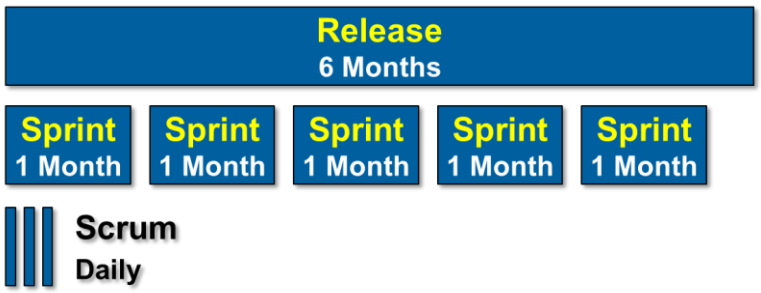


Figure 2 - Basic Agile Structure

Enabling an Agile environment demands some degree of up-front planning and design, but the method emphasizes the importance of beginning development quickly. The fundamental assumption is that requirements, designs, and capabilities will evolve as team members gain information during the development process.

User Story – Description of functionality a user wants, small enough to complete in a single sprint
Epic – A large user story often defined for a release that spans multiple sprints
Theme – A grouping of user stories or epics that may span multiple releases

Teams often capture requirements in user stories and epics to provide a clear operational perspective of the capability's purpose. User stories are simple statements, accompanied by criteria defined for acceptance testing. They are often further decomposed into tasks that define the lower level of detail to be completed. Epics are aggregations of user stories, often used to capture strategic intent. Epics may span multiple sprints and form a release, while user stories should be implemented within a single sprint. Some teams add a third level: themes, which group epics or user stories and span multiple releases.

Teams capture and prioritize themes, epics, and user stories in databases known as backlogs, evolving prioritized queues of requirements identified by the operational and technical stakeholders. A product owner or a scrum master manages the backlogs, updating them based on the results of releases and sprints, changing operational priorities, or technical considerations.

Story Points – Unit of measurement to estimate relative complexity of user stories
Velocity – The amount of work the team estimates it can deliver in a sprint

Development teams, aided by cross-functional representatives, analyze each user story and use measures known as story points to estimate the relative complexity of developing that capability. The development team assigns points to each story following discussion and agreement among the members. Teams may use a Fibonacci series, ideal days, or small-medium-large as units for assigning story points. Over time, as the teams accumulate performance data, this iterative and incremental

process improves accuracy in allocating points. Point values are often unique to the development team and not a valid comparison across teams.

The team uses the size of the stories to determine the team's velocity: the number of story points the team can complete in a sprint. This enables the team to plan the amount of work to accomplish in the next sprint and continually measure its performance. Teams use burn down charts (Figure 3) to track progress during a sprint.

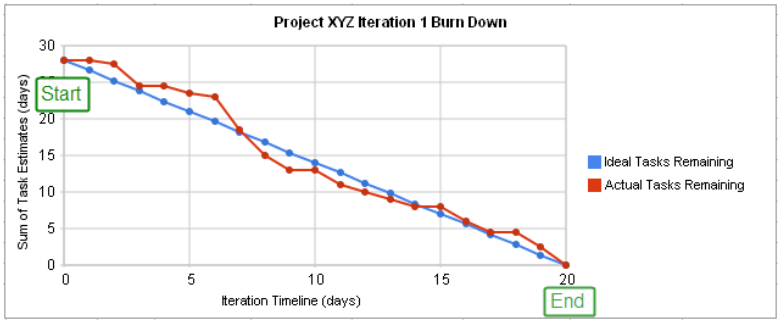


Figure 3: Example Burn Down Chart

Program Backlog – Primary source of all requirements/desired functionality for the program
Release Backlog – Subset of the program backlog listing features intended for the release
Sprint Backlog – Subset of the release backlog listing the user stories to implement in the sprint

Planning occurs continually throughout the development activity. Teams populate the program backlog during an initial planning session, identifying all features the team considers relevant to building the product. The program backlog serves as the primary source for all program requirements and user stories, and the team must prioritize the contents to ensure that the highest priority items are implemented first. The team need not document all the requirements or features up front, as the program backlog will evolve over time.

Subsequent strategic, high-level planning sessions focus on the release and sprint levels. They outline the intent of a release, not a formal commitment. A release backlog typically comprises the highest priority epics and user stories from the program backlog that the team can complete within the established timeframe. The product owner maintains the release backlog with input from the users and development team.

Sprint planning, during which the development team and product owner commit to a specific set of user stories, then addresses the tactical-level details. During sprint planning, the team moves the highest priority user stories from the release backlog to the sprint backlog, estimates effort for the sprint, and often decomposes the stories into tasks. Typically, the scrum master and development team manage the sprint backlog.

The scope of a sprint, unlike that of a release, is locked. During sprint execution, the development team runs through the full development cycle for each user story in the sprint backlog as shown in Figure 4.

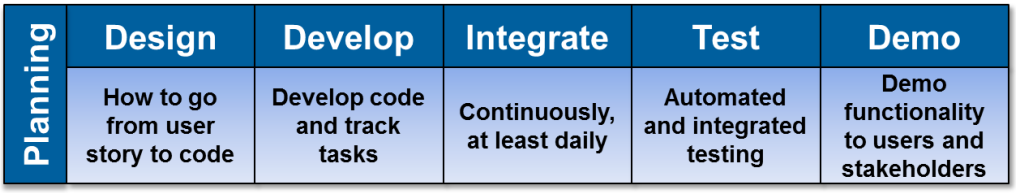


Figure 4 Sprint Execution Cycle

The integrated nature of these lifecycle activities requires frequent communication and collaboration among the team members. This occurs through the daily scrum: a short (e.g., 15-minute) stand-up meeting. Each team member identifies his or her contributions by answering three questions:

1. What did you do yesterday?
2. What are you going to do today?
3. What obstacles are in your way?

At the end of each sprint, the development team demonstrates the functionality to users and other stakeholders and receives feedback. The team then adds user requests for new features to the program backlog, and places tasks that address defects and uncompleted or rejected stories back into the release backlog. Prior to the next planning session, the team revisits the release and program backlogs to reprioritize and adjust the scope accordingly. Upon sprint completion, the development team holds a sprint retrospective to reflect on processes and adjust them as necessary.

Definition of Done – Agreeing on a clear understanding of what it means for a user story or piece of functionality to be complete, or for a product increment to be ready for release.

The Scrum framework includes the concept of "Done," which constitutes a critical aspect of ensuring high-quality software. To be considered potentially releasable, a piece of functionality must adhere to a common definition of completion. During planning sessions, stakeholders and sprint teams agree on the criteria that a user story and release must meet to be considered done. For a user story, the definition may include code completion, the level and types of testing, and (just enough) documentation. For a release, the definition may include more rigorous testing such as regression testing, certification, product owner approval, and release build. The "definition of done" does not change during a sprint, but should be reviewed periodically and updated as processes improve.

"To become Agile, it is not sufficient to just install a handful of new tools, apply some new methods, and rename your milestones to 'iteration exit.' Agile will challenge the way your organization is set up, and it will affect the daily work of each individual."

— Strober and Hansmann

II. Implementing an Agile Approach

3 Deciding to Adopt an Agile Approach

Agile represents a radical shift from industrial age processes to a modern management and development approach suited to the digital age. Agile practices help to make progress and development more transparent, enabling improved decision making by delivering more timely and accurate information. However, Agile is not a panacea: it does not promise to solve all IT and program management problems, and may not be appropriate for use in all cases. Even successful adoption of Agile practices does not guarantee program success, as many variables that affect success lie outside the control of the government program manager and his team.

In the government context, Agile represents a good development approach when customizing an existing system or commercial off-the-shelf (COTS) product, or building a small-scale or self-constrained application. In other words, Agile works well when the program needs to modify software for government purposes and/or integrate it into an existing operational baseline, system, or platform. Although it may not be the easiest approach, the government can also use Agile to build a large IT system from the ground up; however, in this case, it is absolutely critical that the development of the architecture precede sprint development. Alternatively, a program can initially use a traditional approach to build the initial increment that meets the baseline architecture requirements. Once the program has established the baseline and framed the overall conceptual design, program managers can consider shifting to an Agile approach for subsequent increments that build additional functionality into the operational baseline. Several large acquisition programs, such as the Global Combat Support System-Joint (GCSS-J), have adopted Agile methods to build a future increment or block of capability.

Before deciding to adopt Agile practices, program managers should first identify the best approach for the project as a whole and/or for any subprojects within it. This includes assessing the project's volatility (to include requirements and technology), criticality, availability of resources, organizational culture, and availability and commitment of the customer and stakeholders. Specifically, program managers should examine the aspects listed in Table 1 when weighing adoption of Agile or traditional development practices.

Table 1 Traditional Versus Agile Considerations

Consider Agile Practices	Assessment Areas	Consider Traditional Practices
Requirements cannot be well defined upfront due to a dynamic operational environment.	**Requirements Stability**	Requirements have been relatively well defined by the operational sponsor.
Requirements can be decomposed into small tasks to support iterative development.	**Requirements Divisibility**	Requirements are tightly integrated and are difficult to decompose.
Users welcome iterative development	**User Timelines**	Operational environment does not

Consider Agile Practices	Assessment Areas	Consider Traditional Practices
and require frequent capability upgrades (<1 year).		allow iterative development or lacks the ability to absorb frequent updates.
User representatives and end users are able to frequently engage throughout development.	**User Involvement**	Users cannot support frequent interaction with the development team or the target end user cannot be accessed.
Program scope is mostly limited to the application layer while using existing infrastructure.	**Program Scope**	Program spans core capabilities and underlying platform or infrastructure.
The government is responsible for primary systems integration.	**Systems Integration**	The government does not want to own systems integration responsibilities.
Capabilities are operational at a basic level, with some defects that can be addressed in future releases.	**System Criticality**	Program supports a critical mission in which defects may result in loss of life or high security risks.
Industry has relevant domain experience and Agile development expertise.	**Developer Expertise**	Agile development expertise is unavailable or lacks domain experience.
Program office has Agile training, experience, and/or coaches.	**Government Expertise**	Program office has no Agile experience or funding for Agile training or coaches.
Program contract strategy supports short Agile development timelines.	**Contracting Timelines**	Contract strategy cannot support short Agile development timelines.
Program Executive Office (PEO) or subordinate has authority for most program decisions.	**Level of Oversight**	Office of the Secretary of Defense (OSD) or Service Acquisition Executive (SAE) is the Milestone Decision Authority (MDA) and requires most decisions to be made at that level.
Development can be effectively managed by a small cross-functional government team.	**Team Size**	Many government stakeholders will be involved in the software development and decision-making process.
Government and developers can collaborate frequently and effectively.	**Collaboration**	Stakeholders physically located across multiple locations and have limited bandwidth to support frequent collaboration.
One or a few contractor(s) or teams can perform development.	**Complexity**	Many contractors are required to develop program elements.
Program can leverage test infrastructure and automated tests, and testers are active throughout development.	**Test Environment**	Extensive development and operational testing is conducted serially following development. Limited resources and tools available to conduct parallel development testing.
Leadership actively supports Agile development practices and provides "top cover" to use non-traditional processes and methods.	**Leadership Support**	Leadership prefers a traditional development approach or is unfamiliar with Agile practices.

Breaking large requirements into smaller iterations and working in small teams can help build agility into a program, whether or not the program has officially adopted an Agile development approach. Thus, a program can incrementally adopt Agile practices over time to position itself for success when it is ready to commit to full-scale Agile adoption. However, when a program has decided to "go Agile" and formally adopt the methodology as a development approach, the government must commit to making changes across a number of areas. Engaged leadership is needed to support this transition and enable adoption of Agile development processes.

The move to Agile requires time both to learn new practices and replace traditional and entrenched DoD acquisition and development practices. Effective transition calls for some tailoring, because the nature of government contractor relationships, DoD-unique processes and regulations, and dispersion of government decision making authority make it impossible for the government to institute the pure Agile environment that exists in the commercial sector.

Before committing their programs to the transition, program managers must understand and appreciate each stakeholder's risk tolerance and legal responsibilities, and provide clear and compelling evidence that an Agile approach can reduce risk. Application of Agile practices may appear at first glance to encroach upon traditional DoD risk reduction practices, which are optimized for weapon systems acquisition. These traditional methods most often involve extensive analysis, planning, and documentation, as well as large-scale reviews and milestones that ensure readiness to begin development of highly complex and tightly integrated hardware and software. However, Agile inherently serves as a risk mitigation strategy, since early working software products reduce risk by validating requirements and performance characteristics rather than by conducting exhaustive paper analysis. The requirements process prioritizes steps in the development to deliver the highest priority capabilities to the user with each release. Moreover, smaller scale development efforts inherently carry less risk, permitting a lightweight approach to documentation and review that is consistent with the lower risk.

4 Embracing the Agile Culture

Agile practices, processes, and culture often run counter to those in the long-established defense acquisition enterprise. The Agile model represents a change in the way DoD conducts business, and programs must rethink how they are staffed, organized, and managed, as well as whether the business processes, governance reviews, and funding models that support an acquisition are structured to support Agile.

To succeed, the Agile model depends on strong commitments at all levels of the acquisition process. First, Agile requires dedicated government involvement throughout the entire process in order to plan and integrate multiple releases, oversee development cycles, manage evolving requirements, facilitate collaboration, and obtain committed, active, and consistent user engagement. The government must establish a **strong** team to manage and complement

the Agile development contractor. The optimal structure to foster a collaborative environment features physical co-location of the acquisition support team, which consists of the program staff, contractor, and supporting functional areas. A good example can be found in the Air Force Integrated Strategic Planning and Analysis Network (ISPAN) acquisition, an ACAT IAM MAIS program, which has the Government program office, the contractor and end-users all located within a 7-mile radius. This close physical proximity has enabled the ISPAN's development team's adoption of an Agile approach. In cases where close physical proximity of the team is not practical or feasible, programs can use virtual teams with synchronized daily meetings to develop predictable and routine meeting schedules to enhance coordination.

Second, a culture of trust that spans the decision authority, developers, testing organization, acquirers, program management, and users is critical to Agile delivery. This demands a technically competent government team in addition to the development contactors. Close, dedicated acquisition teams facilitate this model, but it must be further reinforced at the top. Leadership can signal that trust by empowering team members with decision-making authority based on clearly communicating a high-level strategy, requirements, and vision for the acquisition.

An analogy to the military term "Commander's Intent" can clarify this concept. Commander's Intent is a concise expression of the purpose of an operation – a description of the desired end state and the way that goal facilitates transition to future operations. It allows adaptation, so that a mission can continue even when the operation does not go as planned. For Agile, the overall plan represents the intent. If the plan does not work as expected, the team alters the plan while continuing to focus on the intent. This requires the team to build relationships that promote trust, collaboration, transparency, and shared responsibility.

The Government Accountability Office (GAO) report on "Effective Practices and Federal Challenges in Applying Agile Methods," recommends four organizational commitment and collaboration practices for Agile implementations:

- Ensure all components involved in projects are committed to the organization's Agile approach
- Identify an Agile champion within senior management
- Ensure all teams include coaches or staff with Agile experience
- Empower small, cross-functional teams

Implementing these practices will help facilitate the cultural changes necessary to make Agile effective.

As noted in the GAO report, several challenges relate to significant differences in how projects are managed in an Agile environment versus a traditional development approach. Previous government attempts with Agile have produced mixed to poor results because the government tried to implement portions of Agile without changing some of the underlying development environments, tools, processes, and culture, which remained oriented toward traditional development strategies. Rather than simply follow a recipe of Agile methods and steps, programs should try to adopt the Agile philosophy and

mindset to instill program agility. Establishing an organizational change discipline and clear vision can help to communicate the organizations strategy of adopting the Agile philosophy.

Lastly, the functional communities supporting acquisition today remain relatively divided and have only weak ties to the program office. Agile practices require a complementary business environment to provide the structure, discipline, and support for Agile development practices. For Agile to succeed, the environment must facilitate close collaboration across multiple disciplines to support rapid development cycles. A successful Agile framework depends on active support from multiple stakeholder communities, including users, developers, systems engineers, and testing and certification staff, as well as DoD leadership and oversight. Program managers must engage these key stakeholders to garner their endorsement and feedback and to build an Agile culture that can be sustained throughout the lifecycle of DoD IT programs.

5 Agile Teams

Agile development requires a new set of roles and responsibilities for a contractor development team and the government program office. Figure 5 shows a potential Agile team construct. At the heart of an Agile development effort is the release team responsible for execution. The release team includes a core team composed of the project manager, product owner, tester, and system engineer, and a development team led by a scrum master (scrum master and product owner are roles specific to the Scrum methodology, and may vary if using other Agile methods). The broader **extended team** includes primary stakeholders and representatives of functional support activities, to include the acquisition leadership, contracting, test, Certification and Accreditation (C&A), the user community, external systems, and cost and financial support.

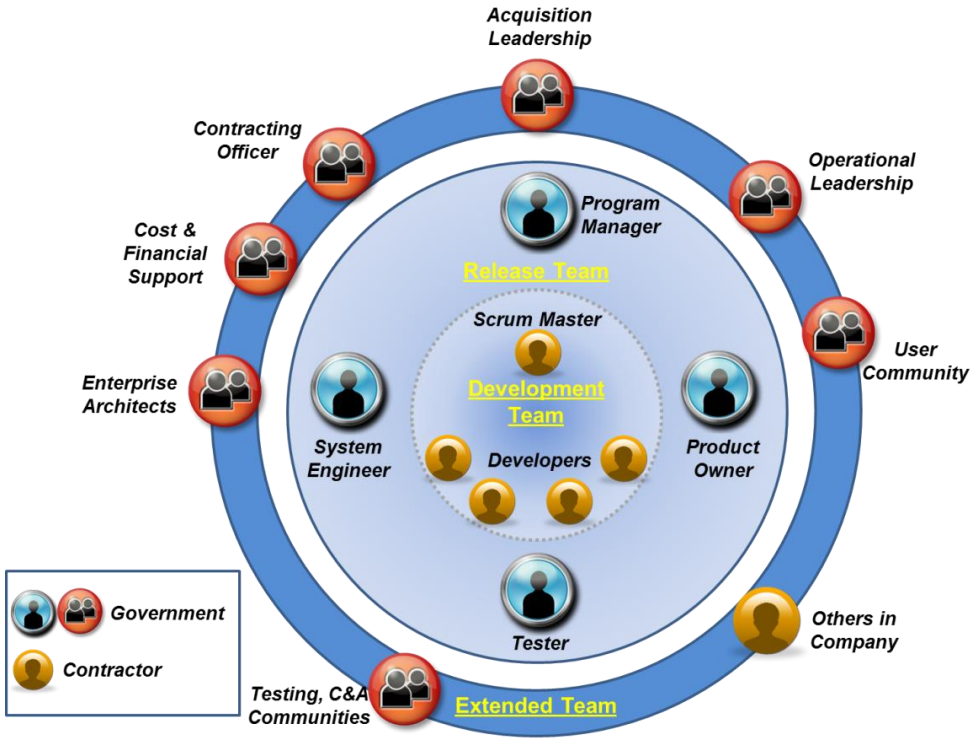


Figure 5: Potential Agile Team Construct

While the roles may vary based on the Agile methodologies and level of integration, four roles recur in almost all Agile activities.

- The **product owner** is the authoritative user community representative who manages the backlog(s) and requirements prioritization, communicates operational concepts to the development team, and provides continual feedback to development team on their developments, demonstrations, storyboards, mockups, etc.
- The **scrum master** facilitates the processes, enforces the team's rules, and keeps the team focused on tasks.
- The **release team** is a self-organizing team composed of <10 government and contractor members who work closely together on the release.
- The **development team** typically is the contractor team of software developers, including software and security engineers, data specialists, testers, quality assurance, and configuration managers.

Ideally these participants are co-located in the same physical space.

While Agile practices depend upon highly skilled and disciplined team members, a cross-functional team of mentors and experts working alongside junior-level team members can also succeed. Program offices realize the best results when the team has at least one staff member with Agile or related expertise who provides on-the-job training to the other staff and is committed to leading teams through successful adoption of Agile practices.

Members of effective program teams understand their roles and responsibilities within an Agile program and have confidence in their ability to perform them. The applicable Agile guidelines for obtaining the necessary talent include:

- Recruit personnel with Agile, Lean, and related expertise to join the program team. Program managers should look across programs in their higher level organization (e.g., PEO level) and should advocate sharing of those critical skill assets.
- Bring in experts in Agile, Lean, and related methodologies to serve as Agile coaches and to conduct on-the-job-training for the program office staff. These experts can make invaluable contributions by guiding the program office to identify and improve roles, processes, and techniques, while addressing any adoption issues. Agile coaches help bring all program participants together under a common set of practices and guidelines.

Agile development often involves multiple teams working on a single program. For example, one team could be assigned a release to develop in parallel with one or more other teams developing other releases. This requires active coordination of efforts across the teams to ensure they are developing towards a common solution. Adding development teams enables more software to be delivered sooner, yet comes with increased risk to coordinate, integrate, and manage these developments and teams. There are various approaches to structure a multi-team environment.

Figure 6 highlights multiple development teams (all contractors) each working on their own release. The scrum masters of each team would meet regularly (e.g. weekly) with government personnel. This approach may limit the government, particularly the product owner's, involvement with the development teams, but enables the development team to focus with limited interruption.

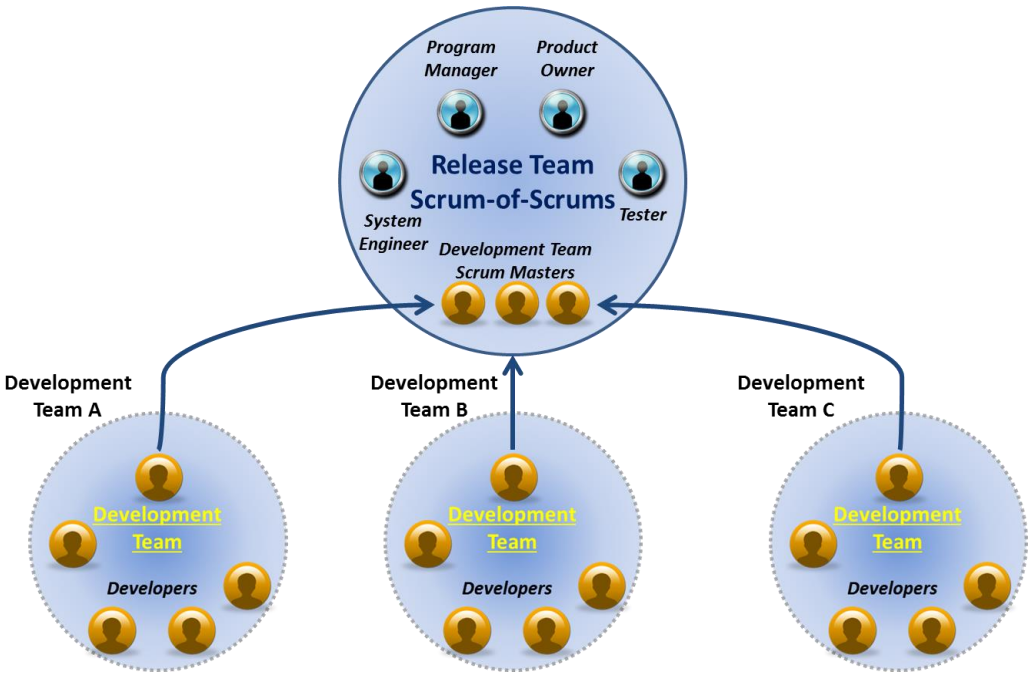


Figure 6 Multiple Development Teams Example

Figure 7 highlights multiple release teams of government and contractor personnel, with government and contractor reps from each team regularly collaborating via a scrum-of-scrums. These reps can be the PM and scrum master. A single product owner could support all the release teams and scrum-of-scrums efforts. Team members can regularly collaborate with those in similar roles on other teams on progress, designs, dependencies, issues, and resources. This approach focuses on an integrated government-contractor release team, but could be more resource intensive.

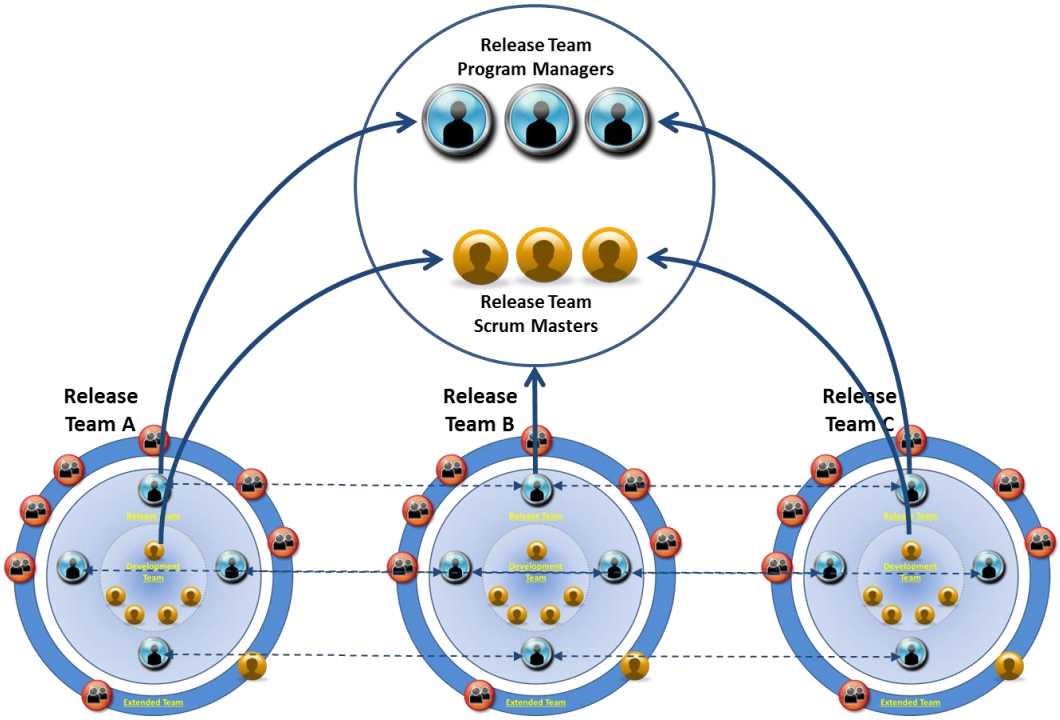


Figure 7 Multiple Release Team Example

Management of Agile development requires a shift from the traditional model of directing projects from the top down to one of trusting and empowering teams to make decisions and to collaborate on solutions. This document refers primarily to program managers, developers, and users in a generic manner; however, programs must operate in a trusting and cooperative manner across the government, contractors, and development teams. This may require the government to refine or redefine existing roles and consider new roles specific to Agile. Table 2 identifies recommended roles and responsibilities for members of an Agile team. Appendix B contains a more complete description of roles and responsibilities.

Table 2 Agile Roles and Responsibilities

Role	Responsibilities
Program Manager	Manages the requirements, funding, and acquisition processes while also overseeing the planning of each release and high-level program increment.
Project Manager*	Manages the development process for a release within a program increment using time-boxed iterations that lead to releases and sprints of capability.
Product Owner	Manages the requirements, tradeoffs, and collaboration between the

	acquisition and user communities.
Scrum Master	Facilitates the developers as they execute their processes, shielding the team from distractions. Enforces the development team rules and keeps the team focused on tasks. Often manages the sprint backlog.
Developers	Develops the software, to include software developers, database administrators and architects, testers, and other technical experts.
End Users	Conveys operational concepts and requirements/needs, participate in continuous testing activities, and provides feedback on developed capabilities.
Enterprise Architect	Creates architectures and designs in an iterative manner to ensure that designs evolve in a controlled way over the course of releases.
Independent Tester(s)	Validates the capabilities produced against the end users' top-priority needs, the design specifications, and standards.
Systems Engineer	Manages releases, overseeing system implementations, O&M, and transition, and integrates all the engineering sub disciplines and specialty groups into a team effort to create a structured development process.
Contracting Officer	Manages the solicitation, award, and execution of Agile development contracts, and facilitates communication between government and contractor staff.
Cost Estimator	Tracks and manages the overall program budgets; provides low-fidelity cost estimates at the program-level for high-level increments, followed by detailed cost estimates prior to the development of each release.

*In some small scale developments the program manager may fulfill the role of the project manager.

Key Questions to Validate Agile Teams/Roles:

- Are the major stakeholder roles and responsibilities clearly defined?
- Is there a clear owner of the requirements backlog?
- Is there a clear government integrator (e.g., program manager and/or systems engineer) who coordinates and integrates programmatics (e.g., schedules, metrics) and deliverables (e.g., code)?
- Is there a clear owner of the program (or broader enterprise) architecture?
- Is there a clear, early commitment from user representatives and the broader user base?
- Are users co-located with, or within close proximity to the program office and/or contractor?
- How frequently do users meet (face-to-face, phone, VTC) with the PMO and developers?
- Is the product owner empowered to speak on behalf of the user community?
- Does the effort actively engage a collaborative, cross-functional community of stakeholders?
- Is the team size small enough to effectively adopt Agile principles?
- Are the number of teams manageable per the size, risk, complexity, and integration required?
- Is there a system-of-systems, scrum-of-scrums, or related cross-team integration role?
- Have approval authorities been delegated to a low enough level to enable rapid decisions?
- Do teams comprise both government representatives and developers?
- Do the team members have sufficient Agile experience, training, and/or coaches?
- Do external stakeholders understand and support their roles in an Agile environment?

See also: Roles in Disciplined Agile Delivery by Scott Ambler

6 Tailoring Program Structure and Processes for Agile Development

The Agile methodology focuses on assembling the right set of experts to plan, develop, and iterate products regularly, rather than on gate reviews and extensive upfront documentation. Programs using an Agile development methodology can tailor the acquisition framework in the DoDI 5000.02 and the Business Capability Lifecycle (BCL) to deliver IT capabilities on a regular basis. In the early stages, programs can still hold to a Materiel Development Decision (MDD), Milestones A and B, and the material solutions analysis and technology development phases. The key is to reach a Milestone B quickly – ideally within 18 months. Programs can accomplish this if PEOs and related portfolio managers can instill common processes, platforms, and documents that individual programs can leverage to meet the DoDI 5000.02 requirements.

Figure 8 provides a potential framework that adapts Models 2, 3, and Hybrid B from the Interim DoDI 5000.02 for an Agile program acquisition.

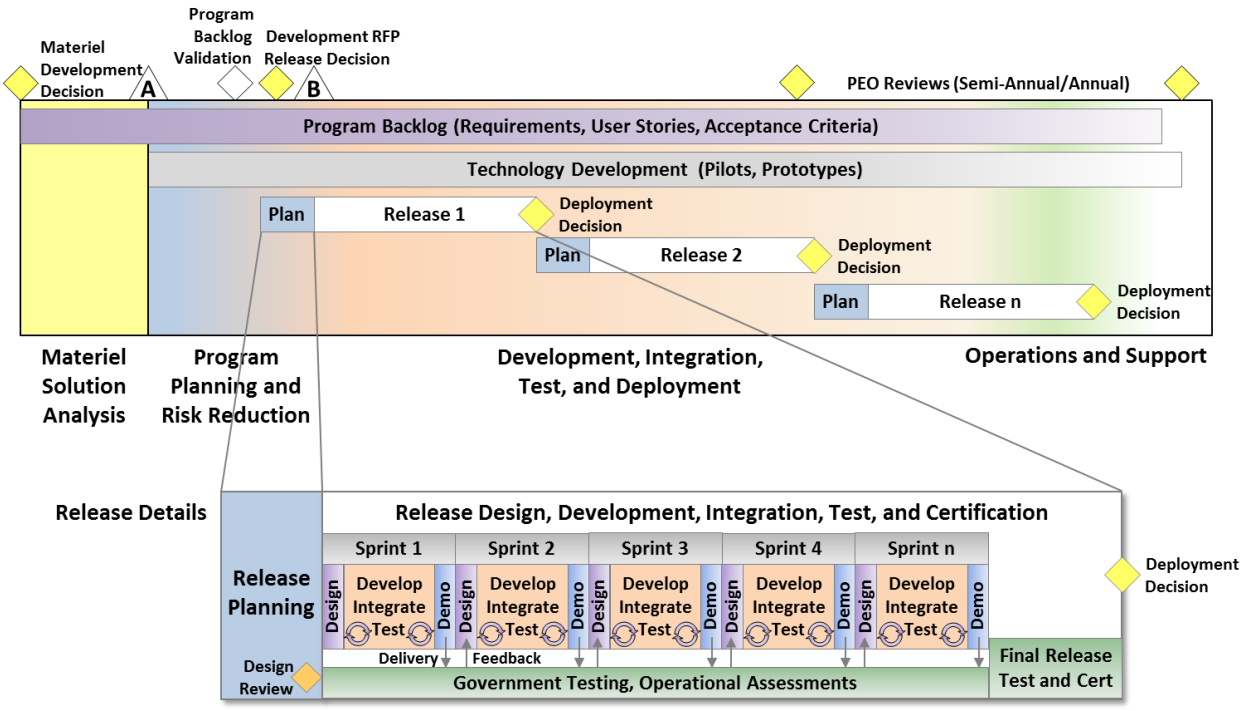


Figure 8: Potential Agile Development Model

The way a program is structured into releases and sprints from the start can play a key role in its success over the lifecycle. Program managers must determine the optimal timelines for releases and sprints on the basis of various constraints, risks, and factors. The primary drivers include how frequently the operational community wants and can integrate new releases, and the ability of the development environment to regularly build, integrate, test, and deploy capabilities. Acquisition, contracting, budget, test, and related processes constrain development, but PEOs should tailor these processes to support smaller, more frequent releases. Active stakeholder engagement and contributions can aid the PEO in designing the acquisition and development processes to fit the planned structure.

While the Agile culture supports changes and continuous process improvement, the program should quickly establish a regular battle rhythm of time-boxed releases and sprints. Releases constitute the foundational structure for deploying useful military capabilities to the operational community. This guide recommends that DoD programs consider six months as the release timeframe. Many private sector companies using Agile techniques deploy capabilities every month, but that cycle is likely too short for the DoD environment. Conversely, some programs may need to structure themselves around 12–18 month releases due to various constraints. While Moore's Law often drives an 18-month schedule, that still represents a significant improvement over five-year increments that end up averaging 81 months for large DoD IT programs.[1] **The key is to dismiss misconceptions about what the acquisition process can allow and challenge the art of the possible.**

Within a sprint (e.g., one month), the development team designs, develops, integrates, and tests the software to enable the user stories selected. A guiding principle is to involve stakeholders early and often to examine features, identify issues, and respond to changes. Engaging testers and certifiers early in the release development process reduces the need for rigorous and lengthy operational test and evaluation and certification processes following development. Ideally, developers deliver the interim capability to the government at the end of each sprint so that testers, certifiers, and users can perform a more thorough evaluation. The testers, certifiers, users, and stakeholders then give feedback to the development team, and the product owner makes official changes to the program backlog. The product owner also grooms the release backlog prior to the next sprint planning session to ensure that successive sprints address identified issues (e.g., operational test, information assurance) and avoid deferring them until the end. At the completion of each sprint, the development team holds a sprint review to identify what did and did not succeed in that sprint so that the team can continually improve its operations going forward.

Programs should base the length of each release and its sprints on operational and programmatic constraints, risks, and objectives. The figures below show potential ways to structure releases and sprints for a government program. Figure 9 shows a six-month release that could include five monthly sprints followed by a month for final testing and certification.

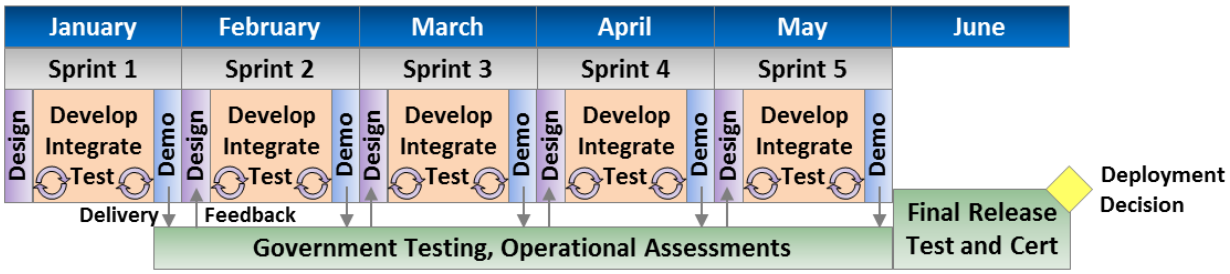


Figure 9: Potential Six-Month Release Structure

[1] Defense Science Board Report on IT Acquisition, Mar 2009

An alternative structure (

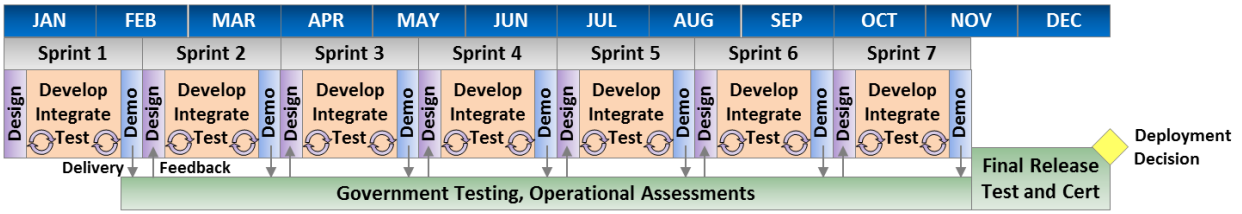


Figure 10) is to construct a 12-month release composed of seven sprints, each taking six weeks, followed by a six-week block for final testing and certification. Programs could also establish 11 month-long sprints in a year.

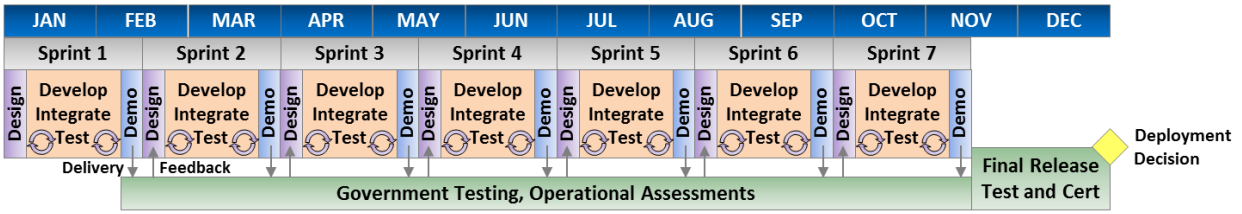


Figure 10: Potential 12-Month Release Structure

Regardless of the structure used, developers must keep in mind that release scope, not dates, is variable, while sprint scope is locked. All development teams should work to the same release and sprint structure to manage scope, schedules, and dependencies. As more programs adopt Agile practices, future programs can leverage an increased knowledge base of best practices and lessons learned.

The government will also need more dynamic, innovative strategies that comply with policies and laws. Within the DoD, the Defense Acquisition Framework (to include BCL) and the Joint Capability Integration and Development System (JCIDS) are two key elements that guide program structure. Both have made progress in enabling more Agile approaches, with JCIDS adopting an IT Box model for IT programs and OSD/AT&L leadership advocating that program managers "design alternative program structures rather than default to the 'school solution'."[2]

Key Questions for Structuring Sprints and Releases:

- Does the team hold planning sessions before each sprint?
- Does each release and sprint have a defined schedule?
- Are sprint and release durations consistent (e.g., all sprints are one month long)?
- Are users actively engaged throughout the design, development, and testing process to provide feedback for each sprint?
- Does the team hold sprint reviews to evaluate and refine processes and tools?

[2] Frank Kendall, Defense AT&L Magazine, "The Optimal Program Structure," August 2012

7 Planning

In an Agile environment, continuous planning occurs at every level – from the program to each release and sprint – and incorporates feedback mechanisms to inform planning for future releases and sprints. The full team and key stakeholders execute the process collaboratively and iteratively. Program teams should work closely with enterprise architects to plan how their efforts fit within the broader enterprise (see section 9.2). As the participants begin to understand program requirements, structure, and solution space, planning should focus strongly on the near term. Programs should review and approve plans at the lowest possible level to support rapid timelines.

While traditional acquisition programs for large weapon systems develop detailed acquisition documents, designs, and strategies that span more than a decade, IT programs have a much shorter and more fluid lifespan. Agile programs can establish some high-level planning, requirements, processes, and structure; however, activities focus on what teams can develop and field in the next few releases. Small, frequent releases enable rapid adaptation of plans, requirements, processes, and capabilities. As noted previously, programs should attempt to begin development within 18 months of the program start.

Agile requires upfront planning to streamline processes to enable rapid and frequent delivery of capabilities. Simply trying to execute traditional processes faster or to cut corners does not equate to an Agile methodology. Program managers must consider different approaches to many of the acquisition processes to truly embrace the speed of Agile development. A portfolio-based approach can allow programs to take advantage of efficiencies gained across several Agile development efforts. For example, competing and negotiating contracts at the portfolio-level can streamline the contracting processes for individual programs and allow them to take advantage of release-level task orders. Additionally, programs should negotiate streamlined testing and certification processes at the portfolio level to drive commonality and repeatable processes. Programs adopting Agile should establish or leverage capstone -level documents as part of their initial program planning.

In Agile programs, planning occurs at both the release and sprint levels. Release planning identifies the capabilities that will be developed over a series of sprints for delivery at the end of the release period (e.g., six months). During release planning, the release team establishes release dates, scope, number of sprints, number of teams, and allocation of user stories to teams. Release planning activities also track risks and manage inter-team dependencies if using a multi-team approach.

Going into release planning, the release team and development team have a program backlog of prioritized user stories. The team goes through the initial exercise of estimating complexity or size of the top features to assist in scoping the release outcome (see release-level estimating section 11.2.2 for further information). The team also defines the criteria for determining that a user story is done. The release planning sessions should result in a full understanding and agreement by all stakeholders and

sprint teams as to what will be delivered, when it will be delivered, and which team (if the project involves a multi-team approach) is responsible for each requirement/user story.

Sprint planning follows release planning. Sprint planning occurs prior to each sprint to confirm understanding and completeness of the user stories, and add detail to user stories by decomposing them into tasks. The development team usually assigns a time estimate to tasks, typically in hours, with a goal of keeping tasks under one day. These refined task estimates, in combination with the team's actual velocity, generate a higher degree of certainty of the team's capacity for delivery. The sprint planning session produces a sprint backlog that defines the set of user stories each sprint team has committed to delivering.

Development teams often have planning sessions periodically throughout the releases and sprints. This is where the team will review the user stories on the program, release, or sprint backlog to ensure a common understanding of what is required, the definition of done, and refinement of the estimated time to complete each story. This along with the product owner's grooming of the backlogs based on operational priorities will ensure the user stories are well understood and properly prioritized.

Key Questions to Validate Planning:

- Is sufficient time allocated to program/release/sprint planning?
- Does the full development team participate in the planning process?
- Are external dependencies identified, discussed, and documented during planning sessions?
- Has the team established a communication plan to coordinate across teams and externally?
- Does the plan align to existing enterprise architectures, frameworks, standards, or interfaces?
- Can the development team access architecture documents and systems?
- Are owners/representatives from these enterprise areas involved in the planning?
- Are assumptions, constraints, and rationale discussed and documented during planning?
- Has the team established standard time boxes for releases and sprints?
- Has the team clearly defined "done" for a release to be fielded (beyond specific content)?
- What methods are used during planning to decompose and translate requirements (e.g., storyboarding, prototyping, user shadowing, interviews, demonstrations, etc.)?
- Does the program have clear, agreed-upon expectations for the depth and breadth of documentation?
- How much planning occurs in early phases and documents vice continually throughout the releases?

III. Agile Acquisition Processes

The following sections recommend strategies for implementing Agile within each acquisition discipline.

8 Requirements

The Agile methodology does not force programs to establish their full scope, requirements, and design at the start, but assumes that these will change over time. Even so, the program must maintain a big picture, long-term view and focus on the next six-month release. All participants must recognize how the program fits within the enterprise architecture from both a business and technical perspective, and have a foundational understanding of the desired operational capabilities and technical infrastructure required.

8.1 Documentation

The government can develop capstone documents at the larger system or portfolio level to capture processes and overarching strategies. As illustrated in Figure 11, programs adopting Agile can establish or leverage such capstone-level documents as part of the initial program planning. Each development effort could either point to those documents or quickly create an appendix to capture details unique to that program. Because Agile embraces continuous process improvement, the government should update capstone documents to reflect any changes to strategies or processes.

The Agile manifesto emphasizes "working software over comprehensive documentation." Plans should be developed by the team, for the team, to provide some level of consistency and rigor, but the level of documentation should not impede the team's ability to focus on capability delivery.

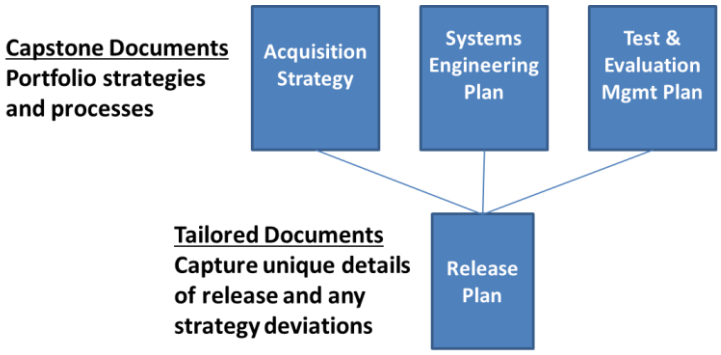


Figure 11 Capstone Documentation Example

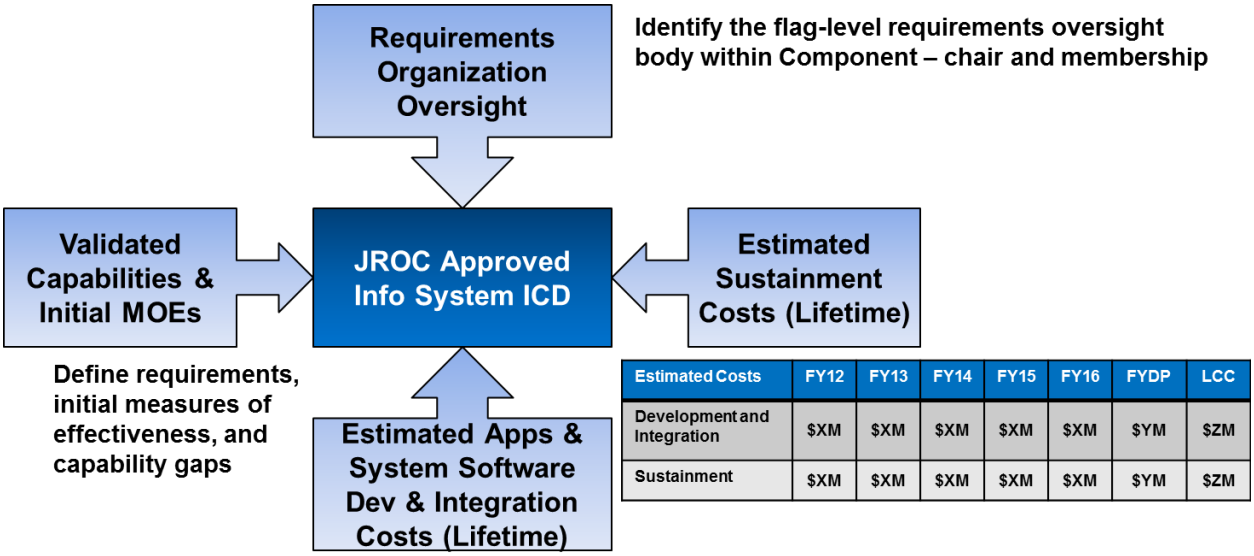
Appendix D lists required DoDI 5000.02 and BCL acquisition documents, and contains a table that identifies the applicability of each acquisition document and how it can be streamlined or combined with other documentation at the portfolio-level. Programs must address requirements for Agile in the

context of both the DoD requirements process under JCIDS and the user story management process within Agile.

8.1.1 IT Box

The Joint Requirements Oversight Council (JROC) published an update to the JCIDS Manual on 19 Jan 12 that includes an IT Box model for Information Systems (IS). The policy applies to programs with software costs over $15M and with COTS or Government off-the-Shelf (GOTS) hardware installation or technology refresh. It does not apply to business systems or IS embedded in weapon systems.

Acquisition programs must have an IS Initial Capabilities Document (ICD) for JROC approval, while the traditional Capability Development Documents (CDDs) and Capability Production Documents (CPDs) are no longer required. As illustrated in Figure 12, the four sides of the IT Box identified in the IS-ICD include:



Estimated Costs	FY12	FY13	FY14	FY15	FY16	FYDP	LCC
Development and Integration	$XM	$XM	$XM	$XM	$XM	$YM	$ZM
Sustainment	$XM	$XM	$XM	$XM	$XM	$YM	$ZM

Figure 12 IT Box (Source JCIDS Manual 19 Jan 12)

As long as the program operates within these four sides of the IT Box, they need not return to the JROC for approval or oversight. In lieu of CDDs and CPDs, programs can develop Requirements Definition Packages (RDPs) to capture a subset of the IS ICD scope and/or Capability Drop (CD) documents for smaller items such as applications (see Figure 13). Services and requirements oversight organizations have the flexibility to identify alternative names for these documents, along with their scope, content, and approval processes. Most important, the requirements documents are designed for a smaller scope of work and approval at a lower level. This flexibility and streamlining of IT requirements enables Agile development within a DoD program. Programs should take advantage of this flexibility and avoid developing a CDD or CPD. Managers can formulate the requirements process for the overarching acquisition using the JCIDS IT Box process to build in flexibility from a high-level operational standpoint. Once an Agile approach has been designed into the program, the process for managing requirements from a functional capability standpoint must also be flexible.

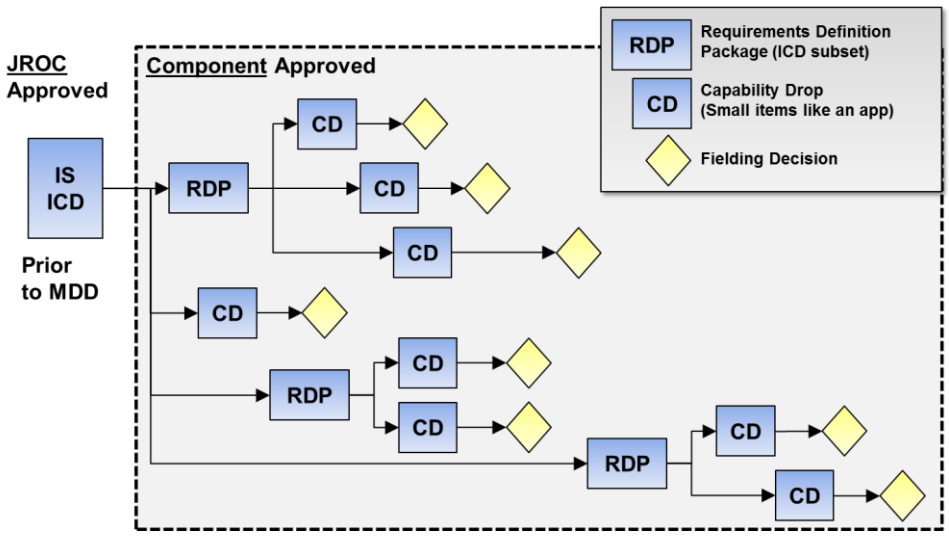


Figure 13: Example of Requirements Documentation (Data Source: JCIDS Manual)

8.1.2 Aligning IT Box Requirements to Agile Development

The IS ICD provides an initial requirements scope for a program, system, or portfolio at the MDD. From the IS ICD, a program develops RDPs that identify the subset of the IS ICD scope that can be chunked into releases. As illustrated in Figure 14, each RDP captures a set of requirements prioritized for a release. A CD captures the release backlog requirements allocated to a sprint and forms the basis for a sprint backlog.

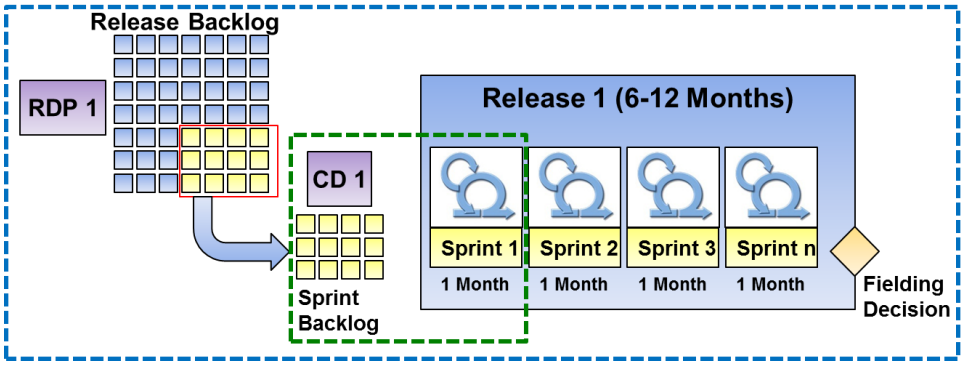


Figure 14: Mapping IT Box Terms to Agile Structure

8.2 Backlogs

As noted in section 2, requirements in an Agile environment are usually managed via program, release, and sprint backlogs rather than through formal requirements documents. Backlogs could take the form of databases, Excel spreadsheets, or Agile-based software tools. The product owner actively manages (grooms) program and release backlogs, working with the user community and other stakeholders to identify the greatest level of detail for the highest priority requirements.

Figure 15 shows the relationships among the program, release, and sprint backlogs. The program backlog contains all desired functionality and requirements. A release backlog typically comprises the highest priority requirements from a program backlog that a team can complete within the established timeframe. A sprint consists of the highest priority requirements from the release backlog. Once the development team commits to the scope of work for a sprint, that scope is locked. Sprint demonstrations conducted by the contractor at the end of a sprint may identify new features or defects that the team would add to the release or program backlogs.

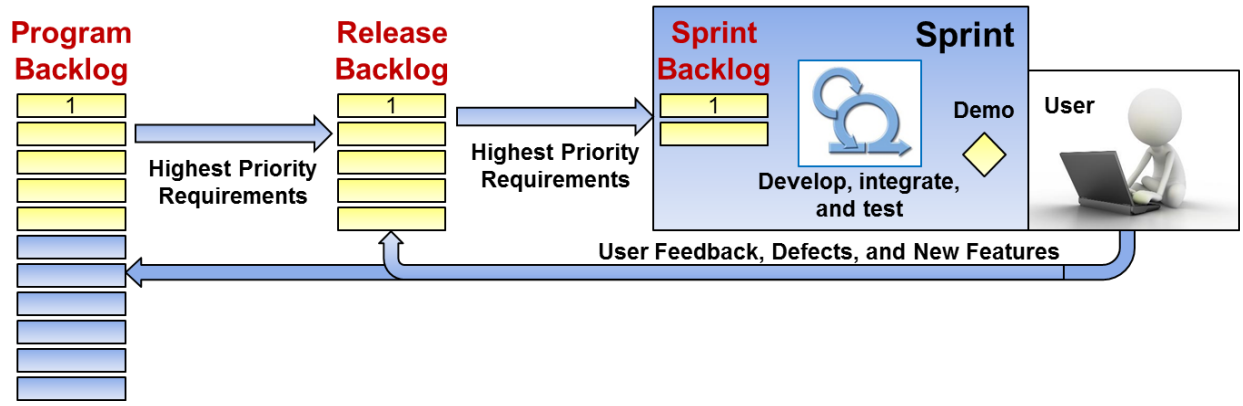


Figure 15 Program, Release, and Sprint Backlogs

The product owner, actively collaborating with users and stakeholders, is responsible for grooming the backlog to ensure the content and priorities remain current as teams receive feedback and learn more from developments and external factors. Users and development teams may add requirements to the program or release backlog or shift requirements between them. The release and development teams advise the product owner on the development impacts of these decisions, while users advise the release team about the operational priorities and impacts. To address a specific user story the dependencies on existing or planned capabilities must be understood. Some programs may use a Change Control Board for some of the larger backlog grooming decisions.

In an Agile environment, users often translate requirements into epics and user stories to concisely define the desired system functions and provide the foundation for Agile estimation and planning. They describe what the users want to accomplish with the resulting system. User stories help ensure that users, acquirers, developers, testers, and other stakeholders have a clear and agreed-upon understanding of the desired functions. They offer a far more dynamic approach to managing requirements than large requirements documents. Development teams periodically review the stories on the backlog to ensure the fidelity of details and estimates. Engineers may also write user stories to cover underlying characteristics of security, technical performance, or quality. Interfaces with other systems are usually captured as user stories.

User stories require little maintenance; they can be written on something as simple as an index card. A common format for a user story is:

"As a [user role], I want to [goal], so I can [reason].

For example, *"As a registered user, I want to log in so I can access subscriber-only content."* User stories should have the following characteristics:

- Concise, written descriptions of a capability valuable to a user
- High-level description of features
- Written in user language, not technical jargon
- Provide information to help teams estimate level of effort
- Worded to enable a testable result
- Traceable to overarching mission threads

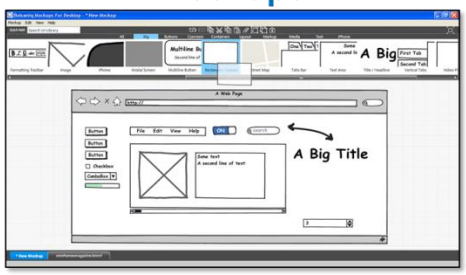
Each user story should be associated with defined acceptance criteria to confirm when the story is completed and working as intended. This requires the stakeholders to have a clear "definition of done" to ensure common expectations. Acceptance criteria consist of a set of pass fail statements that specify the functional requirements. Defining acceptance testing during the planning stages enables developers to test interim capabilities frequently and rework them until they achieve the desired result. This approach also streamlines independent testing following development of a release.

The development team, users, and other stakeholders may use storyboards and mockups to help visualize the system use and features.

Storyboards

Narrative visual depictions set in time to describe system use

Mockups



Visual depictions of the feature of the system

Teams update the backlogs based on what the users and developers learn from demonstrated and fielded capabilities. The new items may include making fixes to software or generating new ideas. As operations change, teams may propose adding or changing requirements and user stories, both in content and priority. For example, the team may add integration items to the backlog as the program interfaces with other systems. Systems engineers and enterprise architects may add items that support the release integrity or technical underpinnings of capability delivery and information assurance. Ideally, teams should address issues discovered by government and contractor testers within the next a sprint or release, but they may add those issues to a backlog based on the scope of the fix.

8.3 Active User Involvement Throughout the Development Process

A close partnership between users and materiel developers is critical to the success of defense acquisition programs and is a key tenet of Agile. Users must be actively involved throughout the

development process to ensure a mutual understanding across the acquisition and user communities. While users will often have operational responsibilities of their day job, the more actively they engage during development, the better chances for success. There needs to be a commitment from operational commanders to allocate time for users to engage. Users share the vision and details of the concepts of operations (CONOPS) and the desired effects of the intended capabilities. Through ongoing discussions, the program office and developers gain a better understanding of the operational environment, identify alternatives, and explore solutions. Users then describe and validate the requirements, user stories, and acceptance criteria. The program office must make certain that the requirements can be put on contract and are affordable based on funding, schedule, and technological constraints. Testers should take an active part in these discussions as well to ensure common expectations and tests of performance. In situations where primary users are not available to engage with the Agile team on a regular, ongoing basis, the end users can designate representatives to speak on behalf of the primary users.

8.3.1 User Forums

User forums enhance collaboration and ensure that all stakeholders understand and agree on the priorities and objectives of the program. They can serve as a valuable mechanism for gathering the full community of stakeholders and fostering collaboration. They give users an opportunity to familiarize developers with their operational requirements and CONOPS and to communicate their expectations for how the system would support their needs. Continuous engagement of users, developers, acquirers, testers, and the many other stakeholders at these forums also enables responsive updates and a consistent understanding of the program definition.

Suggestions for successful user forums include:

- Hold regularly scheduled user forums and fund travel by stakeholders across the user community; alternatively, or in addition, provide for virtual participation.
- Arrange for developers to demonstrate existing capabilities, prototypes, and emerging technologies. These demonstrations give users invaluable insight into the art of the possible and the capabilities currently available. User feedback, in turn, guides developers and acquirers in shaping the program and R&D investments.
- Allow the full community to contribute to the program's future by holding discussions on the strategic vision, program status, issues, and industry trends. Program managers should not rely on one-way presentations.
- Give stakeholders the opportunity to convey expectations and obtain informed feedback.
- Establish working groups that meet regularly to tackle user-generated actions.
- Hold training sessions and provide educational opportunities for stakeholders.

Key Questions to Validate Requirements Management:

- What requirements documents does the program have?
- Who is responsible for managing the program backlog?
- What is the process for adding, editing, and prioritizing elements on the backlog?
- How are requirements translated into user stories, tasks, acceptance criteria, and test cases?
- Are there clear "definitions of done" that the team and stakeholders agree upon?
- Are developers involved in scoping the next sprint (or equivalent)?
- Does the backlog include technical requirements from systems engineers, developers, and testers along with the user requirements?
- What requirements oversight board exists and what does it review/approve?
- How frequently are the backlogs groomed?
- Is there an established change control process for the requirements backlog?
- Are architecture or system-of-systems requirements managed on the backlog?
- Are requirements/user stories/backlogs maintained in a central repository accessible by a broad user base, and is content shared at user forums?
- Does the product owner regularly communicate with the development team about the CONOPS?
- Are testers actively involved in the requirements process to ensure requirements are testable?
- Is there a mapping among requirements, user stories, tasks, and related items?
- Is there a mapping between release/sprint requirements and higher level program/enterprise requirements?
- Does the program have a dynamic process to regularly update, refine, and reprioritize requirements?
- Does each release and sprint have a defined schedule?
- Does each requirement/user story planned for the release/sprint have associated cost and acceptance/performance criteria?
- Are users actively engaged throughout the design, development, and testing process to provide feedback for each sprint?
- Is integration with lifecycle support processes and tools iteratively refined and evaluated as part of each sprint?

See also:

- Agile Requirements Best Practices by Scott Ambler
- Requirements by Collaboration by Ellen Gottesdiener (See also the book by same name)
- Agile Requirements Modeling by Scott Ambler
- Agile Software Requirements: Lean Requirements Practices for Teams, Programs, and the Enterprise by Dean Leffingwell
- Writing Effective Use Cases by Alistair Cockburn
- Articles on Agile Requirements
- Articles on User Stories

9 Systems Engineering

9.1 Overview

"Systems engineering establishes the technical framework for delivering materiel capabilities to the warfighter."

–Defense Acquisition Guidebook

Systems engineering ensures the effective development and delivery of capabilities by using a set of integrated, disciplined, and consistent processes throughout the program lifecycle. In an Agile environment, systems engineering requires tailored methods and processes to deliver incremental capabilities, and therefore demands a disciplined approach to coordinating parallel development, operations, and sustainment activities. Systems engineers play an essential role in technical and programmatic integration, as expressed in the core Agile tenet of active collaboration among developers, users, and other stakeholders. Program leaders must encourage systems engineers to engage developers, testers, users, and other stakeholders in their disciplined engineering processes.

To enable faster, smaller capability deliveries, Agile development requires tight integration among enterprise architectures, platform architectures, and related development efforts. To find the right balance between structure and the flexibility necessary to deliver usable capability aligned with user needs, programs should conduct continuous systems engineering reviews in accordance with DoDI 5000.02 requirements. However, as illustrated in Figure 16 programs should replace comprehensive Preliminary Design Reviews (PDRs) and Critical Design Reviews (CDRs) with more frequent and incremental design reviews during the release planning phases. To demonstrate functionality and provide insight into the program's progress, these reviews should focus on the relatively small scope of a release and how it aligns to the enterprise architecture. Similar technical reviews can be decomposed to the release level.

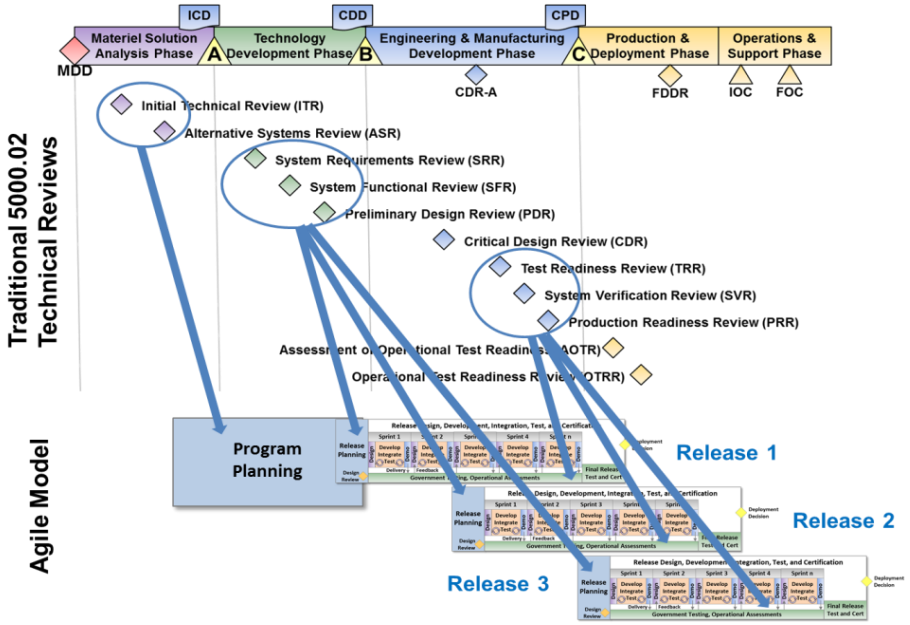


Figure 16: Transitioning Large Formal Technical Reviews to Smaller, More Frequent, Iterative Reviews

While Agile systems engineering involves frequent informal technical and programmatic reviews, this less formal approach does not equate to less rigor. Instead, greater frequency allows key decision makers and other stakeholders to become more familiar and comfortable with processes in the Agile environment, which enables a more collaborative and productive review process. As emphasized in the sections above, full participation by key decision makers and users is fundamental to the Agile approach.

Some key systems engineering practices to consider in an Agile environment include:

- Provide information to all key stakeholders on a consistent, regularly scheduled basis, either through design reviews or program reviews.
- Use the release planning and sprint demonstrations as opportunities to bring users, developers, and stakeholders together in face-to-face sessions to drive collaboration and strengthen teaming arrangements.
- Ensure that once a clear architecture is in place, systems engineers continue to refine it as they learn more from the development sprints and releases.
- Independent of releases, hold periodic technical reviews for larger epics, strategic planning, grooming the program backlog, managing resources, and shaping the program's direction.
- Capture systems engineering processes and related content in portfolio-level systems engineering plans for broad use. Individual releases can capture the scope details in small appendices approved at a low level.

9.2 Enterprise Architecture

The Enterprise Architecture is the explicit description of the current and desired relationships among business and management process and IT. It describes the "target" situation which the agency wishes to create and maintain by managing its IT portfolio.

- Franklin D. Raines, Former OMB Director

In today's environment, IT programs can no longer operate independently; instead, they must interface and communicate with many other systems across the DoD. As the DoD continues to evolve to a common operating environment with enterprise platforms, IT programs will deliver capabilities via a series of applications or web services. IT systems must therefore be designed, developed, and maintained in concert with enterprise architectures. **The enterprise architecture is central to orchestrating technical integration in today's environment and envisioning how a system should operate in the future.** The architecture should highlight both the technical and operational processes and structure across the enterprise. The architecture should take both existing and planned infrastructure and systems into account – from the capabilities they currently provide, to the gaps that must be filled and the required interfaces to other systems. The architecture can also promote the use of open source software and reuse of existing code or applications.

Throughout the IT acquisition lifecycle, the enterprise architects serve as essential members of the program team, helping to effectively scope, structure, and analyze alternatives for delivering capabilities to the warfighter. They ensure the architecture captures the operational and technology environments as well as current processes, components, interfaces, and other features. **An effective architecture is one where the big picture is clearly understood and stakeholders understand how they connect to the enterprise.** As the entire program team learns how the required capabilities will achieve military objectives, the team members must also understand how the program will align with existing and planned efforts across the enterprise, and must provide the architects with updates to the program so they can continually manage and update the architecture. Enterprise architects must ensure that the program effectively digests and integrates additions to the architecture and archives expired elements. Frequent collaboration is critical to success.

When the program develops new software – for instance a new application, or customization of an existing COTS product – it must establish an initial architecture to guide the design and integration aspects of the project. The architecture will help to identify how to prioritize the requirements to deliver initial functionality. The initial design of the architecture and platform infrastructure is usually created separately from the software development portion of the program that employs Agile. Under a Scrum methodology, this upfront architecture and infrastructure design and build are referred to as "Sprint 0."

Enterprise architectures capture vast amounts of information; however, the key guideline in developing and maintaining architectures in an Agile environment is to keep it simple. The value of the architecture decreases as complexity increases, with volumes of documents and artifacts making it difficult for participants to comprehend.

Interfaces with external systems and changes driven by the enterprise architecture are often managed as stories on the program backlog. Many interfaces must be broken up into many stories, but be managed as a theme or epic. Experts in Agile methods recommend use of common (ideally open sourced) platforms, standards, interfaces, and application program interfaces (APIs) over costly point-to-point interfaces.

See also:

- Agile Enterprise Architecture

9.3 Continuous Prototyping

"I have not failed. I have successfully discovered 10,000 ways to NOT make a light bulb." – Thomas Edison

Prototyping plays an essential role in developing tangible, deployable, valuable functionality, but also has other uses. Real innovation always includes a risk of failure, and continuous prototyping gives program managers and users a powerful resource to reduce risk, quickly integrate new technologies, and identify innovative solutions that leverage existing programs, GOTS,

and COTS products. For example, the objective of creating a prototype might be to assess whether a COTS product can perform a particular function. In other cases, a prototype can evolve into the final product or be used in a future release.

Prototypes can enable the government to quickly eliminate ineffective approaches and focus on effective ones. They also provide a way to demonstrate functionality to help solidify requirements for the final design. An ongoing technology demonstration capability creates an environment for prototyping several different implementations of a design feature to benchmark the resulting performance and analyze the tradeoffs of each approach. This can reduce risk, drive better design decisions, and save time during the development process.

The nature of Agile development involves continuously refining capabilities by adding to previous work or refactoring (cleaning up code) when necessary to meet these changing priorities. Teams can reprioritize requirements on the basis of the information learned during prototyping activities.

Some prototyping practices to consider in an Agile environment are:

- Institute rapid prototyping that allows developers and government R&D organizations to quickly demonstrate potential solutions that meet urgent requirements, mature and integrate technologies for the particular solution space, and highlight advantages over alternative options.
- Establish an environment that includes processes and properly configured DoD test platforms to rapidly prototype capabilities that extend beyond the next increment or project. This opens up opportunities for innovative and dynamic solutions. This type of continuous prototyping environment can be supported by a portfolio-level multiple award Indefinite Delivery Indefinite Quantity (IDIQ) contract and can also provide opportunities for innovative small businesses to enter the DoD market.

9.4 Risk Management

Risk management is integral to effective program management and systems engineering in Agile and traditional programs. Programs adopting Agile practices will generally manage risk like traditional DoD programs, but will face different levels and sources of risk. They will still require a rigorous process to regularly identify, assess, mitigate, and track risks. Risks will need to be actively managed with mitigation strategies integrated into acquisition strategies and key program processes throughout the program lifecycle. Agile enables some unique risk management aspects and mitigations.

Within an Agile environment, managing risk is an integral part of release/sprint planning and development. The Agile process itself has built many features into the development process to manage risk; for example, decomposing development into small releases and sprints usually reduces both program cost and schedule risks. Estimates for smaller efforts (e.g., six-month releases) tend to have a higher fidelity than those for entire programs or increments (e.g., 5–10 years), and over time the estimates improve as sprints and releases provide valuable data. Frequent software deliveries of priority user capabilities and responsiveness to changes often mitigate the risk of user dissatisfaction. Using mature technology and designs in shorter releases and sprints helps to manage technical risks. Program

failures in releases and sprints are smaller and therefore have less overall impact; they also provide valuable lessons for future developments.

Agile development reduces overall program risk because the program regularly delivers some degree of useful capability in each release. Thus, even if a program's budget is cut or eliminated along the way, deployed releases provide users with some level of fielded capability. Furthermore, Agile development focuses on users' highest priority capabilities first; as a result, users need not wait five to ten years to receive critical capabilities. Capability demonstrations to users at the end of each sprint reduce the risk that the final product will fail to meet user's expectations, since users can provide ongoing feedback on deployed capabilities to inform and shape future releases. Lastly, short development timelines provide a steady pipeline to infuse mature technologies effectively integrated into the system and enterprise.

An Agile environment does not reduce all risks, as coordinating and integrating many smaller developments involves increased complexity. Similarly as more development teams are involved, possibly including multiple contractors, integration risks increase. Success depends on an effective program manager and chief engineer, working with enterprise architects and stakeholders, who can effectively design and implement solutions.

Programs also run risks when transitioning and adapting to an Agile culture. During the early adoption period it is especially critical to have personnel with Agile experience in the program office and contractor teams. Agile coaches can also provide guidance to assist programs in tailoring and executing Agile roles, processes, and environments.

A key development risk can occur if requirements are continuously deferred to future sprints or releases. Often the developers find that they cannot meet all of the requirements in a sprint, thus requirements continue to shift to the right creating a bow wave effect. One way to manage this requirements shift is to make the 4^{th} or 5^{th} sprint in each release a "catch up" sprint with no new requirements in it. This keeps the program's schedule and cost on track.

A final key risk to consider is the User community's ability to handle the planned releases. When defining the release schedule for a program, one must consider the users' ability to integrate releases in their operational environment. Part of what will dictate this is how much change is visible to the user community and how much they must change their way of doing business. Training and documentation are often highlighted as constraints, but there are strategies for smoother integration. There may be some established periods where the operational community cannot integrate new releases.

9.5 Configuration Management (CM)

The many smaller developments and flexible requirements process that characterize Agile programs make CM an important and challenging practice in Agile environments. CM starts with mapping and managing requirements via backlogs, stories, epics, and themes to the sprints, releases, and acceptance criteria. As backlogs are regularly groomed, the product owner must ensure traceability of changes and from requirements to user stories to delivered capabilities. Regular code check-ins, integration, and builds can occur on a daily basis. Frequent code updates and automated testing create a high demand

for managing the complexity of version control. If issues arise, the team can revert to earlier versions and identify the steps that must be re-accomplished. Leveraging common standards and interfaces enables the development teams to work more independently to reduce integration risks.

Many Agile development and engineering tools have CM functionality. These tools are invaluable for integration across multiple development teams on the program and with external systems. As with CM on traditional programs, there needs to be rigor on managing changes to baselines, designs, engineering, and program documentation. Some additional CM methods for Agile include:

- Assign the daily management of configuration items to the Agile team, while a Configuration Manager focuses on program/enterprise-level activities such as creating a CM strategy and tools.
- Use automated tools to enable continuous integration, reduce delays in integration and testing, and provide CM functionality.
- Continuously observe and adapt the CM tools and process. For example, avoid complex branching or baselining mechanisms if the team does not need them, or observe where long build times slow down the team and focus on improving these procedures.

9.6 Technical Debt

Technical debt is often a term used in the Agile community to describe the eventual consequences of deferring complexity or implementing incomplete changes. As a development team progresses through sprints and releases, there may be additional coordinated work that must be addressed elsewhere in the software code or documentation. When these changes do not get addressed within the immediate sprint or release and get deferred for a later iteration, the program accumulates a debt that must be paid at some point in the future. If the debt is not repaid, then it will keep on accumulating interest, making it harder to implement changes later on and potentially misrepresenting the level of progress the team has made on the development.

This technical debt can have significant impacts to the productivity and completeness of the software releases. There are many causes of technical debt to include: poor/improper coding, changing requirements, issues found during testing, inconsistent programming, lack of process rigor, hastiness to demonstrate progress, and poor enterprise architectures. As the project proceeds, this work must be addressed to avoid compounding issues associated with cost over runs, schedule delays, and product quality. The scrum master and development team must balance completion of planned capabilities with eliminating this technical debit. Addressing technical debt early may hurt short-term productivity, but provides long-term benefits. Technical debt can be minimized with effective architectures and owners, use of automated and regression testing, process disciplines in the development team's coding practices, business rules with the product owner on requirements, and incentives with the government.

Key Questions to Validate Systems Engineering Strategies:

- Does the program leverage existing hardware platforms?

- Does the design integrate mature technologies and align with enterprise architectures, standards, and interfaces?
- How are team engineers collaborating with enterprise architectures to ensure program or enterprise changes are understood, impacts assessed, and technical documentation kept current?
- Does the design anticipate technological evolution?
- Does the design consider mission assurance?
- Has the team conducted a sufficient systems engineering/cost tradeoff analysis?
- How frequently is code integrated and tested in the developer's environment?
- Is all code reviewed by at least one other team member prior to system testing?
- How frequently are scope changes made within a sprint? What is the process to review/approve?
- Does the government have access or insight into the development environment (code, metrics)?
- At the end of a sprint, does the team reflect on what worked and how to improve in future sprints?
- How are development activities coordinated and integrated across releases?
- How are enterprise-level issues identified, managed, and resolved?
- How are program risks managed? Is there a central (single) risk repository? Do stakeholders have access to this repository and can they contribute to it? How quickly are risk mitigations implemented after being identified? How frequently are risks reassessed and reprioritized?
- Does the program have tools and processes in place to ensure rigorous configuration management of requirements, code, technical designs, and baselines?
- What is the continuous improvement strategy?

See also: Agile architecture articles

10 Contracting

Contracting is a challenging, but critical, element in attaining the benefits of Agile practices. Long contracting timelines and costly change requests have become major hurdles in executing Agile developments and enabling small, frequent releases. Contracting strategies for Agile programs must be designed to support the short development and delivery timelines that IT requires. Today, a full and open competition for an IT contract can take as long as 12–18 months. Timelines such as these have driven many IT programs to structure programs in large, five year increments, which in turn drive significant program risk. Understanding the true schedule drivers, constraints, and regulations for the contracting processes is critical to designing an optimal Agile program strategy.

The current contracting environment does not encourage Agile approaches.

Table 3 identifies some of the key contracting areas where traditional contracting practices do not align to Agile contracting needs.

Table 3 Current Contracting Environment Vs Agile Contracting Needs

Current Contracting Environment	Contracting Area	Agile Contracting Needs

Long contracting timelines often becomes the driver for program execution	**Timelines**	Contracting is executed to support short development and delivery timelines
The functional requirements are locked-in at contract award; changes often require costly contract modifications	**Scope**	Contracts allow the program to refine Agile requirements throughout the development process
The contractor executes the technical solution and reports progress to the government	**Government-Contractor Relationship**	The government and contractor are working together on the development with daily interaction and collaboration
Contracting support is often centralized and unable to provide rapid turnaround on contract actions	**Contracting Support**	Embedded contracting support that can quickly and efficiently execute contract actions
Offeror proposes the development methodology and the contract is awarded based on the strength of the technical solution	**Technical Evaluation**	The government identifies the development process and the contract is awarded based on strength of development team and experience with Agile

10.1 Contracting Business Environment

Agile contracting processes must be deliberate and well executed to support regular program delivery timelines. Contracting strategies, processes, and culture must create a business environment that supports small, frequent releases and responds to change. The small, empowered teams central to Agile call for a tight partnership between program managers, users, and contractors in the DoD environment. The government contracting community serves as an invaluable linchpin to enable this relationship in a collaborative, flexible business environment. Dedicated onsite contracting support enables this close partnership with the program team. The program manager should work closely with the contracting officer as a business partner to devise a contract strategy. The contracting officer works with the release team to plan and manage upcoming contract actions, ensure compliance with contract requirements, and manage contractor performance.

The government program office and the Agile contractor development team must have a strong relationship characterized by daily interaction and frequent collaboration. In addition to executing the day-to-day development tasks, the government relies on the expertise of the contractor development team to help prioritize requirements, estimate future sprints and releases, and continuously evaluate and improve deployed capabilities. The contracting officer needs to work with the program office to foster this collaborative environment with the contractor.

10.2 Developing an Agile Contract Strategy

It's essential that both the contracting officer and the program office understand the important distinction between *contract requirements* and the *functional requirements* that are part of the Agile development process. In many cases, the two types of requirements differ significantly. Contract requirements are strictly limited to the tasks and activities the government requires a contractor to

perform, which in some cases have only a distant and indirect relationship to the requirements managed and tracked in the product backlog. For example, the contract requirements for a services contract may refer to the expertise required for the development team (e.g., 6 full-time-equivalent software development staff) as opposed to the functional requirements that are expressed in user stories (e.g., content search capability).

There is no single recommended contracting path or strategy for an Agile implementation, but establishing an environment with contract flexibility is essential to success. Several factors drive the choice of a particular contracting strategy. The program must understand the operational and programmatic priorities, constraints, and considerations involved in Agile development to properly develop a contract strategy. For example, deciding who is responsible for primary systems integration will determine whether the government can pursue a services versus a completion or product-delivery contract. A short development cycle often has more predictable requirements that may allow for a fixed-price contract. Long development cycles involve greater unknowns and may require a more flexible contract type. The political environment may favor a particular contract type or contract vehicle. The program should work closely with the contracting officer to consider the following factors when developing the contract strategy:

- Who is responsible for systems integration?
- What is the overall development timeline?
- What is the frequency of releases?
- Does the current political environment drive the use of a particular contract type or vehicle?
- What is the level of contracting support?
- Does the contracting office have standardized processes or is it willing to pursue them?
- Are government resources available to actively manage contractor support?
- Is the program considered high risk?
- What level of risk is the government willing to accept?
- What level of integration is required?
- What is the level of integration risk if multiple contractors conduct parallel developments?
- Did market research identify available qualified contractors with Agile and domain experience?
- Is an Agile process well defined or already in place within the government program office?
- Are other, similar programs currently using or thinking of pursuing Agile?
- Does the program have executive-level support for Agile development?
- Can the program leverage established contract vehicles - portfolio, enterprise, or external level?

10.3 Contract Vehicles

The government uses many types of contract vehicles. At a basic level, there are a single-award IDIQ contracts (award to one vendor) and multiple award IDIQ contracts. Under a multiple-award contract, several qualified vendors receive an IDIQ contract and all the contract awardees compete for each task order – a practice known as fair opportunity. The government can issue orders faster under a single-award IDIQ than under a multiple-award; however, a single award loses the benefits of continuous competition and the ability to switch easily among contractors in cases of unsatisfactory performance.

Other contract vehicles include Blanket Purchase Agreements (BPAs) off an existing General Services Administration (GSA) schedule contract (e.g., Schedule 70), Government-wide Acquisition Contracts (GWACs) (e.g., GSA Alliant), and Agency-level multiple-award contracts (e.g., Encore II). The government has already awarded such contracts, and makes them available for immediate use. However, these contracts may have disadvantages such as limited selection of vendors with Agile experience, limitations on contract types, and non-dedicated contracting support staff.

Establishing a contract vehicle up front at the PEO, portfolio, or enterprise level would enable many programs to leverage the contract and the benefits of its streamlined processes, allowing them to shorten contracting timelines and focus their strategy and energy on task orders. Programs that fall under the portfolio could expedite the contracting process using task orders rather than individual contracts.

Contract vehicles suitable for Agile development have pre-established contract pricing, terms and conditions, and pre-vetted qualified vendor(s). This allows task orders to be issued in a matter of weeks versus months. A portfolio-level contract vehicle can have a targeted scope that should attract the right pool of vendors with Agile expertise in the technical domain. Programs can also use a portfolio contract to streamline the process for meeting other needs, such as obtaining contractor support in the areas of Agile subject matter expertise, independent testing, and continuous prototyping. However, these types of contracts require that the program invest time and resources up front to compete and award the umbrella contract vehicle.

When developing an IDIQ contract for Agile development, the program can consider a number of steps, shown below, to streamline and improve the contracting and ordering processes.

- Engage users and testers in developing the contract scope, evaluation criteria, incentives, and terms and conditions to ensure the contracting activity fully meets all needs and considerations.
- Develop templates and standard business processes to streamline ordering procedures and ensure the quick execution of orders.
- Work with the contracting office to develop standard Performance Work Statement (PWS) language and proposal evaluation criteria.
- Use past performance and relevant experience as source selection criteria for individual task order awards to incentivize contractor performance.
- Understand the dedicated contracting process and associated timelines for executing task orders. Program managers should become familiar with contracting documentation and approval requirements.

10.3.1 Services Contracts

As noted, the Agile development process is characterized by constant change and reprioritization of requirements. This makes it impractical to select an Agile development contractor using a contract type that locks-in requirements up front and defines end-state products on a completion-basis. Traditional development contracts often use a product-based firm fixed price (FFP) or cost-reimbursement completion type of contract to hold the contractor accountable for delivery of a product or capability.

Under a product- based contract, the contractor proposes to the government a development methodology and the government awards the contract based on the strength of the technical solution. Any change to the original requirements can trigger a potentially expensive and time-consuming engineering change proposal (ECP).

Alternatively, the government can consider using a services-based contract to obtain Agile development support. Under this scenario, the government seeks the time and expertise of an Agile software development contractor, rather than a software delivery end-product. The government chooses the contractor on the basis of the strength and qualifications of the proposed development team rather than the strength of the technical solution. *However, as with any services-type contract, this creates a risk because the contractor cannot be held accountable for delivering the end-product or capability.* The government can only hold the contractor accountable for providing the time and expertise agreed to in the contract, but the government is ultimately responsible for managing the development process to ensure product delivery. The daily interaction between the government and contractor that is fundamental to an Agile development strategy should help to mitigate risks of non-delivery; the government-led development team should be actively managing the development cycle and scaling-back capabilities when needed to meet the time-boxed sprint and release schedule. On the other hand, the team will need to carefully balance the need to meet schedule requirements with accumulating technical debt as outlined in section 9.6.

In addition, when using a services-based contract, the government must assume the role of primary systems integrator and have responsibility over the development process. An Agile development support contractor can provide expertise and integration support to the government, but under a services contract the government is ultimately responsible for delivery. The government must carefully consider the implications of this important responsibility when deciding to adopt an Agile development strategy.

Table 4 identifies some of the contract types available for a services-based contract strategy and the advantages and disadvantages that the program office should weigh.

Table 4 Contract Type Comparisons for Services Contracts

Contract Type	Pros	Cons
Fixed-Price Services Contract (either FFP, or Fixed Price Level-of-Effort)	• Generally preferred contract type in DoD • Easiest contract type to manage	• Requires a deliverable for payment (e.g., monthly report) unless progress payments are authorized • Cannot easily change labor mix and number of hours without contract modification
Cost Reimbursement Term (Level of Effort) Contract	• Provides flexibility to change labor mix and hours as long as it does not exceed contract ceiling • Does not require a deliverable for payment	• Contract ceiling may be difficult to establish based on Agile requirements, which can affect upfront fee determination • The contractor's cost accounting system must comply with

Contract Type	Pros	Cons
		acceptability standards • Government must monitor the contract for cost control assurance • Less incentive for contractor to control costs thus risk of a cost growth that could exceed budget or stakeholder commitments
Time-and-Material (T&M) (Labor Hour) Services Contract	• Provides flexibility to change labor mix and hours as long as it does not exceed contract ceiling • Does not require a deliverable for payment • Profit is built into the hourly labor rate so it does not require extensive upfront fee negotiation	• Unpopular contract type across the government • Requires close government monitoring • Contractor is not incentivized to control costs increasing the risk of a cost growth that could exceed budget or stakeholder commitments

The commercial sector often uses T&M contracts for Agile development, yet T&M is the least preferred contract type in the government because the contractor is not necessarily incentivized to control costs. Programs should consider this option if the government can manage the costs and scope on a proactive, continuous, and frequent basis. At the beginning of a project – a stage with many unknowns – T&M can provide maximum flexibility. Since the Agile development strategy requires daily interaction between the government and contractor staffs, the controls needed to monitor contractor performance are inherently built into the Agile development process and may provide the proper oversight to ensure efficient delivery and effective cost control under a T&M arrangement. Structuring the program and task orders into smaller, frequent releases (e.g., six months) limits the risks often associated with a T&M contract because of the short period of performance and the "build to budget" development cycle. As the project continues down the development path, and the team better understands the requirements and establishes a rhythm, the contract type could shift to a fixed-price or cost-plus arrangement.

Often the political environment or the availability of a contract vehicle forces programs into a particular contract type. If a T&M contract is not feasible or not preferred, a fixed price or cost reimbursement term contract can also be considered. The program should work closely with the Agile cost estimators and engineering team to assess the level of effort involved in establishing the ceiling price (or fixed price) on the contract. This is especially important under a cost-plus-fixed-fee contract, where the fixed fee is based on a percentage of the contract ceiling at the time of award. In addition, under a fixed price contract, the government should establish deliverables (e.g., monthly report) to provide recurring schedule payments.

Why Consider a Services Contract?

Traditional IT acquisition programs contract with a defense firm to deliver an end-product capability based on a defined set of 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 of developers, testers, integrators, database specialists, etc. If using an existing contract vehicle (preferably a portfolio-level contract) the government can issue an order for each 6-month release based on the estimate of the requirements captured in the product backlog. In the course of the release, the Agile requirements will likely change based on reprioritization and changes in the development process. In contrast to a traditional product- or completion-based contract, a services contract provides the flexibility to change the release requirements continuously and still retain a consistent contractor team.

10.3.2 Product or Completion-Based Contracts

The government traditionally uses completion- or product-based contracts for IT acquisition, but these are inappropriate for an Agile development. This type of contract requires upfront definition of requirements so that the contractor can adequately estimate the effort involved and then prepare and submit technical and cost proposals for the work. The nature of Agile would make it very difficult – if not impossible – to identify requirements for an Agile development at the level of detail necessary for a cost proposal estimate. In addition, under a product or completion-type contract, the government should not be directing the contractor to use an Agile development methodology. The government should be using a Statement of Objectives (SOO) to describe the overall objectives of the program (e.g., capability delivery every 6 weeks) and let the contractor propose a development methodology that best meets the objectives of the government. It will be harder to hold the contractor accountable for delivery if the government is directing the contractor to use a specific development methodology.

If the program cannot use a services-type contract, one possible alternative would be to establish a new IDIQ contract or use an existing contract vehicle and issue an order for each well-defined release or sprint. However, sprints typically have very short timelines that make it impractical to issue a new order for every 4-8 week sprint. Releases cover longer timelines, but do not have defined scopes and requirements that would be required to use this type of contract. Using this type of vehicle effectively would require significant coordination with the contracting office, as well as streamlined processes to rapidly issue orders that keep pace with the Agile delivery cycle. This strategy would also require more time and resources to award the initial umbrella contract vehicle and manage the contract and orders.

Under this type of arrangement, a cost-type completion contract would provide more flexibility than a fixed-price product-based contract. A cost-reimbursement contract normally requires the contractor to complete and deliver the specified end product (release or sprint) within the estimated cost, as a condition for payment of the entire fixed fee. However, if the contractor cannot complete the work within the estimated cost, the government may demand that the contractor expend more effort without an increase in fee, but this would still involve an increase in the estimated cost.

Programs could also consider a fixed price incentive fee (FPIF) contract in this scenario. This type of contract incentivizes the development contractor to deliver the release or sprint below the target cost to obtain a higher fee according to the negotiated profit adjustment formula. DoD promotes the use of FPIF contracts under Better Buying Power. This also requires the government to have a more in-depth understanding of the requirements, development work required, and estimated costs to effectively negotiate the fixed price. However, such contracts are more difficult to manage because the government must negotiate the target cost, target profit, ceiling price, and project adjustment formula with each order. Further, as is the case with any completion type contract, frequent changes and reprioritization of requirements may significantly change the end product, driving the need for one or more ECPs and contract modifications.

10.4 Contract Incentives

Under an Agile development, the team should be 100% focused on delivery and a speedy and efficient contracting process needs to support short delivery cycles. As a result, the use of complicated contract incentives (e.g., incentive fee, award fee) is not recommended when using a services-type contract for Agile development. Contract incentives are time consuming and resource intensive to manage, and can lead to a contentious working relationship between the government and contractor. Quick contract turnaround and a close working relationship between the government and contractor is absolutely critical under an Agile development; contract incentives can become a distraction to the program impeding the delivery cycle.

Contract incentives are often used when the government does not have the capacity or control to actively monitor contractor performance. However, under an Agile development, the government is actively interacting with the contractor on a daily basis. The controls needed to manage contractor performance are inherent to the Agile development process and additional incentives can be burdensome and unnecessary. However, the program should still consider using a performance-based contract when contracting for services. The program can issue a Performance Work Statement and use past performance as an incentive, or use the metrics recommended in Section 12 of this guide to manage contractor performance.

See also:

- OMB Contracting Guidance to Support Modular Development
- GAO Report 12-681 Effective Practices and Federal Challenges in Applying Agile Methods, July 2012

Key Questions to Validate Contract Strategies:

- Do the contract strategy and timelines support frequent capability releases?
- Is the program pursuing a services-based contract or completion/product delivery contract?
- Is the government the prime systems integrator?
- Does the program have dedicated contracting support?
- Is the contracting officer co-located with the program?
- Does the contracting environment support Agile development processes?

- Do existing contract vehicles support Agile delivery?
- What is the level of engagement between the contracting officer and Agile team?
- How is the government monitoring the contractor's performance?

11 Cost Estimation

Estimating costs in an Agile environment requires a more iterative, integrated, and collaborative approach than in traditional acquisition programs. While a program can develop rough order of magnitude estimates in the beginning, it cannot gain an understanding of costs and schedule with any true fidelity until the development teams are in a rhythm.

Contrary to the myth that Agile is an undisciplined approach that downplays cost aspects, cost estimation is a critical activity in programs that use Agile practices. However, cost estimation in an Agile environment is challenging, especially for teams new to Agile processes. It consists of an ongoing "just in time" program activity tightly integrated with the activities of the development team and engineers. During the program execution phase, a high-level program estimate undergoes refinement to create detailed release and sprint-level estimates as requirements become better defined. The fidelity of the cost estimate increases once a development team is established to help estimate the level of work for each requirement (i.e., as translated into user stories), and can further improve with subsequent releases as the team captures performance productivity metrics for deployed releases. A comprehensive and competent cost estimation process and methodology gives senior stakeholders the confidence to relax rigorous oversight, and provides the program with valuable cost information to continuously improve performance and management.

Traditional programs often treat cost analysis as a separate activity, rather than an integrated team endeavor, but cost estimation on an Agile program is a team-based activity. Ideally, the government cost estimator should be co-located with the systems engineers and development team as each Agile release is scoped, developed, and tested. Ongoing collaboration among the users, development team, systems engineers, cost estimators, and other stakeholders is critical to ensure agreement on requirements prioritization in the product backlog, and to gain a thorough understanding of the amount of effort required for each release. It also enables an integrated assessment of the operational and programmatic risks, technical performance, cost drivers, affordability, and schedules.

11.1 Program Cost Estimate

Programs use cost estimates to create the spending plan during the acquisition phase. This spending plan outlines how and at what rate the program will expend its funding over time. Because a reasonable and supportable budget is essential to efficient and timely execution of a program, a competent cost estimate creates the key foundation of a good budget.

Cost estimating techniques for an Agile development do not necessarily differ from the way estimates are created for a traditional development program. Product size is usually the biggest cost factor when developing a software cost estimate. Programs frequently estimate size based on source lines of code (SLOC) or function points for traditional software developments. The government cost estimator should

work with the program staff and systems engineers to estimate the product size resulting from the effort, drawing on technical baseline requirements for the IT system.

The government may need to consider some nuances of the Agile development process when developing the program cost estimate. The impact of each cost factor listed below varies depending on the Agile methodology employed, program structure, and unique attributes of the program.

- Program Management – Agile programs require intensive government participation and involvement to manage the overall development process. The government program office must closely engage on a daily basis with the contractor development team, and must enlist dedicated support from additional government resources (e.g., cost estimators, testers, users, contracting officer). In many cases, this may require an increase in government resources, especially in the areas of program management, system engineering, and testing.
- Testing –The cost estimate for testing will have to consider the impact of the short, frequent, integrated real-time testing characteristic of Agile developments, and determine if it differs from the costs for a traditional waterfall testing approach. Additionally, the government must evaluate the impact on regression testing.
- User Participation – User representation on the Agile release team is necessary to help prioritize requirements, assist in creating user stories, conduct user acceptance testing, and report feedback on deployed capabilities. The costs of maintaining continuous long-term user representation on the Agile team should be factored into the cost estimate.
- Deployment – An Agile release deploys new capabilities every 6–12 months. The degree of change and level of complexity of each release may require further consideration of deployment costs. For example, additional end user engagement and training may be required with the rollout of each release.
- Sustainment – As new capabilities are deployed, the program will have to operate and maintain capabilities deployed from prior releases. The program must evaluate the cost impacts of sustaining multiple, frequent, and overlapping releases.

11.2 Alignment to Budgets

The budget for an Agile development program is based on the government program cost estimate, given the technical baseline requirements for the IT system. The program cost estimate must consider the Agile development costs along with all the other costs to plan, manage, acquire, maintain, and dispose of the program. One benefit of Agile is that once a budget has been established the program can be structured to "build to budget." The funding that the program receives then drives the number of releases it can manage in a given year and the totality of delivered requirements within the entire development period of performance. The challenge within DoD is often a resistance to allocate budget for a program until all the requirements are fully defined and approved. There needs to be a clear understanding of the level of requirements maturity required for budget authorizations.

Given the iterative, segmented nature of Agile development, Agile programs can scale up or down more easily than traditional programs that deliver capability in 5–10 year increments. Even if the Agile

program experiences a major funding cut or cancellation during development, deployed releases will already have provided capability to the user, which is not guaranteed under a traditional development.

As illustrated in Figure 17 under traditional waterfall development projects, scope and quality are fixed, while schedule and cost vary and generally increase as requirements are further defined during the project. Agile development projects attempt to fix schedule, cost, and quality while adjusting the scope of specific releases to fit these constraints. The scope of each delivered release is defined by the prioritization of requirements that deliver end-user functionality within the constraints of the cost, schedule, and quality parameters. As a result, treating cost as an independent variable drives the prioritization of requirements and facilitates release planning.

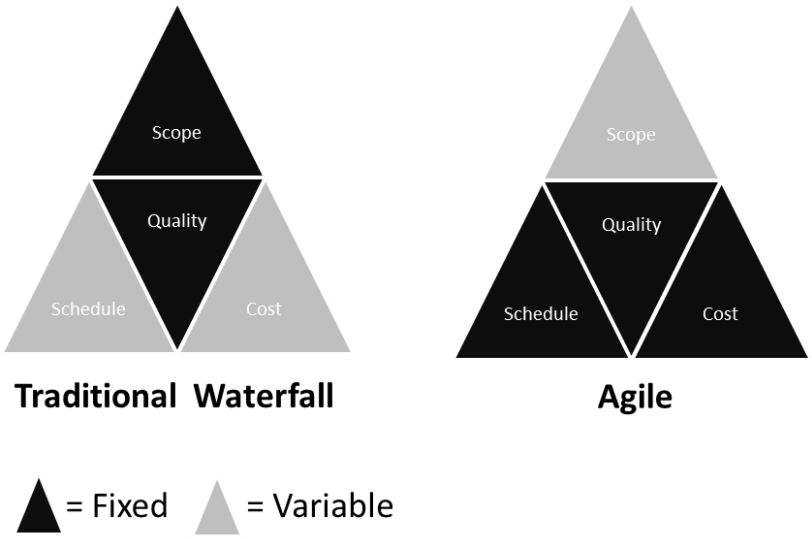


Figure 17: Differences In Traditional Waterfall vs. Agile Cost Variables

As noted, Agile programs can absorb budget cuts more easily than traditional acquisition programs. It is easier to cut funding from an IT program that can easily defer a small release than to "break" a weapon system increment or significantly delay delivery to the user. As a result, funding agencies often target IT programs for budget cuts rather than weapon system programs that operate under inflexible CDDs. IT programs that employ Agile techniques must ensure they have sufficient support at the Service/Agency, OSD, and Joint Staff levels to avoid constant inroads on their funding.

Decomposing a software development project into several deployable releases enables projects using Agile methods to adapt to both permanent reductions in the lifecycle budget and temporary reductions in the development budget. Traditional, waterfall-based development projects find it far more difficult, both technically and contractually, to accommodate changing budgetary conditions.

11.2.1 Independent Government Cost Estimates (IGCEs)

The IGCE is the government's estimate of the resources and projected costs a contractor will incur in the performance of a contract. The FAR requires a separate IGCE for every contract action, to include each task and BPA order. The IGCE should include only those elements applicable to the contract

requirements, as outlined in the Statement of Work/SOO/PWS. The structure of the IGCE depends on the type of contract pursued (e.g., services vs. delivery, term vs. completion, labor hours vs. deliverable). For example, a services contract for Agile development support would require a cost estimate covering the labor hours and labor categories of the Agile development staff; by contrast, a deliverables contract may require a more complicated IGCE to estimate the costs of each requirement, feature, or functionality.

11.2.2 Release-Level Estimation

Following contract award, the government can work with the contractor to execute the Agile development process. The program should continue ongoing, concurrent cost estimation activities. This facilitates the Agile planning process for future iterations, and provides management with forecasting and performance data to reduce the need for formal reviews.

Agile developments typically use cost estimating strategies based on relative measures of size, such as story points. No set formula exists for defining the size of a story, so release teams can use various techniques centered on small team collaboration to reach consensus on the number of points for each story. Teams base story-point estimates on the amount of effort involved in developing the feature, its relative complexity, and the inherent risks. For example, a small, simple story could be assigned one point, whereas a moderate story could assign eight points –meaning that the associated product would take eight times as long to develop. This strategy provides a way to compare the size of one story to another, thus ultimately enabling programs to comparatively measure the size of the entire pool of requirements. Some common scoring systems that teams may use to assign story point complexity measures include Fibonacci series, ideal days, or small-medium-large.

After developing user stories, the Agile team prioritizes them into what becomes the product backlog, and then constructs a release from the product backlog. The number of user stories that comprise a release is based on the sprint schedule measured against the team's estimated velocity (a measure of productivity unique to the Agile development method). Programs can initially measure team velocity using historical values or by making a forecast based on the skill sets of the team and the team's experience with the specific product or technology. After the first sprint, the team selects the user stories for the next iteration and the team working on each story makes a more fine-tuned estimate of the appropriate story points for that sprint. The team can then estimate whether it can build the proposed set of stories within the sprint timeframe, given the team's actual velocity.

Teams regularly conduct this activity and reassess the points of the backlog items based on insight obtained from recent developments. This iterative approach increases the fidelity of estimates over time. The release-level estimate is at a high level, while sprint-level estimates are more refined and detailed to help the team use the cost estimates to manage the project effectively.

As the program progresses through the development process, the program should examine how closely the initial estimates match actual effort, if stories were added or removed from a release, and if the estimating methodology or value for the story changed. Tracking the planned versus actual progress of

completed iterations against the story points that remain to be completed, using a burn-down chart (see Figure 3), can help to track the progress of the Agile development.

Key Questions to Validate Estimates:

- Does the development team have good historical data that it can refer to when making new estimates?
- Is the team's velocity updated regularly and used to scope sprints?
- Does the development team have strong knowledge/experience in the user domain?
- Does the development team have strong knowledge of/experience with the technologies?
- Do all development teams use a consistent estimation method?

For additional information on cost estimates for Agile development, see:

- Estimating on Agile Projects by Scott Ambler
- Agile Estimating and Planning by Mike Cohn
- Agile Estimating Articles

12 Metrics

Programs that adopt Agile methods must tailor the metrics traditionally used by DoD to reflect the different processes and artifacts used in Agile. Agile metrics focus primarily on the development team during sprints. Programs use work elements (e.g., story points, staff hours, task lists, etc.), burn-down charts, and velocity to track progress and measure productivity, costs, schedule, and performance.

A program office and contractor can track a few dozen metrics for requirements, cost, schedule, performance, architecture, size/complexity, test, and risk. The following paragraphs describe some common metrics a program office could manage.

12.1 Requirements Metrics

User stories offer the best measure of requirements in an Agile program. Evaluating when user stories are implemented or defined depends on how well the team understands the features requested, the features actually delivered, and the degree of change in requirements and user priorities. Recommended metrics to monitor requirements include:

- Number of stories selected for a sprint
- Number of stories completed during a sprint
- Number of new feature stories added to a product backlog
- Number of user stories re-prioritized
- Number of changes to the stories selected for a sprint

Results that should draw attention to problems include metrics that show the team often fails to complete the number of stories selected for a sprint, or user stories that the team constantly defers (lower priority). Managers must recognize that there are several reasons why a team may not complete

selected stories. Perhaps the team routinely underestimates the complexity of the user stories or lacks some necessary skillsets, or the introduction of a new tool added a learning curve for the team. Teams may defer user stories from sprint to sprint because of poor estimation. As previously noted, it may take a few sprints for the team to achieve confidence and accuracy in its estimates. The problem could also lie in dependencies between stories. If a high-priority user story depends on one lower in the backlog, the team cannot execute it until after it has completed the less critical item.

12.2 Cost Metrics

Work elements and the number of stories completed in a sprint determine cost. The costs per work element metrics are subjective, and vary according to the complexity of the stories in a sprint and the number of team members contributing to that sprint. Work elements are typically not equivalent to time, and programs must take this into consideration when evaluating these metrics. As described previously, teams may also need several sprints to "normalize" cost metrics, and the cost per work element may fluctuate in the beginning. Figure 18 provides sample work element tracking charts. This metric also depends heavily on the accuracy of the estimates and stability of the team's velocity. Recommended metrics to manage cost include:

- Planned cost per work element
- Actual cost per work element
- Total work elements committed to sprint (velocity)
- Total hours logged against all stories in a sprint
- Total work elements completed in a sprint

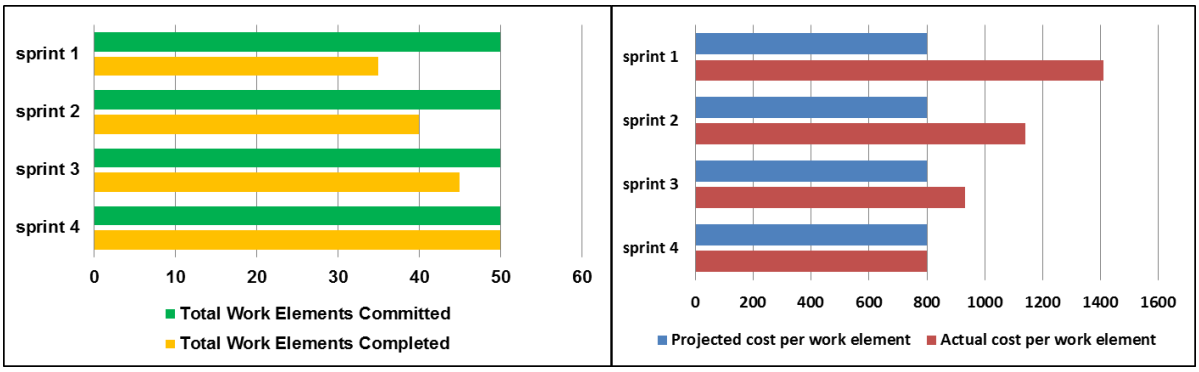


Figure 18 Sample Cost Metrics Charts

Since cost metrics are based on work elements and velocity, inaccurate estimates throw off the planned versus actual costs. Therefore, programs must consider a team's ability to make accurate estimates when looking at program costs in the early stages. If estimation continues to be a problem, the program management should refer to the suggested guidelines in section 11 of this document.

12.3 Performance Metrics

Agile programs deliver a potentially shippable feature at the end of each sprint, with a more fully functional capability delivered at the end of the release. This iterative approach allows for early

detection of potential problems and delays and allows the program to make adjustments that reduce impact on overall program performance and delivery. Figure 19 provides a sample performance metric chart. Recommended metrics to monitor performance include:

- Number of user stories accepted
- Number of bugs discovered by user after release

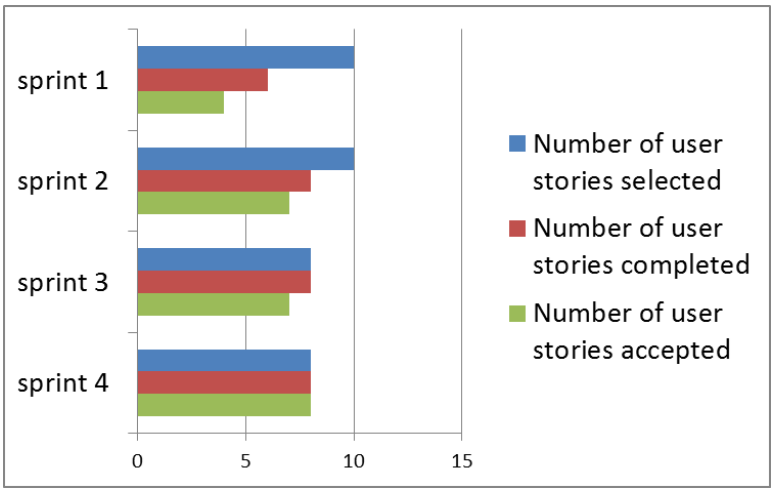


Figure 19 Sample Performance Metric Chart

Agile processes also include and engage the stakeholders throughout the entire project lifecycle. Stakeholder involvement throughout the Agile process helps the development team to quickly and easily clarify requirements and respond to user requests, thus implementing features more likely to be accepted by the user. User acceptance of capabilities for each user story means that the team is fulfilling requirements and responding to change. This means that Agile projects create high-quality products, well aligned with stakeholder expectations.

The iterative nature of Agile allows for more frequent collection of metrics. Programs must balance the benefits of timely and useful metrics against the burden that collecting and reporting the metrics places on the team. Programs should determine a frequency that reflects when significant changes or trends can be observed and changes may be made.

Ideally, programs should use tools that automatically collect and report metrics. Most Agile management tools track, manage, and report metrics out of the box.

12.4 Agile and EVM

Earned Value Management (EVM) for an Agile development program has been debated across federal acquisition and Agile communities. The value and effective implementation of EVM in traditional acquisition programs has also been an ongoing challenge. A 3 Jul 07 OSD/AT&L memo highlights:

> "EVM is considered by many in the project management community to be the best option currently available for holding all parties accountable for the effective management of **large and complex**

projects. EVM provides a disciplined approach to managing projects successfully through the use of an integrated system to plan and control authorized work to achieve cost, schedule, and performance objectives. The fidelity of the information produced by the EVM System is critical to providing an objective assessment of a program's performance from which well-informed management decisions can be made. Moreover, EVM is not just a cost report; it is a tool to help program managers and their team members operate more effectively in managing their programs."

Contract type is a key consideration with EVM. EVM is required on cost or incentive contracts at or above $20 million. Use of EVM on FFP contracts is limited to only when the PM believes there is significant schedule risk, which should not be the case with time-phased releases in Agile. EVM does not apply to T&M/Services contracts. See the DoD EVM website for official policies and FAQs.

Given the dynamic and iterative structure and processes of Agile, implementing an EVM system can pose a significant challenge with little value. Agile embraces responding to changes, not "controlling authorized work". As changes are made throughout development, effective reporting against cost, schedule, and performance against a baseline is difficult. The metrics outlined in this section meet the intent of EVM to provide a disciplined approach to manage programs by providing key insight into progress and issues.

Key Questions for Validating Metrics:

- What metrics does the Program Management Office use to manage the program?
- Which of these are provided by the contractor and which by the government?
- How will these metrics be collected and used? What types of decisions will be made as a result?
- What metrics are shared with program stakeholders? With senior leadership?
- What contractor incentives are tied to these metrics?
- Are a process and culture in place to ensure accurate data is collected?

See also:

- Project Management's Not So Odd Couple (EVM and Agile) By John Zyskowski, FCW
- AgileEVM: Measuring Cost Efficiency Across the Product Lifecycle by Tamara Sulaiman, InfoQ
- Stakeholder Needs and Expectations, Planning your Agile Project and Program Metrics by William A. Broadus III, DAU

13 Testing

Testing in an Agile environment requires upfront engagement and collaboration with the testing and certification/accreditation communities to design processes that support rapid development and deployment cycles. Ensuring early and active involvement of testers and certifiers in the planning stages, and integrating these personnel with the development team, reduces program risk, costs, and schedules, while providing timely insight to inform the development process, thus increasing the software quality. Figure 20 compares the serial approach to testing in a traditional program and the integrated approach of testing in an Agile program. In the Agile model, testing occurs in line with

development during the sprints, followed by independent test and certification The more active testers are during the sprints, the more streamlined the final release testing following the last sprint.

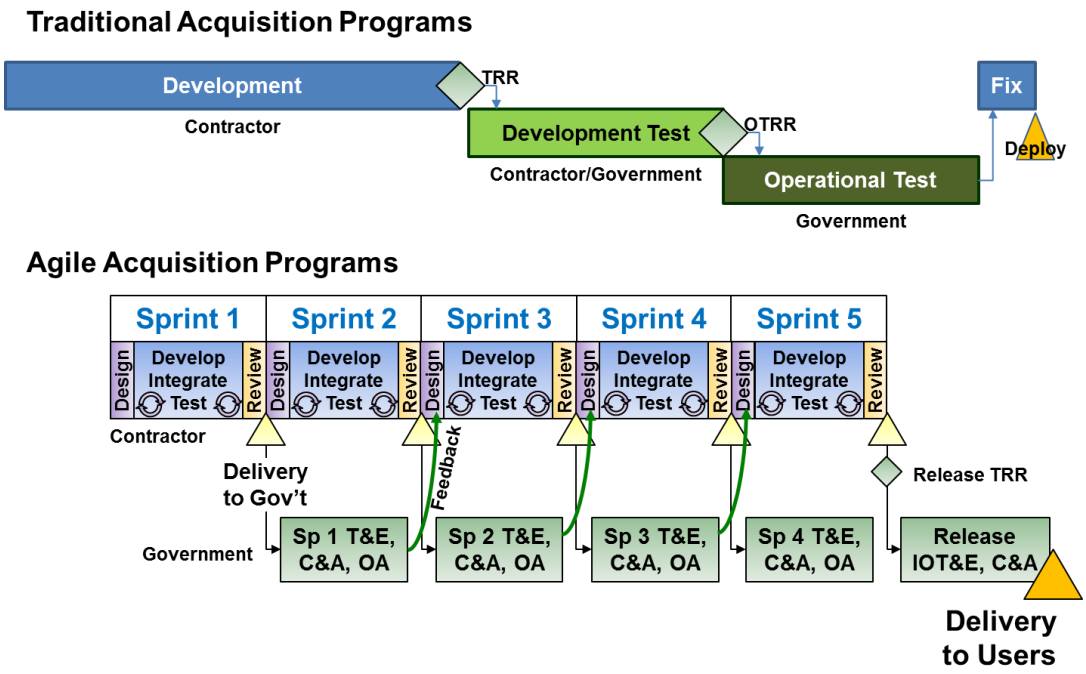


Figure 20: Serial vs. Integrated Testing

When teams convert requirements into user stories, associated acceptance criteria are essential, as testing then can focus less on determining pass/fail and more on how the software performs. Agile approaches place strong emphasis on automated tests and continuous integration throughout development to provide developers immediate feedback on software quality. Automated tests and continuous integration also support regression testing to ensure that code from new sprints integrates with previous code deliveries. These testing activities accommodate frequent updates and deliveries, allowing developers to continue working until software meets the acceptance criteria.

Ideally, the government should establish the test infrastructure at a portfolio or enterprise level, spanning and supporting multiple programs. This requires upfront planning, investment, and resources. Leveraging a common infrastructure enables portfolios to test and certify their component programs more effectively and efficiently. This also reduces infrastructure and staffing costs while increasing interoperability, security, and agility. The DoD (the Defense Information Systems Agency in particular) has made some strategic investments in common test environments and ranges that Agile programs might use.

Verification testing confirms that the system element meets the design-to or build-to specifications defined in the functional, allocated, and product baselines. In an Agile environment verification is an integral element of each sprint. At the end of each sprint, the developers demonstrate the software functionality to the users, testers, program office, and related stakeholders. Testers use acceptance

criteria to verify that the software meets the design, and the stakeholders verify that it meets intended operational and technical objectives. If so, they accept the user story as done.

Another integral element of Agile testing, **validation**, answers the question: "Is this [software] the right solution to the problem?" Validation occurs through frequent meetings and reviews to determine "Is this what the users/customers/stakeholders want?" Active involvement in sprint reviews and planning allows users and stakeholders to see what the team has done and shape what the team will do next.

When developing a test strategy, programs adopting Agile should consider including the following elements:

- Integrate test and certification teams into the acquisition and development cycles at the very beginning, including requirements determination and definition. These teams must include testers knowledgeable about the applicable success criteria in the developmental, operational, interoperability, information assurance, information security, accreditation, system, and mission areas.
- Challenge the resulting test and certification team to create fully integrated test plans that combine required test points and eliminate duplicative and non-value-added testing.
- As test data is collected, make it available to analysts representing the full life-cycle team to permit early identification of discrepancies, limitations, and achievement of success criteria.
- Develop and leverage common test infrastructures to foster effective use of resources, rapid testing, and improved integration.
- Use automated test scripts and tools, including those established for common development and test platforms, to perform regression testing efficiently and to further accelerate testing.
- Establish a continuous integration and deployment methodology to build the system, execute the regression test suite(s), and deploy the working builds to a separate environment on a regular, frequent (e.g., nightly) interval. This translates into early identification of technical, security, operational, and system deficiencies so that appropriate and timely corrective actions can take place prior to deploying the capability.
- Use a robust defect tracking and reporting system to drive immediate fixes or additions to the program backlogs.

Key Questions to Validate Test Strategies:

- Are testers actively involved throughout the requirement, planning, and development phases?
- Do users, government testers, and certifiers review interim capabilities delivered via sprints?
- Do the developers, government, and certifiers make maximum use of automated tests, test data?
- Are joint interoperability testing and security testing integrated in the testing activities and environment?
- Do Agile projects for the government have a common test environment, tools, methods, data collection, and processes available?

- As requirements are translated into user stories (or equivalents), are acceptance criteria and/or test cases well understood and used as the primary measure for the testing community?
- Are test and certification strategy documents written and approved at the capstone level while reserving release-/sprint-specific content for small documents in the release planning phase?
- Do the test strategy and backlog include or consider the following: functional (requirements) testing, non-functional requirements testing (security and interoperability), SW and HW interfaces, integration testing, end-to-end testing, story tests, exploratory testing, scenario testing, usability testing, user acceptance testing, unit testing, component testing, performance and load testing, security testing, maintainability testing, interoperability testing, reliability testing, compatibility testing, and regression testing.
- Are sufficient testers assigned to the Agile development effort? What are their Agile qualifications? What is their Agile experience level?

See also

- Test and Evaluation for Agile by Dr. Steve Hutchinson
- Shift Left by Dr. Steve Hutchinson, Defense AT&L Magazine
- Shift Left! editorial by Dr. Steve Hutchinson, ITEA Journal, June 2013
- Agile Testing Strategies by Scott Ambler and Associates
- Agile Testing: A Practical Guide for Testers and Agile Teams by Lisa Crispin and Janet Gregory
- Articles on Testing in Agile

14 Deployment/Sustainment

Programs will deploy capabilities to users upon a successful deployment decision by the authorities in the acquisition and operational communities. Capabilities are usually deployed at the release level; however, they may also be deployed at the sprint level, given sufficient testing, certification, and approval. The first sprint of a release for example can be fielded independently if it addresses a critical security issue from a previous deployment.

All IT acquisition programs including those using Agile development must comply with the Clinger-Cohen Act and information assurance policies. Testing, certification, and approvals must be integrated early in development and streamlined to support short deployment timelines. As with any IT development, programs must have a clear strategy to deploy and sustain the capability.

The iterative nature of Agile development allows for continual evolution of capabilities and addressing software deficiencies. As users operate and sustain the IT capabilities, there should be continual feedback to the product manager to shape the program backlog. Direct user feedback to the program office and development team is also critical to drive future releases.

15 Pulling It All Together – Potential Agile Program Structures

The structure shown in Figure 21 represents one potential tailoring of acquisition policies such as DoDI 5000.02 and BCL to enable small, frequent releases of IT capabilities to users. The smaller the program, the more flexibility it has to adopt Agile practices and benefit from reduced oversight and regulations. Agile development works best when building additional functionality into an existing platform or GOTS/COTS product. Existing programs could transition to an Agile approach for subsequent Increments or Block upgrades after establishing the operational baseline (Increment 1); for example, GCSS-J transitioned to an Agile approach starting with Block 7 in 2008.

In this potential structure, the program would still have a MDD and Milestones A and B, but these reviews and the early phases would be heavily tailored and streamlined. The goal is to get from MDD to Milestone B in less than 18 months. This would provide sufficient analysis and planning while beginning software development as soon as possible. Programs could use enterprise- and portfolio-level documentation and processes, such as contracting vehicles, enterprise architectures, existing IT platforms, testing environments, and capstone-level DoD 5000 documentation to enable this rapid timeframe.

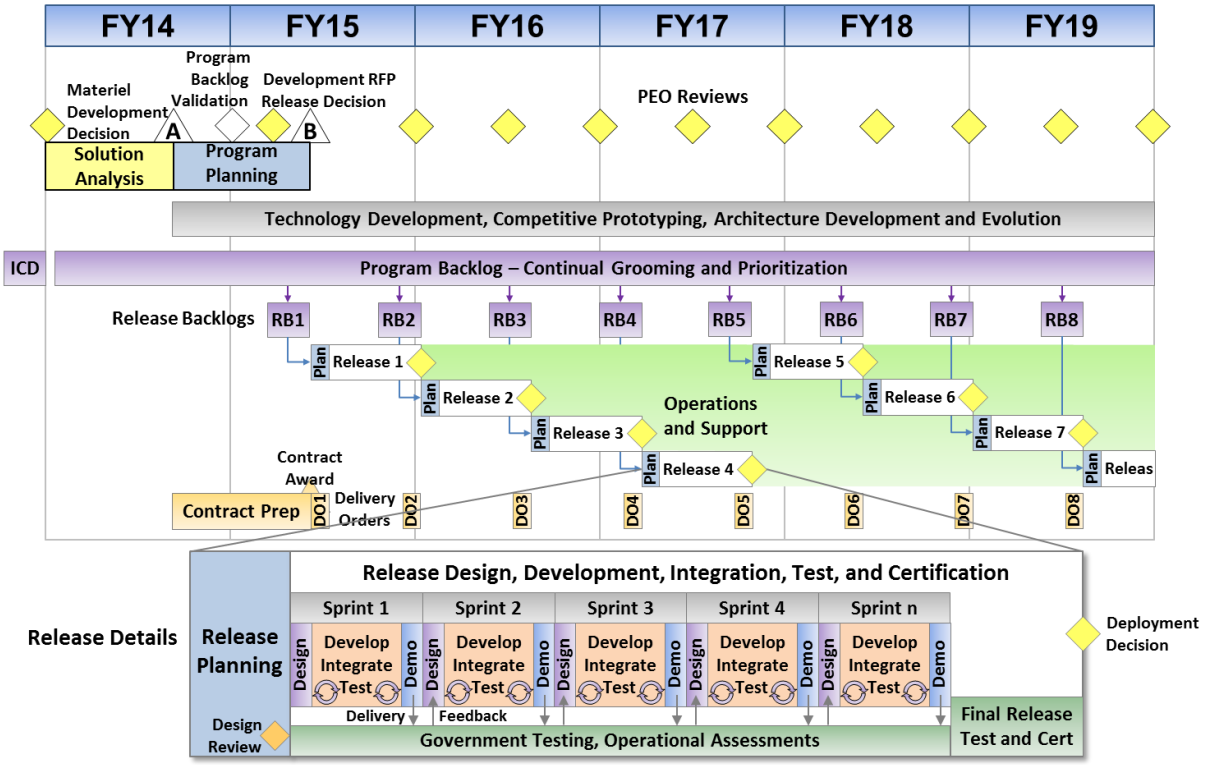


Figure 21: Potential Agile Program Structure

Continuing with the example structure above, the releases shown are six months long, preceded by a one-month planning phase. Various factors, including user demands and environment, program risk, developer environment, budget constraints, contracts, and acquisition oversight drive the duration of the releases and sprints. Each release has a deployment decision – a streamlined and focused review, similar to the traditional Full Deployment Decisions in DoDI 5000. 02 – in which the program draws on

inputs from the acquisition, testers, certifiers, and user communities to decide whether to deploy the capability. Following deployment, the user community operates and maintains the capability according to a defined strategy. Frequent collaboration with the operational community should generate inputs for the program backlog to address deficiencies or add new capabilities. This collaboration also helps to determine the relative priorities of the items on the program backlog so that the next release and sprint address the highest priority requirements.

Throughout development, the program office should develop and evolve an architecture to define the structure of the system. This process could include aligning and integrating the new architecture with an enterprise architecture. The architecture should include the software elements being developed and integrated with other software, and the hardware on which it will operate. This architecture should serve as the foundation that system engineers use to collaborate internally and externally on the development and evolution of an integrated IT system.

The rapid pace of development and response to changes cannot accommodate a series of gate reviews and significant upfront documentation. Oversight in an Agile environment focuses less on a series of gate reviews and more on in-progress reviews of an empowered team. The small, frequent releases should be reviewed and approved at the lowest possible level, likely a Program Executive Officer or related official. SAEs/Component Acquisition Executives, and if necessary the OSD level, should conduct their reviews at the portfolio level, with content highlighting what the program accomplished in the past year and what its plans for the next year.

The approach defined so far implies a single developer team producing releases sequentially. Depending on the nature of the development, funding, and acquisition/contract strategies, a program could have a single contractor with multiple development teams, or multiple contractors developing releases in parallel. The latter approach taps multiple sources of expertise to develop the comprehensive solution. However, it requires increased government planning, coordination, and integration.

Additional considerations for enabling an Agile environment include technology prototyping and software platforms. Ideally, programs would manage or host a software platform at the portfolio or enterprise level for use by multiple acquisition programs. The platform can include an IT environment for development, integration, test, and production/operation, as well as software developer tools and resources that teams can leverage. By using common platforms, programs can focus their energy on the core capabilities unique to their users' requirements, avoid building duplicative, closed infrastructures, and reduce time, cost, and risk to the program.

Technology development and competitive prototyping are continuous activities that span the entire program lifecycle. The program could establish a technology development environment, likely a government – contractor partnership, with a consistent budget to allow for continual evaluation of COTS/GOTS products and maturation of technologies. Future releases could then quickly leverage mature technologies and integrate existing products previously evaluated in a development environment. Meanwhile, maturing technologies external to the time-boxed development releases allows for greater schedule and resource flexibility. Maintaining a continual development environment

enables innovations to emerge from the team or from external sources from government R&D organizations to small businesses.

This approach and the accompanying suggestions represent just a few of many potential structures a program adopting Agile could consider. Throughout acquisition policies and statements, OSD encourages program managers to tailor their programs to best deliver capabilities to the users. Each program should tailor its structures to best fit its environment, the needs of its stakeholders, and the leadership direction. Structures vary based on program scope, risk, technology maturity, complexity, integration, operational urgency, costs and budgets, contractors, government resources, and other factors. As the program is established, the team and stakeholders must ask questions such as those listed below.

Key Questions:

- How is the program broken up into releases and sprints (or related terms)?
- How frequently is capability delivered to users in the operational environment?
- How frequently do contractors demonstrate and deliver capabilities to the government?
- What factors were considered in determining these timelines? (E.g. user demand, integration risks, contracting, technical maturity, training and documentation)
- Do stakeholders agree with the release tempo?

16 Scaling Agile

While Agile works best with small, self-organized, co-located teams, some mid-to-large programs will apply Agile using multiple teams, parallel developments, and multiple contractors. Larger programs may have to use more of a hybrid approach with traditional development methods. Adapting Agile practices to larger projects requires sound engineering discipline to ensure successful integration of multiple smaller development efforts to support the objectives of the larger project. Simple designs, architectures, processes, and requirements enable multiple teams to achieve epic goals.

A program with multiple development teams requires coordination across teams beyond the daily team meetings. This could occur by having the scrum masters of each team meet daily, or as needed based on the level of integration, otherwise referred to as a scrum-of-scrums. Other functional experts will likely need to meet across teams to coordinate on architecture, testing, costs, resources, performance, and other integration touch points. This, in turn, may require staff at a level above the teams to facilitate coordination and integration and take responsibility for enterprise designs, architectures, processes, metrics, and artifacts. The project team must tailor each of these engineering efforts to ensure appropriate use of rigorous methods without introducing heavyweight processes that would negate the benefits of an Agile approach.

Large programs must have clear structure that defines the mission and business environment to guide the partitioning of the larger scope into development efforts of 12 months or less and possibly into multiple parallel development efforts. As previously noted, a technical architecture should frame the

separate development efforts and ensure that individual products can operate in the target environment. Agile emphasizes speed and responsiveness to changing user needs over highly detailed up-front definition of the system architecture. Therefore each program must include periodic evaluation of the evolving technical architecture to ensure that the overall system continues to reflect sound engineering principles such as extensibility, supportability, and scalability.

Integration of multiple, smaller development efforts requires a disciplined approach that begins with high-level "roadmaps" describing the planned evolution of the larger system. As individual development efforts complete their tasks, testing must ensure that the separate components align with the roadmap and conform to the overarching technical architecture. The following items represent additional risk areas related to managing large-scale, multifaceted IT programs using an Agile methodology:

- **Cost Estimation.** When multiple teams work in parallel to complete an effort, estimating costs can become complicated, as the teams estimate and accomplish their tasks at different rates. Program managers must understand the estimation processes the teams use and how they relate to the overall cost of the program.
- **Architectures.** Although the Agile approach centers on delivering capabilities rapidly, program managers cannot ignore significant underlying architectural requirements. Short-term planning to meet iterative capabilities can result in tightly coupled architectures that impose heavy costs if the program must add new capabilities down the line. Especially when a project involves cross-cutting requirements such as security, performance, and availability, program managers must ensure that the developer devotes time early in the project to designing an infrastructure that can support iterative development and overarching quality attributes.
- **Communication.** Agile practices require close and constant communication among all stakeholders. If multiple teams are working on a project, the amount of communication needed increases according to the formula $n(n–1)/2$, where n is the number of teams. Program managers must consider how to expedite communications, especially when the teams are geographically dispersed.
- **Software Code Integration.** Continuous integration involves frequent end-to-end builds of the changing code base, which becomes especially critical when the software development effort increases in scope and requires multiple systems to interact in order to meet the end user's needs. Program managers should ensure their programs use the appropriate development and test tool environments, version control, and change management mechanisms to incorporate continuous integration into the development effort.
- **Testing.** Agile emphasizes the importance of performing tests early in the software development life cycle and testing the capabilities at each release. In addition to unit and acceptance testing, regression testing is critical to delivering shorter and more frequent iterations. As each change is introduced, programs should perform regression testing to ensure the integrity of the overall system.
- **Requirements Derivation.** Larger scale programs naturally include many requirements that may become backlogged. Program managers and product owners should define implied requirements, prioritize quality and architecturally significant attributes, and place high priority

on requirements that support end-to-end capabilities. They should then review the activities backlog periodically and ensure that items are addressed in priority order.

One resource to explore is the Scaled Agile Framework in Figure 22 developed by Dean Leffingwell.

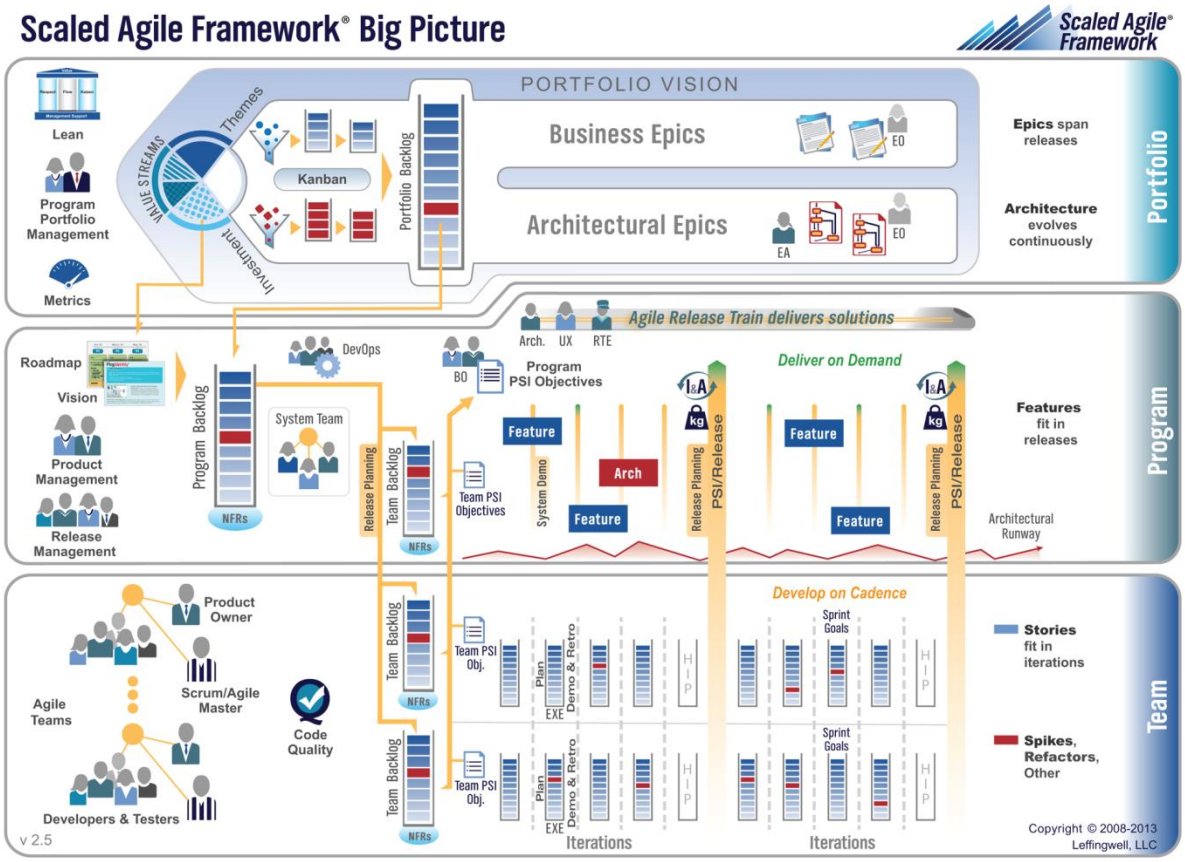


Figure 22 - Scaled Agile Framework

See also:

- Agile Architecture: Strategies for Scaling Agile Development by Scott Ambler
- Scaling Agile Methods by Donald Reifer, Frank Maurer, and Hakan Erdogmus, IEEE, 2003
- Scaling Software Agility: Best Practices for Large Enterprises by Dean Leffingwell
- Five Success Factors for Scaling Agile by Robert Holler and Ian Culling

17 Summary

The DoD can benefit greatly by adopting many Agile development practices. While Agile is not a panacea to address all the IT acquisition challenges, it provides a set of core principles to shape the DoD culture away from the large monolithic systems for IT. Tailoring program structures and processes around small, frequent releases can reduce program risk and be responsive to change. Close user involvement throughout development ensures that each release is addressing the highest operational needs. With the rapid pace of change in IT, the DoD needs the processes and practices of Agile to integrate the latest technological advancements to address an increasingly complex operational environment.

This guide was created to think through the major defense acquisition policies and processes to help program offices and PEOs adopt the benefits of Agile development. While the commercial world has achieved many successes with Agile, federal agencies, and in particular the DoD, are still in the early stages of applying Agile development practices. The government has quite a different set of challenges than the commercial sector. For example, the nature of the government-contractor relationship, as well as other unique federal policies and processes make it difficult to replicate a pure commercial Agile environment. However, these challenges should not dissuade a program from considering an Agile strategy. Agile has the potential to dramatically change how the DoD delivers IT capabilities, when it is used in the right environment, with active stakeholder support, and strong government contractor relationship. It will take time to effectively integrate Agile in the DoD IT acquisition environment. It will take strong leadership to champion the cultural changes needed to enable Agile practices, active support and collaboration across all the major acquisition disciplines, and a strong program management office that is empowered to break from traditional practices.

Appendix A: Agile Resources

Agile Papers, Reports, and Briefings

- MITRE 2011 Handbook for Implementing Agile in DoD IT
- Agile Acquisition briefing to NATO NCIA GM by Pete Modigliani, MITRE
- Considerations for Using Agile in DoD Acquisitions, Software Engineering Institute (SEI)
- Agile Methods: Selected DoD Management and Acquisition Concerns, SEI
- Towards a More Agile Gov't: The Case for Rebooting Federal IT Procurement by Benjamin Balter
- GAO Report 12-681 Effective Practices and Federal Challenges in Applying Agile Methods
- DoD Agile Adoption by Mary Ann Lapham, SEI
- Parallel Worlds: Agile and Waterfall Differences and Similarities, SEI
- More on 804, and Really, Why by James Boston, Defense AT&L Magazine
- The Challenges of Being Agile in DoD by William Broadus, Defense AT&L Magazine

Agile Methods

- Scrum
- Extreme Programming (XP)
- Kanban
- Feature Driven Development (FDD)
- Dynamic Systems Development Method (DSDM)
- Lean software development
- Scrum-ban
- Test Driven Development

Agile Websites

- Agile Manifesto
- Scaled Agile Framework
- Agile Connection

Agile Blogs

- Disciplined Agile Delivery blog by Scott Ambler and Mark Lines
- Agile Management by Version One
- Scott Ambler's IBM Blog
- Leading Answers Blog
- Effective Practices for Software Solution Delivery by Scott Ambler
- Agile Mistakes to Avoid

Agile Books

- Agile Software Development: Best Practices for Large Software Development Projects by Thomas Stober, Uwe Hansmann
- Agile Project Management: Creating Innovative Products by Jim Highsmith
- Scaling Software Agility: Best Practices for Large Enterprises by Dean Leffingwell
- Succeeding with Agile: Software Development Using Scrum by Mike Cohn
- Agile Software Requirements: Lean Requirements Practices for Teams, Programs, and the Enterprise by Dean Leffingwell
- The Human Side of Agile - How to Help Your Team Deliver by Gil Broza
- Disciplined Agile Delivery: A Practitioner's Guide to Agile Software Delivery in the Enterprise by Scott W. Ambler and Mark Lines
- Changing Software Development: Learning to Become Agile by Allan Kelly

Agile Organizations

- Agile Alliance
- PMI Agile
- ADAPT
- AgileDC

Acquisition References

- Interim DoD Instruction (DoDI) 5000.02, 25 Nov 13
- DoD Directive (DODD) 5000.1
- CJCSI 3170.01H
- JCIDS Manual
- Defense Acquisition Guidebook
- Acquisition Community Connection
- Defense Acquisition Portal
- Better Buying Power

Appendix B: GAO Report on Agile

GAO Report 12-681 Effective Practices and Federal Challenges in Applying Agile Methods

The GAO identified 32 practices and approaches as effective for applying Agile software development methods to IT projects. The practices generally align with five key project management activities related to software development: strategic planning, organizational commitment and collaboration, preparation, execution, and evaluation. Officials who have used Agile methods on federal projects generally agreed that these practices are effective. Specifically, officials from at least one agency found each practice effective, and officials from all five agencies found 10 of those practices effective.

The 10 best practices are:

1. Start with Agile guidance and an Agile adoption strategy.
2. Enhance migration to Agile concepts using Agile terms, such as user stories (used to convey requirements), and Agile examples, such as demonstrating how to write a user story.
3. Continuously improve Agile adoption at both the project level and organization level.
4. Seek to identify and address impediments at the organization and project levels.
5. Obtain stakeholder/customer feedback frequently.
6. Empower small, cross-functional teams.
7. Include requirements related to security and progress monitoring in your queue of unfinished work (the backlog).
8. Gain trust by demonstrating value at the end of each iteration.
9. Track progress using tools and metrics.
10. Track progress daily and visibly.

GAO identified **14 challenges** with adapting and applying Agile in the federal environment:

1. Teams had difficulty collaborating closely.
2. Procurement practices may not support Agile projects.
3. Teams had difficulty transitioning to self-directed work.
4. Customers did not trust iterative solutions.
5. Staff had difficulty committing to more timely and frequent input.
6. Teams had difficulty managing iterative requirements.
7. Agencies had trouble committing staff.
8. Compliance reviews were difficult to execute within an iteration time frame.
9. Timely adoption of new tools was difficult.
10. Federal reporting practices do not align with Agile.
11. Technical environments were difficult to establish and maintain.
12. Traditional artifact reviews do not align with Agile.
13. Agile guidance was not clear.
14. Traditional status tracking does not align with Agile.

Appendix C: Agile Roles and Responsibilities

Program Manager

The program manager identifies and sets the vision, roadmap, requirements, and funding for the overall program, and guides the program toward an iterative and Agile approach that supports the frequent delivery of capabilities through multiple, more frequent releases. The program manager retains the responsibility for managing the requirements, funding, and acquisition processes while also overseeing the planning of each release and high-level program increment. In addition, the program manager approves the results of each release.

As user participation is a critical element of the Agile process, the program manager must manage the overall relationship with the user community. In addition, the program manager must collaborate regularly with the project manager and end users to define and prioritize requirements and plan capability releases, and must coordinate with the contracting officer and cost estimator to set up a business environment to support an Agile development strategy.

Project Manager

The project manager organizes the development process in terms of time-boxed iterations that lead to a release of a capability. The project manager primarily facilitates and coordinates participation by the end users and the development team and leads iteration and release reviews. The project manager engages actively with the architecture owner and systems engineer to define and prioritize the project requirements, validate the design, and plan the iterations. In addition, the project manager maintains the list of prioritized requirements and ensures that the defined and derived requirements address the user's most important operational needs. The project manager, acting as the Contracting Officer's Technical Representative, monitors the performance of each iteration-level contract or order and signs off on the results of each iteration.

Product Owner

A product owner is typically a representative of the operational community who is responsible for managing requirements, tradeoffs, and collaboration between the acquisition and user communities. The product owner manages requirements, typically via a series of backlogs, coordinating the priorities with appropriate user, technical, and other stakeholders. The product owner conveys the CONOPS to the release and development teams to ensure a common understanding, provides feedback on interim developments, and coordinates demonstrations and feedback with a broad user base. Ideally, the product owner should have recent practical experience at operational levels and should maintain regular formal and informal contact with the primary user community. While co-location with the program office or developer is ideal, the product owner should maintain frequent (e.g., daily) communication with the release team.

End Users or End User Representatives

End users work closely with the Agile team to convey operational concepts and requirements/needs and provide feedback on developed capabilities. They participate regularly in team meetings and in iteration and release reviews and actively collaborate with the development team, particularly during continuous testing activities and post-development limited assessments and acceptance testing.

In situations where no primary users are available to engage with the Agile team on a regular, ongoing basis, the primary users can designate representatives to advocate their values and needs. These representatives are empowered to speak on behalf of the user community in prioritizing requirements for each release and conveying program progress and issues. Ideally, these representatives should have recent practical experience at operational levels and should maintain regular formal and informal contact with the primary user community. For best results in their capability development role, the representatives should rotate between operational assignments to maintain a relevant experience base.

Architecture Owner

The architecture owner is a government employee and an integral part of the development team. The architecture owner creates architectures and designs in an iterative manner to ensure that designs evolve over the course of the releases, participates in iteration and release reviews to ensure the development complies with the design, and monitors implementation of the design. As issues arise, the owner modifies the architecture.

Independent Tester(s)

The independent tester/team validates the capabilities being produced against the end user's top-priority needs, the design specifications, and standards. The tester/team collaborates with the architecture owner, system engineer, and project manager to understand the requirements and the design, iteration, and release goals and to ensure traceability to the test cases and results. The tester/team also regularly participates in meetings and reviews with the development team and end users. In addition to incorporating testers into the development team, the government should assign an independent tester or test team to execute acceptance tests at the end of each iteration and release.

Systems Engineer

The systems engineer manages the releases; oversees systems implementations, O&M, and transition; and integrates all the engineering disciplines and specialty groups into a team effort to form a structured development process. They collaborate regularly with the project manager and architecture owner to design enterprise and program-specific solutions, manage integration, and identify/prioritize future requirements. They are the architecture owner's primary counterpart on the development team and works with the architecture owner to ensure that the technical baseline integrates architectural artifacts and that architectural and design dependencies are managed across the enterprise. Because Agile systems deliver smaller capabilities more frequently, the systems engineer has an increased workload in managing interdependencies, COTS/GOTS capabilities, and technical baselines.

Contracting Officer

The contracting officer performs overall management of the solicitation, award, and execution of Agile development contract(s). The contracting officer (or contract specialist at a minimum) should be assigned as a dedicated member of the Agile team. The contracting officer works with the program manager to develop the requirements for the Agile development contract(s) and organizes the contractual relationship(s) with the development team. The contracting officer will need to actively manage the Agile contract(s), especially if using a services contract to ensure performance based requirements are being met, and the labor hours and labor categories are appropriate for the contract.

Cost Estimator

The cost estimator tracks and manages the overall program budget and provides rough cost estimates at the program level for high-level increments, followed by detailed cost estimates prior to the development of each iteration. The cost estimator works with the program manager to establish the overall budget and strategy for high-level releases, and works with the project manager and end users to plan and prioritize requirements for future iterations. The cost estimating process also helps to inform decision making on the prioritization of the requirements list. Assigning a cost estimator as a dedicated member of the Agile team provides the continuity needed to keep pace with the frequency of Agile releases.

Appendix D: DoD 5000 Information Requirements

The following tables contain the information requirements for IT systems as referenced in the Interim DoDI 5000.02 Table 2 Milestone and Phase Information Requirements, dated November 26, 2013. The table identifies the applicability of the information requirement for Major Automated Information Systems (MAIS) and Acquisition Category (ACAT) III programs. *Agency-specific policies should be consulted for programs below the ACAT Level III thresholds for applicability*.

Table 5 identifies a core set of acquisition documents prepared by the program. In most cases, these are considered stand-alone documents.

Table 5 Key Acquisition Documents

Document	Applicability
Acquisition Program Baseline	MAIS & ACAT III
Acquisition Strategy	MAIS & ACAT III
Analysis of Alternatives (AoA)	MAIS & ACAT III
Clinger-Cohen Act Compliance	MAIS & ACAT III
Cost Analysis Requirements Description	MAIS
Frequency Allocation Application (DD Form 1494)	MAIS & ACAT III Required only for systems that use the electromagnetic spectrum)
General Equipment Valuation	MAIS & ACAT III (Required only when a deliverable end item meets the requirements for capitalization)
Lifecycle Sustainment Plan (LCSP)	MAIS & ACAT III
Post Implementation Review	MAIS & ACAT III
Post-System Functional Review Report	MAIS & ACAT III (Required for space programs only)
Request for Proposal	MAIS & ACAT III
System Threat Assessment Report	MAIS & ACAT III
Systems Engineering Plan (SEP)	MAIS & ACAT III
Test and Evaluation Master Plan (TEMP)	MAIS & ACAT III

Table 6 identifies the documents required specifically for Defense Business Systems (DBS).

Table 6 DBS Required Documents

Document	Applicability
Business Case	MAIS & ACAT III
Independent Risk Assessment	MAIS (Required only for DBS when directed)
Problem Statement	MAIS & ACAT III
Program Certification to the Defense Business Systems Management Committee	MAIS & ACAT III
Program Charter	MAIS & ACAT III

Table 7 identifies the information requirements that can be combined with one of the above key documents to streamline acquisition documentation.

Table 7 DoD 5000.02 Information Requirements for Consolidated Documentation

Information Requirement	Applicability	Notes
Affordability Analysis	MAIS & ACAT III	Address in Acquisition Strategy (Business Case for DBS)
Bandwidth Requirements Review	MAIS	Documented in the Information Support Plan (part of the Acquisition Strategy)
Benefit Analysis and Determination	MAIS & ACAT III (Applies to bundled acquisition only)	Address in Acquisition Strategy (Business Case for DBS)
Business Process Reengineering	MAIS & ACAT III	Address in Acquisition Strategy and LCSP (Business Case for DBS)
Capstone Threat Assessment	MAIS & ACAT III	Address in Acquisition Strategy (Business Case for DBS)
Consideration of the Technology Issues	MAIS	Address in Acquisition Strategy and LCSP (Business Case for DBS)
Cooperative Opportunities	MAIS & ACAT III	Address in Acquisition Strategy (Business Case for DBS)
Corrosion Prevention Control Plan	MAIS (Only required if the system includes mission critical hardware that will be operated in a corrosive environment.	Address in Acquisition Strategy and LCSP (Business Case for DBS)
Cybersecurity Strategy	MAIS & ACAT III	Included as an appendix to the PPP (addressed in LCSP) or in the Business Case for DBS.
DoD Component Cost Estimate	MAIS & ACAT III	Address in AoA (Business Case for DBS).
Economic Analysis	MAIS	Address in AoA (Business Case for DBS).
Industrial Base Capabilities Considerations	MAIS & ACAT III	Address in Acquisition Strategy (Business Case for DBS)
Information Support Plan	MAIS & ACAT III	Address in Acquisition Strategy and LCSP (Business Case for DBS)
Initial Threat Environment Assessment	MAIS & ACAT III (Required for MAIS, optional for all other programs)	Address in Acquisition Strategy (Business Case for DBS)
Intellectual Property Strategy	MAIS & ACAT III	Address in Acquisition Strategy and LCSP (Business Case for DBS)
Item Unique Identification Implementation Plan	MAIS & ACAT III	Address in the SEP.
Lifecycle Mission Data Plan	MAIS & ACAT III (Only required if	Address in Acquisition Strategy

	dependent on Intelligence Mission Data)	and LCSP (Business Case for DBS)
Market Research	MAIS & ACAT III	Address in Acquisition Strategy and LCSP (Business Case for DBS))
Operational Test Plan	MAIS & ACAT III	Recommend combining with TEMP.
Orbital Debris Mitigation Risk Report	MAIS & ACAT III (Required for space programs only)	Recommend combining with Post-System Functional Review Report.
PESHE and NEPA/E.O 12114 Compliance Schedule	MAIS & ACAT III (Not required for software programs with no hardware component)	Address in SEP and LCSP.
Program Protection Plan (PPP)	MAIS & ACAT III	Address in Acquisition Strategy and LCSP (Business Case for DBS)
Should-Cost Target	MAIS & ACAT III	Address in Acquisition Strategy (Business Case for DBS)
Small Business Innovation Research / Small Business Technology Transfer Program Technologies	MAIS & ACAT III	Address in Acquisition Strategy (Business Case for DBS)
Spectrum Supportability Risk Assessment	MAIS & ACAT III (Required for systems that use the electromagnetic spectrum only)	Address in Acquisition Strategy and LCSP (Business Case for DBS)
Technology Targeting Risk Assessment	MAIS & ACAT III	Part of the PPP, address in the LCSP (Business Case for DBS).

Table 8 identifies the information requirements that will be provided to the program office.

Table 8 Information Requirements Provided to the Program

Information Requirement	Applicability	Notes
Acquisition Decision Memorandum	MAIS & ACAT III	Prepared by the Milestone Decision Authority (MDA).
AoA Study Guidance and AoA Study Plan	MAIS & ACAT III	Prepared by the Lead DoD Component.
Capability Development Document	MAIS & ACAT III	Prepared by requirements organization.
Capability Production Document	MAIS & ACAT III	Prepared by requirements organization.
Development RFP Release Cost Assessment	MAIS	Prepared by Director Cost Accounting and Program Evaluation (DCAPE).

DoD Component Cost Position	MAIS	Prepared by the DoD Component and the Service Cost Agency.
DOT&E Report on Initial Operational Test and Evaluation	MAIS & ACAT III (Required for DOT&E Oversight List programs only)	Prepared by DOT&E.
Exit Criteria	MAIS & ACAT III	Prepared by the MDA.
Full Funding Certification	MAIS	Prepared by the Component Acquisition Executive and the DoD Component Chief Financial Officer
Initial Capabilities Document	MAIS & ACAT III	Prepared by requirements organization.
Independent Cost Estimate	MAIS (Only required for MAIS in advance of a report of a Critical Change)	Prepared by DCAPE
Information Technology and National Security System Interoperability Certification	MAIS & ACAT III	Prepared by Joint Interoperability Test Command or DoD Components
Operational Test Agency Report of OT&E Results	MAIS & ACAT III	Prepared by Operational Test Agency
Operational Mode Summary/Mission Profile	MAIS & ACAT III	Prepared by the DoD Component combat developer

Appendix E: Acronyms

AoA	Analysis of Alternatives
ACAT	Acquisition Category
BCL	Business Capability Lifecycle
BPA	Blanket Purchase Agreement
CD	Capability Drop
CDD	Capability Development Document
CONOPS	Concept of Operations
COTS	Commercial off-the-Shelf
CPD	Capability Production Document
DCAPE	Director, Cost and Program Evaluation
DoD	Department of Defense
ECP	Engineering Change Proposal
FFP	Firm Fixed Price
GOTS	Government off-the-Shelf
IDIQ	Indefinite Delivery Indefinite Quantity
IGCE	Independent Government Cost Estimate
IS	Information Systems
JCIDS	Joint Capability Integration and Development System
JROC	Joint Requirements Oversight Council
LCSP	Lifecycle Support Plan
MAIS	Major Automated Information System
MDA	Milestone Decision Authority
MDD	Materiel Development Decision
MOE	Measure of Effectiveness
OSD	Office of the Secretary of Defense
OT&E	Operational Test and Evaluation
PEO	Program Executive Office
PMO	Program Management Office
PPP	Program Protection Plan
PWS	Performance Work Statement
RDP	Requirements Definition Package
SAE	Service Acquisition Executive
SEP	Systems Engineering Plan
TEMP	Test and Evaluation Master Plan
T&M	Time and Material

© 2014 The MITRE Corporation. All rights reserved.