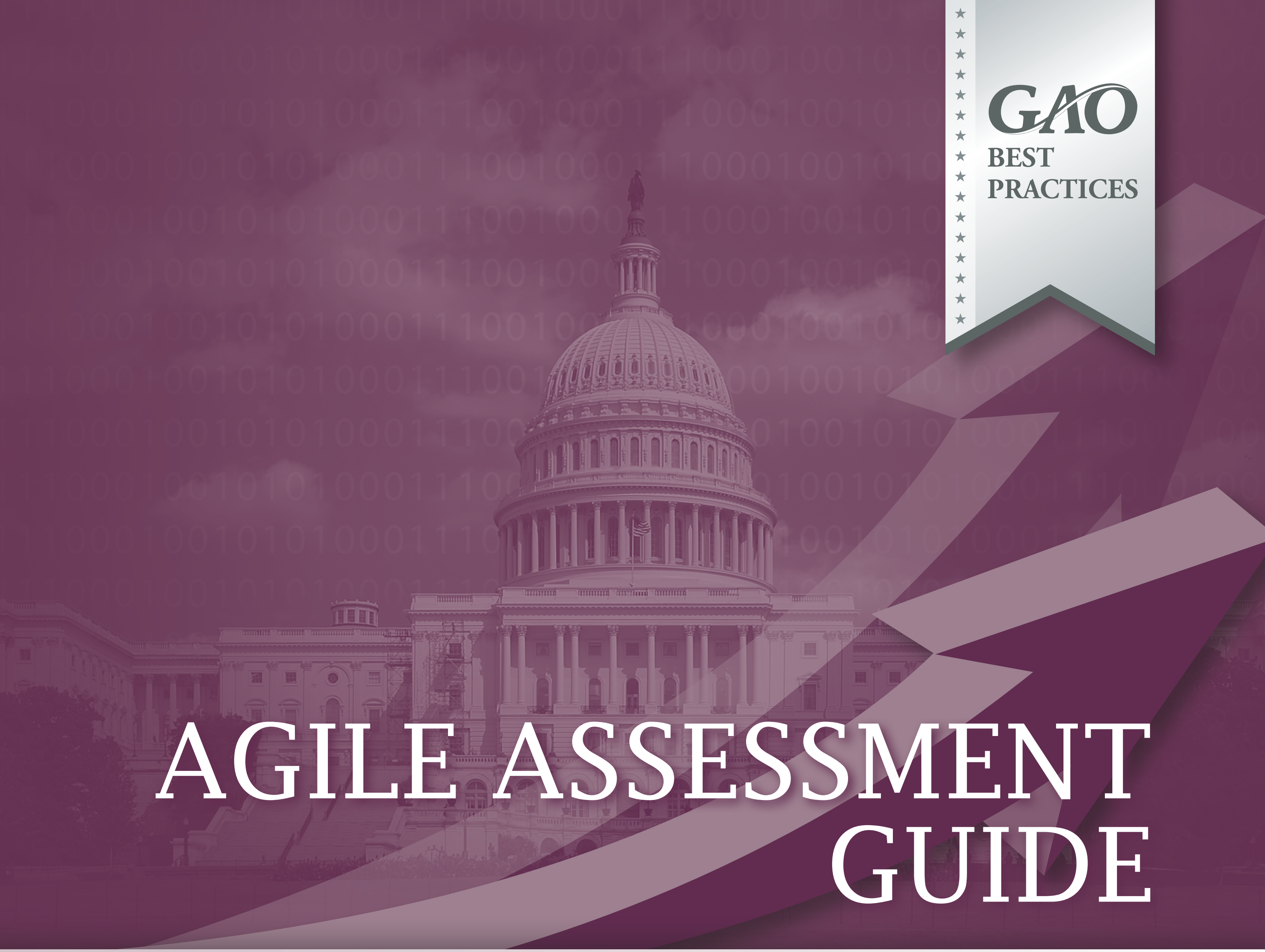
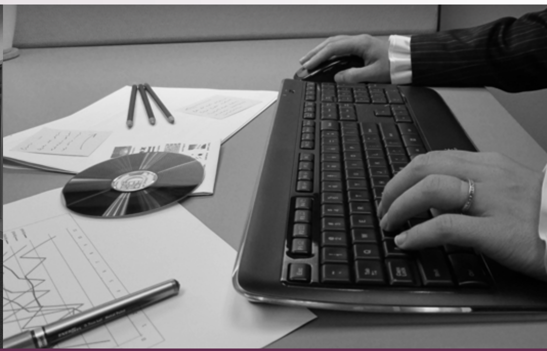


GAO
BEST PRACTICES

AGILE ASSESSMENT GUIDE

Best Practices for Agile Adoption and Implementation



GAO-20-590G
September 2020

From September 28, 2020 to September 27, 2021, GAO is seeking input and feedback on this Exposure Draft from all interested parties. See page 3 for more information.

GAO
U.S. GOVERNMENT
ACCOUNTABILITY OFFICE

Contents

Abbreviations

ALIS	Automated Logistics Information System
COR	contracting officer's representative
CPIC	Capital Planning and Investment Control
DHS	Department of Homeland Security
DevOps	Development and Operations
DAD	Disciplined Agile Delivery
DOD	Department of Defense
DSDM	Dynamic Systems Development Method
EVM	earned value management
ELIS	Electronic Immigration System program
FAR	*Federal Acquisition Regulation*
FDD	Feature Driven Development
FEMA	Federal Emergency Management Agency
FITARA	*Federal Information Technology Acquisition Reform Act*
FOC	full operational capability
G2	Generation 2 (NNSA Program Management Information Systems)
GSA	General Services Administration
GMM	Grants Management Modernization

Abbreviations Continued

ICE	U.S. Immigration and Customs Enforcement
INVEST	Independent, Negotiable, Valuable, Estimable, Small, Testable
LeSS	Large Scale Scrum
MVP	Minimum viable product
MUOS	Mobile User Objective System program
MoSCoW	Must have, should have, could have, won’t have (sometimes *would have*)
NDAA	*National Defense Authorization Act*
NNSA	National Nuclear Security Administration
OMB	Office of Management and Budget
OTC	Office of Transformation Coordination
RFP	request for proposal
SAFe	Scaled Agile Framework
SEVIS	Student and Exchange Visitor Information System program
SOO	Statement of Objectives
SOW	Statement of Work
Space C2	Air Force Space Command and Control program
TIM	Technology Infrastructure Modernization
TSA	Transportation Security Administration
USCIS	U.S. Citizenship and Immigration Services
USDS	U.S. Digital Services
WBS	work breakdown structure
XP	eXtreme Programming

U.S. GOVERNMENT ACCOUNTABILITY OFFICE

441 G St. N.W.
Washington, DC 20548

Preface

The U.S. Government Accountability Office (GAO) is responsible for, among other things, assisting Congress in its oversight of the executive branch, including assessing federal agencies' management of information technology (IT) systems. In prior audits, GAO has reported that federal agencies faced challenges in developing, implementing, and maintaining their IT investments. All too frequently, agency IT programs have incurred cost overruns and schedule slippages while contributing little to mission-related outcomes. Accordingly, in February 2015, GAO added the government's management of IT acquisitions and operations on its list of high-risk programs.[1]

Recognizing the severity of issues related to government-wide management of IT, in 2014, the Congress passed and the President signed federal IT acquisition reform legislation, commonly referred to as the *Federal Information Technology Acquisition Reform Act*, or FITARA.[2] This legislation was enacted to improve agencies' acquisitions of IT and enable Congress to monitor agencies' progress and hold them accountable for reducing duplication and achieving cost savings. Among its specific provisions is a requirement for Chief Information Officers (CIOs) at covered agencies to certify that certain IT investments are adequately implementing incremental development as defined in the Office of Management and Budget's (OMB) capital planning guidance. OMB's implementing guidance requires agencies to use incremental development approaches that would deliver enhanced or new functionality to users at least every six months.[3]

[1]GAO, *High Risk Series: An Update,* GAO-15-290 (Washington, D.C.: Feb. 11, 2015). Some examples of GAO reports showing the struggles of federal agencies in implementing IT systems include: GAO, *Software Development: Effective Practices and Federal Challenges in Applying Agile Methods,* GAO-12-681 (Washington, D.C.: July 27, 2012); *Information Technology: OMB and Agencies Need to More Effectively Implement Major Initiatives to Save Billions of Dollars,* GAO-13-796T (Washington, D.C.: July 25, 2013); *TSA Modernization: Use of Sound Program Management and Oversight Practices is Needed to Avoid Repeating Past Problems,* GAO-18-46 (Washington, D.C.: October 17, 2017); and *FEMA Grants Modernization: Improvements Needed to Strengthen Program Management and Cybersecurity,* GAO-19-164 (Washington, D.C.: (April 9, 2019).

[2]Carl Levin and Howard P. 'Buck' McKeon National Defense Authorization Act for Fiscal Year 2015, Pub. L. No. 113-291, div. A, tit. VIII, subtit. D, 128 Stat. 3292, 3438-50 (2014).

[3]*Office of Management and* Budget, Memorandum M15-14, *Management and Oversight of Federal Information Technology*, (June 10, 2015), at 18.

One framework for incremental development is Agile software development, which has been adopted by many federal agencies. It emphasizes early and continuous software delivery and is defined by values and principles that can be realized through a set of common practices seen in specific Agile frameworks, such as DevOps, eXtreme Programming, Lean, Kanban, Scrum, and others.[4]

This guide has been developed with the assistance of many knowledgeable specialists in the field of Agile and other incremental software development methods to aid federal agencies, departments, and auditors in assessing an organization's readiness to adopt Agile methods.[5]

The best practices in this guide are not comprehensive; that is, they are presented as high-level concepts of software development, contracting, and program management that highlight the aspects of Agile development throughout a program's life cycle and address key risks to an organization, program, or team without prescriptive "how to" steps. Many other publications address how to apply best practices in using an incremental approach to software development and readers can refer to those sources when considering a specific development topic.

GAO, the Congressional Budget Office (CBO) and other organizations have developed projections that show the nation's fiscal path is unsustainable. New resource demands and demographic trends will place serious budgetary pressures on federal discretionary spending, as well as other federal policies and programs in the coming years. When resources are scarce, competition for those resources increases. It is imperative, therefore, that government programs deliver the promised results, not

[4]Agile frameworks are also used to develop hardware programs and manage services. The best practices in this guide are intended to be at a high enough level to be used for any Incremental development program, regardless what type of product or service is being delivered. However, the focus of this guide will be how Agile frameworks are used in software development.

[5]Agile is the name we use to describe incremental software development methods in this guide, with concepts from Lean, Kanban, DevOps, or other more specific methods. For example, Kanban may not be considered an Agile software development methodology, but it may be considered a management method used to improve the flexibility of the activities of knowledge workers during software development. An organization that intends to adopt a specific Agile method should supplement guidance described later in this guide with additional materials that specifically address the practical application of that specific method.

only because of their value to the public, but also because every dollar spent on one program is one less dollar available to fund other efforts.

GAO plans to periodically update this exposure draft based on users' experience and comments. Comments and suggestions from experienced users and knowledgeable specialists regarding the application of Agile principles are welcome. Please click on this link https://tell.gao.gov/agileguide to provide comments on this guide.

If you have any questions concerning this guide, contact Tim Persons at (202) 512-6888 or personst@gao.gov or Carol Harris at (202) 512-4456 or harriscc@gao.gov. Major contributors to this guide are listed in appendix IX and contact points for our Offices of Congressional Relations and Public Affairs are located at the end of this document.

T. M. Persons

Timothy M. Persons, Ph.D.
Chief Scientist and Managing Director
Science, Technology Assessment, and Analytics Team

[illegible]

Carol Harris
Director
Information Technology and Cybersecurity Team

Introduction

The federal government spends at least $90 billion annually on information technology (IT) investments. In our January 2019 High Risk List report, GAO reported on 35 high risk areas, including the management of IT acquisitions and operations.[6] While the executive branch has undertaken numerous initiatives to help agencies better manage their IT investments, these programs frequently fail or incur cost overruns and schedule slippages while contributing little to mission-related outcomes. GAO has found that OMB continues to demonstrate its leadership commitment by issuing guidance for covered departments and agencies to implement statutory provisions commonly referred to as FITARA.[7] However, application of FITARA at federal agencies has not been fully implemented. For example, as we stated in the 2019 High Risk report, none of the 24 major federal agencies had IT management policies that fully addressed the roles of their Chief Information Officers (CIO) consistent with federal laws and guidance.[8]

This Agile Guide is intended to address generally accepted best practices for Agile adoption, execution, and control. In this guide, we use the term best practice to be consistent with the use of the term in GAO's series of best practices guides.[9]

Developing the Guide

Our approach to developing this guide was to ascertain best practices for Agile software development from leading practitioners and to develop standard criteria to determine the extent to which agency software development programs meet these practices. These best practices center

[6]GAO, *High Risk Series: Substantial Efforts Needed to Achieve Greater Progress on High Risk Areas*, GAO-19-157SP (Washington, D.C.: Mar. 6, 2019).

[7]The provisions apply to the agencies covered by the *Chief Financial Officers Act of 1990*, 31 U.S.C. § 901(b). These agencies are the Departments of Agriculture, Commerce, Defense, Education, Energy, Health and Human Services, Homeland Security, Housing and Urban Development, Justice, Labor, State, the Interior, the Treasury, Transportation, and Veterans Affairs; the Environmental Protection Agency, General Services Administration, National Aeronautics and Space Administration, Nation Science Foundation, Nuclear Regulatory Commission, Office of Personnel Management, Small Business Administration, Social Security Administration and the U.S. Agency for International Development. However, FITARA has generally limited application to the Department of Defense.

[8]GAO-19-157SP.

[9]GAO, *GAO Cost Estimating and Assessment Guide: Best Practices for Developing and Managing Program Costs,* GAO-20-195G (Washington, D.C.: Mar. 12, 2020), *GAO Schedule Assessment Guide: Best Practices for Project Schedules,* GAO-16-89G (Washington, D.C.: Dec. 22, 2015) and *GAO Technology Readiness Assessment Guide: Best Practices for Evaluating the Readiness of Technology for Use in Acquisition Programs and Projects,* GAO-20-48G (Washington, D.C.: Jan 7, 2020).

on Agile adoption, execution, and control. We developed each best practice in consultation with a committee of IT and program management specialists and organization executives across government, private industry, and academia. We describe our scope and methodology in detail in appendix I.

The Guide’s Readers

We have developed this guide to serve multiple audiences:

- The primary audience for this guide is federal auditors. Specifically, the guide presents best practices that can be used to assess the extent to which an agency has adopted and implemented Agile methods.
- Organizations and programs that have already established policies and protocols for Agile adoption and execution can use this guide to evaluate their existing approach to Agile software development.
- Organizations and programs that are in the midst of adopting Agile software development practices and programs that are planning to adopt such practices can also use this guide to inform their transitions.

The Guide’s Contents

This guide focuses on best practices surrounding Agile adoption, execution, and controls. For example, chapter 3 groups commonly-recognized best practices for Agile adoption into the areas of team dynamics and activities, program operations, and organization environment. Chapter 4 provides an overview of high-level program management concepts surrounding Agile execution and control best practices, such as requirements development and management, acquisition strategies, and program monitoring and control. Agile execution best practices related to requirements development and management and the federal contracting process are discussed in more detail in chapters 5 and 6, respectively. Program control and monitoring best practices for cost estimating, scheduling, and earned value management are discussed in chapter 7, and best practices for metrics that can be used during the adoption, execution, and monitoring and control periods of the program are discussed in chapter 8.

Certain concepts in the chapters are further explained in the appendixes. Definitions of the key terms and processes discussed throughout this guide are explained in appendix II and related terms that compare terms with similar meanings from different methodologies are displayed in appendix III.

This guide also contains a number of case studies drawn from prior GAO work. The case studies highlight successes and challenges typically associated with Agile adoption and execution in federal settings. These case studies are meant to augment the key points and the lessons learned that each chapter discusses. For example, GAO has found that problems can arise due to the misapplication of Agile software development processes and methods.[10] Similar to the case studies, Agile in Action segments were developed by interviewing agency officials, reviewing documentation, and performing site visits to observe Agile methods in use. To help verify that the information presented in these examples was complete, accurate, and up-to-date, we provided each organization with a draft version of our summary analysis. Appendix VII provides high-level information for each program used in a case study and a summary of the Agile in Action process.

Acknowledgments

The *GAO Agile Assessment Guide* team thanks the many subject matter experts in the federal government, private industry, and academia who helped make this guide a reality. After we discussed our initial plans for developing this guide with GAO's Cost Working Group and at various technical conferences, several members expressed interest in working with us. They formed the initial membership of our Agile working group that convened in August 2016. This number grew as the work developed, and the contributions of all have been invaluable. Thanks to everyone who gave their time and expertise in meetings, provided us with documentation and comments, and hosted us at their facilities as we observed their Agile methods in real time. Contributors of the Agile working group are listed in appendix VIII and GAO staff who contributed to this guide are listed in appendix IX.

[10]For example, in GAO, *Immigration Benefits System: US Citizenship and Immigration Services Can Improve Program Management* (GAO-16-467) we reported that the United States Citizenship and Immigration Service Transformation program was not setting outcomes for Agile software development and in *TSA Modernization: Use of Sound Program Management and Oversight Practices is Needed to Avoid Repeating Past Problems (*GAO-18-46*)* we reported that the Transportation Security Administration's Technology Infrastructure Modernization (TIM) program did not define key Agile roles, prioritize system requirements, or implement automated capabilities.

Chapter 1: Background

The most well-known feature of Agile software development is probably its emphasis on iterative product development and delivery; that is, development of software in iterations that are being continuously evaluated on their functionality, quality, and customer satisfaction.[11] This method is well suited for programs in which the final product is to include distinct features, some of which may be discovered during the process rather than planned at the beginning. Information obtained during these frequent iterations can effectively assist in measuring progress and allowing developers to respond quickly to feedback from customers, thus reducing technical and programmatic risk.[12] With its emphasis on early and continuous delivery of working software, Agile can be a valuable tool for organizations in helping to mitigate schedule and budget risks.

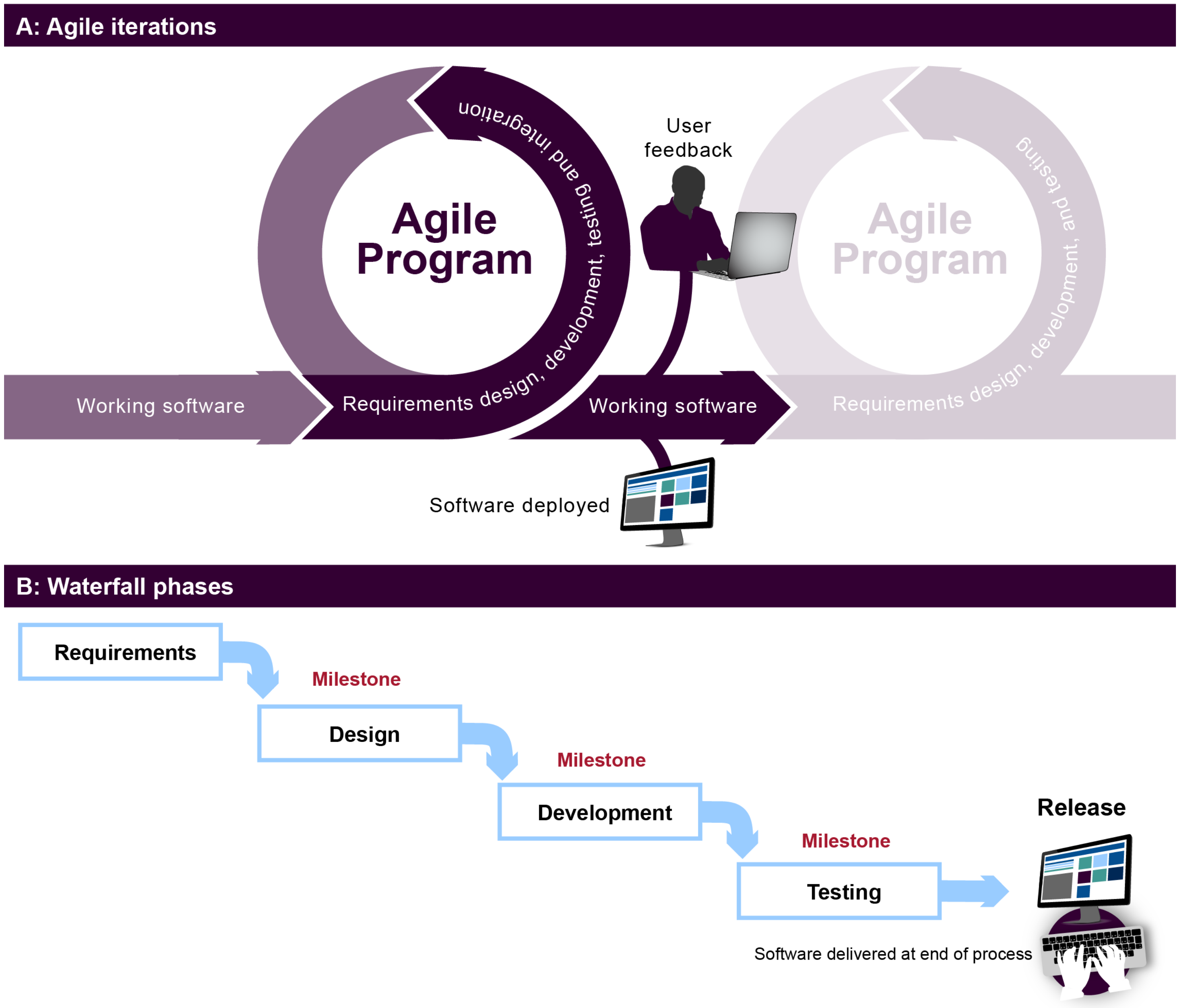
Figure 1 compares requirements, design, development, and testing using Agile software methods to those of the formerly-used Waterfall framework; it illustrates how requirements, design, development, and testing are performed concurrently in small iterations for Agile and sequentially in Waterfall development.[13] In contrast to Waterfall, using an Agile framework can result in an organization producing software using frequent reviews and customer feedback to help ensure that the highest value requirements are being met. Figure 1 compares Agile and Waterfall methods for developing software, assuming that high-level planning for both Agile and Waterfall development has already occurred.

[11]In this guide, an iteration is a predefined, time boxed, recurring period of time in which working software is created. Similarly, a release is defined as a planned segment of requirements that are useable. For more information, see appendix II.

[12]The term ‘customer’ can mean different things depending on the perspective. For example, often a customer refers to the end users of a system but there are also instances where the customer and sponsor are the same individual. The definition of the customer(s) is organizationally and contextually dependent. See appendix II for more information on how we define this term and use it throughout the guide.

[13]A 1970 paper entitled “Managing the Development of Large Software Systems” by Dr. Winston W. Royce is considered by the Software Engineering Institute and others to be the basis for the Waterfall framework. (See Royce, Winston, “Managing the Development of Large Software Systems. Reprinted from proceedings, IEEE WESCOM (August 1970), pages 1-9). Although the paper never uses the term “Waterfall,” the model has sequential phases that flow continuously from one step to the next. While the paper noted that this model is risky because it is unknown how the system will actually work until the testing phase and recommended iterative interaction between steps, it became the foundation for what is known as the Waterfall approach.

Figure 1: Comparison of Agile and Waterfall Methods for Developing Software



Source: GAO analysis of DOD and USCIS Information. | GAO-20-590G

The Value of Using Agile

With an emphasis on the early and continuous delivery of working software, Agile can be a valuable tool for mitigating risks by engaging customers in collaboration early in the program and continuously adapting to changing requirements and environment, thus limiting the chance of continuing to fund a failing program or outdated technology.

While some versions of incremental development were being used as early as the 1950's, the Agile approach was articulated in 2001 by a group of 17 software developers that called themselves the Agile Alliance. In February 2001, the Alliance released "The Agile Manifesto," in which they declared: "We are uncovering better ways of developing software by doing it and helping others do it. Through this work we have come to value:

- Individuals and interactions over processes and tools.
- Working software over comprehensive documentation.
- Customer collaboration over contract negotiation.
- Response to change over following a plan."[14]

The Alliance added that, while they recognized the value in the second part of each statement (e.g., "processes and tools"), they saw more value in the first part (e.g., "individuals and interactions"). The Alliance further delineated their vision with 12 principles. The 12 Agile principles behind the Manifesto are:

1. Our highest priority is to satisfy the customer through early and continuous delivery of valuable software.
2. Welcome changing requirements, even late in development. Agile processes harness change for the customer's competitive advantage.
3. Deliver working software frequently, from a couple of weeks to a couple of months, with a preference to the shorter time scale.
4. Business people and developers must work together daily throughout the project.
5. Build projects around motivated individuals. Give them the environment and support they need, and trust them to get the job done.

[14]©2001-2020 Agile Manifesto authors https://agilemanifesto.org.

6. The most efficient and effective method of conveying information to and within a development team is face-to-face conversation.
7. Working software is the primary measure of progress.
8. Agile processes promote sustainable development. The sponsors, developers, and users should be able to maintain a constant pace indefinitely.
9. Continuous attention to technical excellence and good design enhances agility.
10. Simplicity, the art of maximizing the amount of work not done, is essential.
11. The best architectures, requirements, and designs emerge from self-organizing teams.
12. At regular intervals, the team reflects on how to become more effective, then tunes and adjusts its behavior accordingly.[15]

There are numerous approaches, or frameworks, available for Agile programs to use. A framework is a basic structure to guide customers, rather than a prescriptive process. Each framework is unique and may have its own terminology for processes and artifacts, though the frameworks are not mutually exclusive and so can be combined.[16] When implementing Agile in a federal environment, all staff, both government and contractor, will want to work together to define the Agile terms and processes that will be used for their particular program. Agile, as a concept, is not prescriptive; however, when applied to an organization, it may be. Regardless of the specific frameworks and practices, it is important that applying Agile aligns with the manifesto and Agile principles. A detailed description of commonly-used Agile frameworks is located in appendix V. Table 1 provides a high-level definition for several commonly-used incremental development frameworks.

[15]©2001-2020 Agile Manifesto authors https://agilemanifesto.org.

[16]Some frameworks vary from the Agile Manifesto's values and principles. For example, Kanban must have a customer who has requested a service and an end point where the request is fulfilled and delivered to the customer. In this case, the manifesto's value of "customer collaboration over contract negotiation" applies differently than in a time boxed framework, such as Scrum.

Table 1: Description of Commonly-Used Agile Frameworks

Individual team framework	Description
eXtreme Programming (XP)	XP is a process that originated from taking software best practices to the extreme. XP processes incorporate five key values: 1) communication, 2) feedback, 3) simplicity, 4) courage, and 5) respect. XP values constant communication between customers, developers, user stories, and management as well as having a simple and clean design. Pair programming and 100 percent unit testing are some examples of key XP practices.
Feature Driven Development (FDD)	In FDD, development is driven from the functionality perspective. FDD adheres to the following steps: develop the overall model, build feature list, plan by feature, design by feature, and build by feature. FDD uses a number of best practices, including Domain Object Modeling and Individual Code Ownership.
Kanban	The Kanban framework seeks to limit work in progress in order to alleviate bottlenecks throughout development. Team members “pull” work when they are able to, as opposed to work being “pushed” down to them, to smooth the flow of work and eliminate unevenness. Kanban uses the following practices: visualize the work flow, limit work in progress, manage flow, make policies explicit, implement feedback loops, and improve collaboratively. Kanban’s most prominent feature is a visual task board divided into columns, at a minimum: to-do, in process, and done. Tasks are written on notes and placed on the board, and move horizontally through the columns as the work is completed. As with other team frameworks, electronic means for facilitating flow are available to supplement manual-based visualization.
Scrum	Scrum defines the team by three core roles: product owner, development team, and scrum master. Development is broken down into time boxed iterations called sprints, where teams commit to complete specific requirements. During a sprint, teams meet for daily stand up meetings. At the end of the sprint, teams demonstrate the completed work to the product owner for acceptance. A retrospective meeting is held after the sprint to discuss any changes to the process.
Agile at Scale frameworks[a]	
Disciplined Agile (DA)	Building on different Agile methodologies, DA is a decision framework that can be scaled and is intended to address the whole product life cycle. Key aspects of DA include: people-first, learning-oriented, hybrid methodologies, full delivery life cycle, process goal driven, solution focused, risk/value life cycle, and enterprise aware. DA has defined roles of team members within the framework.
Dynamic Systems Development Method (DSDM)	Previously known as DSDM Atern, this is a framework for rapid development. There are eight principles: 1) Focus on business need, 2) deliver on time, 3) collaborate, 4) never compromise on quality, 5) build incrementally from firm foundations, 6) develop iteratively, 7) communicate continuously and clearly, and 8) demonstrate control. One core technique of DSDM is prioritizing requirements as Must have, Should have, Could have, and Won’t have but would like, or MoSCoW.
LeSS	Large Scale Scrum (LeSS) is a scaled up version of one-team Scrum and it maintains many of the practices and ideas of one-team Scrum. In LeSS you will find: 1) a single prioritized backlog, 2) one definition of done for all teams, 3) one product owner, and 4) many complete, cross-functional teams with no single specialist teams. In LeSS, all teams are in a common iteration to deliver a common, shippable product.

Scaled Agile Framework (SAFe)[b]	SAFe is a framework for implementing Agile at scale. The framework provides guidance for roles, inputs, and processes that can include four configurations (essential, large solution, portfolio, and full), tailored to each unique context. There are ten principles: 1) take an economic view, 2) apply systems thinking, 3) assume variability, 4) build incrementally in cycles, 5) base milestones on evaluation of working systems, 6) visualize and limit work in progress, 7) apply cadence, 8) unlock motivation of workers, 9) decentralize decision making, and 10) organize around value.
Hybrid framework[c]	
Scrumban	A combination of Scrum and Kanban, teams generally abide by Scrum roles while using Kanban to view workload and improve flow. Scrumban can be considered the application of Kanban to a Scrum framework to help an organization tailor its Scrum to better align with their goals. With Scrumban, the amount of work is not limited to the sprint, but to the work in progress limit. Meetings in Scrumban are often scheduled as needed, as opposed to a specific schedule with sprints.
Related frameworks[d]	
Crystal	The Crystal method outlines different methodologies based on the number of people involved and the criticality of the software. The framework that most closely resembles Agile is called Crystal Clear. The methods rely on trust and communication. Unlike other methodologies that dictate discipline to specific practices, Crystal allows freedom for individual preferences and work habits.
DevOps	DevOps, with its name stemming from a combination of development and operations, emphasizes collaboration between development, IT operations, and quality assurance with the goal of more frequent software releases. The overall DevOps values align with Agile, and DevOps is considered an expansion of Agile implementation practices to all areas of a product's life cycle. One common DevOps principle is, "infrastructure as code", meaning operating environments are managed the same as code, with version control, automation, and continuous testing.
Iterative Development	Iterative development breaks down the work into smaller chunks known as iterations, in order to design, develop, and test in cycles.
Lean Software Development	Lean software development applies principles from lean manufacturing to software development. There are seven key principles: 1) eliminate waste, 2) amplify learning, 3) deliver fast, 4) decide late, 5) empower the team, 6) build integrity in, and 7) optimize the whole product.

Source: GAO analysis of information from DHS, DOJ, VersionOne Inc., Scaled Agile Inc., Scrum.org, Booz Allen Hamilton, the DSDM Consortium, and Agile Alliance. | GAO-20-590G

[a]Scaled frameworks are those that are intended to increase Agile processes so that they can be applied to large, complex organizational structures.

[b]The description of SAFe is as of May 2020 and is based on SAFe V5.0.

[c]Hybrid frameworks combine principles and practices from more than one Agile framework.

[d]Related frameworks are those that are very similar to Agile frameworks and often use many of the same principles and practices. Many of these frameworks, such as DevOps, extend Agile principles such as communication to enable additional collaboration.

When selecting a framework, organizations should adopt a deliberative process based on the needs of a given program as well as the culture and structure of the organization. For example, adopting Agile or one of these frameworks might require a dramatic shift in the culture of the organization. This might, in turn, change an organization's structure and result in changes to the physical space used by development teams. A further discussion on Agile adoption best practices for teams, programs, and organizations is included in chapter 3.[17]

[17]For this guide, a program can be defined in various ways for budgeting and policy making purposes. Whether a program is defined as an activity, project, function, or policy, it must have an identifiable purpose or set of objectives if an evaluator is to assess how well its purpose or objectives are met. An evaluation can assess an entire program or focus on an initiative within a program. In the case of IT systems, a single program could be part of a project within a larger program. For that reason, this guide will use the term program; however, that can also refer to a project.

Chapter 2: Agile Adoption Challenges in the Federal Government and Actions Taken in Response

Information systems are integral to many aspects of federal government operations. Congress has expressed long-standing interest in monitoring and improving federal IT investments, which have often been developed in long, sequential phases. Several agencies have tried using an Agile approach, which calls for producing software in small, short increments.

Challenges

In a 2012 report, GAO identified 14 challenges federal agencies reported they encountered while applying Agile methods to an IT software development program.[18] GAO grouped the challenges reported into four areas: organizational commitment and collaboration, preparation, execution, and evaluation. In part, the challenges reported were the result of a cultural or social environment that was not conducive to a successful transition. For example, teams reported difficulty collaborating closely or transitioning to self-directed work due to constraints in organization commitment and collaboration. Moreover, some organizations reported that they did not have trust in iterative solutions and that teams had difficulty managing iterative requirements. Table 2 shows the specific program management activities organized by these four areas and the challenges organizations reported when transitioning to Agile.

Table 2: Iterative Software Challenges, as Reported by Federal Agencies

Program management activity	Challenges
Organizational commitment and collaboration– Actions by management that are necessary to ensure that a process is established and will endure	Teams had difficulty collaborating closely
	Team had difficulty transitioning to self-directed work
	Staff had difficulty committing to more timely and frequent input
	Organizations had trouble committing staff
Preparation– Establish teams and processes prior to implementing Agile for a program	Timely adoption of new tools was difficult
	Technical environments were difficult to establish and maintain
	Agile guidance was not clear
	Procurement practices may not have supported Agile programs
Execution– Establish the concrete steps necessary to conduct the defined Agile approach	Customers did not trust iterative solutions
	Team had difficulty managing iterative requirements
	Compliance reviews were difficult to execute within an iteration time frame
Evaluation– Assess processes to improve the Agile approach	Federal reporting practices did not align with Agile methods
	Traditional artifact reviews did not align with Agile methods
	Traditional status tracking did not align with Agile methods

Source: Summary of GAO-12-681. | GAO-20-590G

[18]GAO, *Software Development: Effective Practices and Federal Challenges in Applying Agile Methods*, GAO-12-681 (Washington, D.C.: Jul. 27, 2012).

Challenges to organization commitment and collaboration

Programs using Agile software development methods require the ongoing collaboration and commitment of a wide array of stakeholders, including business owners, sponsors, users, developers, and cybersecurity specialists.[19] One way Agile promotes commitment and collaboration is by having teams work closely together, in one physical location when possible, to facilitate continuous communication among the team members.

However, officials from the federal agencies that GAO surveyed reported that teams had trouble transitioning to this philosophy. Specifically, they stated that teams were challenged in collaborating because staff were used to working independently and in individual work spaces. For example, some team members preferred to work alone rather than in a team room, viewed open communication (such as posting program status on a team room wall chart) as intrusive, or disliked showing work-in-progress to the federal customer.

Agency officials also reported that staff members struggled in transitioning to self-directed teams. They stated that staff members used to taking direction from a program manager found it hard to take responsibility for their work and then elevate unresolved issues to senior officials. They also reported that cross-functional teams were difficult to form because federal employees tended to be specialists in specific functional areas. For example, a team member who represented specific customers was not always familiar with the needs of all customers.

While Agile stresses frequent input and feedback from all stakeholders, officials noted that federal employees found it difficult to commit to keeping work products, such as schedules, updated to reflect the status of every iteration because they were not used to the rapid pace of development. They also said that teams initially had difficulty maintaining an iteration's pace because they were used to stopping their work to address an issue rather than making a decision and moving on. Officials also stated that federal customers were not always available to review

[19]Each stakeholder will likely have a competing set of priorities and objectives for a system. For example, a sponsor responsible for funding a program may want a particular set of features, while the actual users of the system may want a different set of priorities. It is important to consider all of the views and opinions of stakeholders both up front in planning a program and throughout development. Relying on a sponsor or managers to serve as proxies for actual users presents a risk the same as relying solely on end users or business analysts will present a risk. The product owner, discussed later in this guide, is intended to help coordinate with and filter through these competing priorities while not neglecting the viewpoints of a particular community.

deliverables as they were completed, which was necessary in order to keep the development team on pace with the iteration.

Some agency officials stated that assigning and maintaining staff was an issue because their organization did not have sufficient staff to dedicate to multiple Agile teams. Staff with multiple, concurrent duties could not be spared from their other duties to dedicate themselves to the large time commitment required of an Agile team. Additionally, officials said that frequent work assignment rotations were common in many federal agencies, creating challenges as new staff needed to learn the roles and responsibilities of those being replaced.

Challenges to preparing for Agile adoption

When an organization using Waterfall software development migrates to Agile or other iterative methods, new tools and technical environments may need to be added to support program planning and reporting. However, officials reported that implementing Agile tools could initially create a challenge due to delays in buying, installing, and training staff on the use of these new tools.

Furthermore, officials noted that Agile calls for performing development, testing, and operational activities concurrently, which can create significant demands on a program’s technical environment. One official stated that preparing and maintaining synchronized hardware and software environments for these three activities to support frequent releases can be expensive and logistically challenging, especially for multiple concurrent iterations that can require more complex coordination of staff and resources.

Another complication agency officials identified was the lack of clear guidance for Agile software development, particularly when agency software development guidance was for a Waterfall approach. One official reported that, in order to account for Agile methods, new program policies and procedures would need to be developed, which they found to be a daunting task. They also stated that it was difficult to ensure that iterative programs could follow the standard approach. Another official stated that staff were nervous about moving from Waterfall guidance and, when programs followed a mix of iterative and Waterfall life cycle guidance, added confusion among the teams. The following is a recent example of the development of Agile methods for the U.S. Air Force’s Space Command and Control program.

Case study 1: Updating policy to reflect Agile principles, from *Space Command and Control,* GAO-20-146

In October 2019, GAO reported that the Air Force's Space Command and Control (Space C2) Program was taking an Agile approach to software development to more quickly and responsively provide capability to customers. According to Air Force officials, Agile development was relatively new to Department of Defense (DOD) programs. In the past, requirements were solidified in advance of development and the software was delivered as a single completed program at the end of the development cycle—with no continual involvement or feedback from customers or ability to modify requirements. The Space C2 program was one of the first DOD software-intensive programs to move away from the Waterfall approach and into an Agile framework. However, we reported that the then-current DOD acquisition instruction did not include guidance for Agile software programs.

GAO reported that DOD officials stated that new software guidance was in development, and this guidance was expected to offer pathways for developing Agile programs. DOD had also developed a draft template to assist Agile programs with developing their acquisition strategies, though the template and associated software guidance were in the early stages of development. In the meantime, however, Space C2 program officials confirmed that they were operating without specific software acquisition guidance. Space C2 officials also clarified that, while Agile software acquisition guidance had not yet been formally published, the program office had been actively engaged with the Office of the Under Secretary of Defense for Acquisition and Sustainment in refining draft policy and guidance. The program office noted that its program activities over the past year had been informed by and were consistent with this draft guidance.

Though DOD was taking steps to ensure that the Space C2 program had a comprehensive approach in place for managing, identifying, and mitigating challenges associated with an Agile development approach, GAO reported that key program plans and agency-wide guidance were still in draft form, leaving uncertainty about how program development and oversight would ultimately proceed. Finalizing a robust acquisition strategy containing the key elements for ongoing planning and evaluation would better position the program for success.

GAO, *Space Command and Control: Comprehensive Planning and Oversight Could Help DOD Acquire Critical Capabilities and Address Challenges,* GAO-20-146 (Washington, D.C.: October 30, 2019).

Agile programs depend on having the flexibility to add staff and resources to complete each release and adapt it quickly, based on lessons learned from one release to the next. One official stated that federal procurement practices do not always support this flexibility. For example, contracts that require Waterfall-based artifacts to evaluate contractor performance are not needed in an Agile approach where the contractor is part of the team and their performance is based on the delivery of an iteration. This official added that it can be a challenge for contractor staff to meet iteration time frames when tasks change, since federal contracting officers require structured tasks and performance checks. As a result, adding some flexibility in requirements is a contracting challenge. Chapter 6 discusses contracting best practices that can assist organizations as they work to reconcile Agile methods with contract requirements.

Challenges in executing Agile methods

Programs using Agile methods develop software in increments that are added onto the previous build; however, some agency officials reported that their staff mistrust such iterative solutions. For example, one official stated that federal customers expect to see a total solution; consequently, a demonstration of the functionality provided in one iteration or even one release was sometimes not considered good enough. The small increment of functionality demonstrated caused staff to doubt the Agile team's ability to deliver the remaining requirements, creating a parallel fear that the Agile team would not meet commitments. Officials also stated that this mistrust hindered the federal customer's ability to develop a definition of "done," which is an essential component of the process.

While a key tenet of Agile is prioritizing requirements, one official reported that customers found it challenging to validate and prioritize requirements by release, as they were used to defining all requirements at the beginning of the program and not revisiting them until they had been completed. Additionally, another official said it was difficult to reprioritize requirements when new work was identified.

In addition, iterations may incorporate compliance reviews to ensure that organizational legal and policy requirements are being met. However, one official stated that they found it challenging to complete compliance reviews within the short, fixed time frame of a single iteration because compliance reviewers were used to following a less flexible schedule under Waterfall development. Specifically, the official said that reviewers prioritized requests as they arose and that the reviews took months to perform. This caused delays for the iterations that needed to be assessed within a few weeks in order to proceed with the program in a timely manner.

Challenges in evaluating Agile methods

Agile software development methods stress evaluation of working software over extensive documentation and traditional program management milestone reporting. Officials said that this difference can present challenges in evaluating federal programs due to the lack of alignment between Agile and traditional evaluation practices. For example, federal oversight bodies request status reports for Waterfall development at development milestones and have not adjusted to Agile methods of frequent updates of each increment. As a result, an official reported that Agile teams became frustrated when dashboard statistics appeared negative.

Traditional oversight requires detailed artifacts at the beginning of a program, such as cost estimates and strategic plans, while Agile methods advocate an incremental analysis. One official stated that requiring these artifacts early can be challenging because it can be more worthwhile to start with a high-level cost estimate and vision or road map that gets updated as the solution is more refined through each iteration, rather than spending time estimating costs and strategies that may change. Chapter 6 discusses how program milestones and reviews can be aligned to an Agile cadence and other concerns related to contracting for Agile programs.

Furthermore, officials stated that program status tracking in Agile did not align with traditional methods. For example, one official stated that tracking the level of effort using story points instead of the traditional estimating technique based on hours was a challenge because team members were not used to that estimation method. One official stated that the required use of earned value management can be onerous if there is no guidance on how to adapt earned value management to reflect data about iteration progress or if the organization’s upper management does not embrace an Agile mindset and tracks monthly changes in cost, schedule, and product scope as control problems rather than as revisions to be expected during the iterative process. Chapter 7 of this guide discusses the application of performance management systems, such as earned value management, to Agile programs.

Actions Taken to Address Challenges

Since 2012, the federal government has taken steps to improve its policies and processes to help federal agencies adapt their current processes to Agile methods. Table 3 provides a summary of the laws, policies, and guidance and the entities that have been established to help address challenges.

Table 3: Laws, Policy, Guidance, Reports, and Entities Established to Address Agile Challenges

Effort	Date	Purpose
Office of Management and Budget (OMB), Office of Federal Procurement Policy (OFPP) *Contracting Guidance to Support Modular Development*	June 2012	To provide organizations with contracting guidance to support modular development, as required by item 15 in the *25 Point Implementation Plan to Reform Federal Information Technology,* published on December 9, 2010. The guidance discusses factors that contracting officers, in support of IT managers, will need to consider as they plan for modular development efforts, such as how to ensure that there is appropriate competition at various stages in the process, how broad or specific the statements of work should be, when to use fixed-price contracts, and how to promote opportunities for small business. It states that projects using modular contracting can be designed using iterative or “Agile” development so that subsequent projects add capabilities incrementally and that projects should aim to deliver functional value frequently and produce functionality in as little as six months.
General Services Administration: 18F created	March 2014	18F is an office within the General Services Administration, whose purpose is to collaborate with other agencies to fix technical problems, build products, and deliver digital services and technology products. It was started by a group of presidential innovation fellows to extend their efforts to improve and modernize government technology. 18F effects change using basic Agile tenets to practice user-centered development, testing to validate hypotheses, shipping often, and deploying products to users.
U.S. Digital Services (USDS) created	August 2014	USDS, under the Executive Office of the President, provides consulting and fosters multi-disciplinary teams to bring best practices and new approaches, such as Agile software development, to support government modernization efforts.
U.S. Digital Services: *Playbook*	August 2014	To increase the success rate of USDS projects, this playbook contains thirteen key “plays” drawn from successful practices from the private sector and government that, if followed together, are intended to help government organizations build effective digital services. For example, one of the “plays” is that the government build the service using Agile and iterative practices.
TechFAR: Handbook for Procuring Digital Services Using Agile Processes	August 2014	Highlights flexibilities in the *Federal Acquisition Regulation* (FAR) that can help organizations implement “plays” in the *Digital Services Playbook* that would be accomplished with acquisition support. It is designed to facilitate a common understanding among stakeholders of the best ways to use acquisition authorities in making these investments to set expectations and maximize the likelihood for success. It consists of a handbook, sample language, and a compilation of FAR provisions that are relevant to Agile software development and is not intended to supplant existing laws, regulations, or agency policy.

Effort	Date	Purpose
18F: Digital Contracting Cookbook	2014	Provides organizations with information and suggestions about how to acquire digital services based on the authors' experience. The cookbook is not a "how to" guide for digital services; it recognizes that organizations' requirements are all different. It notes that there are multiple ways to achieve success. For example, the cookbook includes a section on Agile development that states that the contractor shall, among other things, "Use Agile management best practices for estimating, planning, managing risk, and communicating status to enable the effective management of the project team along with use and product-owner expectations as to what will be done and by when."
Federal Information Technology Acquisition Reform Act	December 2014	The *Federal Information Technology Acquisition Reform Act* (FITARA) was enacted to improve the acquisition and monitoring of federal information technology assets. FITARA was intended to enable Congress to monitor organizations' progress and hold them accountable for reducing duplication and achieving cost savings through seven areas: federal data center consolidation, enhanced transparency and improved risk management, agency CIO authority enhancements, portfolio review, expansion of training and use of IT acquisition cadres, government-wide software purchasing program, and maximizing the benefit of the Federal Strategic Sourcing Initiative. FITARA also codified a requirement that agency CIOs certify that IT investments are adequately implementing incremental development, as defined in the capital planning guidance issued by OMB.
Federal Acquisition Institute*: Agile Acquisitions 101*	April 22, 2015	This briefing addresses the differences between Agile development and contracting for Agile programs, citing that both traditional contracting and contracting for Agile programs have defined requirements. It notes that the *Federal Acquisition Regulation* offers several options for implementing agility in federal contracts, which is a basic Agile tenet.
OMB OFPP: *Pilot for Digital Acquisition Innovation Lab*	March 2016	A pilot program aimed at helping organizations drive innovation in acquisition and to provide a pathway to test new or improved practices and help programs successfully adopt emerging acquisition best practices. The Digital Services Council provides funding to USDS and 18F consulting to support their work with pilot agencies, while USDS, 18F and a team of presidential innovation fellows provides hands-on coaching of cross-functional teams, a basic Agile tenet, to agencies.
Defense Science Board: *Design and Acquisition of Software for Defense Systems*	February 2018	The report is intended to provide independent advice to the Secretary of Defense on software development based on commercial best practices from industry and success within the DOD. The Board made seven recommendations on how to improve software acquisitions in defense systems, including the importance of the software factory, continuous iterative development best practices, and other ways to improve the software and acquisition workforce.
Defense Innovation Board: *"Software is Never Done"* report	May 2019	The report is intended to provide specific and detailed recommendations to the Department of Defense (DOD) on implementing modern software practices. The report emphasizes speed and cycle time as the most important metrics for managing software, the need for promoting digital talent in the workforce, and streamlined DOD acquisition processes for multiple types of software-enabled systems. For example, it states that while DOD is moving from Waterfall to Agile development, DOD must also change how programs and contractors are managed, which goes beyond moving to Agile development.

Effort	Date	Purpose
Department of Defense (DOD) *Software Acquisitions Pathway Interim Policy and Procedures*	January 2020	The interim policy and procedures are intended to simplify the DOD acquisition model to enable continuous integration and delivery of software capability on timelines relevant to the warfighter/end user. For software-intensive DOD systems, it requires the program to ensure the use of iterative and incremental software development frameworks, such as Agile; modern technologies to achieve automated testing; continuous integrations and continuous delivery of user capabilities; frequent user feedback and engagement; security and authorization processes; and continuous runtime monitoring of operational software, among other things.

Source: GAO analysis of OMB, GSA, and DOD documentation. | GAO-20-590G

With these efforts helped to address challenges, federal agencies often continue to struggle with software development. Management in these organizations is accustomed to oversight through a series of document-centric technical reviews, such as design reviews that focus on the evolution of artifacts that describe the requirements and design of a system, rather than its evolving implementation, as is more common with Agile methods.

Since reporting on Agile program management challenges in 2012, GAO has continued to examine and report on Agile adoption, execution, and monitoring and control efforts in the federal government. We have found that organizations continue to face challenges with the adoption and execution of Agile programs. Table 4 highlights recent GAO reports related to Agile adoption and execution practices.

Table 4: Recent GAO Reports Highlighting Agile Challenges

GAO report	Summary of findings related to Agile
Immigration Benefits System: U.S. Citizenship and Immigration Services Can Improve Program Management (GAO-16-467)	The Transformation Program has produced some software increments, but was not consistently following its own guidance and leading practices. The program adopted Agile in 2012, committing to the Scrum framework. However, it had deviated from the underlying practices and principles of this framework, such as not setting outcomes for Agile software development. This report made 12 recommendations aimed at improving information technology program management. For example, one recommendation was that the program should establish outcomes for Agile software development.

GAO report	Summary of findings related to Agile
TSA Modernization: Use of Sound Program Management and Oversight Practices is Needed to Avoid Repeating Past Problems (GAO-18-46)	The Transportation Security Administration's (TSA) new strategy for the Technology Infrastructure Modernization (TIM) program included using Agile software development, but the program had not fully implemented many practices necessary to ensure successful adoption of Agile practices. Specifically, the Department of Homeland Security (DHS) and TSA leadership fully committed to adopting Agile and TSA provided Agile training. Nonetheless, the program had not defined key roles and responsibilities, prioritized system requirements, or implemented automated capabilities that are essential to ensuring effective adoption of Agile. This report made 14 recommendations to DHS. For example, one recommendation stated that the TSA Administrator should ensure that the TIM program management office defines and documents the roles and responsibilities among product owners, the solution team, and any other relevant stakeholders for prioritizing and approving Agile software development work.
Department of Defense (DOD) Space Acquisitions: Including Users Early and Often in Software Development Could Benefit Programs (GAO-19-136)	The DOD planned to spend over $65 billion over five years on its space systems acquisition portfolio, including many systems that rely on software for key capabilities. However, software-intensive space systems have had a history of significant schedule delays and billions of dollars in cost growth. GAO found that program efforts to involve users and obtain and incorporate feedback were often unsuccessful. This was due, in part, to the lack of specific guidance on user involvement and feedback. Further, the programs GAO reviewed also faced software-specific challenges in using commercial software, applying outdated software tools, and having limited knowledge and training in newer software development techniques. This report made two recommendations to DOD, with both related to software development. The first was that DOD ensure that the department's guidance addressing software development provides specific, required direction on when and how often to involve users so that such involvement is early and continues through the development of the software and related program components. The second was that the departments' guidance addressing software development provide specific, required direction on documenting and communicating user feedback to stakeholders during software system development.
FEMA Grants Modernization: Improvements Needed to Strengthen Program Management and Cybersecurity (GAO-19-164)	The Federal Emergency Management Agency (FEMA) intended to develop and deploy its own software applications for the Grants Management Modernization program using a combination of commercial off-the-shelf software, open source software, and custom developed code. The agency planned to rely on an Agile software development approach. According to FEMA planning documentation, the agency planned to fully deliver GMM by September 2020 over eight Agile development increments. However, the department had not yet completed critical actions necessary to update its guidance, policies, and practices for Agile programs in areas such as developing life cycle cost estimates, managing IT requirements, testing and evaluation, oversight at key decision points, and ensuring cybersecurity. This report made eight recommendations to FEMA. For example, one recommendation stated that the GMM program management office finalizes the organizational change management plan and time frames for implementing change management actions.

GAO report	Summary of findings related to Agile
Space Command and Control: Comprehensive Planning and Oversight Could Help DOD Acquire Critical Capabilities and Address Challenges (GAO-20-146)	The U.S. Air Force was following Agile development to develop the technologically complex software for Space Command and Control. The program faced a number of challenges and unknowns, from management issues to technical complexity. Additionally, DOD officials had not yet determined what level of detail was appropriate for acquisition planning documentation for Agile software programs. They were also not certain about the best way to provide oversight of these programs, but were considering using assessments by external experts. These knowledge gaps ran counter to DOD and industry best practices for acquisition and put the program at risk of not meeting mission objectives. Additionally, software integration and cybersecurity challenges existed, further complicating program development. This report made two recommendations to DOD. One recommendation stated that the finalized Space C2 acquisition strategy should include, among other things, a manpower assessment identifying program workforce needs and state of expertise in Agile methods.
Agile Software Development: DHS Has Made Significant Progress in Implementing Leading Practices, but Needs to Take Additional Actions (GAO-20-213)	Many of the Department of Homeland Security's (DHS) major acquisition programs have taken longer than expected to develop or failed to deliver the desired value. In April 2016, to help improve the department's IT acquisition, and management, DHS identified Agile software development as the preferred approach for all of its IT programs and projects. GAO found that DHS had addressed four of nine leading practices for adoption of Agile software development. For example, the department had modified its acquisition policies to support Agile development methods. However, it needed to take additional steps to, among other things, ensure all staff are appropriately trained and establish expectations for tracking software code quality. By fully addressing leading practices, DHS could reduce the risk of continued problems in developing and acquiring current, as well as, future IT systems. This report made 10 recommendations to DHS. Among them were that DHS should establish target measures for the department's desired outcomes of its transition to Agile development and that processes and the associated set of controls to ensure Agile programs and projects are reporting a set of core required performance metrics for monitoring and measuring Agile adoption are defined.

Source: GAO. | GAO-20-590G

These reports all found that Agile adoption and execution challenges remain in federal organizations. This may be due to significant differences in focus; many organizations find it difficult to prepare for technical reviews that do not account for implementation artifacts, the availability of requirements, and/or design artifacts that are at different levels of abstraction. On the other hand, some organizations are surprised to discover they are already performing practices that can ease Agile adoption, such as establishing user groups that meet frequently with developers. In addition, while many of the policies and guidance focus on Agile principles, there are others that address cost, schedule, or contracting. It is important that organizations reconcile Agile principles and government policies and guidance with cost and schedule reporting requirements.

Organizations should supplement the Agile software development practices described in this guide with additional internal controls, such as policy and guidance.[20] Establishing such internal controls can help an organization become more efficient and effective. For example, internal controls can contribute to consistent execution of Agile practices across the organization and inform learning and improvement efforts. Such controls also support an organization's ability to report reliable information about its software development efforts.

[20]For more information about internal controls, see GAO, *Standards for Internal Control in the Federal Government*, GAO-14-704G (Washington, D.C.: Sept. 10, 2014).

Chapter 3: Agile Adoption Best Practices

Transitioning to Agile software development methods requires practitioners to do more than implement new or modify existing tools, practices, and processes.[21] Converting to Agile requires adopting the values and principles of the Agile Manifesto, which introduces challenges as an organization shifts from Waterfall development methods to those of an iterative process, such as Agile, which emphasizes rapid, frequent delivery of production-quality software. Yet, an Agile approach also presents an opportunity for an organization to improve its acquisition and development of software.

Organizations can use the best practices described in this chapter to help them manage and mitigate the challenges in making the transition to Agile.[22] The practices described are organized by functional perspective: team dynamics and activities, program operations, and organization environment. The discussion is in general terms in order to be useful regardless of the Agile method used. The practices highlight the aspects of Agile adoption that address key risks to be considered and are not meant to encompass all aspects of software development or program management. They can be used alone, or in conjunction with information from other publications that address similar topics.

This chapter assumes that a team, program, and organization has carefully chosen to adopt Agile software development methods. The decision to adopt Agile will depend on a multitude of factors, such as the stability of requirements, nature of the system, and program complexity. The best practice "Organization culture supports Agile methods" discusses how to make a decision whether or not Agile is the best-suited software development methodology for an organization's program.

There are practices often associated with an Agile approach, such as prescribed roles, events, artifacts, and procedures, but these vary

[21]As with any significant process improvement effort that an organization undertakes, change can be difficult and therefore presents risk. Management should consider the transition to Agile a process improvement or change management effort and manage the undertaking based on organization change management principles.

[22]This guide incorporates materials authored by Carnegie Mellon University with funding and support of the Department of Defense under federal contract FA8721-05-C-00003 for the operation of the Software Engineering Institute. Contact permission@sei.cmu.edu for re-use of such materials for other than US government purposes. Also, see our guide on reducing risks when using Agile methods: GAO, *Software Development: Effective Practices and Federal Challenges in Applying Agile Methods,* GAO-12-681 (Washington, D.C.: July 27, 2012).

depending on the methodology used. Over time, teams may refine and evolve their practices based on experience and lessons learned.

Because the adoption of Agile requires a shift in mindset at all levels of an organization, attempting to address all of the best practices at the same time can be difficult to manage and may lead to an inordinate amount of disruption and change in a short period of time. Therefore, management might consider prioritizing the best practices so that the most important have been implemented before moving on to the next set of practices.[23] Prioritizing the order of adoption may result in an organization prioritizing individual practices from the groups (team dynamics and activities, program operations, and organization environment) of practices described in this chapter, rather than prioritizing an entire set of practices from any single group. Consistent with continuous improvement, some best practices will be more applicable to new adopters, while other practices will be more applicable to organizations with more experience using Agile.[24]

Within each Agile method, specific terms may not fully align with the terms used in the best practices discussed in this chapter.[25] For example, a program might use a different term from the terms used in this guide to capture the concept of a product owner. Use of the specific terminology in this guide is not essential, but the concepts described in each best practice as a whole should be observable. If not, then organization officials should be able to explain why excluding a best practice (or elements of one) does not introduce unacceptable risk to transitioning to Agile. Although identified across varying levels, these best practices are highly interrelated (they all have to be aligned toward common goals) and therefore, each support the success of other practices.

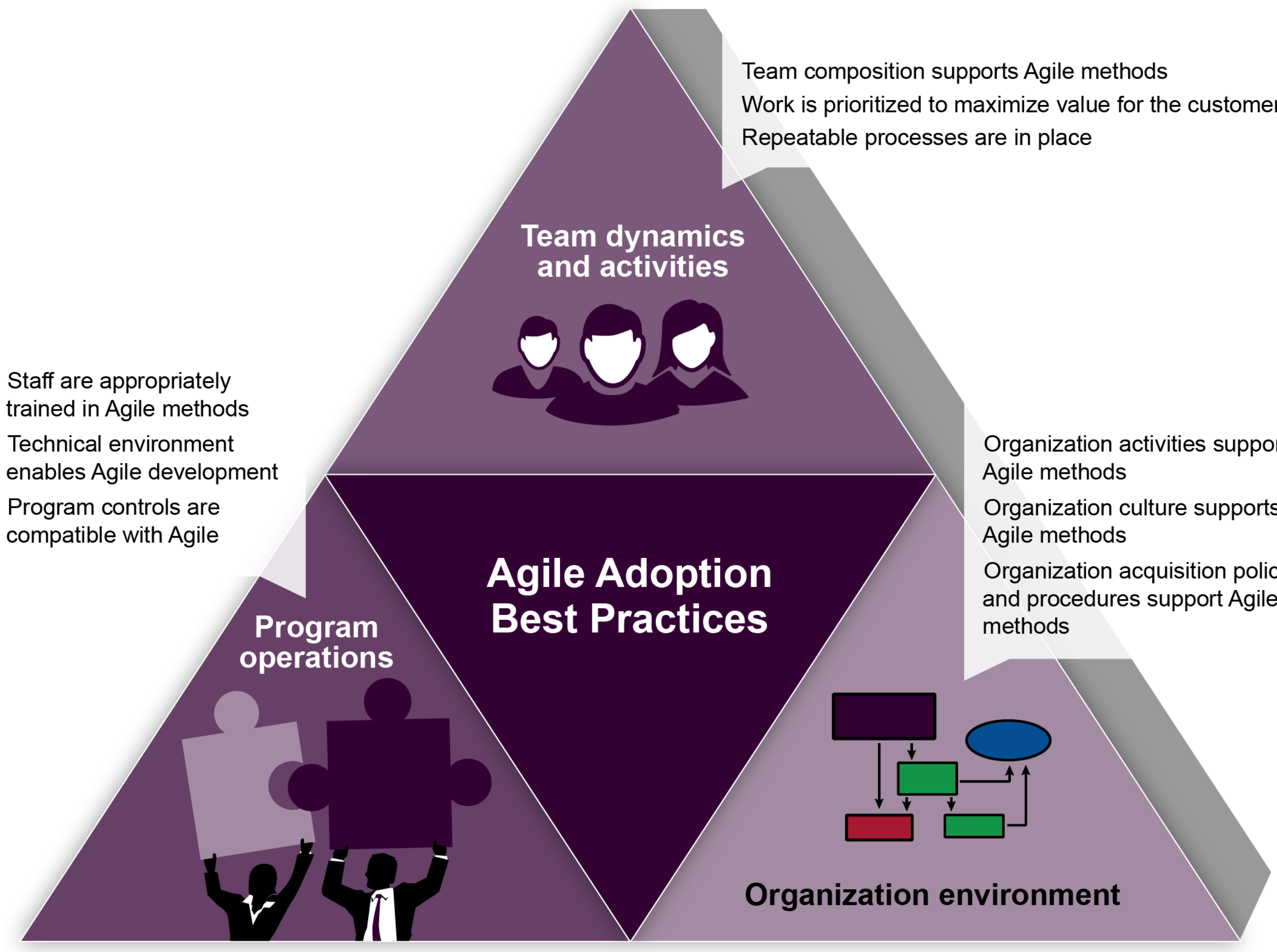
[23]Although not discussed in this guide, some organizations might wish to consider a maturity or readiness model to help in prioritizing practices. Maturity models for Agile are readily available for use independent of this guide, although we cannot attest to the success or appropriateness of these models. In addition, the CMMI® Institute has developed profiles for the use of CMMI in environments using selected Agile methods.

[24]Kanban methods deal with change somewhat differently than other Agile methods and may not limit the cultural barriers that impede change. Kanban methods enable an organization to improve its agility in any professional services or knowledge worker activity, not only software development, without any significant cultural shift and without implementing new processes. Organizations may choose to adopt other Agile methods in a similar fashion, focusing on slow, continuous, incremental change to existing business processes.

[25]See appendix II for definitions of key terms used in this guide.

Figure 2 identifies the best practices associated with each functional perspective of Agile implementation. Table 5 following the figure describes the qualities associated with each practice.

Figure 2: Overview of Agile Adoption Best Practices


Team composition supports Agile methods
Work is prioritized to maximize value for the customer
Repeatable processes are in place
Team dynamics and activities
Staff are appropriately trained in Agile methods
Technical environment enables Agile development
Program controls are compatible with Agile
Agile Adoption Best Practices
Program operations
Organization activities support Agile methods
Organization culture supports Agile methods
Organization acquisition policies and procedures support Agile methods
Organization environment

Source: GAO. | GAO-20-590G

Table 5: Summary of Agile Adoption Best Practices

Agile adoption best practice	Summary
Team dynamics and activities	
Team composition supports Agile methods	• Agile teams are self-organizing • The role of the product owner is defined to support Agile methods
Work is prioritized to maximize value for the customer	• Agile teams use user stories[a] to define work • Agile teams estimate the relative complexity of user stories • Requirements are prioritized in a backlog based on value
Repeatable processes are in place	• Agile programs employ continuous integration • Mechanisms are in place to ensure the quality of code being developed • Agile teams meet daily to review progress and discuss impediments • Agile teams perform end-iteration demonstrations • Agile teams perform end-iteration retrospectives
Program operations	
Staff are appropriately trained in Agile methods	• All members of an Agile team have appropriate training, since techniques used are different from those used for Waterfall development programs • Developers and all other supporting team members have the appropriate technical expertise needed to perform their roles
Technical environment enables Agile development	• System design supports iterative delivery • Technical and program tools support Agile
Program controls are compatible with Agile	• Critical features are defined and incorporated in development • Non-functional requirements are defined and incorporated in development • Agile teams maintain a sustainable development pace
Organization environment	
Organization activities support Agile methods	• Organization has established appropriate life-cycle activities • Goals and objectives are clearly aligned
Organization culture supports Agile methods	• Sponsorship for Agile development cascades throughout the organization • Sponsors understand Agile development • Organization culture supports Agile development • Incentives and rewards are aligned to Agile development methods
Organization acquisition policies and procedures support Agile methods	• Guidance is appropriate for Agile acquisition strategies

Source: GAO. | GAO-20-590G

[a]A user story is a high-level requirement definition written in everyday or business language; it is a communication tool written by or for customers to guide developers though it can also be written by developers to express non-functional requirements (security, performance, quality, etc.). User stories are not vehicles to capture complex system requirements on their own. Rather, full system requirements consist of a body of user stories. User stories are used in all levels of Agile planning and execution. They capture the who, what, and why of a requirement in a simple, concise way, often limited in detail by what can be hand-written on a small paper notecard. While Agile programs may use different terminology, such as product backlog items, for the purposes of this guide we will use the term user story throughout.

Team Dynamics and Activities

Best practice: Team composition supports Agile methods

Agile teams are self-organizing

Agile teams should be self-organizing, meaning they are empowered to collectively own the whole product and decide how work will be accomplished. The Agile teams' duties should be well defined, e.g., covering lower-level decision making and team formation. The teams' authorities should highlight the importance of cross-functionality to allow for autonomy and team stability. The more encouragement and latitude the team is given, the better it can address technical issues in creative ways. If teams are not self-organizing or self-managing, the teams may be inefficient, causing program cost increases and schedule slips.

The Agile team should be structured to allow for its own autonomy so that it need not rely on outside teams to complete its work. Team members should have cross-functional skills that allow them to be capable of performing all of the work rather than a single specialty and collectively the team should have all the skills necessary to perform the work and represent the various sections of the organization that touch on software development, such as business subject matter expertise, quality assurance, and cybersecurity.[26] In addition, the team should be integrated

[26]If operating in a government setting, the Agile team, or a subset of it, may be contractors. Contracting for Agile services can limit autonomy due to legal requirements for performing inherently governmental functions. See, e.g., FAR § 2.101 (defining inherently governmental function). However, whether using government employees or contractor employees, each Agile team should consist of personnel with all of the necessary skill sets. When defining the terms of a contract for Agile services, the program should work closely with contracting personnel (e.g., contracting officer and contract specialist) to promote autonomy while ensuring compliance with federal acquisition regulations. Contracting best practices related to Agile methods are discussed in more detail in chapter 6.

with other areas in the program office.[27] Specifically, the team can include contract specialists, designers, analysts, developers, and testers who, when working together, are able to decompose high-level descriptions of features that need to be accomplished into appropriate user stories and then work to identify logical iteration stopping points for testability. This level of expertise on the team allows it to solve most problems. If a team does not have the requisite skill sets, it will be reliant on other teams that may have other responsibilities, thus delaying progress on the product.

Case study 2: Cross functional teams, from *Defense Management,* GAO-18-194

The cross-functional team approach is thought to, among other things, advance a collaborative culture to address critical objectives and outputs. GAO research identified eight broad categories of leading practices associated with effective cross-functional teams: (1) open and regular communication, (2) well-defined team goals, (3) inclusive team environment, (4) senior management support, (5) well-defined team structure, (6) autonomy, (7) committed cross-functional team members, and (8) an empowered cross-functional team leader.

In February 2018, GAO reported that DOD had established a cross-functional team to address the backlog on security clearances and developed draft guidance for cross-functional teams that addressed six of seven required statutory elements and incorporated five of eight leading practices that GAO identified for effective cross-functional teams. GAO noted that DOD's guidance for cross-functional teams was critical to their consistent and effective implementation across the department and that this guidance would help ensure that such teams were provided with leadership support and resources and it further promoted collaboration across the department. GAO found that fully incorporating leading practices would help the teams be consistent and effective in addressing DOD's strategic objectives.

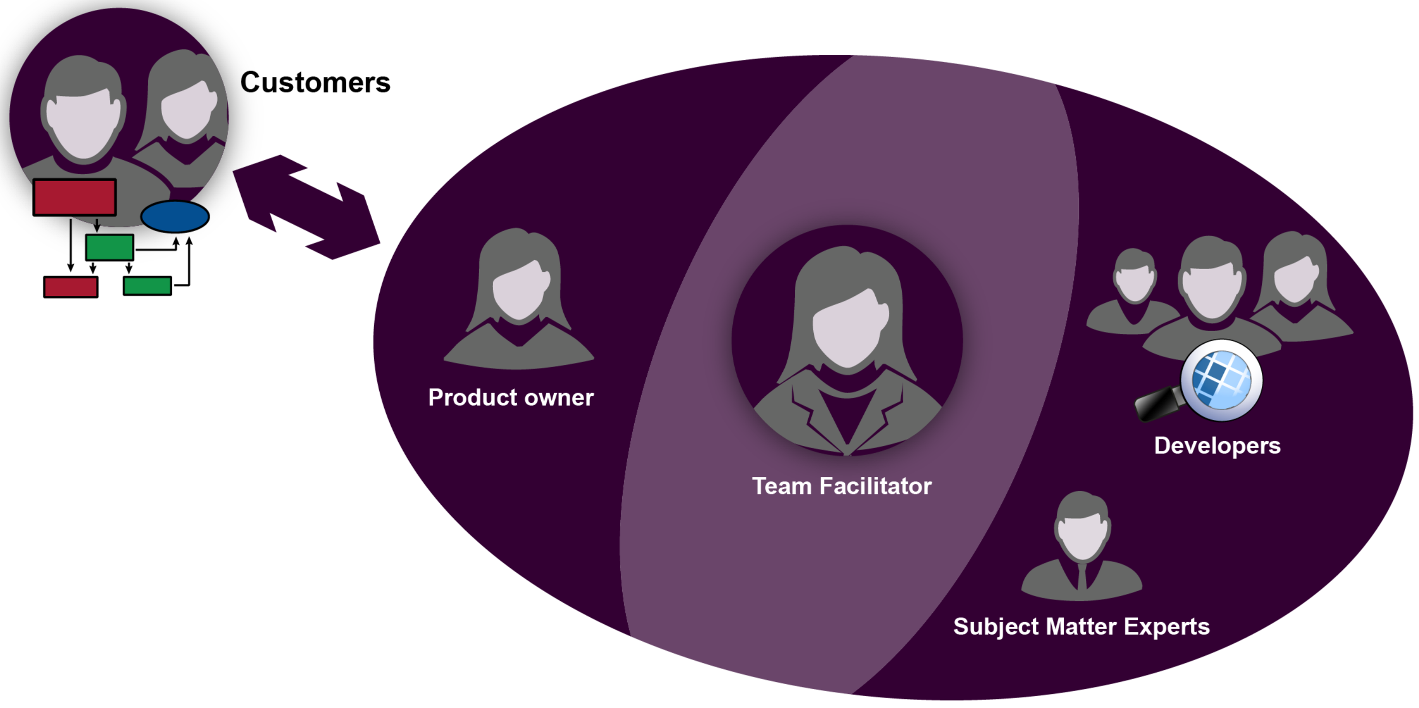
GAO, *Defense Management: DOD Needs to Take Additional Actions to Promote Department-Wide Collaboration,* GAO-18-194 (Washington, D.C.: February 28, 2018).

The roles for an Agile team can vary based on the Agile methods being applied; however, certain roles are similar in all Agile environments, such

[27]See GAO, *IT Workforce: Key Practices Help Ensure Strong Integrated Program Teams; Selected Departments Need to Assess Skill Gaps*, GAO-17-8 (Washington, D.C.: Nov. 30, 2016), for a more in-depth discussion of an integrated program team including critical success factors. GAO also issues a bi-annual series on cross-functional teams at the Department of Defense. For more information see GAO, *Defense Management: DOD Has Taken Initial Steps to Formulate an Organizational Strategy, but These Efforts Are Not Complete*, GAO-17-523R (Washington, D.C.: June 23, 2017).

as the developers, product owner, team facilitator, and subject matter experts.[28] Figure 3 shows the relationship of the Agile team to customers.

Figure 3: Relationship between the Agile Team and Customers



Source: GAO. | GAO-20-590G

Team stability, where team members are dedicated to the team and do not move in and out of the team, is important to ensure consistent productivity. Frequently shifting resources within a team, or between teams, can undo learning and shift team dynamics and skills, thereby diminishing the team's ability to meet commitments. The level of commitment of each team member and stakeholder is based on the needs of the program and should be discussed on a case-by-case basis. For example, involvement of a database administrator may only be required on a part-time basis when the team is working on user stories that require access to, or may indirectly impact, a database. Whether a

[28]See the best practice entitled "Staff are appropriately trained in Agile methods" in this chapter for further discussion of the training and technical expertise needed for a team. Chapter 6 also elaborates on subject matter expertise necessary for the effective contracting of Agile services.

team member is fully or temporarily dedicated to a particular team, all staff should be available when needed, to the extent possible.

The role of the product owner is defined to support Agile methods

In an Agile environment, the developers work daily with stakeholders, including the product owner. The product owner is the authoritative customer representative who manages the prioritization of the requirements (e.g., user stories) and acceptance criteria for those requirements, communicates operational concepts, and provides continual feedback to the developers as a representative of the customer.[29] The product owner also defines the acceptance criteria for stories and ultimately decides if those criteria have been met.[30] A product owner should understand the business and strategic values of the organization and possess subject matter expertise related to the business need in order to draw alignment with the vision of the product. Linking the need, vision, and product includes ensuring that prioritized requirements are evaluated and implemented in a timely manner and that the backlog is managed. If there is not a clearly identified product owner who is the authoritative customer representative and responsible for managing requirements prioritization, communicating operational concepts, and providing continual feedback, the developers may not be sure which requirements are priorities if they receive conflicting information. This uncertainty can result in delays to delivering high-priority features and deployment of the overall system. If the product owner is not a dedicated resource, the developers may find that person unavailable to answer questions when needed, and, if questions are not addressed in a timely manner, the developers may make assumptions in order to continue with its development and meet its commitments. If the team assumptions do not match the expectations of the product owner, significant rework may be necessary. This can slow down the development process.

The product owner role and responsibilities can be fulfilled in more than one way. For example, some organizations may delegate these

[29]Requirements are typically referred to in an Agile environment as user stories, features, or epics, depending on the target audience for level of detail of the work. Chapter 5 elaborates on how we use the term ‘requirements’ throughout this guide and best practices associated with requirements development and management, including the role of the product owner in those processes.

[30]As discussed subsequently in chapter 6, when using a contract for an Agile development effort, the contract must provide sufficient structure to achieve the desired mission outcomes, while also offering flexibility for adaptation of software requirements within the agreed-on scope of the system. Nothing in this guide is intended to suggest that a product owner has legal authority to undertake actions or make decisions that are reserved for contracting officers or contracting officer representatives.

responsibilities through multiple product owners, each of whom has clear boundaries and a clear division of duties, while other organizations may establish a core group of business officials to make key programmatic decisions, with a single product owner interacting with the Agile teams on behalf of the group. Regardless of the structure, the product owner should be empowered and their responsibilities should be well defined (e.g., the product owner's availability to the team). From a functional perspective, a product owner must be empowered to prioritize decisions about development. Without the ability to reprioritize work, the development process can slow down due to waiting on others with competing responsibilities to consider and respond on behalf of the business.

Since the product owner represents the customer, they routinely interact with key stakeholders to weigh the value of each requirement and establish work priorities for the developers.[31] The developers may choose to interact directly with key stakeholders if the Agile team deems it warranted. However, the team should ensure that functionality is prioritized by the product owner and not the stakeholders and that this additional coordination does not impact development productivity.

In order for a product owner to be effective, their responsibilities should be reduced so as to limit the number of Agile teams the product owner works with and allow time to interact with and complete duties with the team, stakeholders, and the customers. Without maintaining contact with both the developers and the customers, a product owner may not be able to represent what the customer priorities are and may misrepresent them to the developers. This could result in a decreased value from the system if the wrong features are given priority in the backlog or cause schedule delays if critical features were not developed.

[31]In this guide, we use the term 'requirement' to refer to a condition or capability needed by a customer to solve a problem or achieve an objective. Requirements will be used to refer to all development work since specific terminology (e.g., epic, capability, feature, sub-feature) may be unique to a specific organization. See chapter 5 and appendix II for more detail.

Case study 3: Product owner, from *Immigration Benefits System,* GAO-16-467

In 2016, GAO found the United States Citizenship and Immigration Services' (USCIS) Transformation program experienced many challenges due to product owners being stretched among multiple development teams. Product owners for the primary Transformation program system, the Electronic Immigration System (ELIS), were responsible for more than four development teams, and, at times, up to twelve teams. Consolidated release assessments, prior product owner testimony, and GAO observations identified that not having a dedicated product owner presented many difficulties for the ELIS development teams. For example, one product owner stated that it was a challenge to accommodate more than one team and she had to stagger her time between the teams to support sprint planning and maintain meaningful dialogue with the team. Additionally, consolidated release assessments indicated that product owners did not attend 21 percent of sprint planning meetings. Product owner availability was an issue voiced by development team members and also observed by GAO during standup meetings and sprint planning.

The more development teams a product owner is responsible for, the less time the product owner is able to spend with each team. Consequently, this can impact product owner effectiveness in performing his or her assigned duties. Furthermore, as we reported in 2016, the program faced challenges in completing work within committed time frames and product owner availability may have been a contributing factor. According to USCIS guidance, lack of inclusion and transparency with the development team's decision making and processes can result in a disengaged product owner, or one that makes decisions without adequate consideration of challenges faced by the team.

GAO, *Immigration Benefits System: U.S. Citizenship and Immigration Services Can Improve Program Management*, GAO-16-467 (Washington, D.C.: July 7, 2016).

Best practice: Work is prioritized to maximize value for the customer

Agile teams use user stories to define work

User stories have become a common method of defining small items of work that can be completed by team members inside of an iteration. Although some methods do not explicitly require the use of user stories (e.g., Kanban), they provide additional information beyond the high level requirement description to help Agile teams work to meet the requirement. A user story defines who needs the requirement and why. Regardless of the form used to communicate low level requirements, ensuring that the team knows who the requirement's customer is and why the requirement is needed are important. While Agile programs may use different terminology, such as product backlog items, for the purposes of this guide we will use the term 'user story' throughout to describe a small segment of work that can be completed in a single iteration and is determined by the product owner and developers.

The Agile team constructs a general outline for developing the user stories that comprise an iteration. The user story's focus is on the value delivered to the customer, often defines who the customer is, what is being developed for that customer, and why there is a need for the functionality. However, striking a balance between too much and not enough detail can be challenging: each user story should provide enough detail to allow developers to estimate the user story's complexity, but not so much information that there is little room for discussion between the product owner and the developers around the intent of the user story. Clearly establishing the components to include in the user story can help strike this balance. Establishing a common structure for the user story helps ensure consistency and can help prevent delays when product owners work with multiple teams or teams are reorganized.

The product owner determines the business value of each user story in consultation with the developers by refining the size, defining the criteria for acceptance, and establishing when the user story will be considered to be done. The value of a user story should be reevaluated based on the current needs of the organization to ensure the greatest return on investment. The practice of backlog refinement, along with a discussion of acceptance criteria and a definition of done is covered in greater detail in chapter 5.

INVEST

The acronym INVEST defines the characteristics of a quality user story: it should be "I" ndependent (of all others),"N" egotiable (not a specific contract for features), "V" aluable (or vertical), "E" stimable (to a good approximation), "S" mall (so as to fit within an iteration), and "T" estable (in principle, even if there is not a test for it yet). If the user story fails to meet one of these criteria, the team may want to reword it, or even consider a rewrite.

Agile teams estimate the relative complexity of user stories

The developers should use relative estimation, which compares the current work with work of similar size and complexity, to determine how much complexity a new user story represents. Relative estimation enables teams to maintain a sustainable software development pace and predict work commitments. The team should size user stories relative to one another, assess the complexity of work based on input from the product owner, refine user stories and estimates over time, and use prior estimates to inform future estimates. If teams are not using relative estimation to compare current size and work estimates to historical completed work, the team may underestimate or overestimate the complexity and time necessary to complete the user story.

Relative estimation

In software development, an estimate traditionally consists of a quantified evaluation of the effort necessary to carry out a given development task; this is most often expressed in terms of duration. Relative estimation is one of several types of estimation used by Agile teams. It consists of estimating tasks or user stories, not separately and in absolute units of time, but by comparison or by grouping of items of equivalent difficulty. Relative estimation, consistent with estimation in units other than time, avoids some of the pitfalls associated with estimating in general: seeking unwarranted precision, confusing estimates for commitments. For example, if a team uses a complexity point scale with the values [1, 2, 3, 5, 8, 13, 21], it should not be assumed that an 8 pt. backlog item will require four times as long as a 2 pt. one (although, if that is the norm the team has agreed upon, it could); rather, it will be more than a 5 pt. and less than a 13 pt. item. Also, because estimates are team- and domain-specific, there is little utility in attempting to use them for cross-team performance or productivity.

When estimating, the team should consider potential factors that can increase the complexity of the work. For example, a piece of functionality that requires passing interface testing before it can be accepted might prove challenging when the team factors in coordination and access to other systems. Team members are providing only a best estimate based on experience to date and will not fully know the complexity of the user story until the work has begun. Accordingly, program management should remember that estimates for the program are likely to change with each iteration. Practices such as affinity estimation can help to identify factors that affect the complexity of a user story.[32] Well-defined acceptance criteria can also help teams estimate a user story's complexity. Less well-defined user stories will carry more risk and uncertainty around size estimates. Additionally, if teams are not estimating user stories consistently, the teams may be committing to too much work, leading to user stories lasting longer than one iteration and team burnout.

The team continually revises the estimates of the program as they learn more about the business priorities and as a user story increases in priority. However, once an iteration has begun, sizing estimates should remain unchanged so that the team can examine variances between

[32]Affinity estimation is a consensus-based technique to estimate the relative effort of work. This term is further defined in appendix II.

estimated and actual work accomplished during the iteration. Estimation is a team-specific activity and estimates for one team should not be compared against estimates for another. For example, different development teams on one program may have a different “idea” of what the relative size of work is.

Requirements are prioritized in a backlog based on value

To prioritize a user story, the product owner determines the business value of each user story based on the needs of the customers, stakeholder priorities, and factors such as its risk level, dependent relationships, frequency of use, alignment with the vision of the product, security requirements, expected return on investment, and learning. The organization and program should have a shared understanding of what value means in terms of how much a feature satisfies strategic priorities. Identifying and measuring value, as with other Agile practices, requires constant collaboration. Agile teams should pull work from a prioritized backlog, providing frequent deliveries of software with immediate value to the customer. A lack of traceability between different levels of backlogs and program planning artifacts could lead to overlooking user stories or features that are critical to the program due to their high value to the customer or key dependencies that those user stories or features might have with other aspects of the system. Further, lack of understanding or insight into the methods used to measure value for user stories could cause a disconnect between the customer and developers and allow delivery of features that do not maximize the value.

The value of the work accomplished by Agile teams should be tracked and monitored. Once software has been delivered, the product owner may survey customers to measure satisfaction with each software release and track the accuracy of initial value estimates.

Value-driven feature development

One way to gauge the value of work is to measure how often a feature of a system is used by the customer. While there may be situations where a critical feature is necessary but used infrequently, often the product owner should be focused on developing features that will actually be used on deployment and therefore are of immediate value. As with any measure, setting a target for usage beforehand can serve as a benchmark to compare against on deployment.

The team should provide an ongoing assessment of value expected versus value delivered. In doing so, the organization has another measure of progress beyond traditional cost or schedule considerations. Without clearly prioritizing work, the developers could work on features that are not “must haves” to the customer, resulting in the delivery of features that may not be used and might contribute to schedule and cost overruns.

MoSCoW

Many Agile methods use the acronym MoSCoW to classify user stories as "must have," "should have," "could have," or "would like to have" for prioritizing the backlog.

Case study 4: Release road map, from Agile *Software Development,* GAO-20-213

In June 2020, GAO reported that the Department of Homeland Security (DHS) modified its acquisition procedures to allow for an ongoing assessment of progress, and indirectly the value of work accomplished, via a release road map. DHS guidance stated that the release road map is to be submitted to the Acquisition Review Board prior to acquisition decision event 2B. During lower-level technical reviews, exit criteria for reviews required the development team to follow the release road map and make adjustments that supported the successful completion of requirements defined at the acquisition decision event 2B. DHS supplemented these requirements with guidance on constructing a road map, including a discussion on how a program can sequence its road map for learning, risk, and economic value.

Within DHS, GAO reported that it reviewed a road map for one development module of the U.S. Immigration and Customs Enforcement (ICE) Student and Exchange Visitor Information System (SEVIS) program. This road map listed areas for development in the order they were intended to be developed and identified the associated business capabilities. The business capabilities identified in the road map aligned with the sub-capabilities listed in the program's operational requirements document. Examples of business capabilities in the road map that were also sub-capabilities identified in the operational requirements document included:

- create nonimmigrant record (including supporting forms),
- align nonimmigrant eligibility information with unique nonimmigrant,
- update nonimmigrant biographical information, and
- add/update dependent information.

GAO, *Agile Software Development: DHS Has Made Progress in Implementing Leading Practices, but Needs to take Additional Actions,* GAO-20-213 (Washington, D.C.: June 1, 2020).

Because the value of requirements is constantly fluctuating based on the state of the program and the organization, the product owner reevaluates requirements frequently to reprioritize if necessary as a result of team discussions. Doing so allows customers to receive the most important functionality (e.g., those features that provide the greatest value) first.

Likewise, this practice usually provides the biggest return on investment for the work performed.

Story board mapping (a.k.a. user story mapping)

Story mapping, a concept first formulated by Jeff Patton in 2005 in an article entitled "It's All in How You Slice It," consists of ordering user stories along two independent dimensions.[33] *The map arranges user activities along the horizontal axis in rough order of priority (or "the order in which you would describe activities to explain the behavior of the system"). Down the vertical axis, it represents increasing sophistication of the implementation. Working through successive rows fleshes out the product with additional functionality. One intent of this practice is to avoid a failure of incremental delivery, where a product could be released that is composed of features that, in principle, are of high business value but are unusable because they are functionally dependent on features that are of lower value and, therefore, deferred to future releases.*

Best practice: Repeatable processes are in place

To successfully meet the demands of rapid development, Agile teams use repeatable processes to establish consistency, thus providing a baseline against which improvements can be evaluated and adapted. Repeatable processes are not to impede the creativity of the Agile team by repeating the same steps in the same way every time the team operates. Rather, they characterize how to approach the Agile cadence. Because iterations are short (often 2-4 weeks in duration), consistency is important as practices will be repeated dozens of times a year.

Agile programs employ continuous integration

Automation of repeatable processes allows software components that are added or modified to be continuously integrated into the system. With short iterations in which to develop working software, integration should be frequent; thus, continuous integration using automation ensures that software handoffs between the various stages of development and testing are performed in a reliable, dependable manner.[34] Without continuous integration using automation, reliable, dependable software handoffs may not occur. Each stage of the continuous integration process should include automated tests of both functional and non-functional requirements with the scope of automated testing tracked and monitored based on established expectations. Without automated build and testing tools, the program may experience challenges in delivering the product

[33]Patton, Jeff. "It's All in How you Slice It." *Better Software.* Retrieved July 27, 2020, from https://www.jpattonassociates.com/wp-content/uploads/2015/01/how_you_slice_it.pdf.

[34]Due to the continuous integration of a code base in Agile, it is important for the program to have a mature integrated version control system in place. This is a critical tool to enable teams to work together and maintain configuration control over the code base.

on time and may have a limited assurance of product quality. Because automation depends on early investments in the technical environment, its success is heavily dependent on the program process best practice, “Technical environment enables Agile development”.

Mechanisms are in place to ensure the quality of code being developed

Adherence to coding standards and the use of automated and manual testing are necessary for improving the quality of code that is ultimately inserted into the continuous integration build process. Software with a large number of defects or an inefficient structure not only affects system performance, it forces the developers to spend critical time and effort to repair defects. While many methods are available for assuring code quality, there will always be some code inefficiencies or redundancies that ultimately limit system performance. These deficiencies can stem from time constraints, an unsustainable pace of development, undisciplined coders, or other reasons. The accumulation of these deficiencies over time is called “technical debt” and can present obstacles to an Agile program if not properly managed.[35] For example, as a code base grows, additional functions will rely on the deficient code, causing a degradation in overall system performance. Moreover, as the interest incurred on technical debt continues to rise, teams will devote more time to cleaning up errors instead of producing new features.

Technical debt can also be incurred mindfully, when it is more important to hypothesize the way a module will work in the eventual system (so that interfaces can be tested, for example) than to wait for the requirements for that part of the system to be written in detail. Eventually, both intentional and unintentional technical debt can increase to the point where the code base no longer functions properly and a complete refresh becomes necessary. Code quality should be tracked and monitored based on established expectations. Table 6 discusses methods that can be used to assure code quality.

[35]Although we only discuss technical debt accrued as a product of development, technical debt may also be generated by factors outside of the team’s immediate control. For example, program vision, architecture, and agency factors may all contribute to technical debt.

Table 6: Manual Coding Quality Assurance Methods

Method and description	Strengths	Limitations
Development is test driven: test cases are written before any code has been produced and only enough code should be produced to address the test case. Subsequent test cases and code are added via a cyclical process until the user story is finished.	• Continuous delivery of working software • Errors easier to identify and correct in smaller batches of code • Erroneous code does not proceed past development stage • Automation of testing can be incorporated	• Strength and accuracy of code depends on developer or tester who writes the tests • Does not ensure execution of tests in the build process if test cases are not part of the automated test suite • Does not ensure adequate maintenance of the test suite over time
Pair programming: Developers work in pairs.	• Working software provided more quickly • Working software has few defects • Raises skill level across the team	• Technique must be learned to be effective • Success can be hampered by incompatible dynamics of the pair • Appearance of not effectively using resources
Refactoring: A portion of time is set aside in each iteration to update and improve the code.	• Addresses technical debt that accrues • Promotes collective ownership • Promotes understanding of the code	• Does not remedy systemic issues that lead to technical debt • Can be challenging to gain management support
Code quality and peer review: A team member who is not the developer of the code reviews portions of the code base to assess its quality and adherence to defined coding standards.	• Catches errors not conceived by the initial software developer • Provides added assurance that code will function as intended when deployed • Enhances collective feeling of ownership of the code base	• Code coverage is limited • Diverts resources from other efforts • Is time consuming • Identifies coding issues after the fact

Source: GAO analysis of Software Engineering Institute literature and other material. | GAO-20-590G

Agile teams meet daily to review progress and discuss impediments

In addition to repeatable technical practices, there are repeatable business practices that increase the likelihood a team will succeed when using Agile methods for its software development. Specifically, teams can meet daily to coordinate the work, demonstrate working software to the product owner either during or at the end of an iteration to verify it meets customer needs, or participate in a retrospective meeting.

The daily progress meeting is to discuss any barriers encountered in completing the work; it is not intended to provide a status update to management.[36] Its purpose is to help the team gauge if it is on track to meet the iteration goals and adjust as necessary, while holding team members accountable. Daily meetings usually discuss these three topics:

[36]This practice comes from the Scrum method and has been adopted by many other Agile methods.

yesterday's accomplishments toward the iteration goals, today's planned work to advance the iteration goals, and any impediments to achieving the iteration goals that need to be removed. The larger purpose of the discussion is to help a team meet its stated goals for an iteration and increase the flow of work.

Without the daily standup meeting, team members may not be held accountable for their work. In addition, duplication of work could occur, or work may not get accomplished because of a lack of communication and understanding of who is doing what for the program. Without daily standup meetings, the team might also not identify impediments, which may result in rework or schedule delays.

Managers can observe the daily meeting and consider actions they might take to help remove team impediments, but the daily meeting should not become a status update for management. If used as a status update for management instead of focusing on progress and impediments, the meeting could last too long. The meeting is also not a place to solve problems or hold discussions with stakeholders. Instead, it is a place to decide what conversations (with what participants) need to take place that day. Teams can invite subject matter experts or other business stakeholders to the meeting, as needed, to answer questions regarding a specific user story they intend to work on that day.

Agile teams perform end-iteration demonstrations

Teams should demonstrate the latest version of the software for the product owner and other stakeholders at the end of each iteration, or as functionality has been completed. These demonstrations offer an opportunity for stakeholders to validate that teams are building the right product, help inform the priorities for the team moving forward, and offer a key opportunity to discover new requirements that can be translated into user stories. During a demonstration, stakeholders review and react to the particular portion of working software being demonstrated, rather than to the whole system. In order for a demonstration to be useful, all participants must be engaged and the sample software should be depicted in a realistic setting. Teams should not spend a significant amount of time preparing for a demonstration, as the focus of this time is to demonstrate working software and obtain feedback. If end-iteration demonstrations are not performed, the team may not be able to identify portions of the software that need improvement or modifications to provide the anticipated functionality.

Agile teams perform end-iteration retrospectives

At the end of each iteration, the team should hold a retrospective meeting to reflect on what went well and what could be improved for the next iteration.[37] It is an effective tool to enable continuous process improvement. The findings of the retrospective are determined and implemented by the team. For example, although retrospectives focus on process improvements instead of product improvements, the team can include action items from the retrospective as user stories in the backlog and track their implementation. If a retrospectives is not held at the end of each iteration, the team may not reflect on or improve the efficiency and effectiveness of its work processes, thereby impacting the timely delivery of a high-quality product. These retrospectives differ from end-of-project retrospectives in that they provide the opportunity to improve in the next iteration, not the next project.

Program Operations

At the program level, best practices address training staff in Agile methods, establishing a technical environment that facilitates Agile development, and implementing controls that are compatible with Agile.

Best practice: Staff are appropriately trained in Agile methods

All members of an Agile team are trained in Agile methods

All members of a team using Agile methods need to have appropriate training, since the techniques used are different from those used for Waterfall development programs. Team members and all staff who will be actively developing software, supporting software development activities, or involved in the acquisition process using Agile should be trained in the specific Agile method they will be using in order to have a common understanding about the processes to be used. Training in specific Agile methods includes the Agile policy and procedures documented by the organization. Without training, there may be a lack of common understanding in the program about the Agile methods to be used.

In addition, training requirements should be tracked and monitored for all team members. Refresher training should occur whenever there are any changes to the development or acquisition process, such as the use of

[37]If following the Kanban method, retrospectives should be held at an agreed-on interval because work is not organized by iterations.

new programming languages, applications, compliance requirements, coding, or security standards. If Agile is adopted throughout an organization, training of all team members should be considered as part of the organization's larger workforce training or strategic human capital management efforts. Without effective training based on a strategic human capital analysis, the program will be challenged in helping to ensure that the required capabilities and mission value will be delivered in a timely and cost-effective manner.

Developers and all other supporting team members have the appropriate technical expertise needed to perform their roles

In addition to training, teams using Agile methods should possess the competencies, skills, knowledge, and process abilities needed to perform their role. A program should consider Agile-centric skills when forming teams. Ideally, team members, including contract specialists, developers, and testers, should be cross-functional and together possess all the skills needed to produce working software, as discussed in the best practice, "Team composition supports Agile methods". If team members do not have all the required skills, programs should ensure that each developer has immediate access to people with specialized skills in, for example, contracting, architecture, database administration, software development, quality assurance, operations, information security, risk analysis, user experience, and business systems analysis. Having qualified staff helps ensure that the flow of development is continuous.

If program development is performed by an Agile services contractor, program officials should include an evaluation of the qualifications of the contractor to perform the work as part of the source selection. For example, on receipt of contractor proposals, a program may require the offerors to conduct a technical demonstration of their expertise. An Agile team needs to have all the appropriate technical expertise, or it could be delayed in completing its work while waiting on input from knowledgeable specialist outside of the team. Moreover, if individual team members are not proficient in the skills necessary to complete the work, then the quality of the product being developed may suffer, requiring substantial re-work.

Case study 5: Technical demonstrations, from *Agile Software Development,* GAO-20-213

In June 2020, GAO reported that the Department of Homeland Security (DHS) offered guidance for preparing acquisition strategies through its Procurement Innovation Lab. Webinars offered by the Procurement Innovation Lab on acquisition strategies for Agile programs discussed the need for interim delivery of software, close coordination between contractors and program office staff, contract oversight mechanisms that were tailored to support Agile development, and refined requirements. For example, the “Transportation Security Administration Agile Services Procurement” webinar discussed planning, executing, and de-briefing technical demonstrations used to select the contract recipient, paying particular attention to the value of transparency and modifying contract oversight mechanisms.

GAO reported that, within DHS, the U.S. Immigration and Customs Enforcement (ICE) Student and Exchange Visitor Information System (SEVIS) program evaluated contractor qualifications to ensure they had the necessary technical expertise. According to the program manager, contractor qualifications were evaluated in two stages; first, by assessing the contractor’s proposal, and second, by conducting a technical challenge to ensure that contractors could demonstrate the technical skills in the proposal. According to the instructions included in the request for proposals, this technical challenge required the contractor to leverage Agile best practices to design, develop, and demonstrate working software that addressed user stories provided by the program. Although the instructions stated that contractors were required to follow Agile methods, the ICE SEVIS program manager stated that the primary goal of the technical challenge was to assess development skills rather than knowledge of Agile.

GAO, *Agile Software Development: DHS Has Made Progress in Implementing Leading Practices, but Needs to take Additional Actions,* GAO-20-213 (Washington, D.C.: June 1, 2020).

Best practice: Technical environment enables Agile development

System design supports iterative delivery

Planning the design of the system is important in order to understand and manage the considerations that can enable a loose coupling of architecture components and to provide architecture to support the Agile methods and end state for the program. An Agile program should refine and build out the architecture over time as it learns more about the system but also allow time to consider system requirements in order to limit future complexity, rework, and loss of investment. If the program does not consider the system architecture during its initial planning and instead relies on building out the architecture as code is developed, the architecture may not support the needs of the system when fully operational and require a complete technical refresh.

Architectural runway

Some programs use the concept of an architectural runway to ensure that the technical infrastructure, dependencies, and interfaces are clearly understood and in place to support implementing the near-term software in an operational environment. The architectural runway is continually extended to meet new and evolving needs in front of development, which avoids the need for large, upfront architectural design.

In designing the system, a loosely structured architecture allows for the rapid development of modular components in incremental releases. From an Agile perspective, this allows teams to produce useable code at each iteration without impacting the larger system, as the architecture provides the platform for new code to be inserted seamlessly into the operational environment. In addition, since large federal programs typically have staff distributed across multiple locations, it is easier for each team to be responsible for a module. This module is then loosely coupled with others, eliminating the need for many point-to-point interfaces that would require significant communication and collaboration between teams. Frequent testing and reviews can help ensure that newly developed components are properly integrated with existing ones. Incremental code delivery can result in more frequent customer reviews that provide valuable feedback to the developers. Because customers are reviewing smaller slices of the system than in a typical Waterfall development, the staff members participating in an Agile development review are likely to be different than those in a Waterfall development. If software design and architecture are not loosely coupled, changes to individual pieces of the system may require a significant amount of testing of the entire system, slowing the pace of development and delivery of the product.

Case study 6: Tools for automated testing and continuous integration, from *Agile Software Development*, GAO-20-213

In June 2020, GAO reported that the U.S. Immigration and Customs Enforcement (ICE) Student and Exchange Visitor Information System (SEVIS) program defined its technical environment to include technical tools for automated testing and continuous integration. The team process agreement for one development module GAO reviewed identified technical tools that supported continuous integration and testing within the project's technical environment. This included a tool known as Jenkins for continuous integration and tools known as MUnit and Soap UI for continuous testing. In addition, the *ICE SEVIS Modernization Test and Evaluation Master Plan* discussed tools for helping to ensure code quality, such as an automated code analytics tool to be used to identify test coverage of code and cybersecurity code vulnerabilities.

The project also defined management support tools in the process agreement. Specifically, it identified support tools for tracking and knowledge management, such as JIRA and Confluence. The team process agreement stated that JIRA should be the main knowledge management tool and that all changes, discussion, and history should be tracked in each ticket. This process agreement also stated that JIRA should be the team's tracking tool with Confluence used to provide transparency.

GAO, *Agile Software Development: DHS Has Made Progress in Implementing Leading Practices, but Needs to take Additional Actions,* GAO-20-213 (Washington, D.C.: June 1, 2020).

Technical and program tools support Agile

To continually monitor progress, Agile program management and technical tools may be needed to assist Agile teams with electronically managing the Agile framework they are using to develop software. The selected tools should be integrated into the program's technology environment (e.g., automated regression testing suites and continuous integration support tools) and access should be available to all team members and stakeholders who need the access. These electronic tools can prevent delays in performing critical tasks. If technical and program tools are not consistently available to those members of the team requiring access, then the productivity of developers may suffer and result in increased costs for development.

Organizations sometimes face limited access to the contractor's tools. This is based on a perception that providing access could lead to micro-management of the developers. This fear of micro-management should be addressed because everyone involved in the Agile development effort, both organization and contractor, should have access to the data. Given the variety of Agile tools available in the commercial market, program managers should analyze their current suite of program management tools to determine to what extent they are aligned with Agile principles and practices.

Since Agile methods deliver software frequently, they require a certain degree of automation to avoid creating lags in the process. For example, to ensure quality products are produced during a delivery cycle, the software is integrated and tested frequently—usually daily. This rapid integration and testing can be labor intensive without the support of automated tools. Automation also reduces the chance of human errors and can perform many functions much faster than people can. Large programs not using automated tracking tools could miss key dependencies between user stories and features. Without automated tools, the program risks inconsistent implementation of processes across teams, which may negatively affect product delivery and understanding the program's progress.

Best practice: Program controls are compatible with Agile

Critical features are defined and incorporated in development

The program strategy should identify the mission, architecture, safety-critical components, and dependencies that ensure that all aspects of a program are considered, and these aspects should be revisited on a regular basis.[38] Some programs define these components during an initial iteration before any software development begins. Doing so can help the program avoid rework and integration challenges from inadequate software and the resulting increase in costs and time to deliver all critical features. Without clearly identifying mission and system-critical architecture features, the program risks developing these features after other software is in place and facing substantial rework and integration challenges, unnecessarily increasing the cost and time to deliver all critical features.

In determining the criticality of the software, the program should evaluate and prioritize the relative value of the work to ensure that each iteration delivers the most business value, this can ensure that the customer's most pressing needs are being met first. Business and mission goals drive the prioritization of the most advantageous requirements, and security requirements should be reviewed throughout development. At the same time, the product owner must consider technical risk relative to

[38]For more information on critical systems in the federal government, see GAO, *Information Technology: Agencies Need to Develop Modernization Plans for Critical Legacy Systems*, GAO-19-471 (Washington, D.C.: Jun. 11, 2019).

business and mission goals–and if there are significant “unknown unknowns,” those features may need to be addressed early to understand what is actually achievable versus what is desired. The program may need to pivot if technology assumptions are made in the program’s conception that are not reasonable for the cost allowed or the state of the technology that must be used. If critical business requirements are not prioritized appropriately, software may not provide the required functionality. Lack of communication between the product owners and developers regarding features’ priorities risks the development of noncritical software in place of critical software and lower customer satisfaction with the completed product.

Non-functional requirements are defined and incorporated in development

Although much of the focus in development is on functional needs, the program must also include nonfunctional requirements, such as security and privacy, in the program strategy.[39] As with critical dependencies, continuous attention to technical excellence and good design requires the developers to consider nonfunctional requirements throughout development. This is particularly true with complex programs such as healthcare and financial systems that process sensitive data with complex non-functional requirements. Teams overlooking nonfunctional requirements may develop a system that does not comply with current federal regulations (e.g., cybersecurity or interface requirements for IT programs), causing unnecessary risks to business operations and resulting in the software not becoming operational until these components have been addressed. See chapter 5 for additional discussion on defining and capturing non-functional requirements.

Agile teams maintain a sustainable development pace

Management should strive to ensure that teams can maintain a sustainable development pace by prioritizing user stories, some of which may be non-functional requirements, establishing an agreed-upon definition of done for those user stories, and reaching a mutual commitment on the work to be accomplished for each iteration. Many teams embrace Agile methods because the software is needed quickly; however, sound engineering and management principles are still required when employing Agile.

[39]Nonfunctional requirements generally specify criteria that can be used to judge the operation of a system rather than specific behaviors. This should be contrasted with functional requirements that specify specific behavior or functions. Typical nonfunctional requirements are reliability, scalability, maintainability, availability, quality, privacy, security, and compliance with section 508 of the *Rehabilitation Act of 1973*, as amended (29 U.S.C. § 794d (discussing information and data accessibility)).

Management should emphasize and encourage teams to maintain a consistent development pace that can be sustained indefinitely. For this to happen, management needs to encourage and promote how this paradigm will benefit everyone. Specifically, teams that can determine a reasonable pace will not suffer from burnout and will take pride in their ability to continually produce quality software that pleases the customer. If teams are not working at a sustainable pace, there is a risk of burnout, which can cause delays in the program. In addition, working at a sustainable pace provides management with historical data, such as the team velocity, that can provide for more accurate cost estimates and time to develop desired features. While an effective measure if collected and interpreted properly, it is important that management understand velocity is team-specific and should not be compared across multiple teams.

Chapter 7 provides additional information related to specific Agile program monitoring and control and chapter 8 addresses the various metrics that can be captured to monitor performance. In addition, appendix V discusses the Scrum and XP methods for achieving a sustainable pace and how it can be planned for and monitored over the program's life. Without establishing a consistent pace, the program cannot reliably use historical metrics, such as team velocity, to estimate future efforts required in product development.

Organization Environment

Organization environment best practices address organization life cycle activities, culture, and acquisition policy and procedures. Although not explicitly called out as a best practice, an organization may also be responsible for directing, monitoring, and/or controlling the implementation of program operations and team activities and dynamics. Best practices related to these topics will be discussed later in this guide.

Organizations all have different missions, goals, existing processes, culture, and requirements. Consequently, they may adopt different and varying levels of Agile methods to suit their needs. Before beginning the process of scaling Agile, management will select or develop a suitable approach that might include using a pilot program to discover problems and then mature its processes and incorporate lessons learned before fully adopting them throughout the organization.[40]

[40]In IT, scaling is the ability of a system, network, or process to absorb a growing amount of work or its potential to be enlarged to accommodate that growth. If the design or system fails when the amount is increased, it does not scale.

An organization may have to consider a possible reorganization to enable a large-scale transformation to Agile software development. This can be simple, such as reviewing traditional roles and responsibilities and realigning them with Agile roles (that is, program manager to product owner), or it can be more complicated, resulting in intensive changes, such as restructuring one or more components or shifting entire IT portfolios. One way to help ease an organization’s reorganization is for management to establish communities of practice or other working groups of motivated or influential individuals to lead the change. Another is to use small pilot programs to showcase success and learn first where the organization’s pain points exist before scaling Agile across the organization. Either a top-down or bottom-up approach can be successful in scaling Agile and helping to drive an organization’s change.

Best practice: Organization activities support Agile methods

Organization has established appropriate life cycle activities

The organization’s life cycle activities should support the iterative and incremental nature of an Agile approach. They should also allow for the organization to tailor life cycle activities to encourage frequent collaboration between the customer and the developers to support rapid development. When making the transition to Agile, sponsors may need to make structural changes at the organization level in order to support the iterative nature of Agile. These changes include allowing programs that are applying Agile methods to tailor life cycle activities, including technical reviews, and associated artifacts to their cadence of delivery. These changes may affect the organization, staffing, and interactions with other groups, such as information assurance and operational test and evaluation. If programs are unable to tailor life cycle activities, then the organization’s oversight process could negatively affect the cadence established by the Agile team, resulting in less predictable development efforts.

The organization’s life cycle must also allow for refining detailed requirements. The highest priority of federal IT programs is to satisfy customers through early and continuous delivery of valuable software. In order for the mission to succeed, federal organizations’ acquisition policy and guidance need to allow for refining detailed requirements while maintaining the high-level program vision and frequently delivering value in small deployments. There must be frequent collaboration between the organization and the developers so that the most valuable work is always completed first. If collaboration is not occurring regularly, then priorities regarding requirements will not be known and the result may not meet the program's vision or customer's needs.

Programs can respond to changing business needs when early requirements are defined at a level high enough that the program (or organization) can fine tune or modify the requirements to reflect a better understanding of what is needed (see chapter 5 for a discussion of requirements decomposition). Organizations can do this by considering whether refined policies and procedures governing life cycle activities and oversight allow for lower-level requirements to be refined and the speed with which updated work can be approved. For example, in determining the appropriateness of the life cycle activities associated with using Agile methods, an organization can state in policy that satisfaction of the customer is the main focus and accommodating refining requirements is acceptable. (See chapter 7 for further discussion of how to monitor changing requirements with respect to cost, schedule, and scope commitments.) Where detailed requirement refinement is not understood or defined at an organization level, the adoption and full realization of the benefits from Agile methods will be difficult to achieve.

Goals and objectives are clearly aligned

A proven method for nurturing a strong relationship among customers, the developers, and the organization is to align program goals with strategic IT objectives and to ensure that program goals clearly reflect stakeholder needs and concerns.[41] While this alignment is important in non-Agile settings, its urgency in an Agile environment derives from the fact that software will be available earlier to test and interact with other parts of the system. To effectively implement Agile processes, the organization's mission or strategic goals are key inputs for decision making. If the organization's goals are not clear or do not adequately reflect stakeholder concerns and mission needs, then lower-level decision

[41]Agency plans for capital acquisitions, including plans for IT, should align with and support advancement of these goals. Alignment to mission and goals is required for major IT investments subject to Capital Planning and Investment Control (CPIC) reporting. See chapter 2 for further discussion of legislation impacting Agile adoption in the federal space.

making may be misaligned with the organization's focus.[42] This misalignment can, in turn, erode trust and often results in overbearing governance and bureaucracy, leading to delays. While a program may need to build trust with developers, the organization needs to trust that the program office can properly manage itself through delegation and more targeted governance.

Additionally, it is important that the organization's software-related goals are clearly aligned with its program goals. The continuous delivery of working software depends, for example, on systems engineers and quality assurance teams having sufficient resources to respond to repeated software deliveries. If these software-specific needs are not considered to be part of the larger program goals, then the implementation of software applications may not fulfill minimum requirements established by the organization or by the federal government.

In determining whether software, program, and organization goals and objectives are strongly aligned, an organization should collect objective measures, such as data from road maps and product portfolios that are well defined. These measures should be clearly communicated to the entire organization so that stakeholders, sponsors, customers, and developers know exactly which features and capabilities have been achieved according to the goals and objectives. Doing so will not only allow an organization to regularly track its productivity but will also determine how an individual program fits into the organization's portfolio and mission. If approved program goals do not align with both the IT and business goals, then lower-level decision making runs the risk of being misaligned with the organization's focus.[43] Chapter 8 provides a detailed discussion of metrics and their use in continuous improvement of organization processes.

[42]The best practice, "Program controls are compatible with Agile" discusses how programs should consider and capture both critical features as well as non-functional requirements. Both steps within the practice can help to ensure strategic alignment between the goals of the organization and those of the program.

[43]The best practice, "Work is prioritized to maximize value for the customer", discusses the need for the team, and ultimately the program, to routinely deliver the most valuable functionality each iteration. Ensuring alignment between the user stories delivered in an iteration and the goals of the program and organization via an agreed-upon artifact such as a road map that tracks feature prioritization is one way to exhibit the delivery of high value functionality.

Finally, goals should be clear but not static. Many organizations adopt Agile precisely because it allows for rapid response to changes in either the external or internal environment. This rapid change makes it even more important that an organization effectively and routinely ensures that program goals are clearly communicated.

Best practice: Organization culture supports Agile methods

In most organizations, adopting Agile methods involves new behaviors and a different mindset. This is a major shift in how an organization operates and will affect the overall climate. For some agencies, the life cycle management process for an IT system includes not just the program office, but also outside support functions that are shared across the organization, such as certification and accreditation or operational test and evaluation. Policies and regulations can make it difficult to include these areas when adopting Agile. However, cascading sponsorship helps ease these problems by having advocates in many places within the organization who can model new Agile values and behavior, thereby instilling confidence in the people who are actively trying to adopt the new practices.

Sponsorship for Agile development cascades throughout the organization

Implementing Agile requires that stakeholders and sponsors embrace and fully understand the implications of this approach. Without high-level encouragement, Agile implementation might become a paperwork exercise, leading to a failure to complete software development. For example, without encouragement and commitment from upper-level management, Agile teams may not appropriately collaborate with product owners when they are unsure about the importance of certain functionality, causing confusion that ultimately can result in a poor product. Thus, functionality developed using a process that does not embrace an Agile mindset might require heavy investment in the post-deployment correction of errors or functionality enhancements to meet customer needs.

Sponsorship for a program should start with senior stakeholders openly and explicitly supporting the use of Agile processes in the organization. One way to initiate a successful transition is to identify influential individuals within the organization who are interested in transformation and can become Agile champions. These champions may or may not be senior stakeholders but should always be someone who has the respect of Agile adopters as well as the support of senior leaders. The champion’s role is to help protect early Agile programs from being derailed by those who do not understand the new methods or are skeptical of change. Therefore, the strategy for winning over skeptics will be for the champion to demonstrate how programs have flourished under this new approach. Senior stakeholder sponsorship will be helpful to organizations in transitioning to Agile methods and help to ensure success with the use of Agile practices. Without sponsorship from senior stakeholders and the presence of an Agile champion or multiple champions, the organization may not embrace the transition, which can lead to inconsistent Agile practices and lackluster results.

Case study 7: Agile sponsor, from *DOD Space Acquisitions,* GAO-19-136

A practice of Agile development is to identify an Agile sponsor within senior management—someone with formal authority within the organization to advocate for the Agile approach and resolve impediments. GAO's 2019 review of the Mobile User Objective System (MUOS) program found that the MUOS contractor lacked an Agile advocate in the program office, which undermined its ability to fully employ an Agile development approach. For example, even after the contractor adopted an Agile approach, the program office directed the contractor to plan out all work across software builds in order to maintain control over requirements—similar to a Waterfall approach but inefficient in Agile. According to the Software Engineering Institute, without an Agile advocate in a program's leadership, organizations only tend to use a partial Agile or Agile-like approach.

GAO, *DOD Space Acquisitions: Including Users Early and Often in Software Development Could Benefit Programs,* GAO-19-136 (Washington, D.C.: March 18, 2019).

While having a clearly defined policy for Agile programs can be effective in many cases, using a policy or mandate to force adherence to Agile principles does not produce the healthy adoption of new practices. For example, putting policies in place too early, before the appropriate transition mechanisms are solidified, may lead to basic compliance but without consideration for the organization's culture and mindset change that should occur during a successful transition.

Further, since Agile may not be appropriate for all programs, each program should consider its rationale for the use of an Agile approach in accordance with defined program and software goals. For example, the following could be considered indicators that a program is ready to adopt Agile practices, although this is not the only scheme for evaluating program readiness for Agile:[44]

- requirements are flexible;

[44]One approach for determining if Agile is best for a program is the Stacey diagram. This diagram measures requirements agreement against technology certainty.

- an established process is in place to further refine the requirements over time;
- an Agile champion or program sponsor is available to help the team overcome impediments;
- customers and/or subject matter experts are readily available to provide feedback;
- teams have been trained in a specific Agile framework or set of methods;
- a facilitator is available to assist teams in applying Agile methods;
- supporting functions like contracting embrace organizational changes needed to make Agile work;
- the program is large with a variety of risks, particularly technological obsolescence; and
- teams desire more responsibility and ownership.

Sponsors understand Agile development

Sponsors and champions should not only be assigned to enable an Agile transition; they should understand and be able to differentiate between traditional and Agile roles, Agile cadence, and processes. It is also important that they are accountable for results. Sponsors should be committed to supporting the specific Agile approach adopted so that processes are applied consistently across the organization. While the roles and responsibilities in a traditional acquisition are well documented in regulations, policies, and training documents, in an Agile environment they are more flexible and may not be as easily understood. One of the biggest obstacles to an Agile transformation can be that very few people in the organization know and understand Agile methods or that they implement Agile based on limited experience and understanding of them. As a result, sponsors and senior stakeholders may need training and/or coaching regarding their new responsibilities.

Organization policies, therefore, should require sponsors and senior stakeholders to be fully educated regarding Agile values and principles and committed to implementing the chosen Agile approach, and organizations should monitor completion of that training. In doing so, sponsors can then transmit or reinforce learning from their training to staff, as needed. If sponsors are unable to effectively differentiate between Waterfall and Agile implementation, they may hamper or impede the effective adoption of Agile principles, leading to a breakdown in processes.

Organization culture supports Agile development

In addition to senior stakeholder and policy support, certain physical and social environments should be provided by the organization to allow Agile teams to succeed. For example, Agile environments typically call for locating cross-functional teams in common areas where the teams can work together and converse regularly. Designating a team space for physically co-located teams to work with appropriate network and IT access can be as simple as dedicating a conference room to the team for the duration of the program. Even if the teams are physically separated, modern communications and social media methods (such as video-teleconferences or instant messaging chats) can be used to promote continuous discussion. For example, some distributed teams may establish a continuously "open" chat room where team members can talk about their work. Whether distributed or co-located, the end goal is for all team members, including the product owner, to be immediately accessible to ensure questions are answered promptly and team pace is not delayed.

To facilitate the delivery of a "just enough, just in time" product, a climate of trust should exist throughout the life cycle between the organization and the developers. However, since the federal acquisition environment is built on strong oversight, traditional acquisitions can often result in adversarial relationships between the acquirers and the developers. Conversely, in an Agile environment, a climate of trust, built by shared experiences in which all parties feel respected and accepted, is needed so that the program team can achieve its fullest potential. A first step toward developing trust between the developer and the organization could be a joint workshop or event that focuses on the effort but provides opportunities for working together across organization boundaries. Additionally, organizations should consider granting greater autonomy to Agile teams by providing them with the skills and knowledge necessary to succeed and an awareness of the long-term goals of the system.

Another method to develop a climate of trust is to consider communication practices across groups and the amount of transparency coming from the organization both bottom up and top down. For example, one option could be to make all artifacts that contribute to the development of the system broadly accessible to everyone associated with a program, including oversight boards.[45] Availability of team message boards, instant messaging software, and other collaborative workspaces can facilitate such communication practices. This can be helped by having a process and terminology in place that are commonly understood in order to prevent misunderstanding.

After Agile has been implemented, the organization can continue to learn and adapt from the feedback from key stakeholders and Agile teams. To do this requires continuous inspection and adaptation to improve the entire development process, such as in a more formal meeting, a retrospective, or an informal set of discussions among sponsors. In addition, ongoing demonstrations of working software can then serve as touchpoints where an oversight body can gain added assurance that the Agile teams are developing a system of value in line with its intentions.

To effectively apply lessons learned, relevant, reliable data should be collected during the transition to help facilitate and support senior stakeholder adaptation and decision making, since stakeholders are often removed from day-to-day Agile operations. In addition, modifications to appropriate policies and processes, such as systems engineering life cycle documentation, will help ensure that needed changes to Agile practices and processes are effectively communicated and consistently applied throughout the organization.

Establishing an environment supportive of Agile can aid team and program operations in meeting program goals; however, if an environment supportive to Agile methods is not in place, then team and program operations might not have the resources necessary to be successful, thus impeding delivery of the product and not meeting agreed-upon goals for cost, schedule, and performance.

Incentives and rewards are aligned to Agile development methods

Open and explicit support by the senior stakeholders also means that traditionally rewarded behavior is no longer the norm. This is often one of the hardest concepts for senior stakeholders to consistently practice when advocating for change. Sponsorship from senior executives takes a step toward tangibly expressing this larger commitment and fostering an environment of trust. To that end, an organization should also examine its

[45]The best practice, “Technical environment enables Agile development”, discusses the need for a program to consider program management and technical support tools early in program planning. As part of these deliberations, the program should think about access to these tools and the level of transparency it might afford to stakeholders that are less active in the day-to-day operations of the team or program.

existing incentives and rewards systems and consider the extent to which they might interfere with or reinforce Agile behavior and make changes to bring those systems in alignment with Agile principles.

Changes to incentives and rewards systems may be slow and ineffective, thus preventing team cohesion and unity and restricting productivity unless there is active involvement from the appropriate organization entities, such as human resources and employee unions. To ease the transition, organizations should identify and include such entities early and establish an organization goal to align related incentives and rewards with Agile values and principles. For example, one step to achieve such an environment and demonstrate support from senior stakeholders is to establish appropriate incentives to work on Agile teams and offer rewards to teams that satisfy business needs. That is, rewards should be tied to accomplishments (e.g., working software) and not to the outputs of an Agile process.

Most organizations have incentives and rewards that focus on individual accomplishments. However, in an Agile environment, incentives should be established to supplement traditional individual rewards with those that also focus on team success. For example, the reward system should be closely related to achieving software and program goals. If organization rewards are not structured to promote team performance, competitiveness or a lack of respect among team members might increase, impacting team behavior, productivity, and outputs.[46]

The organization can use other mechanisms to reward team performance. For instance, rewards such as public acknowledgment by presenting a program's success story at conferences and other networking events and team access to certificate programs might be used to supplement individual-focused performance rewards. However, for such a rewards system to be effective, managers should understand the kinds of rewards that different individuals value and seek to reward successful teams accordingly. Structuring organization incentives to promote improved team performance and behavior will help productivity and outputs.

[46]There are certain awards that can be provided to federal employees and other forms of recognition available to recognize contractor employees. As a result, when awarding team success, a distinction may have to be made between federal and contractor staff.

Best practice: Organization acquisition policies and procedures support Agile methods

Guidance is appropriate for Agile acquisition strategies

The organization’s Agile acquisition policy and guidance should align with the planned acquisition strategies.

Before entering into any contract, the program office should analyze the risks, benefits, and costs associated with the acquisition. In a federal agency, this can be accomplished with acquisition planning as outlined in the *Federal Acquisition Regulation* (FAR) and other agency acquisition policy and guidance documents. For example, the Department of Defense has established the *Defense Federal Acquisition Regulation Supplement* (DFARS), which provides additional information for DOD programs as they implement the FAR. Additionally, FITARA grants the agency Chief Information Officer the authority to approve all information technology contracts, either directly or as part of active participation in agency governance.[47]

Mechanisms should also be in place in the contract and acquisition strategy to allow for close collaboration between the developers and stakeholders in order for everyone to agree on what features have the highest priority. In a commercial environment, the business workforce includes managers and customers of the product being developed. In the public sector, these roles may vary and span different organizations, not to mention the multiple business-related stakeholder roles to be allowed. These roles can include program office personnel, information assurance, logisticians, trainers, and others.

Further, the overarching acquisition strategy should match the program’s Agile cadence. While many contract types can be used to effectively support Agile development efforts, the way the contract is structured determines how effective it will be. Therefore, the contract structure and

[47]The law requires CIOs to review and approve IT contracts and OMB’s implementing guidance states that CIOs may review and approve IT acquisition strategies and plans, rather than individual IT contracts. 40 U.S.C. § 11319(b)(1)(C)(i).

the acquisition strategy need to support Agile implementation, such as by allowing for interim demonstration and delivery between official releases. In addition, Agile program contracts should specify the cadence of delivery and to what extent product demonstrations will be relied on to obtain customer feedback. These agreements can be defined as contract deliverables in the contract data requirements lists.

Accordingly, the contract should be structured to request frequent deliverables, rather than milestones that span several months, taking care to ensure that the deliverables meet the requirements. However, requirements should be written in such a way as to allow the government representative reviewing the deliverables for acceptance (e.g., the technical team in coordination with the product owner) enough flexibility to adjust requirements prioritization and the delivery schedule as the program evolves. If an acquisition strategy and contract structure do not allow for interim delivery and product demonstrations, then the organization may lose opportunities to obtain information and face challenges when adjusting requirements to meet and adapt to customer needs. This may negatively impact continuous delivery of software.

Contracts should be structured to align oversight reviews with Agile practices (e.g., frequent, interim deliverables and product demonstrations), frame the acquisition strategy to match the Agile cadence, allow for flexibility to refine detailed requirements, and encourage close collaboration between the developers and stakeholders.[48] The organization contract oversight mechanisms should also be aligned with Agile practices. In the federal government, contracts for large acquisition programs often mandate document-centered capstone reviews, such as preliminary design reviews and critical design reviews, which are based on an organization’s policies and guidance governing the system development life cycle. These reviews analyze requirements, preliminary design, and detailed design documentation; software coding does not typically begin until after all these documents have been approved following the critical design review. However, contracting language for Agile methods should enable incremental and frequent progress reviews at key points. If the organization does not adjust its oversight process to account for Agile methods, then there may not be adequate insight into the contractors’ productivity may decrease.

[48]The U.S. Digital Services’ TechFAR handbook offers guidance on how to acquire products and services in an Agile setting: https://playbook.cio.gov/techfar/. Guidance in the TechFAR handbook can be supplemented by the *U.S. Digital Services Playbook*: https://playbook.cio.gov/.

Contracting and the federal acquisition process are discussed in more detail in chapter 6.

Best Practices Checklist: Adoption of Agile Methods

Team dynamics and activities

1. Team composition supports Agile methods
   - Teams are self-organizing.
   - The role of the product owner is defined to support Agile methods.
2. Work is prioritized to maximize value for the customer
   - Agile teams use user stories to define work.
   - Agile teams estimate the relative complexity of user stories.
   - Requirements are prioritized in a backlog based on value.
3. Repeatable processes are in place
   - Agile programs employ continuous integration.
   - Mechanisms are in place to ensure the quality of the code being developed.
   - Agile teams meet daily to review progress and discuss impediments.
   - Agile teams observe end-iteration demonstrations.
   - Agile teams observe end-iteration retrospectives.

Program operations

4. Staff are appropriately trained in Agile methods
   - All program staff have appropriate training since the techniques used are different from those used for Waterfall development programs.
   - Developers and all other supporting team members have the appropriate technical expertise needed to perform their roles.
5. Technical environment enables Agile development
   - System design supports iterative delivery.
   - Technical and program tools support Agile.
6. Program controls are compatible with Agile

	• Critical features are defined and incorporated in development. • Non-functional requirements are defined and incorporated in development. • Agile teams maintain a sustainable development pace.
Organizational environment	7. Organization activities support Agile methods. • Organization has established appropriate life cycle activities. • Goals and objectives are clearly aligned. 8. Organizational culture supports Agile methods • Sponsorship for Agile development cascades throughout the organization. • Sponsors understand Agile development. • Organization has established an environment supportive of Agile development. • Incentives and rewards are aligned to Agile development methods. 9. Organizational acquisition policies and procedures support Agile methods • Guidance is appropriate for Agile acquisition strategies.

Chapter 4: Overview of Agile Execution and Controls

Once a program has adopted an Agile framework for developing its software, it should also apply effective practices for Agile execution and control. Effective program management can help programs achieve strategic goals and increases the likelihood that a program will deliver promised capabilities on time and within budget. Program management encompasses many disciplined practices needed to execute and oversee a program, including requirements development and management, acquisition strategy development, and program monitoring and control (e.g., cost and schedule estimating). This chapter provides a high level background for each of these three areas and chapters 5, 6, and 7 describe best practices for each area and how those best practices apply for an Agile program.

- **Requirements development and management.** Having a documented strategy for developing and managing requirements helps to ensure that the final product will function as intended. Developing the requirements includes planning activities, such as establishing program objectives to outline the course of action required to attain the desired end result and developing plans for understanding and managing the work. Effectively managing the requirements includes assigning responsibility for identifying the requirements and tracking their status as well as controlling refinements made to lower-level requirements. Doing so helps to ensure that each requirement traces back to the business need and forward to its design and testing. When done well, requirements management practices provide a mechanism for helping to ensure that the end product meets the customers' needs. Agile integrates planning with design, development, and testing to deliver small amounts of working software over a shorter time period, making requirements management an ongoing, continuous process versus a single phase in a series of processes.
- **Acquisition strategy development.** Among other things, acquisition strategies and solicitations for requirements where a contract will be awarded should define standard Agile terms so that both the government and contractor know what each term represents. OMB guidance specifies that all acquisition strategies and plans include principles that allow for adequate incremental development, which is defined as "Planned and actual delivery of new or modified technical functionality to users and occurs at least every six months."[49] The

[49]Office of Management and Budget, Memorandum M-15-14, *Management and Oversight of Federal Information Technology* (June 10, 2015), Attachment B: Definitions of Terms for the Purposes of this Guidance, "Adequate Incremental Development".

acquisition strategy is also where it is appropriate to establish expectations, such as the overall development cadence (e.g., iteration length, release length, synchronization activities among multiple teams) that should carry forward into the solicitation and resulting contract. In turn, Agile program contracts should be flexible enough to allow for lower-level requirements to be refined over time. These contracts should also provide the means for management to mitigate risks, track deliverables, and easily monitor contractor performance.

- **Program monitoring and control.** The ability to generate reliable estimates is a critical program management function. Typical estimates include cost and schedule estimates that are updated throughout the program's life cycle, forecasts of costs at completion for work in progress, and plans to establish an Agile work breakdown structure to identify discrete features that can be monitored.

At first glance, it might appear that applying these more traditional program management practices to an Agile program would be in conflict with the principles of the Agile Manifesto. However, existing Agile artifacts, such as the feature's lead and cycle time (as described in chapter 8), the number of defects discovered, and team velocity trends can be used to effectively oversee an Agile program in a real time fashion, allowing program management to quickly address risks and make better decisions. The following sections provide more details about each of these program management practices and refer to other chapters for more information, where applicable.

Overview of Requirements Development and Management

Agile methods integrate planning, design, development, and testing using an incremental life cycle to deliver small amounts of software to customers at frequent intervals. These frequent iterations provide program management with an effective way to measure progress continually, reduce technical and programmatic risk, and respond to feedback from stakeholders.

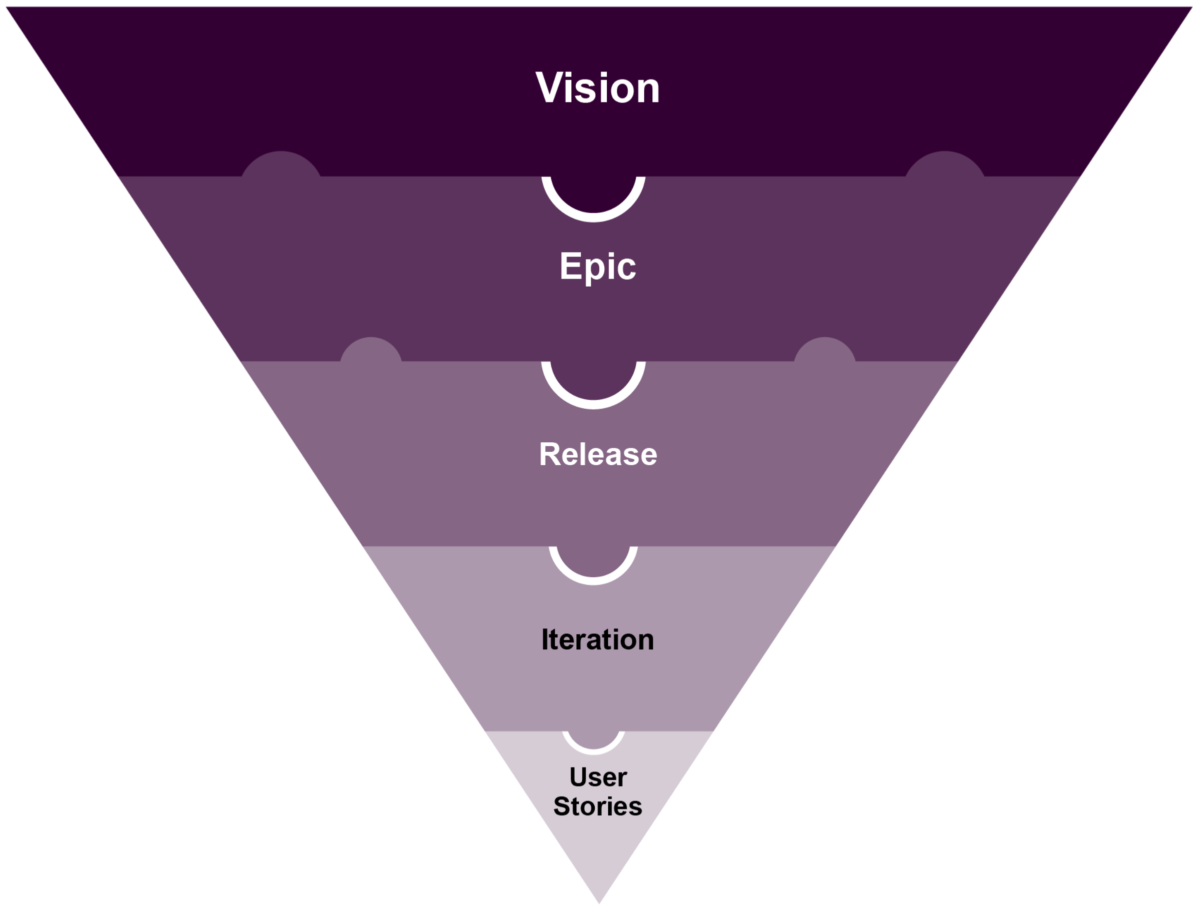
Agile teams typically embrace rolling wave planning in which near-term work is planned in detail, while all future work is identified at a high level.[50] Planning near-term work in detail provides the building blocks for constant updates from feedback and lessons learned that characterize Agile methods. However, the magnitude associated with requirements refinement must be confined to the scope of the capabilities in the

[50]GAO, *GAO Schedule Assessment Guide: Best Practices for Project Schedules,* GAO-16-89G (Washington, D.C.: Dec. 22, 2015).

program road map. Using an Agile approach is not and should not be viewed as an opportunity for boundless development.

All remaining work is summarized and documented in what is commonly referred to as an epic. As time passes and future elements of the program become better defined, epics are decomposed into features for release planning and user stories for iteration planning. This incremental cycle of rolling wave planning continues for the life of a program until all work has been sufficiently converted into user stories. Agile programs typically use five levels of planning to progressively define work, as illustrated in figure 4.

Figure 4: Agile Planning Levels



Source: GAO representation of Agile planning levels. | GAO-20-590G

The **vision** level provides a strategic view of the program goals expressed at a broad level so that the vision remains basically static and changes only infrequently; it is similar to a mission needs statement.

The **epic** level describes large concepts which, when developed, will move the program toward accomplishing the vision. An epic is useful as a placeholder to keep track of and prioritize larger ideas.

The **release** level provides the foundational structure for deploying needed capabilities to the operational community. It begins with a planning segment where the team prioritizes the requirements and establishes preliminary cost and schedule estimates. Releases occur in fixed intervals throughout the life of a program. An important difference exists between releases and deployments. A release is typically an internal hand-off of functioning code, whereas a deployment makes the functionality available to external stakeholders. For some commercial programs, a release may happen daily or even multiple times a day, though that is typically not the case for government programs.[51]

At the **iteration** level, the developer designs, codes, integrates, and tests whether the software provides working capabilities that satisfy the needs of the selected user stories.[52] More detailed planning done at the iteration level ensures that the Agile teams develop software that satisfies the customer's prioritized needs. An iteration should always be the same amount of fixed time, typically 2-4 weeks in length, so that a cadence can evolve.

The **user story** level is broken down into tasks that are the daily work of the teams.

Terminology

Agile programs may use different terminology when referring to the same things. For example, an epic can be referred to as a theme or high-level requirement; however, it is important that all members of an Agile program use the same terminology to avoid confusion.

[51]"Release" in the commercial community may not mean the same thing as in the government. In government settings, the working product at the end of a release may go to a certifier or independent test organization rather than directly to the end user.

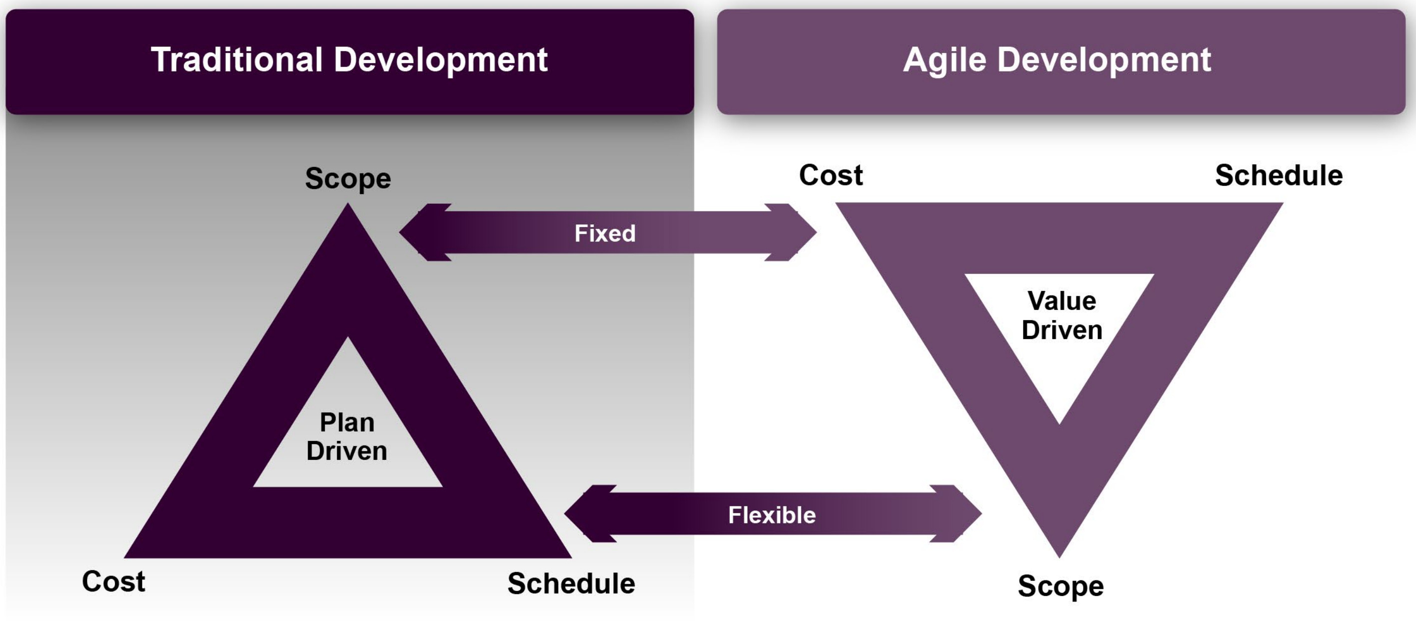
[52]Agile teams may assign a specific meaning to terms such as "iteration" and "release." We have used the terms in this guide as they are most commonly understood by Agile teams.

As discussed previously, Agile programs do not identify all of their low-level requirements up front; instead, the Agile team refines requirements by soliciting feedback from the customer. Because the product owner, as part of the Agile team, is very much involved in prioritizing and reviewing requirements that have already been developed, the risk that the team will produce requirements of little value diminishes. For each iteration, the Agile team focuses on creating only what provides the customer with value. Since software is developed in smaller increments, stakeholders can provide immediate feedback on demonstrated capabilities. Using this information, the team updates the program backlog so that it reflects desired updates.

Requirements are initially expressed as high-level capabilities in a program’s road map and are prioritized in the backlog on a regular basis. As the highest-priority capabilities are pulled from the backlog during each iteration, they are further refined based on customer feedback. As requirements get more specific, the team must ensure that full traceability to the business need remains apparent. In addition, the Agile software team is developing requirements and developing their test plans to determine acceptance criteria to confirm whether the chosen requirements have been satisfied at the end of the iteration at the same time.

As discussed in chapter 1, one of the key differences between a Waterfall development process and Agile development methods is that Waterfall starts by developing a plan for all requirements and ends when those requirements have been completed. Conversely, Agile starts by developing a high-level program goal and priority requirements and ends when the program goal has been met with an understanding from everyone involved in the program that the requirements will be refined over time as small segments of software are developed and presented to customers for feedback. In addition, program management tradeoffs are different for Waterfall and Agile development frameworks. In a Waterfall development, the requirements are fixed but schedule and cost are variable, while in Agile development, the program cost and schedule are fixed but the requirements are variable for each iteration. The different constraints associated with these two software development approaches are shown in figure 5.

Figure 5: Comparison of Traditional and Agile Development Program Management Constraints



Source: Federal Aviation Administration. | GAO-20-590G

Government programs generally do not have the autonomy to manage a completely flexible scope. If scope cannot be completely flexible, it is vital for teams and customers to understand and differentiate the requirements; that there are “must have” requirements that are different from the “nice to have” requirements early in the planning effort. Having a hierarchy will help facilitate delivery of the “must have” requirements first, thereby providing customers with the greatest benefits as soon as possible. See chapter 5 for more information on requirements development and management.

Overview of Acquisition Strategy Development

While there are numerous frameworks available to Agile practitioners, there are no standard terms for Agile processes and artifacts from the acquisition viewpoint. Therefore, when implementing Agile methods, the organization and the contractor must work together to define the Agile terms and processes that will be used during the development. These definitions will help establish common Agile terms that can aid everyone related to the program in understanding the relationship between Agile and program monitoring and control. Communicating this kind of information is often overlooked, especially as new employees join the program.

Chapter 6 addresses contracting in an Agile environment in greater detail, and discusses three best practices: (1) tailor the contract structure and inputs to align with Agile practices; (2) incorporate Agile metrics, tools, and lessons learned from retrospectives during the contract oversight process; and (3) integrate the program office and the developers. These best practices highlight that acquisition strategies should reflect contracts that are flexible enough to allow for lower-level requirements to be refined over time while allowing management to mitigate risks, track deliverables, and easily monitor contractor performance. As previously stated, reasonable risk in the contracting process is appropriate as long as risks are controlled and mitigated. These best practices help to mitigate those risks common to contracting in an Agile environment by tying the contracting process and an Agile approach together.

Overview of Program Monitoring and Control

There are several advantages that program monitoring and control documentation provide for an Agile program. First, since effort is commonly used as a proxy for cost, estimating effort can determine not only the program cost, but it can also reasonably predict how long both near-term and long-term deliverables will take to develop. Second, understanding capacity (or the total amount of work that Agile teams can accomplish in one iteration) helps prioritize work and predict the cost of a delay when “must have” features cannot be accomplished as expected. Finally, having the Agile team commit to near-term deliverables is important because those commitments materially affect customer planning and business objectives while at the same time make the developers accountable for their work.

Estimating is the key to unlocking the team’s ability to predict and commit what deliverables can be accomplished in the near-term. Therefore, while any cost estimate will always be based on the best information available at a given time, Agile program cost estimates have an advantage over traditional program cost estimates because they can be regularly updated to reflect new information in accordance with the program’s cadence. The regular cycle of iterations and releases provides numerous opportunities to continuously refine the estimate based on learning what the customer wants. Even so, it is important to bear in mind that a cost estimate is typically created or updated before financial commitments have been made and used to establish a performance measurement baseline. While the estimate should be updated regularly, the original baseline is only developed once. For example, the estimate at completion may be revised, but the original cost estimate should rarely be changed so that variances can be observed.

While Agile supports change and continuous process improvement, the program should quickly establish a regular cadence of time boxed releases and iterations so that teams can estimate the cost and time it takes to develop features with some degree of precision. Since both releases and iterations are time boxed, estimating the number of iterations in a release should be relatively straightforward. For example, if a program has a release every 12 weeks and iterations are two weeks long, then there should be six iterations for every release. After several iterations, program office personnel can track a team's cadence to better forecast the remaining effort.

Estimating the cost and time it will take to develop software is inherently challenging because not enough is known at the start about what exact requirements and functionality are going to be needed. As a result, requirements need to be iteratively fleshed out and may shift as the program evolves. Typically, developing an accurate estimate will be difficult until the team learns more about the program's requirements, For these reasons, cost and schedule estimates should always quantify the effect of changing assumptions using risk and uncertainty analysis. Additionally, it is important that managers and stakeholders understand that, because an Agile program's requirements will be iteratively determined, collaboration between the customer and developers is paramount.

Note that, even though Agile methods typically provide working code more quickly, this approach is often used as an excuse to avoid documenting traditional program management efforts like formal cost and schedule estimates. However, formal cost and schedule estimates remain important. The following case study provides an example of a review of the cost and schedule for an Agile program.

Case study 8: Cost and schedule estimating for an Agile program, from *FEMA Grants Modernization,* GAO-19-164

In April 2019, GAO reported that the Federal Emergency Management Agency (FEMA) Grants Management Modernization (GMM) program's May 2017 initial life cycle cost estimate was reliable; however, key assumptions made about the program had changed. Thus, the initial cost estimate no longer reflected the current approach for the program. Additionally, GAO found GMM's program schedule was inconsistent with leading practices. Of particular concern was that the program's final delivery date of September 2020 was not informed by a realistic assessment of GMM development activities but by imposing an unsubstantiated delivery date.

Key assumptions about the GMM program changed after the May 2017 cost estimate was approved, including a change in technical approach, an increase in the number of system development personnel, and significant delays and complexities with data migration. FEMA officials reported that they anticipated the cost estimate to increase as a result, and that this increase might be high enough to breach the $251 million threshold set in GMM's May 2017 acquisition program baseline. The program informed the DHS Acquisition Review Board of this anticipated breach, and, on September 12, 2018, the board declared that the program was in a cost breach status. In December 2018, program officials stated that they had completed a revised cost estimate using a new cost estimating methodology that was developed by DHS's Cost Analysis Division and tailored for Agile programs, but it was still undergoing departmental approval.

In addition to an outdated estimate, GAO found GMM's schedule to be unreliable. One of the most significant issues was that the program's final delivery date of September 2020 was informed by an unsubstantiated delivery date. Program officials stated that they had been uncertain about the level of rigor that should be applied to the GMM schedule, given their use of Agile development. However, leading practices state that program schedules should meet all the scheduling practices, regardless of whether a program is using Agile development. Program officials also stated that the delay in awarding and starting the Agile contract delayed other important activities. A more robust schedule could have helped FEMA predict the impact of delays on remaining activities and identify which activities appeared most critical so that the program could ensure that any risks in delaying those activities were properly mitigated.

We reported that establishing an updated cost estimate should help FEMA better understand the expected costs to deliver GMM under the program's current approach and time frames. However, without a robust schedule to forecast whether FEMA's aggressive delivery goal for GMM is realistic to achieve, leadership will be limited in its ability to make informed decisions on what additional increases in cost or reductions in scope might be needed to deliver a complete system.

GAO, *FEMA Grants Modernization: Improvements Needed to Strengthen Program Management and Cybersecurity,* GAO-19-164 (Washington, D.C.: April 9, 2019).

GAO has developed processes and best practices for program monitoring and control in formal guides available on its website. A summary of the two most relevant guides are included here.

GAO Cost Estimating and Assessment Guide: First released in 2009, the guide was revised using solicited comments and the new version was released in 2020.[53] The guide establishes a consistent methodology based on best practices that federal agencies can use for developing, managing, and evaluating program cost estimates. Best practices related to program monitoring metrics, such as earned value management, are also included. The importance of having a reliable cost estimate that reflects best practices cannot be emphasized enough because, as resources become scarce, competition for them will increase. It is imperative, therefore, that government acquisition programs deliver capabilities as promised, not only because of their value to their customers, but also because every dollar spent on one program will mean one less dollar available to fund other efforts.

GAO Schedule Assessment Guide: First released in 2016, the Schedule Guide is a companion to the Cost Guide.[54] Because a cost estimate cannot be considered credible if it does not account for the phasing of costs over time as well as the cost effects of schedule slippage, the guide provides an effective methodology for developing, managing, and evaluating program schedules. It draws on the scheduling concepts introduced in the Cost Guide and presents them as ten detailed best practices associated with developing and maintaining a reliable, high-quality schedule. The Schedule Guide also presents guiding principles for auditors to evaluate certain aspects of government programs.

While cost estimating, earned value management, and scheduling best practices apply to Agile development programs, there are some considerations that must be understood, such as recognizing that specific Agile documents may already contain metrics and data that can be mapped to traditional management tools to accomplish the same results. Chapter 7 will examine in more detail how program monitoring and control processes and best practices can be used in partnership with an Agile work breakdown structure and Agile principles to ensure a successful program.

[53]GAO, *GAO Cost Estimating and Assessment Guide: Best Practices for Developing and Managing Program Costs*, GAO-20-195G (Washington, D.C.: Mar. 12, 2020).

[54]GAO, *GAO Schedule Assessment Guide: Best Practices for Project Schedules,* GAO-16-89G (Washington, D.C.: Dec. 22, 2015).

Chapter 5: Requirements Development and Management in Agile

Sound management practices are critical for the success of any program, including one using incremental development methods such as Agile. These practices include establishing what the system is to do, how well it will perform those functions, and how it will interact with other systems.[55] GAO has developed a body of work that defines the activities and best practices used to develop and manage the requirements for a system development program.[56] This chapter identifies how traditional requirements development and management processes can be adapted for Agile programs and highlights key considerations when assessing compliance with policy and standards for requirements development and management.

For the purposes of this guide, we use the term 'requirements' to represent all development work because it is a generally understood concept from Waterfall development. However, in Agile development, the term requirement is rarely used. Instead, it is replaced with terms such as 'epic' or 'user story' and often represents a capability, feature, sub-feature, or more granular expectation for the system being developed. The specific terminology will be unique to each organization, which means it is important for the organization to be explicit in defining each term and applying that definition consistently within a team, program, or organization. The terminology will also be based on the duration of the work or planning exercise. For example, a feature or epic may be

[55]It is important to distinguish between development and acquisition when discussing requirements. Many federal programs rely on contractors to develop a system where the government is responsible for managing and evaluating the contractor's completion of requirements defined in a contract. Variability will often occur in the actual management of those requirements rather than the high-level requirements themselves. For example, in an acquisition, criteria are established to designate appropriate channels or official sources from which to receive requirements. Those who receive requirements conduct analyses of them with the provider to ensure that a compatible, shared understanding is reached on the meaning of requirements. The result of these analyses and dialogs is a set of approved requirements reflected in a contract. Chapter 6 offers further discussion of how to structure a contract to allow for requirements flexibility during development.

[56]GAO, *Framework for Assessing the Acquisition Function At Federal Agencies,* GAO-05-218G (Washington, D.C.: Sept. 1, 2005)*; Information Technology: Management Improvements Are Essential to VA's Second Effort to Replace Its Outpatient Scheduling System,* GAO-10-579 (Washington, D.C.: May 27, 2010)*; FEMA: Action Needed to Improve Administration of the National Flood Insurance Program,* GAO-11-297 (Washington, D.C.: June 9, 2011)*; Information Technology: Critical Factors Underlying Successful Major Acquisitions,* GAO-12-7 (Washington, D.C.: Oct. 21, 2011)*;* and *Defense Major Automated Information Systems: Cost and Schedule Commitments Need to Be Established Earlier,* GAO-15-282 (Washington, D.C.: Feb. 26, 2015).

discussed and committed to for a release, whereas an iteration may focus on the individual user stories that make up the feature or epic.

As discussed in chapter 4, Agile programs typically incorporate five levels of planning to progressively define all work. At the highest level, the vision provides teams with a top-level plan, while at the lowest level, the daily work reflects specific activities that team members can accomplish in a single workday. After establishing a vision, the program will typically elicit a preliminary set of very general operating requirements from all customers. The process for eliciting requirements could take the form of surveys, face-to-face communication, or a combination of different techniques. Requirements are often still vague after this exercise. In Agile, the requirements gathered at this phase are called epics and they are grouped into general themes.

An epic can help the program to reach agreement with governance bodies on the priorities for the larger objectives of the program. It is up to the organization to determine the level of specificity that requirements are committed to for each governance body and to weigh the benefits of added governance from, for example, an additional layer of review and approval.[57] A program may commit to a set of operating requirements with a department investment review board, refine capabilities with a component review board, detail features or sub-features within a component's or program's integrated program team, and define discrete user stories with a dedicated product owner. These commitments are then reflected in artifacts associated with those touch points, such as a program road map approved by an investment review board, a release plan associated with the component review board or lower-level integrated program team, and a backlog for management by the product owner.[58]

As an Agile program anticipates the development of a theme or epic in the near-term, the program should define the requirements into smaller efforts with more granularity so that the team can properly plan and

[57]In chapter 3 of this guide, we highlight the potential risks an organization may incur if it does not modify the acquisition and software life cycle processes to accommodate Agile methods.

[58]Requirements in Agile development can be thought of as both strategic and tactical. A set of strategic requirements are necessary to justify a program, and one can generally assign a work breakdown structure and some form of earned value management measurement to achieving these goals. The tactical requirements are the lower-level requirements capturing the features for which customers and stakeholders are looking.

execute the work. This process may occur at various levels and with different personnel, depending on the stage of requirements decomposition. However, the end goal of the program is to have a set of user stories that can be discussed and further understood by the Agile teams and the product owner on a routine basis.

Agile in Action 1: Requirements decomposition

In July 2016, we observed release planning for the National Nuclear Security Administration Program Management Information Systems, Generation 2 (G2) program. G2 used a requirements hierarchy that allows teams to plan for, manage, and execute a project. Officials said that this was helpful for clearly defining and communicating requirements from National Nuclear Security Administration stakeholders and customers through the federal program manager, product owners, and development team. According to documentation provided, the requirements hierarchy decomposed a project down into smaller, more manageable efforts. Specifically, there were four levels to G2's hierarchy: road map, feature, user story, and task, with specific periods of time associated with each level.

Officials said that the road map was the program's strategic vision, which provided release planning information for the current development cycle and next three cycles (three months of work, each). The road map was used to facilitate conversations with the program's multiple customers to define and time box desired system features. Features comprised level 2 of the requirements hierarchy. Requirements were captured as uniquely numbered features in the backlog; each feature was the starting point for estimating level of effort and requirements were approved for work at the feature level.

Documentation provided showed that Level 3 of the requirements hierarchy was composed of user stories. As features were entered in the backlog, they were decomposed into user stories (e.g., requirements that can be addressed in one iteration). Officials said that to ensure requirements traceability, as both features and user stories are entered, a work breakdown structure (WBS) number was assigned. Because of the widely varying scope of application requirements, a designated WBS numbering scheme (as defined in the G2 *System Requirements Specification*) was used. Tasks were level 4 of the requirements hierarchy. They were the detailed requirements that could be completed in one day and were assigned to one person to help maintain accountability. This four-level requirements hierarchy provided traceability for the requirements through all the program's planning documents, visibility for multiple customers engaged in the program, and accountability for the development team.

Agile values and principles provide guidance for the process an Agile team uses to develop and manage the requirements for a program. Agile does not provide a detailed, specific method to be used to perform these tasks and allows the team flexibility to choose a method. For example, a team may follow the Scrum concept of product backlogs consisting of ordered backlog items that are represented on a task board based on specific commitments made each iteration. Alternatively, a team may follow the Kanban concept of continuous flow and rely on a Kanban board that is not reset because Kanban deemphasizes the use of time boxed iterations.

Because Agile affords such flexibility in requirements development and management, each program will be unique, depending on the Agile framework it has adopted and the organization's governance requirements. This guide considers both product backlog items and user stories to be a form of requirements. The difference comes in the structure and expectations for communicating those requirements. In an Agile environment, the techniques, resulting work products, and frequency for each goal may change, impacting how an auditor might evaluate compliance with existing best practices. The following sections describe how a best practice might be modified in Agile and possible associated artifacts that can help a program to meet the intent of the best practice.

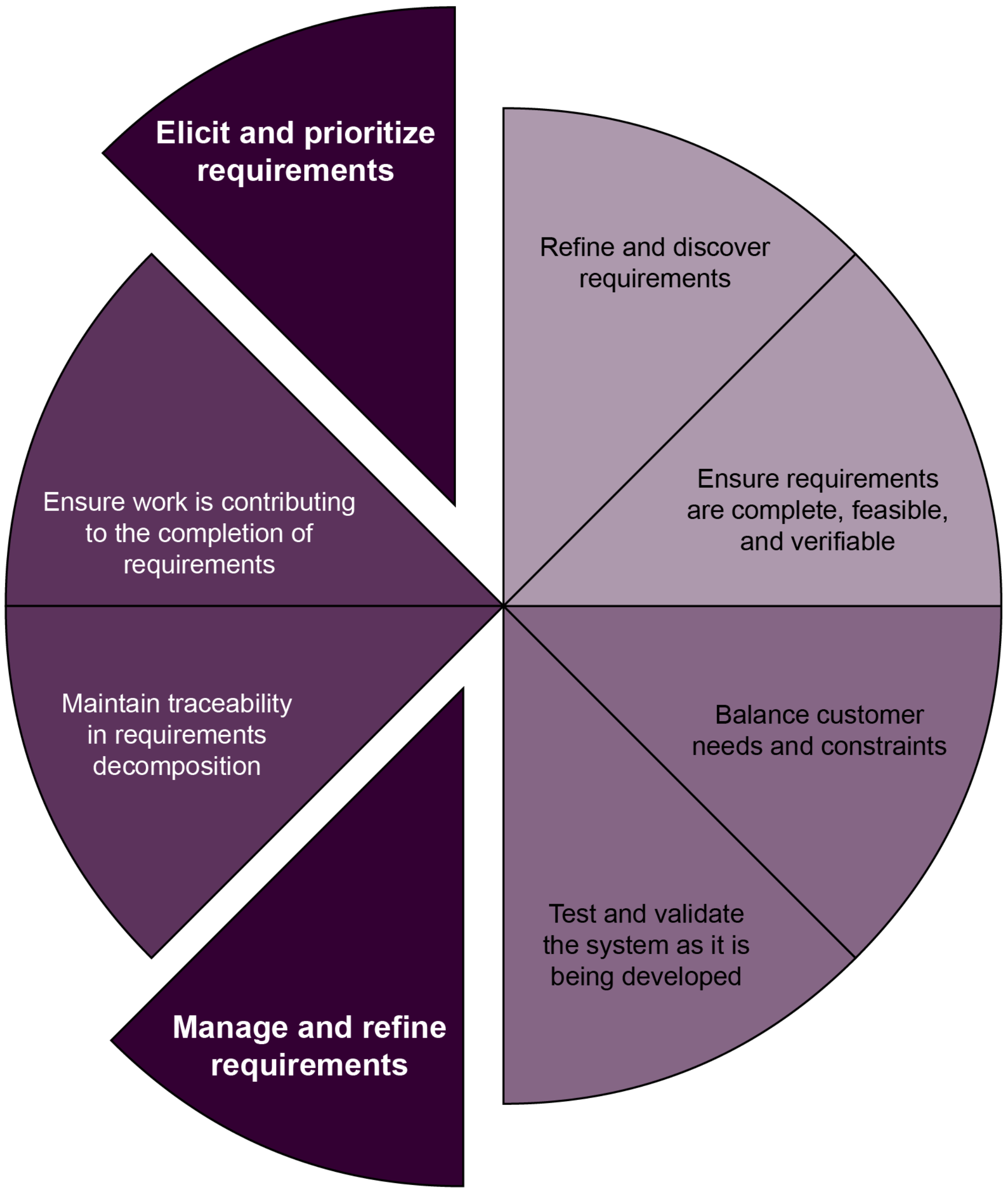
The following best practices will be discussed in this chapter:[59]

- Elicit and prioritize requirements.
- Refine and discover requirements.
- Ensure requirements are complete, feasible, and verifiable.
- Balance customer needs and constraints.
- Test and validate the system as it is being developed.
- Manage and refine requirements.
- Maintain traceability in requirements decomposition.
- Ensure work is contributing to the completion of requirements.

Figure 6 shows an overview of requirements management best practices and table 7 following the figure summarizes the best practices.

[59]These practices were developed as explained in appendix I.

Figure 6: Overview of Requirements Management Best Practices


Elicit and prioritize requirements
Refine and discover requirements
Ensure requirements are complete, feasible, and verifiable
Balance customer needs and constraints
Test and validate the system as it is being developed
Manage and refine requirements
Maintain traceability in requirements decomposition
Ensure work is contributing to the completion of requirements

Source: GAO analysis of CMMI, PMI, and SEI documentation. | GAO-20-590G

Table 7: Summary of Agile Requirements Management Best Practices

Best practices for Agile requirements management	Summary
Elicit and prioritize requirements	• A strong commitment exists to ongoing elicitation and refinement of new requirements to meet the changing needs of the customer and the evolving technical landscape while managing requirements already defined. • The process relies on surveys, forums, and other mediums in order to effectively understand the needs of the organization. • Non-functional requirements are accounted for using regulations or elicited through coordination with customers throughout the organization.
Refine and discover requirements	• Requirements are further refined as part of ongoing backlog refinement.
Ensure requirements are complete, feasible, and verifiable	• Prior to development, an overall definition of done and acceptance criteria for requirements are established. • A definition of ready may also be established as Agile teams work to set an expectation of the level of detail needed before teams can start development on a user story.
Balance customer needs and constraints	• A consistent process is in place to measure the value of work to ensure that user stories are developed based on relative value. • Backlog refinement is an ongoing, collaborative process between the product owner and the developers.
Test and validate the system as it is being developed	• Continuous integration and automated testing is used in the build process. • The product owner agrees and accepts the definition of done for each user story.
Manage and refine requirements	• Additions and refinements to requirements are managed efficiently and effectively in an evolving prioritized backlog. • The backlog contains functional and non-functional requirements and bugs or defects representing revisions to existing functionality.
Maintain traceability in requirements decomposition	• Requirements can be traced from the source requirement (e.g., feature) to lower level requirements (e.g., user story) and back again. • The program uses Agile artifacts, such as a road map, to ascertain requirements traceability.
Ensure work is contributing to the completion of requirements	• Agile teams are continuously working on tasks that directly contribute to the completion of user stories committed to for that iteration. • The product owner and Agile teams ensure that the committed user stories contribute to the commitments made to oversight bodies.

Source: GAO analysis of CMMI, PMI, and SEI documentation. | GAO-20-590G

Elicit and Prioritize Requirements

Officials can analyze and validate Agile program requirements through various tests; however, the amount of time devoted to the up-front planning and identification of the requirements will be much shorter than when using a Waterfall or another non-Agile development approach.[60] Instead of hardening all requirements at the outset of the program, Agile methods require a strong commitment to ongoing elicitation and refinement of requirements to meet the changing needs of the customer and the evolving technical landscape while continually managing the requirements that have already been defined. If there is not a strong commitment to ongoing elicitation and refinement of requirements, the delivered software may not meet the changing needs of the customer or address the evolving technical landscape.

The process for eliciting customer needs, expectations, and constraints that comprise the vision and the initial set of epics for an Agile program provides an opportunity for customer feedback. The process relies on surveys, forums, and other mediums to understand the needs of the organization. The overall vision for a program should not change over the life of it, but because detailed requirements remain flexible in an Agile program, ongoing elicitation can occur. Furthermore, an organization may have various levels at which requirements are defined and each layer of requirements might have a different approach to eliciting customer needs, expectations, and constraints, and a different process for prioritizing decisions. The minimum viable product (MVP) is a valuable tool to elicit feedback by demonstrating aspects to the developing solution.

Minimum viable product (MVP)

A concept popularized in Eric Ries' 2011 book, The Lean Startup, the MVP is a version of a working product that allows the team to learn from and interact with their customer with the least amount of effort.[61] An MVP allows the team to better understand their customers' needs and interests without committing a large number of resources or developing a completed product. If done correctly, the MVP can allow a team to refine the product early in development to ensure it meets customer' needs rather than later in development when updates might be expensive or cost-prohibitive. This could mean significant updates to the product or even abandonment of the product altogether, but ensures the team is working on a product that the customer actually wants. However, teams must remember that an MVP is only valuable if the product is sufficiently developed to allow for customer interaction and to elicit feedback and learning. The MVP should not simply represent the smallest piece of functionality.

[60]The importance of modifying the acquisition life cycle to accommodate flexible requirements is discussed further in chapter 3 under the practice "Organizational processes support Agile methods".

[61]Ries, Eric. *The Lean Startup: How Today's Entrepreneurs Use Continuous Innovation to Create Radically Successful Businesses.* New York, New York: Crown Publishing Group, 2011.

Stakeholders and customers will continue to propose modifications to the system (e.g., new lower-level requirements) in response to demonstrations of the functionality of the user stories. Reviews allow the organization to observe the system and communicate additional functionality or modifications to existing functionality for the developer. The product owner can capture this feedback in the backlog for consideration, even if the suggested functionality cannot be incorporated into the system. To do this, the program must have a process in place to field suggestions from customers interacting with the system. In doing so, the product owner should also proactively seek out customers to inform future requirements. If the product owner does not capture feedback from reviews for consideration, there is no historical record of proposed requirements or modifications for reference. The lack of a documented change control process could hinder decision makers' insight into the true value of delivered features.

Agile methods emphasize customer-facing requirements. These are requirements for the system to perform a specific function, such as the ability to search information or aggregate data. However, when the focus on customer functionality becomes exclusive, the underlying system (non-functional) requirements can go unnoticed.[62] For example, when building out a search function, the team may not account for potential privacy issues associated with access to customers' data. Non-functional requirements can be derived from regulations or can be elicited through coordination with customers from other divisions within the organization, such as security or privacy groups. As with functional requirements, non-functional requirements will be added, modified, or removed over time based on ongoing communication between the product owner and customers.

There are several options for capturing non-functional requirements. One option is to define each discrete requirement as a separate user story that traces to a non-functional feature such as architecture. Another option is to continue building the "definition of done" or acceptance criteria for each functional requirement to include the non-functional requirements. For example, a product owner might require the developers to demonstrate that they have successfully load or stress tested a piece of functionality in the pre-production environment before accepting the user story as complete. Due to time and resource constraints, a team or program may

[62]Non-functional requirements are discussed in chapter 3 under the practice "Technical environment enables Agile development".

adopt the practice of testing some of the non-functional requirements outside of the iteration. For example, although unit, integration, and functional testing may be required prior to user story acceptance, an architecture team may test performance and customer satisfaction separately just prior to a full release.

Refine and Discover Requirements

In an Agile environment, refining and discovering requirements will heavily overlap with the elicitation of customer needs and the prioritization of customer requirements. Due to the ongoing flexibility in requirements, Agile teams may employ more unique mechanisms, such as the use of visualization tools. However, the approach to requirements discovery will vary by team, program, and organization.

Customer requirements are further refined in the backlog as part of ongoing backlog refinement. Because requirements are the least understood at the outset of an Agile program, programs are expected to learn as they progress through development. In order to take advantage of this learning, a program can incorporate newly discovered requirements or eliminate requirements previously thought to be essential. If Agile programs do not learn to discover and refine requirements throughout the development process, a program may miss an opportunity to incorporate newly discovered requirements or eliminate requirements previously thought to be essential, which could create a disconnect between deployed software and the customer's needs. The concept of backlog refinement is addressed in our discussion of other practices in this chapter. The following case study is an example of an organization refining its backlog.

Case study 9: Backlog refinement, from *TSA Modernization,* GAO-18-46

In October 2017, GAO reported that the Transportation Security Administration's Technology Infrastructure Modernization (TIM) program was expected to manage a backlog for each software release. The backlog was to identify features and their derived user stories (the smallest and most detailed requirements) that were to be delivered in a specific release. Each feature and user story was to be assigned a priority level to determine the order for development of the next release and associated sprint.

GAO found the program's backlogs did not contain prioritization levels for each of the features and user stories, as called for in Department of Homeland Security (DHS) guidance. According to program officials, instead of assigning specific prioritization levels, they identified which features should be developed within the near term (e.g., in the next several Agile releases). Program officials recognized that they still needed to prioritize their backlogs by assigning priority levels to all features and user stories, but they did not have a time frame for completing this effort.

Without ensuring full prioritization of current and future features and user stories, the program was at risk of delivering functionality that was not aligned with the greatest needs of the customers, who were responsible for conducting security threat assessments to protect the nation's critical transportation infrastructure.

GAO, *TSA Modernization: Use of Sound Program Management and Oversight Practices is Needed to Avoid Repeating Past Problems,* GAO-18-46 (Washington, D.C.: October 17, 2017).

Ensure Requirements are Sufficiently Complete, Feasible, and Verifiable for the Current State of the Program

Prior to development, the team is expected to define, overall, what completion, or "done," is for that team. If there are multiple teams working on the system or product release, the teams should also agree on a mutual definition of done. As teams mature, their definitions of done will become more comprehensive. However, not having clear criteria and an established definition of done allows uncertainty into the development process.

Case study 10: Definition of done, from *Agile Software Development,* GAO-20-213

In June 2020, GAO reported that the Department of Homeland Security (DHS), in guidance available to programs on requirements engineering, highlighted that acceptance criteria defines the boundaries of a user story and confirms when a story has been completed and is working as intended. Further, the definition of done identifies all of the activities/artifacts besides working code that must be completed for a feature or sub-epic to be ready for deployment or release, including testing, documentation, training material development, certifications, etc.

Within DHS, the U.S. Immigration and Customs Enforcement (ICE) Student and Exchange Visitor Information System (SEVIS) program generally followed this guidance with most of its user stories including acceptance criteria. The program also developed a “definition of done” for all user stories. According to the definition, a user story was “done” when the following steps had been addressed:

- All code to meet the story’s needs was written according to the system’s development standards.
- Unit tests were written and run successfully.
- All code was checked in and the build completed successfully.
- All database changes (if required) were complete and checked in (a functional test could be run).
- The software had been deployed to the system test environment and passed system tests.
- The product owner agreed that the implementation met the acceptance criteria written in the story as appropriate.
- All documentation required to support the story was completed (test cases, interface updates, etc.).

GAO, *Agile Software Development: DHS Has Made Progress in Implementing Leading Practices, but Needs to take Additional Actions,* GAO-20-213 (Washington, D.C.: June 1, 2020).

In addition to a definition of done and acceptance criteria, Agile teams may also use a “definition of ready” for user stories. A definition of ready sets expectations for the level of detail required before a team begins work on that user story. For example, the team may agree that no work on a user story can begin until it estimates the relative complexity of the user story and defines the acceptance criteria for the user story. Since detailed requirements evolve throughout the program, a definition of ready helps to ensure that participants work on only the most current and prioritized requirements and that those requirements always reflect updates to plans, activities, and work products. Without clear definitions for ready, acceptance, and done, the team may be working inefficiently and on requirements that are not high priority.

Spike

As requirements evolve and an Agile team begins to decompose, prepare for, and estimate user stories, there can be instances where the user story is challenging to estimate. This might be due to design questions or a technical challenge that the team is not experienced in working through. Derived from eXtreme Programming (XP), a spike can serve as a placeholder user story that represents the research a team needs to undertake in order to better understand a user story and thereby more effectively estimate its size.

Balance Customer Needs and Constraints

Waterfall development sets an expectation that all requirements are established at the start of the program and their value is relatively fixed. In Agile, where requirements are continuously being discovered and refined, the program is continually developing functionality to match the requirements. In doing so, the program developer can maintain flexibility and offer the option, at any point, for the organization to end the program if it feels the system is not meeting the original vision and the needs of its customers or if external constraints require that the program be discontinued. Moreover, the agency may determine that enough incremental value has been delivered that the system is meeting agency needs and the remaining features are no longer necessary.

Case study 11: User story prioritization, from *DHS Acquisitions,* GAO-20-170SP

In December 2019, GAO reported that, in November 2018, Department of Homeland Security (DHS) leadership approved the Transportation Security Administration's (TSA) Technology Infrastructure Modernization (TIM) program's request to descope and change its definition of full operational capability (FOC) to include only the Transportation Worker Identification Credential (TWIC) and TSA Pre√® capabilities. By the time TIM had fully delivered capabilities for TWIC and TSA Pre√®, TSA had made ongoing updates and improvements to the remaining legacy vetting and credentialing systems to meet security and mission demands, which had also sufficiently met end user needs. According to TSA officials, any additional system development would produce redundant functionality.

The program updated its key acquisition documents, including its acquisition program baseline and life-cycle cost estimate to reflect the change in scope. In July 2019, DHS leadership approved the program's revised acquisition program baseline. DHS leadership granted the program acquisition decision event 3 and acknowledged the program's achievement of full operating capability—fulfilling TSA Pre√® and TWIC mission needs for vetting and credentialing—in August 2019. We reported that DHS attributed a $220 million decrease in the program's baseline acquisition cost goal to this scope decrease; however, the program's operations and maintenance cost goals increased by $205 million. This increase was primarily due to maintenance of legacy systems to address user needs.

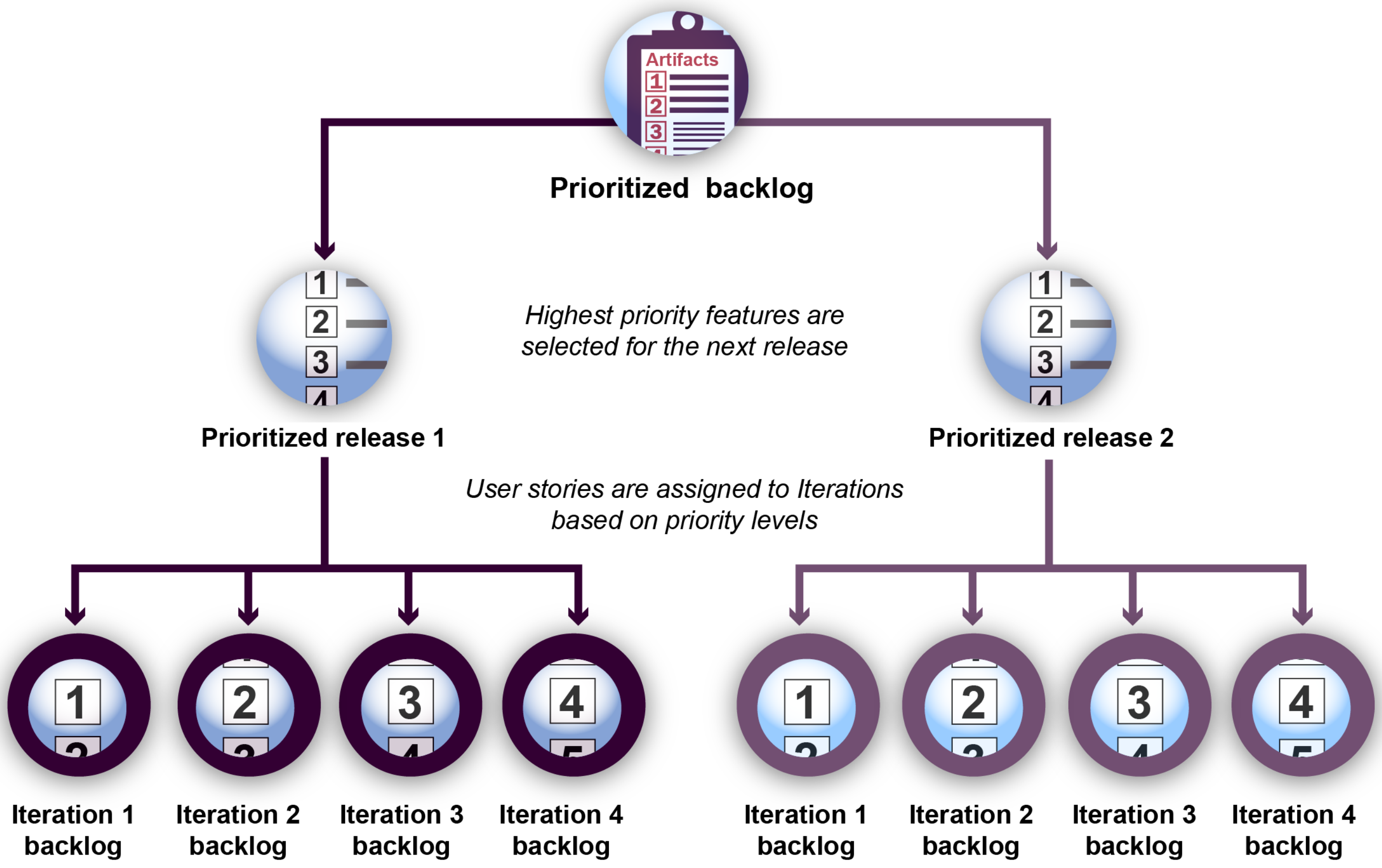
GAO, *Homeland Security Acquisitions: Outcomes Have Improved but Actions Needed to Enhance Oversight of Schedule Goals,* GAO-20-170SP (Washington, D.C.: December 19, 2019).

The value of individual requirements is subjective and, in Agile, its determination is often left up to the product owner. The product owner should have some consistent process for calculating the value of work and ensuring that user stories are being developed based on relative value (e.g., that the work is prioritized based on its value to the customer). For example, a product owner may choose to value high-risk work early in a release to mitigate the likelihood of encountering delays later in development that can require substantial re-work. Alternatively, a developer may prioritize work based solely on resource availability with regard to time, money, or staff. Other times the work will be valued based on a holistic consideration for cost, complexity, risk, availability of staff, or any number of other categories. Each consideration represents the developer balancing organization needs and constraints.

The product owner reviews and prioritizes user stories in a backlog based on the relative value of each user story at a specific point. As part of backlog refinement, the product owner adds detail, estimates, and orders the user stories based on priority in the backlog. The Agile team, or at times the entire program, decides how and when refinement is to be performed. However, user stories can be updated at any time at the discretion of the product owner. Suggestions from organization personnel should also be incorporated into the backlog and considered by the product owner.

Higher priority user stories are usually clearer and more detailed than lower priority user stories. More precise estimates are made based on the greater clarity and increased detail of a requirement; the lower the order, the less detail. Figure 7 illustrates the concept of backlog prioritization.

Figure 7: Backlog Decomposition for an Agile Program



Source: GAO analysis of Transportation Security Administration documentation. | GAO-20-590G

Problems can arise if the product owner does not consider the relative value of the work: all of the user stories can end up being developed just prior to deployment. While there are situations where this can occur, such as a very mature requirements decomposition process with an experienced product owner, often this is a sign that the product owner is not prioritizing the requirements and is developing functionality that is not immediately necessary. This practice of developing each and every user story can lead to problems if funding is reduced mid-iteration, mid-release, or mid-program or other external factors impede the progress of the development work. Further, when the product owner does not consider the relative value of work, the team may develop functionality that is not immediately necessary to meet customer needs. If the highest value requirements are not completed first, the customer may be left without necessary functionality. The best practice is to rank order requirements with those of the highest value being completed first so that if funding ends, the customer will still benefit from the work that has been completed to date.

Test and Validate the System as it is Being Developed

In an Agile environment, teams routinely build and test the software through continuous integration and automated testing.[63] Continuous integration merges all developer working copies to ensure they function as intended through an automated process by repeatedly integrating the code multiple times a day. However, continuous integration is only as strong as the automated testing used in the build process. If a build fails, the developer should address the issue and resubmit the code for continuous integration. Once successfully built and adopted into the code base, the developer and organization can gain confidence that the code will execute properly in the future.

Code may not meet the requirements of the original user story even if the quality of the code is good. Then, as part of the backlog refinement process, the team establishes the definition of done and defines acceptance criteria for each user story, so that the developers and product owner have a shared understanding of what it means for a piece of work to be considered complete. The definition of done encapsulates both the completion of acceptance criteria and the completion of additional activities, such as testing or compliance checks. User story acceptance criteria is specific to just one user story and documents the

[63]In chapter 3 of this guide, we highlight the potential risks an organization and program may incur if the organization does not stand up an environment for automated testing and instead relies on manual tests.

product owner requirements that must be met, whereas the definition of done applies to all user stories. In order to validate that requirements have been met, the product owner should identify acceptance criteria for every user story prior to development of the story (often as part of backlog refinement or planning for an iteration) and the program should agree on a definition of done (e.g., must meet acceptance criteria and be 508 compliant).[64]

The acceptance criteria and definition of done constitute the expectations for the user story against which the requirement will be validated and either accepted or rejected by the product owner. Depending on the nature of the acceptance criteria, this may require manual interaction with the system by the product owner and/or organization. Validation of a user story is performed either as part of a user story demonstration or as part of a review at the end of each iteration. Although the product owner is ultimately responsible for the user story, such demonstrations and reviews allow other customers to observe the functionality and weigh in on whether it meets the intended purpose or requires further refinement. Just because a product owner accepts a user story as complete does not mean that it has been adequately tested according to traditional standards for testing in order to fully validate the requirement.[65] If customers are not involved in the review and acceptance process for software functionality, the software may not meet the intended purpose required by the customer.

Manage and Refine Requirements

Detailed requirements can change as work proceeds and new requirements are defined. As with developing requirements at the start of a new program, it is important that the additions and refinements are managed efficiently and effectively. In Agile, there will be less formality around the refinements process as a program has flexible lower-level requirements and Agile empowers the product owner to prioritize requirements as necessary. Agile does not prescribe how a product owner should elicit requirements or order and refine the backlog. Instead, the product owner selects a process that allows them to maximize the value of software delivered during each iteration. If this process is too inflexible, it becomes a change prevention process and user needs will not be adequately incorporated into the program, making it less useful to

[64]Section 508 of the *Rehabilitation Act of 1973*, as amended, 29 U.S.C. § 794d, requires federal agencies to make their electronic information accessible to people with disabilities.

[65]The level of testing will depend on the product being developed and the rigor defined in the agreed-on definition of done.

customers than intended. However, if this process is too flexible, then boundless development can occur and the organization may not receive the full value that it requires. Chapter 6 discusses how this can be managed from a contracting perspective.

As previously discussed, requirements are maintained in the prioritized backlog for an Agile program. However, a backlog is never complete; it constantly evolves to meet new requirements. The earlier backlogs lay out the initially known and best-understood requirements. As the backlog evolves, the system being developed and the processes governing development are better defined. As long as a program exists, its backlog will contain user stories representing discrete pieces of new functionality to be developed and bugs or defects representing revisions to existing functionality. User stories may represent both functional and non-functional requirements.

Maintain Traceability in Requirements Decomposition

When requirements are managed well, they can be traced from the source requirement to lower-level requirements and from those lower-level requirements back to the source requirement. Such traceability helps to determine whether all source requirements have been completely addressed and whether all lower-level requirements can be traced to a valid source.

Agile considers only the work without regard to the terminology or hierarchical structure used to define that work (e.g., capability versus feature versus sub-feature). However, the product owner must justify to oversight groups the value that is being developed each iteration. This means tracing a user story back to its high-level requirements that the program committed to with oversight bodies. Without such traceability, a program cannot justify whether it is meeting the commitments made to various oversight bodies and, in turn, cannot establish whether the work is contributing to the goals of the program and thereby providing value.

In a Waterfall development, traceability is demonstrated through a requirements traceability matrix. In lieu of a requirements traceability matrix, Agile development requirements can be traced through Agile artifacts, such as the backlog.

Case study 12: Requirements traceability, *from Agile Software Development,* GAO-20-213

In June 2020, GAO reported that the Department of Homeland Security (DHS) guidance on requirements engineering recognized that, as a program progressed through the acquisition and systems engineering life cycles, it was important to trace requirements from the top-level mission needs or capabilities and/or business requirements down to the system/sub-system, component, or configuration item level that enabled those requirements to be met. This helped ensure continuity across various DHS artifacts, such as the program's mission needs statement, concept of operations, and operational requirements document, to vendor specifications (or applicable equivalent artifacts). This guidance recommended a series of artifacts that an Agile program could develop to ensure this traceability.

Within DHS, GAO reported that the U.S. Immigration and Customs Enforcement (ICE) Student and Exchange Visitor Information System (SEVIS) program generally followed this guidance. The program developed user stories based on business capabilities and other requirements as determined by the product owner and the business stakeholders. The program's operational requirements document described eight business capabilities that represented core SEVIS functions. According to ICE SEVIS officials, these business capabilities were addressed through user stories, and there was traceability in the backlog from user stories to epics to business capabilities/operating requirements.

GAO, *Agile Software Development: DHS Has Made Progress in Implementing Leading Practices, but Needs to take Additional Actions,* GAO-20-213 (Washington, D.C.: June 1, 2020).

Ensure Work is Contributing to the Completion of Requirements

Agile focuses on iterations and the extent to which working software is delivered rather than on plans and work products.[66] Each iteration, teams are expected to deliver software in accordance with a goal. As such, an Agile team should always be working on tasks that directly contribute to completing the user stories committed to for that iteration.[67] Any work not associated with those commitments (e.g., a tiger team initiated to fix an issue for an unrelated team) is a misalignment between the requirements and work and presents a risk to the program.

[66]As discussed earlier in this guide, teams applying the Kanban method will not rely on iterations to time box development work. Instead, these teams will pull in new user stories on a flow basis as user stories already being developed are completed.

[67]As the Kanban method does not use time boxed development, teams using the Kanban method for development will not make commitments each iteration. However, teams will still rely on a Kanban board and all work should contribute toward completing a user story on that Kanban board.

From a high-level planning perspective, programs will make commitments to oversight bodies. As part of those commitments, teams should prepare the streamlined artifacts required by oversight bodies. At least one of these artifacts will require the phases and overall structure of program development to be defined. It is then contingent on the team, and primarily the responsibility of a product owner during development, to ensure that the user stories contribute to the commitments made to the oversight bodies. For example, a management plan may discuss development in phases and a series of projects within each phase. If the schedule of projects and phases and the scope of each project are defined and committed to in advance, there should be alignment between the user stories being developed and the scope of a specific project. In an Agile program, a management plan can take the form of a program or release road map, whereby capabilities or features for development are laid out in a timeline and planned for future iterations.[68]

Best Practices Checklist: Requirements Development

1. Elicit and prioritize requirements
    - There is a strong commitment to ongoing elicitation and refinement of lower-level requirements to meet the changing needs of the customer and the evolving technical landscape while managing requirements are already defined.
    - The process relies on surveys, forums, and other mediums to effectively brainstorm the needs of the agency.
    - Non-functional requirements are accounted for using regulations or elicited through coordination with customers throughout the organization.
2. Refine and discover requirements
    - Requirements are further refined as part of ongoing backlog refinement.
3. Ensure requirements are complete, feasible, and verifiable
    - Prior to development, an overall definition of done and acceptance criteria for requirements are established prior to development.
    - A definition of ready may also be established as Agile teams work to set an expectation of the level of detail needed before developers can start development on a user story.

[68]In chapter 7, we highlight how Agile programs estimate cost and schedule. This chapter discusses how requirements are defined and decomposed in order to create an overall plan for the program.

4. Balance customer needs and constraints
    - A consistent process in place to measure the value of work to ensure that user stories are developed based on relative value.
    - Backlog refinement an ongoing, collaborative process between the product owner and the developers.
5. Test and validate the system as it is being developed
    - Continuous integration and automated testing are used in the build process.
    - The product owner agrees to and accepts the definition of done for each user story.
6. Manage and refine requirements
    - Additions and refinements to requirements are managed efficiently and effectively in an evolving prioritized backlog.
    - The backlog contains functional and non-functional requirements and bugs or defects representing revisions to existing functionality.
7. Maintain traceability in requirements decomposition
    - Requirements can be traced from the source requirement (e.g., feature) to lower level requirements (e.g., user story) and back again.
    - The program uses Agile artifacts, such as a road map, to ascertain requirements traceability.
8. Ensure work is contributing to the completion of requirements
    - Agile teams are continuously working on tasks that directly contribute to the completion of user stories committed to for that iteration.
    - The product owner and Agile teams ensure that the committed user stories contribute to the commitments made to oversight bodies.

Chapter 6: Agile and the Federal Contracting Process

Agile programs depend on using lessons learned from one release to the next and should have flexibility to add staff and resources to adapt. Federal procurement practices used for Waterfall programs can be adapted to support this flexibility for Agile programs. The *Federal Acquisition Regulation* (FAR) was established for the codification and publication of uniform policies and procedures for use by all executive branch organizations in acquiring goods and services.[69] The FAR helps organizations ensure that contracts deliver, on a timely basis, the best value product or service to the customer. Prior to entering into a contract for information technology, organizations should analyze the risks, benefits, and costs involved.

What does the FAR say?

"The FAR outlines procurement policies and procedures that are used by members of the Acquisition team. If a policy or procedure, or a particular strategy or practice, is in the best interest of the Government and is not specifically addressed in the FAR, nor prohibited by law (statute or case law), Executive order or other regulation, Government members of the Team should not assume it is prohibited. Rather, absence of direction should be interpreted as permitting the Team to innovate and use sound business judgment that is otherwise consistent with law and within the limits of their authority. Contracting officers should take the lead in encouraging business process innovations and ensuring that business decisions are sound."

FAR § 1.102-4(e)

Contracts for Agile development must likewise be consistent with FAR guidance. While the FAR does not specifically discuss Agile development, it does discuss contracting approaches that can be beneficial for Agile development efforts. For example, the FAR implements authority to use modular contracting, a contracting method intended to reduce program risk and incentivize contractor performance while meeting the government's need for timely access to rapidly changing technology.[70] Similar to the Agile principle to deliver working software at intervals, modular contracting may be divided into several smaller acquisition increments.

[69]While the FAR applies to executive branch agencies, not all agencies are subject to the FAR. For example, the FAR does not apply to the Federal Aviation Administration pursuant to 49 U.S.C. § 40110(d)(2)(G).

[70]Modular contracting was established in 41 U.S.C. § 2308 and is implemented in section 39.103 of the FAR.

The FAR also authorizes the use of simplified procedures for the acquisition of certain commercial items that fall between specified dollar thresholds. This is intended to maximize efficiency and economy, and to minimize burden and administrative costs.[71] In addition, OMB and GSA have developed guides to help organizations apply the flexibility offered by the FAR to facilitate the use of Agile practices. For example, OMB issued the TechFAR handbook, which highlights flexibilities in the FAR that can be used in partnership with the "plays" from the *Digital Services Playbook.*[72]

As discussed in chapter 2, one challenge the federal government faces for Agile adoption is ensuring that acquisition strategies and contract structures truly support Agile programs. For example, government contracts may be designed with heavily structured tasks and performance checks that are not necessarily aligned with a program's Agile methods or cadence. These structured tasks can slow down the program's Agile cadence by establishing long contract timelines and costly change requests that can cause major hurdles in executing Agile development. Furthermore, contracts without the flexibility to add staff and other resources needed to meet the work planned for each release or that cannot adapt to updates from one release to the next can work counter to Agile adoption best practices and negatively impact a program's ability to perform well.

As discussed in chapter 3, long timelines to award the contract and costly change requests are major hurdles in executing Agile programs, which require frequent releases. Rather than avoiding using Agile for development or relying solely on contracting methods that clash with Agile development, organizations can mitigate their risks by ensuring the contract supports Agile methods.

As with any contract, the government must determine the appropriate contract vehicle based on its assessment of risk and the extent that such risk will be shared with the contractor. Accepting reasonable risk in contracts for IT is appropriate as long as risks are controlled and mitigation processes are put in place. Risks can include schedule

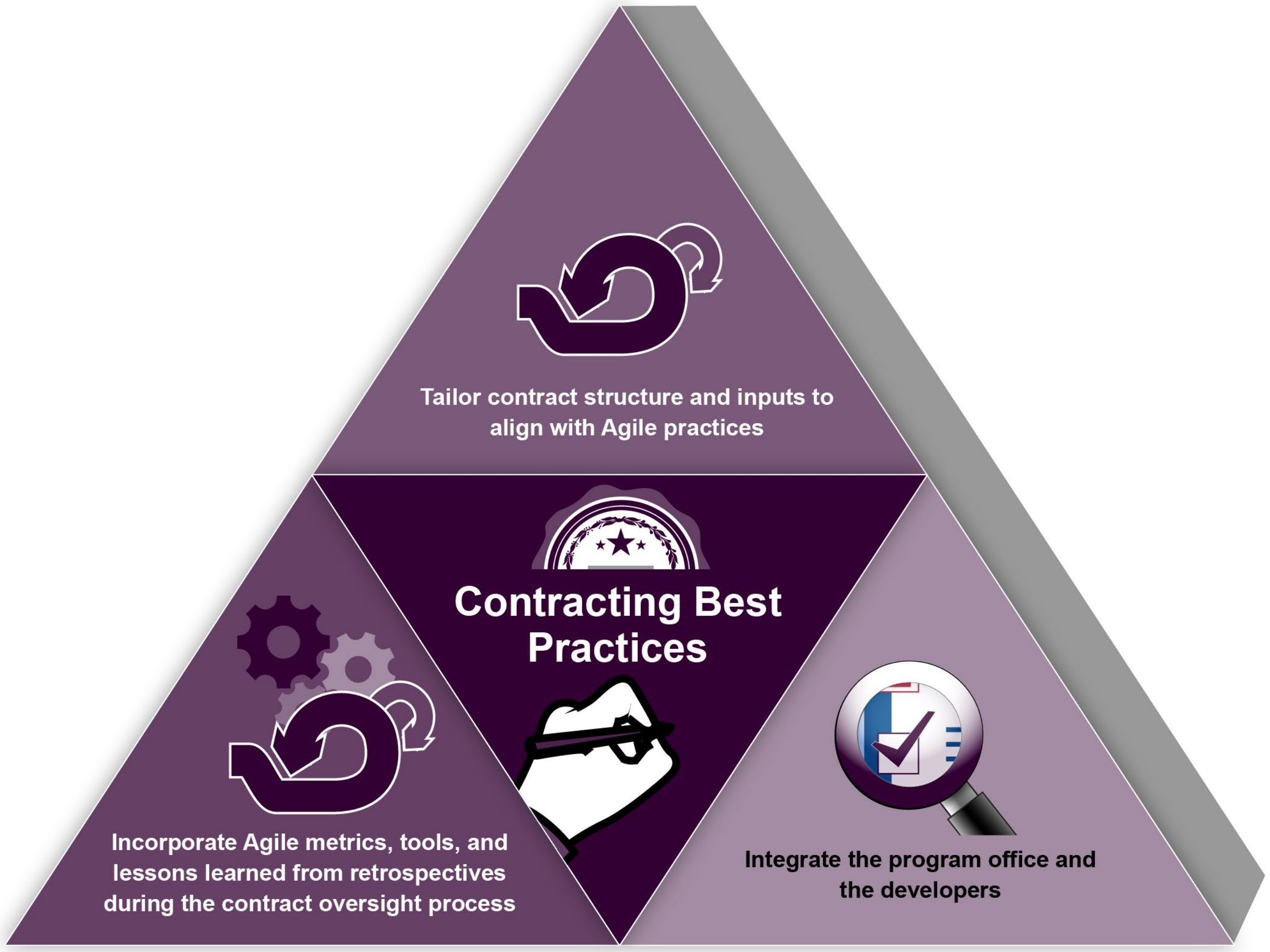
[71]FAR §13.500.

[72]The U.S, Digital Services TechFAR handbook offers guidance on how to acquire products and services in an Agile setting: https://playbook.cio.gov/techfar/. Guidance in the TechFAR handbook can be used in partnership with the U.S. Digital Services Playbook: https://playbook.cio.gov/.

problems, technical feasibility, dependencies between a new program and other programs, the number of simultaneous high risk programs to be monitored, funding availability, and program management issues. While all risks cannot be controlled, the best practices in this chapter highlight aspects of contracting for Agile IT acquisitions to help address key risks that should be considered when awarding and monitoring a contract.

Figure 8 shows an overview of acquisition best practices and table 8 following the figure summarizes the best practices.

Figure 8: Overview of Agile and Contracting Best Practices


Tailor contract structure and inputs to align with Agile practices
Contracting Best Practices
Incorporate Agile metrics, tools, and lessons learned from retrospectives during the contract oversight process
Integrate the program office and the developers

Source: GAO. | GAO-20-590G

Table 8: Summary of Agile and Contracting Best Practices

Contracting best practice	Summary
Tailor contract structure and inputs to align with Agile practices	• Encourage the use of modular contracting. • Enable flexibility for the contracts requirements. • Decide to structure the contract for goods or services.
Incorporate Agile metrics, tools, and lessons learned from retrospectives during the contract oversight process	• Ensure that contract data requirements rely on Agile metrics • Enable contract oversight through data from the program's Agile artifacts • Conduct retrospectives to continually improve Agile methods based on lessons learned. • Ensure that contract oversight reviews align with the program's Agile methods and cadence.
Integrate the program office and the developers	• Train program office acquisition, and contracting personnel. • Identify clear roles for contract oversight and management. • Ensure that all personnel are familiar with the contract's scope.

Source: GAO. | GAO-20-590G

Tailor contract structure and inputs to align with Agile practices

Encourage the use of modular contracting

An organization's contracting process must be deliberate and well executed to support regular program delivery timelines. Contracting strategies, processes, and the culture should create a business environment that supports small, frequent releases and responds to change, taking into consideration programmatic risks and the scope and purpose of a program (e.g., whether it is a large weapon system or small web application). One technique to accomplish this is called modular contracting. Modular contracting is when an organization's need for a system is satisfied by successive acquisitions of interoperable increments.[73] It is intended to reduce program risk and to incentivize contractor performance while meeting the government's need for timely access to rapidly changing technology.[74]

Agile development is designed to provide usable capabilities rapidly. Use of modular contracting practices can help an organization achieve these

[73]41 U.S.C. § 2308(b).

[74]FAR § 39.103(a).

compressed time frames by eliminating the costly lag between when the government defines its requirements and when the contractor begins delivering workable solutions. Achieving timely results requires the contracting cycle to be in alignment with the technology cycle.

For IT investments that use modular contracting, an acquisition may be divided into several smaller acquisitions, each of which comprises a system or solution that is not dependent on any subsequent increment in order to perform its principle functions.[75] In other words, the acquisition of any single program should not commit the government to acquiring any future systems. In addition, the program should avoid vendor lock-in by making sure deliverables are properly tested and documented so that a new vendor can continue work already begun if necessary. If each program is not separable, then the government may need to acquire future programs, which could be costly and burdensome.

Modular contracting divides investments into smaller parts in order to reduce risk, deliver capabilities more rapidly, and permit easy adoption of newer and emerging technologies.

Enable flexibility in the contract's requirements

Similar to writing a solicitation for a Waterfall program, schedule achievement, software quality, user acceptance, and product complexity should all be considered when drafting a solicitation for an Agile program. Furthermore, a contract governing an Agile development effort must provide sufficient structure to achieve the desired mission outcomes, while also offering flexibility for adaptation of software requirements within the agreed-on scope of the system. Contract structure for Agile programs must be designed to support the short development and delivery timelines that Agile requires.

While contracting for all development methods requires definition, Agile contracts often define the Agile process and program objectives rather than detailing specific detailed requirements. For example, many contracts often rely on a statement of work in section C of the uniform contract format.[76] The statement of work lays out a detailed presentation of the technical requirements so that contractors can provide an offer based on their unique technical solution to a well-defined need. However,

[75]FAR § 39.103(b)(3).

[76]The uniform contract format is a standardized format for structuring government solicitations and contracts. FAR § 15.204-1 provides a list identifying the parts of the uniform contract format.

having this level of detail early in the program’s life is typically not the case with Agile development because the underlying detailed requirements are unknown and not well-defined at the beginning of the acquisition process. Therefore, instead of establishing a detailed presentation of the technical requirements in a statement of work, section C could use a statement of objectives, whose goal is to develop a broadly defined statement of high-level performance objectives to provide offerors with maximum flexibility. The statement of objectives can be used with any performance-based contract and can include goals and desired outcomes of the development effort, expected performance standards, and “build iterations” for software development.

The statement of objectives should include a purpose, scope, period of performance, location for conducting the work, background, performance standards (e.g., the required results), and any identified operating constraints. Performance standards establish the expected accomplishment level required by the government to meet the contract requirements. If performance standards are not measurable and structured to enable performance assessments, the government may not be able to assess the expected accomplishments.

The statement of objectives focuses on measuring outcomes, rather than on specific tasks that the contractor is to perform. Table 9 highlights the differences between a statement of objectives and a statement of work.

Table 9: Differences between Statement of Objectives and Statement of Work

Contract factor	Statement of objectives	Statement of work
Organization understanding	The organization understands the objectives but expects the end state to evolve. Additionally, the government provides sufficient resources to ensure the work identified can be completed.	The government has a high level of confidence in the end state and provides more “hands on” oversight to ensure that tasks are performed as specified.
Change	Change may be a significant factor in achieving the end state.	Change is expected to be minimal; if encountered, changes to the statement of work can be disruptive.
Constraint	This approach provides the offerors trade space flexibility in developing their proposals. It should probably not be used unless a high-level vision or road map of the work effort required has been established by the government.	Constrains offerors to the specific tasks identified, so it must be unambiguous and comprehensive. The government needs to apply specific constraints on the tradeoff space of life cycle cost, performance, interoperability, logistics/training, etc. Additionally, the government will hold the contractor accountable for delivery of all tasks described in the statement of work.

Source: GAO analysis of Software Engineering Institute literature. | GAO-20-590G

While a statement of objectives provides the additional flexibility necessary for Agile programs, a statement of work can also be used. Ideally, the government can provide either as long as it includes the product vision, strategic themes, an initial road map, and an initial backlog of features and capabilities. If a statement of objectives is used, the government can request contractor proposals to include a performance work statement based on their proposed solution, an Agile development management plan, and a quality assurance plan, along with other data required, for a thorough evaluation of the proposal. Focusing on these items in the statement of work or statement of objectives helps organizations describe their needs in terms of what is to be achieved rather than how it is to be performed, thus providing the developers more flexibility in their processes.

Contract structure and type

The FAR provides a wide selection of contract types and structures to provide the needed flexibility to organizations to acquire a wide variety of goods and services. However, to ensure that the contractor does not perform inherently governmental functions, the organization should carefully delineate the responsibilities of the contractor in the solicitation. It may identify the types of decisions expected to be made and ensure that federal employees oversee and make the final decisions regarding the disposition of the requirements. These actions should guarantee the contractor’s work does not become so extensive or close to the final product as to effectively preempt the government officials’ decision-making process, discretion, or authority.

Choosing the appropriate contract type and structure depends on many factors, such as the complexity of the requirements and risk associated with the work. Typically, any type of contract can be used for Agile development; however, a critical consideration as part of this decision is driven by whether the contract is structured for end items (e.g., products such as the number of features completed) or services (e.g., the work performed by a specified quantity of developers). For example, the program office and the contracting officer must communicate whether they will purchase structured goods or services, since this can provide the contracting officer with insight into key contracting decisions, such as contract type, contract vehicle, and what data to request as part of their contract oversight mechanisms.

The following illustrates how different agencies have used different contract types and vehicles for Agile programs.

Agile in Action: Contracting for an Agile program	
General Services Administration (18F)	**United States Air Force**
The 18F office was established in March 2014 as an office within the U.S. General Services Administration (GSA) that collaborates with other agencies to fix technical problems, build products, and improve how government serves the public through technology. In November 2018 we met with 18F officials to discuss their experience with contracting for Agile programs. According to officials, not long after its inception, 18F noticed an increased demand from partner agencies for help to support efforts to build new digital services. In early 2015, 18F created and tested an Agile blanket purchase agreement (Agile BPA), under GSA Federal Supply Schedule 70, Information Technology. This Agile BPA was intended to allow organizations to select developers from a pool of vendors that use Agile methodologies and customer-centered design principles. Once the Agile BPA was established, it provided GSA with the flexibility to quickly award flexible contracts through a streamlined ordering process. As part of competing the Agile BPA, GSA wanted prospective vendors to demonstrate their ability to use Agile practices, GSA asked vendors to publicly demonstrate their commitment to customer-centered design and iterative development by building open source prototype software. In order to help other organizations streamline their own Agile BPAs, 18F has provided examples of solicitation documentation on GitHub. 18F found that, by using an Agile BPA, 18F did not need the vendors to state that they could operate in an Agile manner each competition but could instead focus on the specific details for that contract and avoid duplicative administrative acquisition work.	In March 2019, we met with Air Force officials to discuss how they have developed contracts for Agile programs. Air Force officials said that they have chartered an acquisition agency that helps establish and manage Agile programs for components within the Department of Defense that have a similar software need. Working with these components through a memorandum of understanding, the Air Force is able to act as a hub to optimize different platforms for multi-domain operations. The Air Force initiates small, short term contracts (e.g., ~3 months) for a low cost through a broad agency announcement, which helps them scout capabilities through many contractors at once. They also establish mid-term length contracts (e.g., <3 years in duration) when promising programs progress into a longer-term development. Lastly, as an operational capability completes development, the necessary contracts are put in place to ensure a smooth transition without a loss in productivity. An indicator that a program is ready to move to sustainment is when there are no new capabilities in development. In addition, officials said that the Air Force typically uses an indefinite delivery/indefinite quantity contract to procure Agile development teams to fill specific software needs for specific mission areas. Through these three different categories of contracts, officials said they identified common factors that facilitate a successful Agile program. For example, the contract should have a scope broad enough to provide leeway for decisions. This flexibility allows for continuous evaluation of capability delivery throughout the contract's life cycle. The Air Force also found that it is important to document relationships and responsibilities among the interested parties and have an active

The Agile BPA, a simplified method of filling anticipated repetitive needs for goods or services by establishing “charge accounts” with qualified contractors, was an experiment in modular contracting. Based on lessons learned from the BPA, 18F warned against using open source prototype software and organizations pre-establishing vendor pools with large durations without the ability to onboard or off-board vendors. Without this capability, a permanently fixed vendor pool could yield stagnated competition with vendors only competing for larger buys.	and engaged product owner. Officials said that having defined roles helps to manage expectations of stakeholders and empowers the product owner.
Both of these examples show that keys to successful contracting include ensuring that the contract is structured so that it reflects the program and can react to updates in the program without overly burdensome paperwork. A contract should also reflect learning from previous contracts to further improve the contracting process.	

Generally, the decision regarding which contract type to select should be based on which contract type will allow the most efficiency in delivering a product. That is, the contract type should enable the program to continuously deliver working software. However, if the contract does not provide sufficient structure to achieve the desired mission outcomes, while offering flexibility for adaption of software requirements within the agreed on scope of the system, it may not be able to support an Agile development approach. A lack of balance between structure and flexibility increases the likelihood of disruption and delays.

Incorporate Agile metrics, tools, and lessons learned from retrospectives during the contract oversight process

Contract data requirements rely on Agile metrics

Typically, an IT contract is structured with contract data requirements (referred to as a contract data requirements list) and relies heavily on documentation and major reviews at predetermined milestones. However, the primary deliverable for an Agile program is working code; that is, code released to the customer that adds value to the program. Therefore, programs that adopt Agile methods should tailor the contract's contract data requirements list to align with Agile metrics to reflect the different processes and artifacts used in Agile.

Contract data requirements list: technical data package

Obtaining data rights for developed software can be useful if the government changes contractors for software maintenance. Since the government works closely with the development contractor in Agile, it is important to tailor the contract to protect the government's interests. These data rights can prevent the government from getting tied to one contractor.

There are many points throughout the Agile development life cycle that offer the opportunity to collect data about the quality of the software products. The quantity and type of contract data requirements established in the contract should account for the program environment. Due to the anticipated close and continuous work coordination between the government and contractors, the number of formal deliveries for contract data requirements list may be less than what is collected for a traditional IT acquisition. To that end, mindful tailoring of the contract data requirements list as part of the contract should be performed. If the contract data requirements list does not account for the Agile development program environment, the program may miss the opportunity to collect data about the quality of its software products.

Data from Agile artifacts enables contract oversight

Programs should also collect actual data associated with the program's releases, features, and capabilities to enable contract oversight and hold contractors accountable for producing quality deliverables. Agile metrics primarily focus on the developers during an iteration. Programs use work elements (e.g., story points, staff hours, task lists, etc.) and burn down charts to track progress and measure productivity, costs, schedule, and performance. As previously discussed in chapter 3, the definition of "done" in the user story should have identified all requirements that must be demonstrated before a user story is implemented.

A program office and contractor can track several different Agile metrics for requirements, cost, schedule, performance, architecture, size/complexity, test, and risk in order to ensure that the organization is adequately monitoring the contracted development effort. If the program does not collect Agile metrics for technical management, program management, and Agile methods, the government may not have the right information for effective contract oversight and will not be able to hold the contractors accountable for producing high quality deliverables. Table 10 provides an overview of these three metric categories that can be used throughout the Agile development life cycle to help enable effective contract management and oversight. Additional information regarding best practices to establish program-specific metrics is included in chapter 8.

Table 10: Examples of Agile Metrics by Metric Category

Metric category	Description
Technical management	Includes metrics that measure the quality of the product delivered. For example, technical debt provides valuable information regarding the accumulation of deficiencies over time. Observing technical debt provides insight into the code quality, ensuring that code quality meets expectations and does not result in an excess of technical debt and the need for a complete program refresh if the code base no longer functions properly.
Program management	Includes metrics that monitor and report on the cost, schedule, and performance of an Agile program. For example, lead time provides information about how long it takes to move a feature from identification to release to management. This allows managers to observe how rapidly developers are able to meet customers' needs.
Agile methods	Includes metrics that measure how well the program leverages Agile methods. This can be observed at an organization level through policies in place to support Agile, at a program level through training staff, and at a team level by implementing repeatable practices and forming Agile teams that have direct contact to customers through a product owner. Metrics in this category can include how much customers use a new feature or how often working code is delivered and demonstrated.

Source: GAO. | GAO-20-590G

Documentation for these metrics can be found in the backlog, design documents, test scripts, or other sources and is typically updated regularly when using Agile methods. For additional information about these metrics and reports, see chapter 8.

Beware of self-reporting

The process of choosing which metrics to use for contract oversight should include thoughtful consideration of what information most clearly shows how the contractor's work adds value to the program. Some metrics can be collected via self-reporting. For example, velocity is a measure of the rate at which the team delivers a user story. It is not comparable from team to team, and should not be used to distinguish one team from another. It is very easy to show an increase in velocity without adding value to the program by inflating story point estimates. In other words, increasing velocity does not always indicate a change in productivity.

Conduct retrospectives to continually improve based on lessons learned

In addition to including metrics in a contract for an Agile program, the organization should require reviews with stakeholders to interact with the developers and product owner in order to better understand the goals for the end product. The interaction can provide valuable insight to help inform contract oversight. The following are sample questions that can be asked at a retrospective with developers and what their answers might imply.

1. When was the last time a program delivered working software to the customer?

   Implication: The longer the time frame from when the customer need is identified to the delivery of working software, the greater the risks to the program.
2. Does the program build a minimum viable product to test the riskiest assumptions?

   Implication: A minimum viable product solves the core customer needs as soon as possible and helps to validate needs, reduce risk, and help the program's course correct quickly, if necessary.
3. What impediments currently facing the program can the sponsor help remove?

   Implication: Agile values leadership through empowerment rather than power; that is, those who enable others' success by the ability to use their position to make others' jobs easier and more efficient.
4. Does the program have lessons learned to share with the organization?

   Implication: Sharing this information with sponsors and throughout the organization will help organizations identify efficiencies across programs.
5. Do you need better clarity regarding feature prioritization?

Implication: Goals and priorities are critical in Agile planning and work better if aligned with organizational strategic goals.

6. What is your biggest bottleneck?

   Implication: A key Agile principle is to promote sustainable development. Normalizing the workload at a system level helps developers to meet schedules and find additional organizational efficiencies.

7. How has the program improved since its last review?

   Implication: Improvement shows that the team is reflecting on how to become more effective and adjusts the behavior accordingly. In Agile, it is important that teams review processes so that they can improve.

8. Is the customer satisfied with the results?

   Implication: Working software is the primary measure of progress and customer satisfaction is an indicator that the program is prioritizing early and continuous delivery of valuable software.

9. Are iterations finished as planned or are unfinished requirements pushed to the back of the backlog for future iterations?

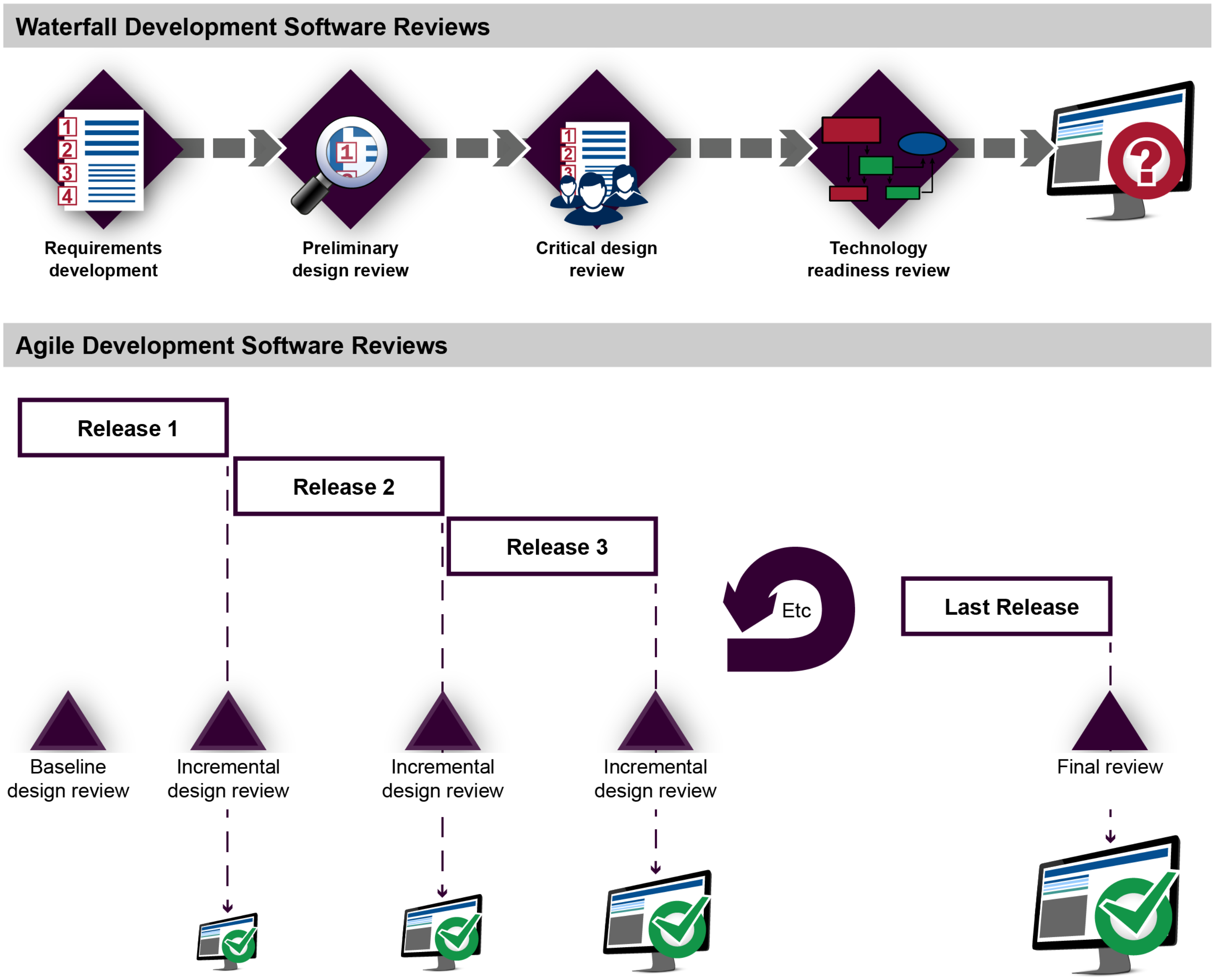
   Implication: Moving unfinished work to the end of the backlog should not be done without input from the product owner as the backlog should be maintained so that the program can ensure it is always working on the highest priority requirements to deliver the most value to customers.

Contract oversight reviews align with the program’s Agile cadence

Contract oversight reviews should align with the program’s Agile methods and cadence. For example, in a Waterfall model, technical reviews are used as control gates to move from one sequential phase to the next. These formal reviews provide traditional programs the opportunity to discover risks so they can be mitigated before moving on to the next phase of development. However, in an Agile program, the focus is on completing each work unit quickly in order to provide a releasable product in a short period of time. As a result, Agile programs tend to use technical reviews as an opportunity to share information face-to-face and to build team confidence. A byproduct of this approach is that problems are discovered early, often before they become too big to control.

As a result of these key differences between Waterfall and Agile program structures, the same contractual review gates may need to be tailored as part of the contract deliverables in order to successfully align the contract requirements with the functional requirements. For example, the traditional requirements development, preliminary design review, and critical design review events may be replaced by incremental design reviews, and, if needed, system-level reviews. The incremental reviews should be tied to the program's Agile cadence for completing releases and will likely occur more often than traditional reviews. As each release commences, developers will continuously pull and refine features for development from the backlog that is being constantly prioritized based on the program's road map. These recurring efforts provide Agile program managers the oversight they need to help ensure that the right features are being developed. Following this approach, reviews may occur incrementally, following the program's cadence, throughout the life of the contract. Figure 9 shows a comparison of Waterfall development reviews and how an Agile program's reviews would align with the program's Agile cadence.

Figure 9: Comparison of Waterfall and Agile Programs' Review Cycles


Waterfall Development Software Reviews
Requirements development
Preliminary design review
Critical design review
Technology readiness review
Agile Development Software Reviews
Release 1
Release 2
Release 3
Etc
Last Release
Baseline design review
Incremental design review
Incremental design review
Incremental design review
Final review

Source: GAO. | GAO-20-590G

Generally, if reviews for the program are not tailored to align with the program's Agile cadence, the review structure could impede progress and cause delays.

Integrate the program office and the developers

Train program office, acquisition, and contracting personnel

Proper training in Agile implementation for all personnel is a key element for success. Without properly trained program office personnel, including contracting personnel, staff will not be capable of assisting the program in making business decisions and trade-offs that come with the implementation of an Agile effort. Agile practices stress the need for government program management personnel to be highly involved with the program and available daily to provide input for the developers. This may involve both a culture shift and training in new roles and responsibilities for these program management personnel. To accomplish this, program office personnel should work with developers to establish a common understanding of Agile techniques so that an acquisition strategy can be properly structured to establish a development cadence. This common understanding often depends on effective training and collaboration between developers and the acquisition team.

In turn, the small, empowered teams need to have a close partnership with the program managers, customers (through the product owner), and contractors (typically the developers). The government contracting community serves as an invaluable linchpin to enable this relationship in a collaborative, flexible business environment. Dedicated onsite contracting personnel, properly trained in Agile implementation, can assess any impact Agile cadences may have on the program's acquisition strategy, enabling a close partnership with the developers.[77]

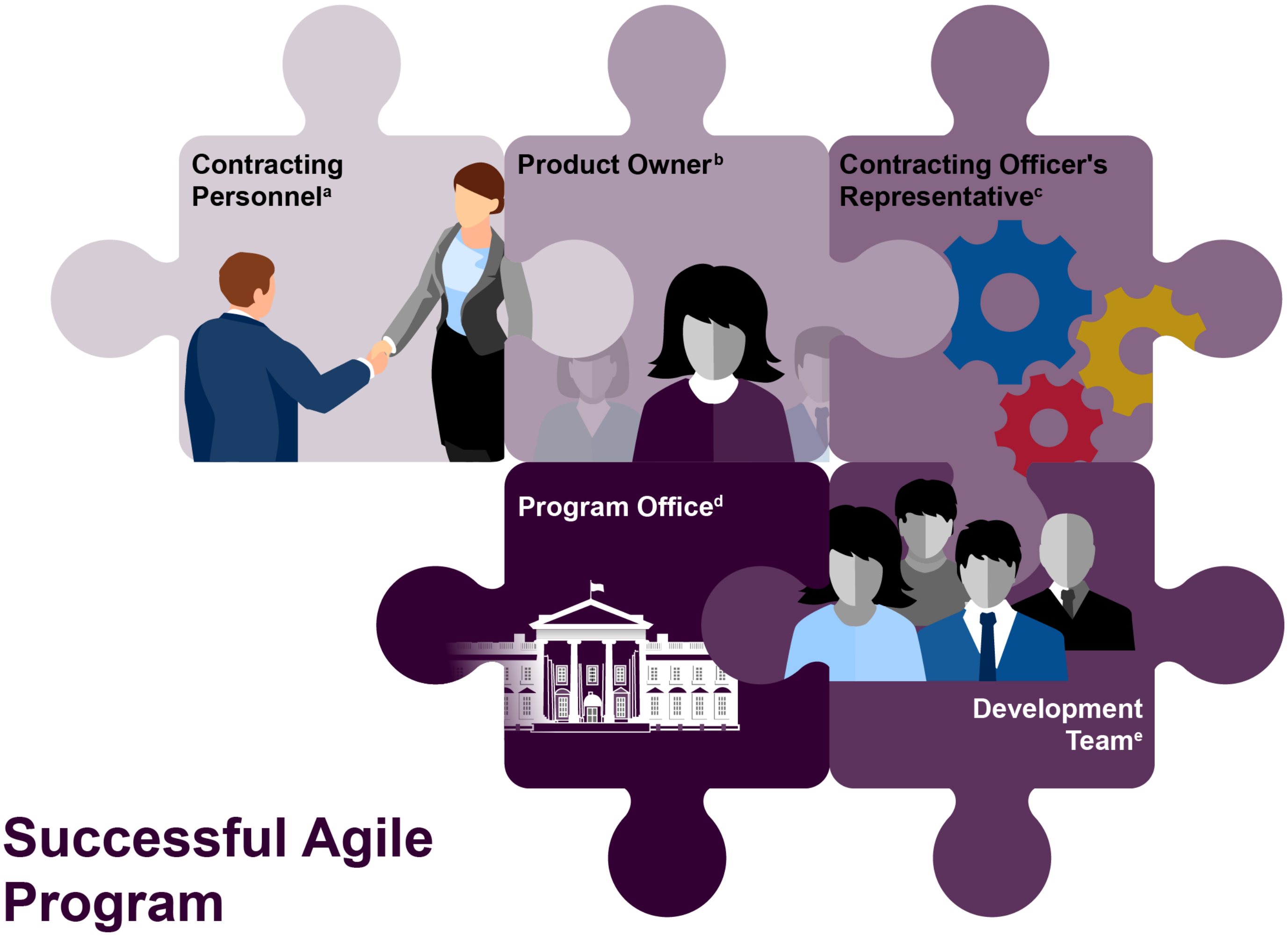
Management can create an environment that empowers and motivates the team. An empowered team has the authority and responsibility to make decisions rather than a manager. Management can accomplish this by adopting the role of a mentor to foster an environment of trust and communicate a positive perception of Agile.

Identify clear roles

[77]How dedicated onsite contracting staff is to the Agile team is a decision for the program office to make and depends on many factors, such as the complexity, duration, and size of the contract. Another consideration is the availability of adequately trained staff. In order to maximize effectiveness, the program's acquisition personnel should have a thorough understanding of the program's Agile methods.

There are various roles and offices involved in the planning, managing, and executing an Agile contract. Figure 10 depicts these roles.

Figure 10: Roles When Planning, Managing, and Executing an Agile Contract



Source: GAO. | GAO-20-590G

[a]Contracting personnel typically includes a warranted contract officer, who has express authority to enter into and administer a contract on behalf of the government and a contract specialist who can act as a business advisor to program managers; contracting personnel typically assist in planning the acquisition of goods and services, help negotiated the terms of the contract and provide contract management and administration services.

[b]The product owner is the "voice of the customer," and is accountable for ensuring business value is delivered by creating customer-centric items (typically user stories), ordering them, and maintaining them in the backlog. See appendix II: Key Terms for more about the definition of a product owner.

[c]The contracting officer's representative (COR) is a technical expert designated by the contracting officer to perform specific technical and administrative functions. The COR may provide day-to-day oversight of the contractor and reviews deliverables to ensure that they meet government requirements for quality, completeness, and timeliness.

[d]The program office refers to all other personnel who support the program. This can include legal support, program monitoring and control support, and program management support. It is important that all personnel who support the program are familiar with Agile processes.

[e]As discussed in chapter 3, the development team consists of the software developers, team facilitator, and subject matter experts who code the features for the program.

These roles must be clearly defined and responsibilities should be faithfully carried out in order to help prevent bottlenecks and ensure that rapid feedback channels are clearly established from the start of development. One area of potential confusion can be between the role of the contracting officer and contracting officer's representative (COR), and the product owner. For example, in Agile, the product owner approves the work delivered by the team, while the COR is responsible for ensuring the work is technically sufficient and the contracting officer accepts the work. This confusion could be due to the product owner role not being a typical role that is used in Waterfall development. While there are similarities between these roles, each role has distinct responsibilities.

The product owner is typically associated and familiar with the business aspects of the program office, while the COR has more technical skills. However, it is important that the product owner and the COR both understand the program and teams' Agile methods and that there is one government focal point for the team to interact with. In other words, the product owner serves as a bridge between the COR (who generally judges the technical quality of the contractor's work for acceptance on behalf of the contracting officer) and contractor, while also working to integrate the program office and developers to ensure that the customer receives the expected business value for the work. As long as the product owner and the COR remain in close communication, they can continue as separate roles. Additionally, the product owner and COR work closely to align the program's business and technical requirements. Typically, both the COR and product owner should be government employees so that they can be empowered to make day-to-day decisions for the development effort. If the product owner is not a government employee, the product owner may not be empowered to make day-to-day decision for the development effort, thus causing development delays.

As stated earlier, dedicated contracting personnel should work closely with the developers and the product owner. The product owner should represent a government commitment to providing an empowered customer, a representative who can make decisions quickly and prioritize requirements within the scope of the program’s road map. Together, the contracting officer, product owner, and government members of the development team (e.g., the developers, subject matter experts, and team facilitator) should consider the following questions as the acquisition strategy is developed:

1. Have the program’s vision and goals been established?
2. Has the program’s scope been established and are the cost and schedule constraints identified?
3. Are Agile methods well defined or already in place within the government program office?
4. Does the program office have executive-level support for Agile development?
5. Did market research identify qualified contractors with Agile experience?
6. Are there multiple contractors who will conduct parallel development?
7. Who is the systems integrator (e.g., government or contractor) and what level of integration is required?
8. What is the overall development timeline?
9. What is the release schedule?
10. How much contracting support is available for the program?
11. Are government resources available to actively manage contractor support once the contract has been awarded? For example, is there a dedicated product owner?
12. Is the program considered high risk and what level of risk is the government willing to accept in its contracting strategy?
13. Are other, similar programs currently using or thinking of using Agile for development?
14. Can the program leverage previously-established contract vehicles in order to shorten acquisition times?
15. What are the program’s scope and key deliverables?
16. What are the milestones and how frequently do they occur?
17. What performance metrics are defined in the contract?

Furthermore, contracting personnel and other program office personnel should understand the distinction between contract and functional requirements that are part of the Agile development process. In many cases, these two types of requirements differ significantly. For example, in an Agile environment, contract program requirements are broken down into high-level capabilities that, over time, are further decomposed into features, while Waterfall programs define requirements (e.g., functional requirements) in detail in the statement of work and system segment specifications. Furthermore, customer collaboration should include any legal, security, and compliance with section 508 of the *Rehabilitation Act of 1973*, or privacy concerns, and should be included as part of the program's contract and functional requirements. If these are not clear, all compliance and security requirements may not be included in the program.

Awareness of the contract's scope

Contracting personnel must also watch for "scope creep," where requirements are added to the contract without other work being identified as lower priority. If additional requirements are identified by the customers after a contract has been awarded, but are still within the scope of the contract, contracting personnel (along with other members of the Agile team) should ensure that there is enough time on the contract to complete the additional work or whether these requirements can substitute for currently-identified features. This is done by examining the statement of work or sequence of operations to ensure that new work is within the scope of the contract and by prioritizing the work in the backlog. If new work, outside of the contract's scope is identified and found to be a higher priority to accomplish goals that are outside the scope of the current contract, then the contracting officer may issue an out-of-scope modification to add work to the contract.[78] Contract modifications should be infrequent if the program's vision and high-level capabilities are broad enough so that features resulting from breaking down requirements can be added to the backlog within the contract's current scope.[79]

[78]Out-of-scope contract modifications are considered non-competitive contract actions under the FAR, which requires an approved justification describing the exception to competition under which the modification is issued and the rationale for the exception to competition prior to issuance. The FAR, in part 6 and similar sections in parts 8, 13, and 16 (for federal supply schedules, simplified actions, and task orders), describes exceptions to competition and the requirements for documenting and using these exceptions.

[79]Contract modifications raise numerous legal considerations. For instance, depending on the scope and circumstances of the modification, an agency may not be able to use the same appropriation that it used to fund the contract at award. Prior to substituting any work that is outside the scope of the contract, the product owner and COR must consult with the contracting officer, contracting specialist, and other government program office personnel, as appropriate.

While a contract cannot be modified without a contracting officer's authority, the product owner should be empowered to prioritize the detailed requirements within the scope of the product vision and thereby help to avoid scope creep. The engaged and empowered product owner considers the requirements consistent with the program vision, participates in incremental planning, iteration and release planning, and retrospectives in order to minimize contractual changes as the Agile program evolves. Lack of involvement by the product owner and limited empowerment can result in bottlenecks in the contracting process.

Successful delivery of a software program requires planning, management, information gathering, and continual assessment of performance under the contract by both program office personnel and developers. All levels are involved in the acquisition and contracting phases and must understand the Agile process to achieve a successful outcome. For example, the product owner should be engaged in the award process to help clarify the customers' needs from the very start of the development of the contract. Likewise, contracting personnel should understand the program's Agile methods to develop a contract structure that aligns and supports those specific processes.

Best Practices Checklist: Contracting for an Agile Program

1. Tailor contract structure and inputs to align with Agile practices
   - Encourage the use of modular contracting.
   - Enable flexibility for adapting software requirements.
   - Decide to structure the contract for goods or services.
2. Incorporate Agile metrics, tools, and lessons learned from retrospectives during the contract oversight process
   - Ensure that contract data requirements rely extensively on Agile metrics.
   - Data from the program's Agile artifacts enables contract oversight.
   - Conduct retrospectives to allow stakeholders to interact with developers and product owners to continually improve Agile methods based on lessons learned.
3. Integrate the program office and the developers
   - Train program office acquisition and contracting personnel.
   - Identify clear roles for contract oversight and management.
   - Ensure that all personnel are familiar with the contract's scope.

Chapter 7: Agile and Program Monitoring and Control

Program monitoring and control provide the oversight needed for legislators, organization officials, and the public to be able to assess whether government programs are achieving their goals. The government uses many traditional practices that have been developed for Waterfall development. These practices may need to be adapted so they can be applied to an Agile program. For example, as discussed in chapter 4, traditional methods of tracking and reviewing the status of a program focus on the big picture, whereas Agile methods are focused on short-term efforts with the most attention and detailed planning paid to the current iteration. In spite of this apparent conflict, program monitoring and controls can be adapted to an Agile program. This chapter discusses how to adapt a work breakdown structure (WBS) for an Agile program and how cost estimating, scheduling, and earned value management (EVM) are applicable to Agile programs.

The WBS is the framework used by federal agencies to organize the work into manageable, smaller components. It is an essential input to three principle program controls used by federal agencies: cost estimating, scheduling, and EVM.[80] Using the work breakdown structure, a program's cost estimate and schedule are developed and, if warranted, they can be combined into one baseline used to measure program performance. A major benefit performance management tracking provides is the identification of cost and schedule variances from the overall baseline plan so that program risks can be quickly discerned, tracked, and managed. GAO has established cost, schedule, and EVM best practices in best practices guides.[81]

[80]A WBS breaks down product-oriented elements into a hierarchical parent/child structure that shows how elements relate to one another as well as to the overall end product. A WBS provides a basic framework for a variety of related activities including estimating costs, developing schedules, identifying resources, and determining where risks may occur. It also provides a framework to develop a schedule and cost plan that can easily track technical accomplishments—in terms of resources spent in relation to the plan, as well as completion of activities—enabling quick identification of cost and schedule variances.

[81]GAO, *GAO Cost Estimating and Assessment Guide: Best Practices for Developing and Managing Program Costs,* GAO-20-195G (Washington, D.C.: Mar 12, 2020) and *GAO Schedule Assessment Guide: Best Practices for Project Schedules,* GAO-16-89G (Washington, D.C.: Dec. 22, 2015).

Work breakdown structure in an Agile environment

A WBS or similar document can be used by management and Agile teams to provide a clear picture of the total scope of work necessary to meet a program's vison and requirements.[82] A well-structured WBS decomposes the program scope into discrete deliverables that can be measured and tracked, thus providing a framework for planning and accountability by identifying work products that are outcome-focused.

With the end product in mind, the WBS creates a hierarchical structure that shows how program elements relate to one another and the overall program. Similarly, Agile programs break down the work into successive levels of effort: epic, feature, and user story. Each of these Agile levels depict what the work entails and how it relates to higher-level program goals and the final product.[83] The levels of effort are described here.

Epic: The epic captures high-level capabilities. It generally takes more than one or two iterations to develop and test. An epic is usually broad in scope, short on details, and will commonly need to be split into multiple, smaller user stories before the team can work on them.

Feature: A feature is a specific amount of work that can be developed within one or two reporting periods. It can be further segmented into a user story or stories. The functionality is described with enough detail that it can remain stable throughout its development and integration into working software. It is this level that should be tracked through program management products like the life cycle cost estimate and schedule. The features in the WBS should be fully traceable to the program's road map.

User story: The user story is the smallest level of detail in an Agile program and is subject to change based on customer feedback. For this reason, a user story can be added to or deleted without altering the overall scope of the features. A user story is weighted for complexity using story points. It can be used as quantifiable backup data for EVM;

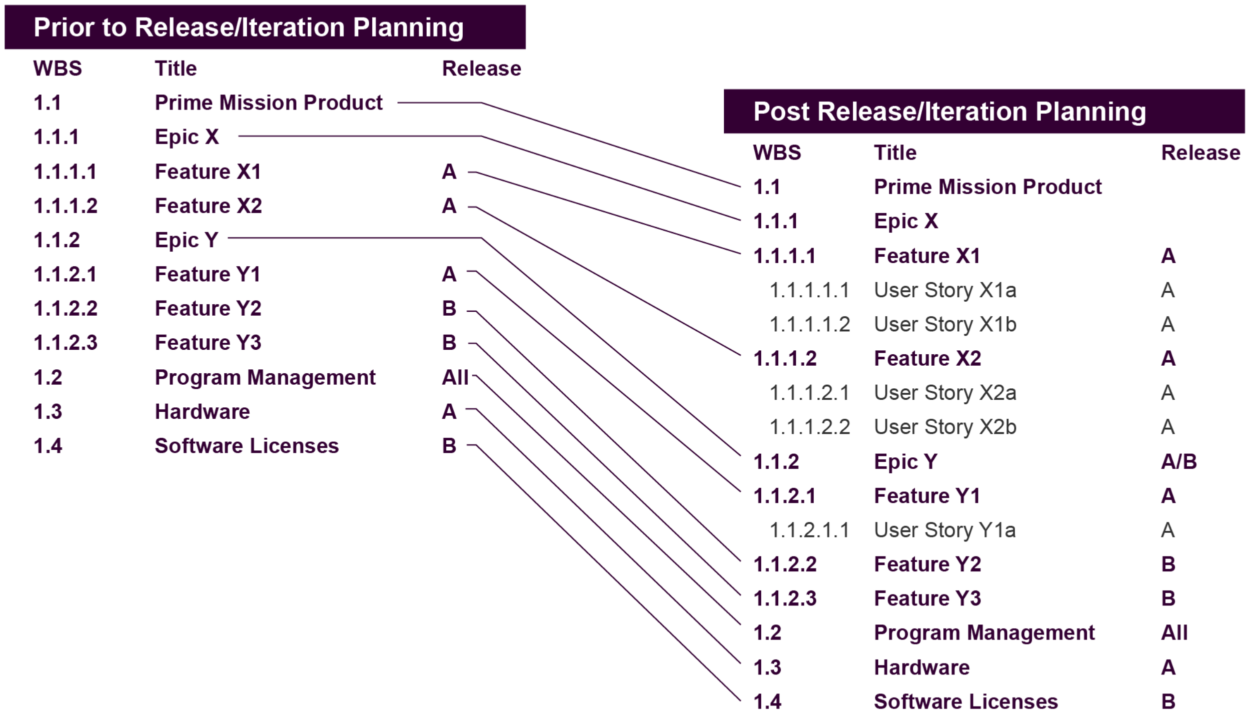
[82]A separate WBS document is not the only solution. Chapter 5 discusses how the concept of backlog refinement is used to decompose requirements as more information becomes known. It also describes a set of strategic requirements necessary to justify a program that can be used to develop a WBS and some form of EVM measurement to achieving these goals.

[83]As with any WBS, more detail can be added as additional information is discovered about the program. The WBS should ultimately be based on existing Agile artifacts (e.g., the road map) to reinforce traceability between program monitoring and controls and Agile planning documents.

however, a user story should only be added to the WBS after release or iteration planning and be traceable to the prioritized backlog.

Figure 11 shows a typical WBS for an Agile software development program and how more detail can be added over time.

Figure 11: Work Breakdown Structure in an Agile Program



Prior to Release/Iteration Planning

WBS	Title	Release
1.1	Prime Mission Product	
1.1.1	Epic X	
1.1.1.1	Feature X1	A
1.1.1.2	Feature X2	A
1.1.2	Epic Y	
1.1.2.1	Feature Y1	A
1.1.2.2	Feature Y2	B
1.1.2.3	Feature Y3	B
1.2	Program Management	All
1.3	Hardware	A
1.4	Software Licenses	B

Post Release/Iteration Planning

WBS	Title	Release
1.1	Prime Mission Product	
1.1.1	Epic X	
1.1.1.1	Feature X1	A
1.1.1.1.1	User Story X1a	A
1.1.1.1.2	User Story X1b	A
1.1.1.2	Feature X2	A
1.1.1.2.1	User Story X2a	A
1.1.1.2.2	User Story X2b	A
1.1.2	Epic Y	A/B
1.1.2.1	Feature Y1	A
1.1.2.1.1	User Story Y1a	A
1.1.2.2	Feature Y2	B
1.1.2.3	Feature Y3	B
1.2	Program Management	All
1.3	Hardware	A
1.4	Software Licenses	B

Source: GAO. | GAO-20-590G

Figure 11 shows that, as more information is learned, additional detail can be added to the WBS. For example, when features are decomposed during iteration or release planning, user stories can be added to the appropriate parent level item in the WBS. Updating the WBS with additional information and tying it to Agile documents as more information is discovered helps provide additional traceability through Agile artifacts and program control files. A WBS can also help show the relationship between the Agile development effort and other parts of the program, such as program management or software licenses.

An Agile environment should have established methods and measures to ensure progress is monitored objectively. Specifically, lower-level user stories and tasks are flexible and subject to change throughout the development process. Because of this instability, the majority of traditional program monitoring and control best practices should be maintained at a higher level of the WBS, namely at the epic or feature level.

The program WBS provides a common structure for cost, schedule, and EVM. In an outcome-based Agile environment, the WBS is hierarchical, product-based, and contains the total program scope. Further, the WBS should reflect high-level capabilities identified in the road map as well as varying levels of detail at the epic and feature levels when this information is available, typically only for near-term work. At any specific time, the WBS should inform the necessary technical activities needed to sufficiently complete a feature. As program requirements are decomposed to capabilities (or epics) and features, the derivation of the WBS should remain traceable to the program's cost, schedule, and EVM work, as appropriate. The program should also establish defined completion acceptance criteria to ensure that performance measurement is consistent and traceable. In this way, the relationship between a program's progress and its technical achievement can be maintained.

Cost estimating best practices in an Agile environment

The *GAO Cost Estimating and Assessment Guide*[84] (Cost Guide) establishes a consistent methodology based on best practices used by federal agencies to develop, manage, and evaluate a cost estimate. The Cost Guide recommends the use of a 12-step process that, when followed, can result in a high quality, reliable cost estimate:

1. Define the estimate's purpose.
2. Develop the estimating plan.
3. Define the program.
4. Determine the estimating structure.
5. Identify ground rules and assumptions.
6. Obtain the data.

[84]GAO, *GAO Cost Estimating and Assessment Guide: Best Practices for Developing and Managing Program Costs,* GAO-20-195G (Washington, D.C.: Mar. 12, 2020).

7. Develop the point estimate and compare it to an independent cost estimate.
8. Conduct sensitivity analysis.
9. Conduct a risk and uncertainty analysis.
10. Document the estimate.
11. Present the estimate to management for approval.
12. Update the estimate to reflect actual costs and changes.

These steps mostly occur in a sequential order; however, steps three through seven are iterative and can be accomplished in varying order or concurrently. While the Agile methods differ from Waterfall development process, the need for a high-quality, reliable cost estimate is applicable. Whatever development framework is used, every program needs to establish a budget and be accountable for delivering a value-based outcome. To that end, an Agile program should follow the GAO 12-step cost estimating process to develop an estimate that reflects cost estimating best practices. One advantage of Agile development is that this approach follows an iterative process that results in new data that is generated and collected after every iteration to keep the estimate updated. Furthermore, while Agile lowers the technical risk through incremental delivery, sensitivity, risk, and uncertainty analyses can be performed to inform management decisions. Finally, while Agile places the value of working software over comprehensive documentation, documenting the cost estimate assumptions is still important.

Agile measures and documenting the cost estimate

Agile software development produces documentation, measures, and artifacts that can be used as evidence to assess whether the program is following GAO's cost estimating process and best practices and to aid auditors in assessing the planning and assumptions made by the program office to develop their cost estimate. Table 11 compares GAO's cost estimating process to a typical Agile process.

Table 11: 12-Step Cost Estimating Process and Agile Cadence Examples

12-Step estimating process step and definition	Agile environment and the GAO cost estimating process
Step 1: Define the estimate's purpose The purpose of a cost estimate is determined by its intended use, which determines its scope and detail.	During release or initial planning, determine how any cost estimates will be used.
Step 2: Develop the estimating plan This step involves selecting the estimating team members and developing a schedule that includes enough time to perform all steps commensurate with the estimate's purpose.	During initial planning, identify the cost estimating team that will develop the estimate and any technical experts that will be needed to support the estimating effort. The estimate plan should also include details about when the government program office plans to update the estimate with Agile metrics.
Step 3: Define the program Program personnel identify the technical and programmatic parameters on which the team will base the estimate. This information should be kept updated at all times so that it remains current.	These steps (steps 3-7) should first occur during initial program planning with the development of a road map or vision and be updated as the estimate is refined at established intervals, such as after a release, in support of program milestone reviews, or whenever there are updates to the road map. Agile performance measures and artifacts such as burn up/burn down charts, velocity metrics, and the product backlog can be used to update the estimate accordingly. It is important that the cost estimating team is integrated into release planning so that team members can fully understand the changes to the plan and update the estimate to reflect those changes that occur naturally during the Agile process (e.g., additional detail is provided through a requirements decomposition process). An independent cost estimate should be developed after the initial cost estimate and then repeated at major milestone reviews for the program. Between estimates, a sufficiency review of the cost estimate after major updates should be performed to assess the credibility of the program office estimate.
Step 4: Determine the estimating structure This step defines at various levels of detail what the program needs to accomplish to meet its objectives. Typically, estimators will have access to a work breakdown structure (WBS) that decomposes the work into a product-oriented, hierarchical framework supplemented by common elements like program management, systems engineering, and systems test and evaluation, etc. A WBS promotes accountability by clearly identifying work products and enables managers to track technical accomplishments. It also outlines how program elements progressively subdivide into more detail as new information becomes available.	
Step 5: Identify ground rules and assumptions The estimating team establishes ground rules that represent a common set of agreed to estimating standards such as what base year the team will use to express costs, the number of expected program quantities, and the anticipated contracting strategy. When information is unknown, the estimating team must fill the gaps by making assumptions so that the estimate can proceed. Because many assumptions profoundly influence cost, management should fully understand the conditions the estimate was structured on. Well-supported assumptions include documentation of their sources along with a discussion of any weaknesses or risks.	
Step 6: Obtain the data The team collects, normalizes, documents, and archives the cost, schedule, programmatic and technical data it will use for the cost estimate.	

12-Step estimating process step and definition	Agile environment and the GAO cost estimating process
Step 7: Develop the point estimate and compare it to an independent cost estimate The team creates a time-phased cost estimate for each WBS element using a variety of techniques including analogy, parametric, and engineering build up. Once each WBS element has been estimated using the best methodology from the data collected, the estimating team adds all WBS costs together to determine the program's point estimate. This "point estimate" represents the best guess at the cost estimate, given the underlying data and represents one potential cost among a range of many possibilities. To validate the estimate, the team reviews it for errors as well as any omissions and compares it to an independent cost estimate to understand where and why there are any differences.	
Step 8: Conduct sensitivity analysis The team examines the effect of changing one assumption or variable at a time while holding all other estimate inputs constant in order to understand which factors most affect the cost estimate so that cost drivers are evident.	Sensitivity analysis should be performed after the initial point estimate has been developed and then updated whenever the point estimate is refined to determine if there are any changes to the estimate's cost drivers and what impact those changes have on the overall cost estimate.
Step 9: Conduct a risk and uncertainty analysis Using a risk and uncertainty analysis, the team quantifies the cumulative effect that uncertain data inputs, changing assumptions, and variations underlying estimating equations have on the estimate. Based on probabilities produced from the analysis, the team determines a range of costs associated with the point estimate so that management can decide how much contingency funding it needs to mitigate potential risks.	Risk and uncertainty analysis should be conducted to better understand the risk range around the cost estimate due to variations in estimating assumptions such as sizing metrics, velocity, number of iterations, and labor rates. This analysis should be repeated after major updates to the cost estimate.
Step 10: Document the estimate The team documents its entire estimating process including what assumptions, data sources, and methodologies it used. The documentation should reflect sufficient detail so that someone unfamiliar with the program can easily recreate the estimate and get the same result.	Documentation of the cost estimate should be updated regularly following the same cadence that the Agile program has established for updating other program management documents such as the vision, road map, and product backlog.
Step 11: Present the estimate to management for approval The team presents management with an overview of the estimate that contains enough information about the basis for the estimate including the quality of the program definition, availability and reliability of the data, and key assumptions made. The presentation should also include the outcome of the risk and uncertainty analysis so that management can approve the estimate at a confidence level of its choice.	Management should review and sign off on the estimate and its underlying ground rules and assumptions before any major program reviews (as established and required by the organization) so that the most recent and applicable information is available to inform decisions. Management should review and approve the presentation to show their understanding of the documented assumptions.

12-Step estimating process step and definition	Agile environment and the GAO cost estimating process
Step 12: Update the estimate to reflect actual costs/changes The team continually replaces the original estimate with actual data and records reasons for variances and any lessons learned. The team refreshes the estimate on a regular basis using EVM information and updates the estimate to reflect major changes.	The estimate should be updated with information taken from Agile artifacts and measures (e.g., burn up/down charts, velocity, actual vs. planned work, changes in requirements, program risk assessments, etc.) at predetermined times that align with the program's Agile cadence. While new data is created and should be captured in each iteration, it is recommended that the cost estimate be updated before any major milestone decision in order to provide the most recent information to decision makers. For example, if the program plans to negotiate a new development contract, the estimate should be updated to help provide information for the contract negotiation process.

Source: GAO. | GAO-20-590G

These 12 steps are tied to four best characteristics of a high-quality, reliable cost estimate. These four characteristics, used to determine how reliable a cost estimate is, also apply to Agile programs with some unique considerations, as discussed here.

Well documented: The cost estimate should be clearly documented, showing step-by-step how the estimate was constructed so that a person unfamiliar with the program could, from the documentation alone, reconstruct the cost estimate. Once the teams have been determined, cost estimates for Agile programs tend to be straightforward, with the number of iterations needed to work off the product backlog being based on relative sizing methods such as assumed function points or story points, and dividing the total number in the backlog by an average team-specific velocity factor.

Comprehensive: An Agile cost estimate should reflect all effort contained in the product backlog and each item in the product backlog should be directly linked to value-based high-level requirements captured in the program vision and road map. Ideally, all of the lower-level items that are defined in the release or the iteration are hierarchically linked to the product vision. A product-oriented WBS consisting of epics, features, user stories, and other supporting items should provide a consistent framework for the cost estimate, the schedule, and the EVM system.

Accurate: Historical data from other software programs should be used as input to the initial point estimate. Additionally, Agile cost estimates should be developed in constant year dollars and appropriately time-phased to account for inflation, updated frequently as more information becomes available or a new contract is awarded, and provide documentation for any variances between planned and actual costs in order to develop lessons learned to better inform future estimates.

Credible: Agile cost estimates are credible when input (e.g., the assumed number of iterations, velocity, etc.) has been tested for sensitivity, and a confidence level for the point estimate has been determined based on risk and uncertainty analysis, cross checked by cost estimators using another estimating method, and compared to an independent cost estimate with similar results. These analyses can provide insight, whether extra iterations or additional resources are needed to deliver the must have features identified by stakeholders and customers.

Table 12 shows GAO's characteristics of a reliable cost estimate and examples of Agile measures that can be used to ensure that the Agile cost estimate meets GAO's characteristics of a reliable cost estimate.

Table 12: Characteristics of a Reliable Cost Estimate and Agile Measures

Characteristic	Examples of Agile measures and documentation
Well-documented	• Release notes that discuss what features and enhancements are included in that release, any known defects, and a summary of the spend plan. • Iteration commitments based on number of story points or other unit of measure used by the developers • Contracted labor rates. • Number and composition of teams developing software. • Program documentation that is updated regularly. For example, a plan that captures technical changes to the system, a process plan that outlines the business rules and workflow for the program, a quality assurance plan, a cybersecurity plan, etc. • Retrospective reports that discuss lessons learned and highlight features where more attention is needed in future releases. • Release planning session executive briefings showing changes made to the road map during the planning session.
Comprehensive	• Road map and prioritized backlog that indicate must have features to be developed with input from stakeholders and subject matter experts. • Road map or vision aligned with program requirements (e.g., a Statement of Objectives or Statement of Work). • Schedule reflecting all activities the organization, its contractors, and others need to perform to deliver the must have requirements. • Prioritized backlog consisting of epics, features, and stories. • Backlog queues and unfinished work and any defects, listed in priority order. • Relative sizing estimates and assumed velocity, number of iterations, and blended labor rate.
Accurate	• The road map and vision documents are used to time phase the estimate to properly account for inflation. • The estimate should be updated using actual data from the burn up/down charts so that decisions impacting the budget can be based on the most recent information. • After the estimate has been updated, retrospective and release planning executive briefings should discuss variances between planned and actual costs to provide lessons learned for future estimates.

Credible	• Customer feedback from retrospective to provide insight into risks and priority of requirements. • Retrospective and release planning executive briefings should discuss threats and opportunities, including team size, management support to avoid distractions, availability of tools to aid Agile efforts, and external dependencies. • Daily standup meetings and other techniques used to mitigate threats and take advantage of opportunities for the program.

Source: GAO. | GAO-20-590G

Considerations for developing a cost estimate for an Agile program

Since Agile programs have flexible requirements and fixed budgets for an iteration, some have argued that conventional performance management tools, such as life cycle cost estimating, are not applicable to Agile programs. Those arguments are made because Agile programs have structures and processes that are dynamic and iterative and spread planning activities throughout the program's duration instead of conducting extensive planning upfront, as in traditional program development. However, as previously mentioned, reliable cost estimates are still applicable as all federal programs must follow the federal budgeting process. In addition, program controls provide necessary oversight that legislators, government officials, and the public can use to observe whether government programs are achieving their goals. The following are three areas should be examined for Agile programs when developing a cost estimate:

- **Consistent sizing.** Developers typically rely on relative estimation methods to determine the software size. However, these methods are not consistent across different Agile programs, or even across different teams working on the same Agile program. Consistent sizing is a key data quality consideration for reliable cost estimates.
- **Expertise of the developers.** The cost estimate is dependent on the expertise of the developers.
- **Cost estimating benefits.** Since Agile programs have fixed costs, the benefits of developing and updating an Agile program's cost estimate may not be recognized as important by technical personnel.

Next, we discuss estimating issues and provide examples of how to apply traditional cost estimating concepts to an Agile program.

Consistent sizing

While relative estimation methods, as discussed in chapter 3, are typically used by developers throughout the estimating process, these methods can vary from team to team on a single Agile program and do not provide a consistent measure that can be used to develop a cost estimate. This lack of consistency creates a challenge for cost estimators to normalize the data received from the program's reporting metrics (e.g., the burn up/down charts).

However, traditional size metrics can be used with Agile relative estimating metrics. Each sizing metric can serve a unique purpose for an Agile program. For example, developers use relative estimating techniques to determine how many story points to accomplish in an iteration. After a release, estimators can use a traditional sizing technique to establish the size of the effort and productivity rate achieved for the features developed in that release. Then, the cost estimators work with the developers to understand how the features relate to the traditional sizing metrics. While traditional sizing metrics would not eliminate the challenges associated with the initial program estimate, data collected could help cost estimators refine the initial estimate with respect to the remaining requirements. This could also help establish an Agile program database based on traditional sizing metrics, to be used to help the government program office develop initial cost estimates for future Agile programs.

Table 13 provides an overview of the different measurement techniques used by developers and cost estimators.

Table 13: Comparison of Consistent Sizing and Relative Sizing

	Cost estimating team: consistent sizing	**Developers: relative estimating**
Purpose	To develop a life cycle cost estimate for the program.	To scale the size of work to assist in iteration and release planning.
Strengths	Provides a method that can be used across programs and teams to measure work. From this, cost estimates can be developed and databases can be started to provide a basis to estimate future programs.	Is performed by the team performing the work at a granular level to increase the accuracy of the estimate.
Limitations	Using consistent sizing is typically performed at a higher level and requires insight into the program's scope, complexity, and interactions.	Relative estimating is team-dependent so measures cannot be used for comparison between programs or even different teams on the same program. Additionally, it is also performed later in the life cycle so it cannot be used at the start of the program.
Examples	Source lines of code, function points.	Story points, t-shirt sizing.

Source: GAO. | GAO-20-590G

Choosing a consistent sizing method depends on the software application (purpose of the software and level of reliability needed) and the available information. Cost estimators should work with the developers to determine the most appropriate method. Further, when completing a software size estimate to develop a total program cost estimate, it is preferable to use two different methodologies, if available, rather than relying on a single approach.

Agile in Action 3: Sizing and estimating before Agile teams are established

In March 2019 we met with the Department of Homeland Security's Cost Analysis Division to discuss how they piloted a simple function point analysis methodology to estimate Agile software development costs for acquisition programs. According to officials, this methodology uses the correlation of transactions to functional size, which is known to be correlated to program cost. The Simple Function Point Association, an international non-profit association dedicated to evolving and promoting Simple Function Point methods, has published a measurement manual, examples of counting in simple function points, and templates to facilitate the calculation of measurements. The Cost Analysis Division used this measurement manual as the basis for elementary process factors.

The Cost Analysis Division used a program's concept of operations document as the basis for determining the number of function points. The concept of operations is to describe the functional capabilities of a program, including a comprehensive list of business functions and a list of all applicable stakeholders. In addition, the Cost Analysis Division decomposed the functional capabilities into three types of transactions: elementary processes, saves, and interfaces.

Officials described the process used to develop a simplified function point estimate: over a period of two to three days, one Cost Analysis Division analyst will review the concept of operations and manually count the number of action verbs (e.g., "create," "submit," "maintain," "receive," "respond," "withdraw," "register," "appeal," "cancel"). Each verb is associated with a number of transactions. Each type of transaction is weighted based on difficulty to develop, with elementary processes having the lowest weight and interfaces having the highest weight. The analyst then estimates function points by summing the weighted value of each transaction. After the count has been completed, a second analyst will review the number. Once finalized, the Cost Analysis Division works with the Chief Technology Officer, Program Office, and Chief Information Officer to validate the count.

After the function point count has been validated, the cost analysis division uses historical data and industry standards to develop and apply a productivity factor to the count along with other factors to account for growth, complexity, and uniqueness. The function point count multiplied by the various factors results in a cost estimate. Lastly, the Automated Cost Estimator tool of the ACEIT software suite is used to determine a final risk adjusted output.

Because Agile considers high-level requirements in the long term as opposed to knowing requirements up front, the Cost Analysis Division believed that using simple function point analysis would give cost estimators a faster, reliable, repeatable process for cost estimates and will allow program managers and oversight groups to track and manage progress toward completion by tracking estimated function points.

Cost Analysis Division officials noted that the simple function point analysis methodology still needs further research and refinement to properly calibrate the tool they created and discover appropriate uncertainty distributions.

While traditional estimating methods can be used by the organization's program office to develop a cost estimate for an Agile program before development begins, Agile development metrics can be used to refine a program's cost estimate. For example, Agile uses velocity as a measure of productivity that captures the amount of work each team can deliver in each iteration. Because velocity is a team-specific metric, it should not be used to dictate how much work any team should complete in an iteration; however, a team-specific velocity that is traceable in their Agile tool can be used as an input for a cost estimate once development has begun.

No matter which sizing method is chosen, actual costs can vary widely from the estimated costs. As a result any point estimate should be accompanied by an estimated range of probability, as identified in step 9 of the GAO 12-step estimating process listed previously in this chapter. This is especially important for initial program estimates that are used to develop a budget.

Expertise of the developers

There is no generally recognized standard unit of measurement for any of the common approaches to Agile estimation. That is, story points, user stories, etc. are all subjective and dependent on the experience and skills of the developers. As a result, cost estimators for Agile programs rely on the composition and expertise of the developers. Therefore, to improve the quality of the estimate, cost estimators should be integrated with the developers and should participate in release planning sessions to understand the relationship between the backlog and the developers' relative estimating techniques so that they can further refine the total program's cost estimate.

In other words, Agile cost estimating requires a more iterative, integrated, and collaborative approach. Traditional programs often treat cost analysis as a separate activity, rather than as an integrated team endeavor. For an Agile program a cost estimator should be co-located with the systems engineers and developers as each release is scoped, developed, and tested. This ongoing collaboration among the customers, developers, systems engineers, cost estimators, and other stakeholders is critical to ensure agreement on requirements prioritization in the 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.

Cost estimating benefits

Cost estimating for an Agile program can be challenging, especially for teams new to Agile development.[85] However, a reliable cost estimate can provide benefits to an Agile program. For example, the cost estimate can be used to support the government budgeting process and to help inform management decisions.

Cost estimating techniques for an Agile program are similar to traditional development programs, since the federal budgeting process requires an estimate of the total cost of the program before it has been approved. However, as discussed in the GAO Cost Guide, because cost estimates predict future program costs, they are associated with uncertainty. This level of uncertainty decreases over time as the program definition increases for both Agile and traditional programs due to a better understanding of the work and more insight into the programs' productivity.

While a program can develop a rough order of magnitude cost estimate early in its life cycle, it may be challenging to precisely understand costs or schedule until the teams have established a rhythm to their work. As a result, cost estimating for an Agile program consists of an ongoing *just in time* activity tightly integrated with the activities of the developers and engineers. Moreover, the fidelity of the cost estimate increases once teams have been established to help estimate the level of work for each requirement, as described in chapter 5, and can further improve with subsequent releases as the estimating team captures performance productivity metrics for deployed releases. Furthermore, the 12-step cost estimating process described earlier in this chapter provides a framework that can be used to develop a reliable estimate and provide information for use during negotiations and in justifying acquisition decisions.

Maintaining an integrated cost estimating effort throughout the course of the program allows Agile programs to collect the data necessary to estimate the requirements/features that fit within the program's total budget as it progresses. A budget may be fixed for a single iteration, but,

[85]As discussed in the *GAO Cost Estimating and Assessment Guide: Best Practices for Developing and Managing Program Costs,* GAO-20-195G, there are many challenges to estimating software costs. These challenges will apply to Agile programs, especially when deriving an initial estimate for program initiation. See the GAO Cost Guide software appendix for more information.

if the requirements are not completed at the end of an iteration, management may need information to provide justification for additional funds and a change in the schedule. Cost estimating can provide managers with valuable information about the budget needed to maintain a certain level of support.

Scheduling best practices in an Agile environment

The *GAO Schedule Assessment Guide* (Schedule Guide) was developed in 2015 to establish a consistent methodology based on best practices for developing and maintaining high-quality schedules that forecast reliable dates. The GAO Schedule Guide discusses 10 best practices that, when followed, should result in a high quality, reliable schedule. These best practices are part of a cyclical process where each best practice is one step in that process.

These steps have been collapsed into four general characteristics for sound schedule estimating: comprehensive, well-constructed, credible, and controlled.

Just as in any other approach to program execution, developing and executing a schedule for an Agile program is important because it provides a focus on deadlines for specific goals and activities to ensure that all required actions are (planned to be) completed. For example, it identifies predecessor and successor relationships to ensure that prerequisites are covered in planning for iterations and user stories, it identifies timelines, and provides an estimate for the amount of time required to complete various functions/activities.

While the Agile software development philosophy is different from that of Waterfall development methods, the need for a high-quality program schedule is still applicable to all federal programs. All programs need to establish a schedule to be accountable for delivering a value-based outcome. To that end, Agile programs should adhere to GAO's scheduling best practices to develop a schedule. The following narrative describes the applicability and benefits of scheduling best practices for an Agile software development program by identifying key documentation differences between Agile and traditional scheduling and highlights key considerations when scheduling an Agile program.

Agile measures and scheduling best practices

Agile methods provide many useful progress indicators to inform management about the status of high-priority features. Many measures used to manage Agile development programs can be evidence that the program is meeting GAO's scheduling best practices. These items can aid in assessing the planning that program offices performed in developing their schedule. Table 14 shows GAO's ten scheduling best practices, a description of each best practice in an Agile environment, and examples of Agile measures that can be used to support the findings of that best practice.

Table 14: 10-Step Schedule Estimating Best Practices and Agile Cadence Examples

Scheduling best practice	Agile environment and scheduling best practices	Examples of Agile measures, artifacts, and documentation
Step 1: Capturing all activities The schedule should reflect all activities as defined in the program's work breakdown structure (WBS), which defines in detail the work necessary to accomplish a program's objectives, including activities both the owner and contractor are to perform.	During planning, the road map should be prioritized with input from stakeholders and subject matter experts. The schedule should include epics and features from the road map that are linked to the contract statement of work, the backlog, and all organization-specific tasks.	• Road map • Prioritized backlog
Step 2: Sequencing all activities The schedule should be planned so that critical program dates can be met. To do this, activities need to be logically sequenced—that is, listed in the order in which they are to be carried out and joined with logic.	The program schedule should reflect work at the epic and feature levels. The order of work should align with the prioritization included in the road map and iteration backlog. Additionally, any key dependencies between features should be identified, where applicable.	• Kanban board (or similar) • Government work/oversight • Road map • Prioritized backlog
Step 3: Assigning resources to all activities The schedule should reflect the resources (labor, materials, overhead) needed to do the work, whether they will be available when needed, and any funding or time constraints.	During release planning, each team member should assess his or her availability for development activities with respect to other commitments (e.g., vacations, holidays, etc.). Additionally, these assessments should account for team facilitator and other subject matter experts that could be needed to complete the planned work.	• Kanban board (or similar) • Team calendars
Step 4: Establishing the duration of all activities The schedule should realistically reflect how long each activity will take. Durations should be reasonably short and meaningful and allow for discrete progress measurement.	Durations are time boxed in Agile, which makes each release a consistent duration in the schedule. However, since requirements can fluctuate, it is important to track what work has been accomplished for each release in the schedule (see best practice #9 for more information).	• Prioritized backlog • Release plans • Road map
Step 5: Verifying the schedule can be traced horizontally and vertically The schedule should be horizontally traceable, meaning that it should link products and outcomes associated with other sequenced activities. The schedule should also be vertically traceable—that is, varying levels of activities and supporting sub-activities can be traced.	To be horizontally traceable, the program schedule should include the sequenced plan for developing all epics and features, along with all dependency information. To be vertically traceable, the program schedule should align with the Agile road map, prioritized backlog, and burn up/down charts.	• Road map • Releases included in program schedule • Prioritized backlog • Kanban board (or similar) • Burnup/down charts

Step 6: Confirming that the critical path is valid The schedule should have a valid critical path—this path defines the program's earliest completion or minimum duration. The critical path should be the path of longest duration through the sequence of activities.	In the schedule, critical path management should be performed at the epic and feature levels, since Agile software development could impact the critical path with non-Agile software development tasks. For an Agile program, the critical path is managed during iteration planning and daily standup meetings.	• Epics and features sequenced according to the road map
Step 7: Ensuring reasonable total float The schedule should identify reasonable float (or slack), which represents an estimate of the overall flexibility of the program.	Float is tracked at the epic and feature levels.	• Burnup/down charts
Step 8: Conducting a schedule risk analysis A schedule risk analysis uses a good critical path method schedule and data about program schedule risks to predict the level of confidence in meeting a program's completion date.	As mentioned, even though iterations are time boxed, a schedule risk analysis can help provide a confidence level to the schedule's end date to determine if additional resources need to be added to deliver all must have features. To do this, risk ranges should be applied to all assumptions, including the number of iterations needed and the team velocity measures. Risk analysis also helps identify threats and opportunities (e.g., team size, management support, availability of tools, etc.) facing the program. Additionally, iteration planning sessions provide valuable information on particular risks that could impact the delivery of must have features that can be used to inform the risk analysis.	• Iteration 0 planning • Iteration planning sessions • Retrospectives • Assumptions regarding the number of iterations, story points, and velocity
Step 9: Updating the schedule using actual progress and logic Progress updates and logic provide a realistic forecast of start and completion dates for program activities.	In Agile programs, feature development progress is updated at the end of each iteration and the cumulative results for all of the features and epics are displayed through burn up/down charts. Quantifiable back-up data regarding the completion of user stories should inform feature progress. Additionally, retrospectives are conducted to capture lessons learned at the end of each release to reduce future risks, improve customer commitment, and motivate teams. Demonstrations of working software determine stakeholder and customer satisfaction. Finally, daily standup meetings are conducted to check feature development status during iterations and any impediments the team is encountering. If the program requires more time to finish the epics and features, then the schedule should be extended to reflect this delay.	• Epics and features are included in program schedule • Prioritized backlog • Burnup/down charts • Retrospective summaries

Step 10: Maintaining a baseline schedule A baseline schedule is the basis for managing the program scope, the time period for accomplishing it, and the required resources. The baseline schedule is designated the target schedule.	The road map and release plans become the baseline from which to measure schedule variances. Demonstrations of working software determine stakeholder and customer satisfaction. Additionally, retrospectives are conducted to capture lessons learned at the end of each release to reduce future risks, improve customer commitment, and motivate teams.	• Iteration planning sessions • Prioritized backlog • Releases plans and reports • Retrospective reports

Source: GAO. | GAO-20-590G

Considerations for scheduling an Agile program

Since an Agile program consists of time boxed units with a fixed schedule, some have argued that conventional performance management tools, such as an integrated program schedule, should not be applied to an Agile program. Those arguments are made because Agile programs have structures and processes that are dynamic and iterative and spread planning activities throughout the program duration, compared to the traditional methods where extensive planning is performed upfront. However, we explore the following five areas in more detail to further encourage the use of high-quality, reliable schedules to help manage program risk:

- Planning for all activities
- Minimize the use of schedule constraints
- Assign resources
- Conduct a schedule risk analysis
- Develop and use a schedule baseline

The following discusses these issues and provide examples on how to apply these scheduling concepts to an Agile program.

Planning for all activities

While Agile emphasizes that only near-term work is planned in detail (e.g., the next iteration), programs need to define their overall goal in a vision and plan the releases needed to satisfy the vision. The detailed plan is subject to change, but the vision provides a high-level view and direction for the work to be accomplished for the entire program. Additionally, while the team self-organizes its own work, it must be cognizant of dependencies with other teams, related Agile and non-Agile programs, and equipment.

An integrated master schedule or similar artifact that includes Agile software development efforts should capture all the planned features needed to accomplish the program goals at an appropriate level of detail using rolling wave planning. This schedule should include all government and contractor activities. Developing an integrated master schedule for the whole program provides a comprehensive, end-to-end view of all the features necessary to accomplish the program's goals. Including features enhances the utility of the schedule as a coordination and communication tool and allows for better performance tracking and measurement. For example, additional information in the schedule helps to ensure that it can serve as the summary, intermediate, and detailed schedule. Including high-level features in the schedule is also a foundational best practice for most other scheduling best practices, because if the schedule does not contain planning for all the features for the duration of the program, it will lack horizontal and vertical traceability, a valid critical path will not exist, and the schedule's risk analysis will not be valid.

Minimize the use of schedule constraints

A common approach in Agile software development is to develop and deliver working software in fixed-length iterations, typically 2-4 weeks in length. Constraints may appear to provide a straightforward way to model the fixed start and end dates of iterations; however, using constraints reduces the utility of the schedule as a coordination tool among Agile teams, management, and other resources. The value of this coordination is highlighted by several effective practices for applying Agile methods on federal IT programs, such as effectively involving experts and other resources, addressing requirements related to security and progress monitoring, and identifying and addressing impediments at the organization level as well as within the program.[86]

Additionally, removing constraints from the schedule end dates allows the schedule to supplement the duration planning information included in other Agile tools for tracking. By managing teams to observe what work is scheduled to occur after milestones are set during early Agile planning, program managers can make key decisions (e.g., whether more resources are needed to complete the work in the set time frame or if those requirements can be completed after the Agile deadline).

Using constraints only when necessary and justified in the schedule documentation helps to ensure that planned dates in the schedule can respond dynamically to changes. Minimizing constraints increases the ability to meet other best practices as well, because constraints can make resource allocations unrealistic, reduce horizontal traceability, and make it

[86]GAO, *Software Development: Effective Practices and Federal Challenges in Applying Agile Methods*, GAO-12-681 (Washington, D.C.: July 27, 2012).

difficult to track the scope represented by giver/receiver milestones, produce an invalid critical path and result in inaccurate float calculations.

Assign resources

Because Agile emphasizes stable and self-organizing teams, one might think that resources (e.g., the developers) do not need to be explicitly assigned and managed. However, many activities require interfacing with resources outside of the program, such as activities involving subject matter experts and non-labor resources. Agile emphasizes working at a sustainable pace, including resources in the schedule can help ensure this occurs by providing insight into developers' availability and when additional equipment is needed.

Furthermore, the amount of available resources affects estimates of work and duration, so the schedule should include the labor and non-labor resources needed to accomplish the work. The level of detail used in assigning resources should be commensurate with the level of detail of activities in the schedule. For example, as more information is known about the program, additional resources, such as automated testing tools, could be identified for purchase in order to increase the productivity rate. Among other things, assigning resources helps ensure that the schedule is a useful tool for coordinating among resources so they are available when needed, that schedule estimates are valid, and that the schedule risk analysis provides a full understanding of schedule risk.

Conduct a schedule risk analysis

Agile self-organizing teams and iterative processes can be viewed as ways to mitigate risk in complex software programs. Accordingly, some might argue that conducting a schedule risk analysis is unnecessary. However, all programs face risk and uncertainty and the likelihood and consequences of each risk should be examined. For Agile programs, effective practices include developing initial plans at a high level and updating frequently as more is learned about the program. Further, the potential impact of some issues, such as technical debt or team size, should be considered earlier rather than later.

A schedule risk analysis should be conducted throughout an Agile program's iterative process to identify the risks, paths, and activities most likely to delay it and to serve as a basis for determining schedule risk contingencies or other mitigating measures. If time or resources are insufficient to conduct a schedule risk analysis for the full program or the level of detail is unclear because of rolling wave planning, the analysis should be performed on a summary version of the schedule. Additionally, as Agile emphasizes trading off scope in order to meet a fixed completion date, potential delays or opportunities and mitigating contingencies should also be considered in terms of fixed time boxes aligned with the program's cadence (e.g., number of iterations) and what desired scope (e.g., user stories in the prioritized backlog) may be affected or re-prioritized. Lastly, the schedule risk analysis should consider the risks that are most likely to delay a program. For an Agile program, this could include risks affecting team performance, such as team size or the availability and feasibility of tools and practices necessary to achieve the team's goals. For example, a commonly accepted Agile practice is the use of continuous integration to automatically run unit and integration tests every time code is checked in. This greatly increases the speed of testing and provides instant feedback on code quality, so if the team plans to use continuous integration but is not provided the resources to implement it, the program would likely not be able to meet all the requirements in the time allotted.

Develop and use a schedule baseline

A central tenet of Agile is to welcome change. As a result, teams practice rolling wave planning, in which only near-term work is planned in detail. However, welcoming change does not mean that software is developed and delivered in an undisciplined or ad hoc manner. Agile's priority to deliver software in iterations, typically in time boxed iterations of 2-4 weeks, is guided by the program's vision, which establishes a high-level definition of the cost, schedule, and scope goals for the program and provides a basis for specifying expected outcomes for each iteration. These critical features identify the program's schedule baseline and thus allows product owners to reprioritize work in accordance with the vision at the end of each iteration.

In creating the baseline using the rolling wave planning process, updates should contain enough detail to enable a collaborative agreement between product owners and developers without making schedule updates overly frequent or cumbersome. As the schedule is updated, changes should be documented in progress records and the schedule narrative. For example, this could include using data from the completed backlog and burn up/down charts. Schedule trends should be used to identify deviations from the baseline and to understand the need for changes. Developing and using a schedule baseline provides a good basis for measuring and understanding progress and maintaining accountability.

Earned value management best practices in an Agile environment

The goal of any software development process should be to maximize the flow of value to the customer. One method frequently used in the federal government to measure the value of work accomplished is earned value management (EVM), which can alert program managers to potential problems sooner than they might be discovered if only tracking expenditures. In fact, EVM is often required for programs once they reach a certain threshold.

There are other methods besides EVM that can be used to track performance for Agile programs; however, effective performance management practices should still be in place, regardless of the development paradigm. For example, volume 1 of the *DOD section 809 Report* states that the program manager should approve the appropriate program monitoring and control methods, which may include EVM.[87] The report states that these methods should provide faith in the quality of the data and, at a minimum, track schedule, cost, and estimate at completion. It adds that program managers should select the appropriate resources for their toolkit based on program characteristics. For example, an Agile programs should use real-time tools designed to track and monitor Agile software development. In other words, for EVM to work with Agile, program office staff must tailor EVM to integrate into the overall program management approach.

The *GAO Cost Estimating and Assessment Guide* methodology for developing, managing, and evaluating cost estimates is based on best practices across the federal government. It also outlines 13 activities that are fundamental to the earned value management process.

1. Define the scope of effort with a work breakdown structure.
2. Identify who in the organization will perform the work.
3. Schedule the work to a timeline.
4. Estimate the resources and authorize budgets.
5. Determine objective measure of earned value.
6. Develop the performance measurement baseline.
7. Execute the work plan and record all costs.

[87]Section 809 Panel, *Report of the Advisory Panel on Streamlining and Codifying Acquisition Regulations, Volume 1 of 3*, section 4: “Earned Value Management for Software Programs Using Agile”, (Arlington, VA: January 2018).

8. Analyze earned value management performance data and record variances from the performance measurement baseline plan.
9. Forecast estimates-at-complete using earned value management.
10. Conduct an integrated cost-schedule risk analysis.
11. Compare estimates-at-complete from earned value management (step 9) with estimates-at-complete from risk analysis (step 10).
12. Take management action to respond to risks.
13. Update the performance measurement baseline as changes occur.

To evaluate the consistency of an organization's EVM system, GAO identified three characteristics of a high-quality, reliable earned value management system that can be used to determine the overall quality of that EVM system. Table 15 displays the characteristics and best practices identified in the *GAO Cost Estimating and Assessment Guide*.

Table 15: GAO Earned Value Management Best Practices

Characteristic	Best practice
Comprehensive: a comprehensive earned value management system is in place	The program has a certified earned value management (EVM) system
	An integrated baseline review verified that the baseline budget and schedule capture the entire scope of work, risks were understood, and available and planned resources were adequate
	The schedule reflects the work breakdown structure, the logical sequencing of activities, and the necessary resources
	EVM system surveillance is being performed
Accurate: the data resulting from the earned value management system are reliable	EVM system data do not contain anomalies
	EVM system data are consistent among various reporting formats
	Estimates-at-complete are realistic
Informative: the program management team is using earned value management system data for decision-making purposes	EVM system data are reviewed on a regular basis
	Management uses EVM system data to develop corrective action plans
	The performance measurement baseline is updated to reflect changes

Source: GAO. | GAO-20-590G

The GAO Cost Guide also describes key benefits of using EVM. These include improving insight into program performance, reducing cycle time to product delivery, focusing management attention on the most critical issues, fostering accountability, and providing objective information for measuring progress. While Agile approaches should reduce program technical risks through early delivery, EVM can provide additional insight into the relationship between scope, cost, and schedule and this integrated data can be used to better inform management decisions.

Agile measures and Earned Value Management

According to the *Federal Acquisition Regulation* (FAR), an EVM system is required for major acquisitions for development, in accordance with OMB Circular A-11. The FAR also states that the government can require EVM systems for other acquisitions, in accordance with agency procedures. For example, the Department of Defense requires compliance with EVM guidelines for cost or incentive contracts greater than or equal to $20 million.[88] However, just as EVM is not applied to all traditional programs, it should not necessarily be applied to small Agile programs. The amount of effort implementing EVM on small programs may pose unnecessary costs for little value in return. However, in contrast it can be implemented on medium and large Agile programs. Table 16 shows the 13 activities of an EVM system implementation and execution with examples of how a program can meet each of the steps for an Agile program.

Table 16: 13 Earned Value Management Activities and Agile Examples

EVM Activity	Agile environment example
Activity 1: Define the scope of effort with a work breakdown structure	The work breakdown structure should be based on the prioritized backlog; however, the work breakdown structure should not extend below the feature level. Lower levels, such as user stories, should not be in the work breakdown structure, but metrics from these levels provide quantifiable backup data for measuring performance at the feature level and higher.
Activity 2: Identify who in the organization will perform the work	As in conventional programs, work assignments should be consolidated at the level of a control account manager. This is often done during iteration and release planning sessions and tracked in Agile program management tools.
Activity 3: Schedule the work to a timeline	The schedule should be based on the product road map, which shows a plan for epic and feature development across releases.
Activity 4: Estimate resources and authorize budgets	Features should be the basis for identifying work package scope and budget.
Activity 5: Determine objective measures of earned value	Progress should be tied to the completion of scope and not the completion of time boxed events. The technique used for taking credit for performance should be documented. Additional information on measuring earned value is described in this step.
Activity 6: Develop the performance measurement baseline	The performance measurement baseline should be based on the work breakdown structure and the integrated master schedule and be traceable to the product road map. The smallest building block for the performance measurement baseline is at the control account level where each control account is based on a feature or group of features.

[88]*DOD Instruction 5000.02T*, table 9 (Jan. 7, 2015, incorporating change 6, Jan. 23, 2020).

Activity 7: Execute the work plan and record all costs	The level at which effort is converted into cost in the performance measurement baseline should be defined and traceable to Agile metrics captured by the program. These metrics can vary from program to program, but some common ones to consider tracking are the iteration burn down chart, cycle time, and cumulative flow diagram.
Activity 8: Analyze EVM performance data and record variances from the performance measurement baseline	Variances should be determined at the work package level within each control account based on quantifiable backup data that supports each associated feature. For example, an iteration burn up chart can show what work that was planned was not accomplished during the iteration. Further, release retrospectives can highlight impediments that occurred during a release and highlight whether feature development is on track according to the road map developed at the beginning of the release.
Activity 9: Forecast estimates-at-complete using EVM	Metrics generated from Agile tools can typically be used to forecast estimates-at-complete. Adding the completed work and the remaining work divided by an efficiency factor yields an estimate-at-complete. The efficiency factor is calculated by dividing the completed work by the effort used to perform that work.
Activity 10: Conduct an integrated cost-schedule risk analysis	Similar to a cost risk/uncertainty analysis and schedule risk analysis, an integrated cost-schedule risk analysis can be completed by developing risk distributions around Agile-specific metrics to provide a range around the program's cost and schedule related to the total number of requirements in the prioritized backlog.
Activity 11: Compare estimates-at-complete from EVM (step 9) with estimates-at-complete from risk analysis (step 10)	These two steps should be performed for Agile programs as they are for other programs.
Activity 12: Take management action to respond to risks	
Activity 13: Update the performance measurement baseline as changes occur	Activities in the product backlog and road map at the feature level should have an assigned budget that is under baseline control. Changes to the backlog at this level should be documented and should occur in accordance with baseline change processes. Any changes that occur can be documented and reviewed by management in release retrospective notes.

Source: GAO analysis of data from DOD, National Defense Industrial Association's Integrated Program Management Division, and GAO. | GAO-20-590G

Rework, such as developers modifying or revising existing code to improve performance, efficiency, readability, or simplicity without affecting functionality, may be needed for program completion. Agile programs should include adequate budget and schedule for rework in the performance measurement baseline and integrated master schedule so these will also appear in EVM. Some programs may assign rework to a separate planning package from the original task. Alternatively, adjustments to earned value can reflect that specific features were not completed or that rework is occurring.

Some Agile programs are required to provide EVM reporting based on guidance and established reporting thresholds. These data can assist the program manager in providing oversight officials with vital program performance information. Much of the data already associated with implementing Agile can be used to support EVM reporting, so providing EVM reporting does not have to be an overly time consuming task.

Ultimately, EVM is effective for Agile programs when it is integrated with technical performance and EVM processes are augmented with a rigorous systems engineering process. The following is an example of how one program supported by existing Agile metrics reported to the Office of Management and Budget.

Agile in Action 4: Performance reporting requirements

In February 2018, we met with The National Nuclear Security Administration's (NNSA) Generation 2 (G2) to discuss how they meet the Office of Management and Budget's (OMB) Capital Planning and Investment Control (CPIC) reporting requirements for major IT Investments. Officials said that they worked closely with OMB and NNSA senior management to meet the program's CPIC reporting requirements to align the program's Agile methods. For example, officials said that G2 defines "project" as a program increment (e.g., 14 weeks comprised of seven two-week iterations).

However, because CPIC's project reporting structure did not align with G2's Agile cadence or contractors' cost reporting requirements, officials said that reporting cost and schedule variances for CPIC reports posed a challenge to G2. As a result, the program developed a repeatable and transparent way to proportion their cost and Agile cadence to the CPIC reporting structure. To determine the prorated project cost of a program increment within a month, G2 calculates the number of days for the program increment in a month compared to the total days and proportion it has to the actual effort charged for the whole program. Since the activities are time boxed with variable scope, there is no schedule variance.

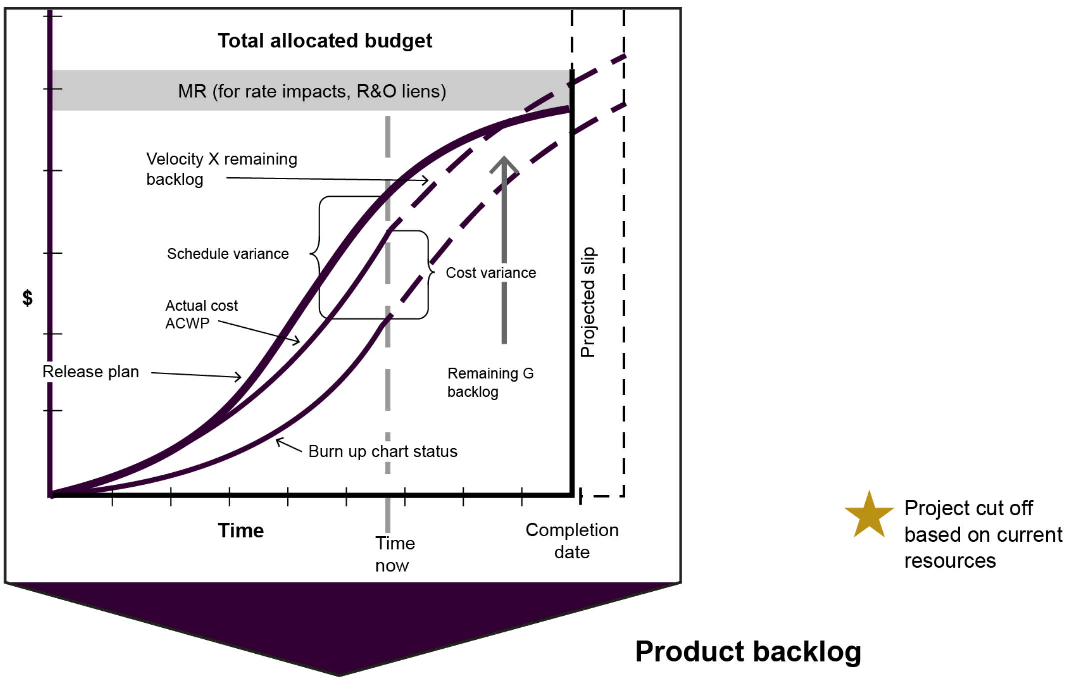
Officials said that, although this allows G2 to follow CPIC reporting requirements, resulting variances may be misleading and require further explanation. For example, G2 provided the following rationale for a cost variance in its August 2019 CPIC monthly report: "Project/activity PI12 completed on schedule and finished with a positive 3% financial variance as previously projected."

CPIC reporting also requires a documented risk register. Officials said that, while G2 addresses high-level risks through a traditional risk register, the program primarily addresses risk through activities (e.g., release planning and retrospectives) as part of using the Agile methodology. For CPIC reporting, risk actions are typically reported at a high level, tying updates to formal risk reviews for each program increment in the reporting period.

Traditional programs analyze and review EVM data on a monthly basis so that problems can be addressed as soon as they occur and cost and schedule overruns can be avoided. Then, using the EVM data, managers assess cost and schedule performance trends. When cost and schedule are not fixed for a program, EVM data shows a negative cost variance if the program will be over budget and a negative schedule variance if the program is behind schedule.

Agile programs use alternative methods to track risk in combination with a flexible scope and fixed cost and schedule; however, EVM concepts can provide managers with important insights since, for government programs, scope is flexible for an iteration or release, but is not necessarily flexible for the program as a whole. To highlight this difference, instead of monthly reports that show projected cost or schedule variances, reports could be included as part of a release retrospective summary that show, along with other metrics familiar to Agile practitioners, what the estimated cost and schedule overruns are for the program if it completes all work in the backlog. Figure 12 shows a how to visualize EVM tracking for traditional and Agile methods.

Figure 12: Traditional and Agile Earned Value Management Tracking Methods



Product backlog				Product backlog	
Release	User Stories	Feature	Work package	Work package	Budget
A	X1, X2, X3, X4, X5	X	A-00X	A-00X	$10,000
	Y1, Y2, Y3, Y4	Y	A-00Y	A-00Y	$15,000
	Z1, Z2, Z3	Z	A-00Z	A-00Z	$5,000
B	N1, N2, N3, N4, N5	N	B-00N	B-00N	$15,000
	TBD	Q	B-00Q	B-00Q	$5,000
C	TBD	R	B-00R	B-00R	$10,000
	TBD	S	B-00S	B-00S	$10,000
	TBD	T	B-00T	B-00T	$10,000

Source: GAO analysis of DOD documentation. | GAO-20-590G

Figure 12 shows that, for an Agile program, it might be appropriate to view a “cut off” point, where based on the current budget and schedule, which features can be accomplished. The figure shows that this project will be able to accomplish releases A and B, but not release C.

Considerations for applying earned value management to an Agile program

Since Agile differs from Waterfall development with respect to its treatment of requirements, some say that conventional performance management tools, such as those for EVM, should not be applied to Agile programs. Those arguments are made because Agile programs have structures and processes that are dynamic and iterative and spread planning activities throughout the program duration, whereas traditional methods perform extensive upfront planning. However, EVM can be a valuable performance management tool that decision makers can use to see how the program is progressing compared to its initial plan. The following areas should be examined for Agile programs when using EVM:

- Tracking work breakdown structure detail
- Measuring earned value
- Calculating variances
- Controlling baseline changes

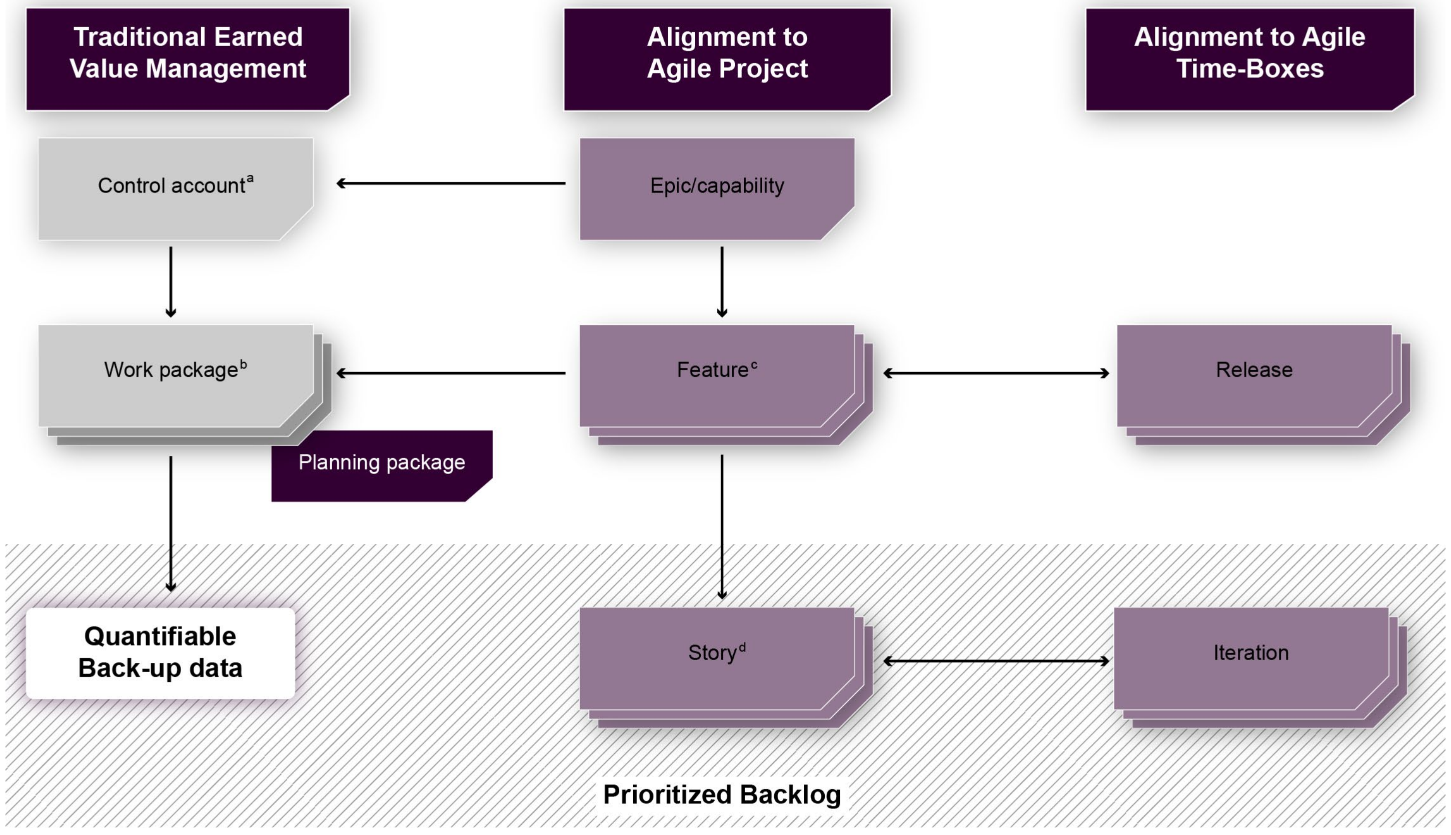
The following narrative will discuss these issues and the application of traditional EVM concepts to an Agile program.

Tracking work breakdown structure detail

One of the major concerns with applying EVM to Agile programs is the level of detail tracked in the work breakdown structure. As previously discussed, the WBS used for EVM, like the one for the integrated master schedule, should not track Agile data at the level of iterations or user stories, but should be monitored at a higher level, including features and epics. Given the dynamic nature of Agile, tracking at too low a level will not yield valuable data because of the frequent changes made. However, the Agile data at the iteration level (e.g., the prioritized backlog) should be available for use as quantifiable backup data for the work tracked in the EVM system.[89] Figure 13 shows a hierarchy of Agile products, time boxed elements, the relationships among them, how they relate to the EVM system, and the different levels where EVM data are tracked along with where Agile metrics can be used to provide quantifiable backup data.

[89]Quantifiable backup data is information that is used to gauge the progress of a capability based on the technical completion of each feature, which, in turn, is based on the accomplishment of the feature’s acceptance criteria.

Figure 13: Comparison of Traditional and Agile EVM Products



Source: National Defense Institute Association Integrated Program Management Division. | GAO-20-590G

[a]One or more sized epics define the scope for the control accounts.

[b]One or more features define the scope for the work packages.

[c]A feature consists of multiple user stories. Multiple features are implemented in each release.

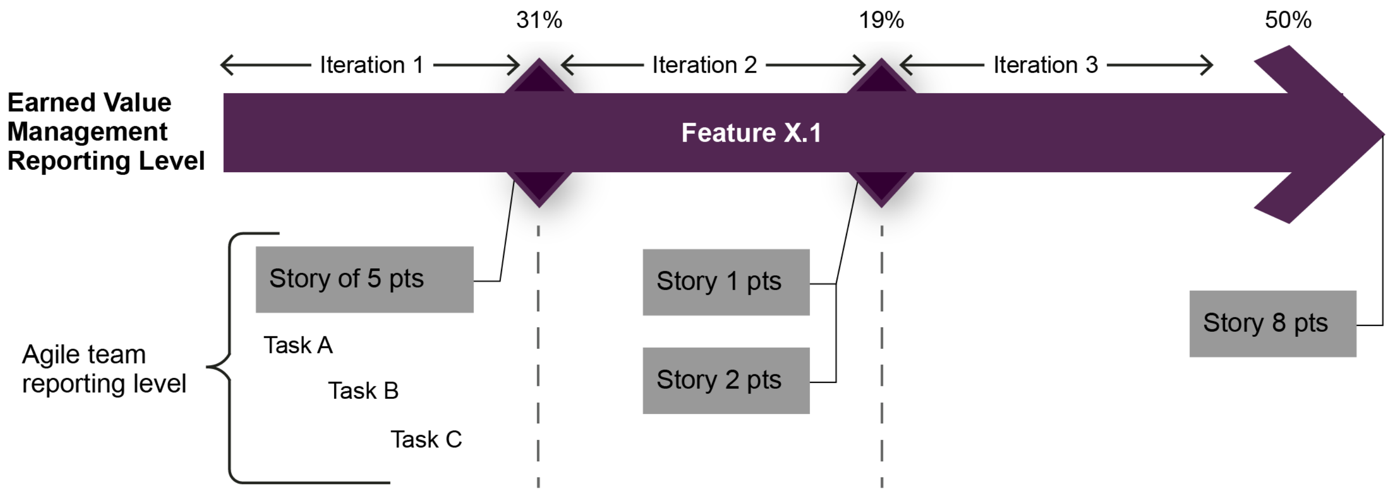
[d]A user story is a small, well-defined system function that can be developed within one iteration. Multiple user stories are implemented in each iteration.

Other structures mapping EVM to Agile relationships can be developed, but should be documented so that decision makers can easily observe what the data collected means in relationship to the work to be performed.

Measuring earned value

One way to establish EVM measures is to use the percent complete[90] method at the feature level. For example, at the feature level, percent complete is calculated based on the number of associated user stories that have been completed and some measure of the user story's weight, using the 0/100 method[91] to determine if a user story has been completed. On completion, the full credit is taken for the user story. This measure can be based on the number of story points. Figure 14 illustrates this method of measuring earned value at the feature level.

Figure 14: Example of Measuring Earned Value for an Agile Feature



Source: National Defense Industrial Association Integrated Program Management Division. | GAO-20-590G

In this example, the feature contains user stories with a combined 16 story points. When the first user story is complete, the feature is 31 percent complete because five of the total 16 story points within the feature have been completed.

Calculating variances

Since lower-level Agile requirements (captured in the prioritized backlog) are updated frequently, calculating variances between completed work and planned work can be difficult. However, every program (including

[90]In the percent complete method, performance is equal to the percent a task is complete. Percent complete should be based on underlying quantifiable measures as much as possible and be measured by the status of the resource-loaded schedule.

[91]In the 0/100 method, no performance is taken until a task has been finished. This aligns with the Agile concept of user stories; only user stories that are 100% complete are counted at the end of each iteration.

Agile programs) needs a method to measure performance. Meaningful variances require measuring performance against a baseline. As noted, the work breakdown structure should not extend below the feature level. When the work breakdown structure is established at the feature level, the variance would be calculated as follows:

- If a feature is planned to be completed in 100 hours, but it takes 200 hours to complete it, then the cost variance for that feature would be -100 hours.
- If the feature is planned to occur over three iterations, but is not complete until after four iterations, then the schedule variance for that feature is -25%, or the length of one iteration.

A similar concern is calculating the program's estimate-at-complete. In general terms, an estimate-at-complete is computed as follows, where the completed work represents the actual costs to date and the remaining work is the budgeted cost of the remaining work.

$$estimated\ total\ cost\ = completed\ work + \frac{remaining\ work}{efficiency\ index}$$

The efficiency index is based on program performance to date. For Agile programs, we can present the estimate at complete equation by replacing cost and work with effort:

$$estimated\ total\ effort\ = completed\ effort + \frac{remaining\ effort}{efficiency\ index}$$

For a feature, effort could be measured in story points and the efficiency index calculated as the ratio of total hours expended to total story points for completed iterations for that feature. Estimated total effort for larger elements, such as epics, could be calculated similarly, using story points and hours expended; however, this requires the estimation of story points to be consistent across the features that make up the epic. If different teams have different story point estimation schemes, then the estimate-at-complete will not be as accurate. In that case, it may be preferable to use feature-level data to calculate estimated total effort for the epic that comprises those features. A program level estimate-at-complete could be composed of the sum of epic-level estimates at completion. Alternatively, the program-level estimate-at-complete could also be calculated as follows, where the velocity is the completed weighted user story value across the program's development teams divided by the total length of iterations completed to date.

$$total\ project\ effort = actual\ effort\ expended\ to\ date + velocity * remaining\ effort$$

The remaining effort is the work remaining in the prioritized backlog and is measured in story points. This formulation assumes that the development teams have attained a stable velocity that will remain consistent through the end of the program. These estimated effort equations can be easily converted to estimated costs by replacing the completed effort with actual costs to date and replacing the remaining effort with the budgeted costs for the remaining backlog. That is, multiplying the effort hours by the average labor rate will convert effort to cost.

Controlling baseline changes

As mentioned previously, in order to have the ability to measure program performance, there must be a baseline to measure against. Accordingly, a process should be established to manage baseline changes. The goal of this process is to preserve the integrity of the performance baseline and to ensure it reflects the most current plan so that credible performance measurement can occur. This process creates reliable data for management to rely on for making program decisions. Initially, it may seem that a formal change process interferes with the flexibility of an Agile program to reprioritize the backlog from iteration to iteration. However, a properly designed change process will not restrict the Agile process while also maintaining a credible baseline.

The following three examples are of possible baseline changes.[92]

- If a feature was originally planned for the current release and then moved to a future release, then the associated baseline change action would be to re-plan the feature into the future release and the associated user stories would be returned to the backlog.
- If a feature is worked on during the current release, but not finished, then the unfinished user stories are moved to the next release. In most cases, this move does not constitute a baseline change; however, failure to finish the feature within the planned release will create a schedule variance and could possibly create a cost variance.
- If a feature is worked on during the current release, but the product owner removes scope from the feature or associated epic, this would

[92]Derived from the National Defense Industrial Association's Integrated Program Management Division, *An Industry Practice Guide for Agile on Earned Value Management Programs*, (Arlington, VA: May 26, 2019).

necessitate a baseline change. The feature should be finished with the reduced scope. Any budget associated with the eliminated scope should be removed from that feature and reassigned.

Best Practices Checklist: Agile and Program Monitoring and Control

Detailed best practice checklists are found in the companion guides; the *GAO Cost Estimating and Assessment Guide* (GAO-20-195G) and the *GAO Schedule Assessment Guide* (GAO-16-89G).

Chapter 8: Agile Metrics

GAO has consistently emphasized the need for organizations to collect and use data about program performance to help inform and measure organization operations and results.[93] Performance information can be measured at various stages of software development and at different levels of an organization. Such information can be used to, among other things, identify problems and take corrective actions, develop strategies and allocate resources, recognize and reward performance, and identify and share effective approaches. Accordingly, regardless of their preferred Agile development framework, organizations and programs should establish an appropriate set of metrics and associated processes to use to measure their performance goals early in the development cycle. In keeping with the Agile Manifesto, Agile metrics should be geared toward measuring outcomes and meeting customer needs.

Organizations can use the following best practices to help them develop meaningful metrics.[94]

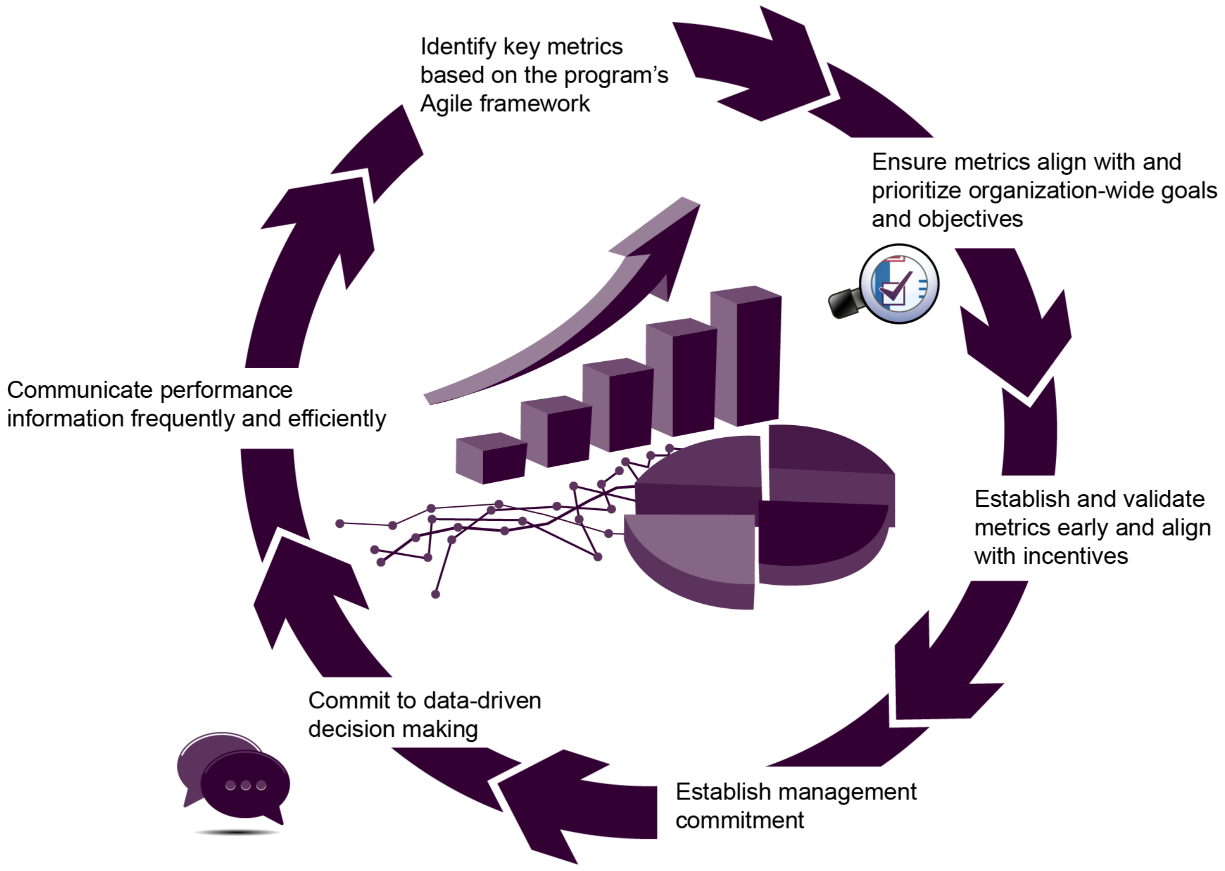
- Identify key metrics based on the program's Agile framework.
- Ensure metrics align with and prioritize organization-wide goals and objectives.
- Establish and validate metrics early and align with incentives.
- Establish management commitment.
- Commit to data-driven decision making.
- Communicate performance information frequently and efficiently.

Figure 15 shows an overview of these best practices to develop meaningful metrics and table 17 following the figure summarizes the best practices.

[93]For example, see GAO, *Managing for Results: Government-wide Actions Needed to Improve Agencies' Use of Performance Information in Decision Making*, GAO-18-609SP (Washington, D.C.; Sept. 5, 2018); *Managing for Results: Further Progress Made in Implementing the GPRA Modernization Act, but Additional Actions Needed to Address Pressing Governance Challenges*, GAO-17-775 (Washington, D.C.: Sept. 29, 2017); *Government Performance: Lessons Learned for the Next Administration on Using Performance Information to Improve Results*, GAO-08-1026T (Washington, D.C.: July 24, 2008); and *The Government Performance and Results Act:1997 Government-wide Implementation Will be Uneven*, GAO/GGD-97-109 (Washington, D.C.: June 2, 1997).

[94]Programs are unique, as are the needs of organizations where they operate. For these reasons, organizations are in a position to establish the appropriate thresholds and guardrails associated with performance metrics.

Figure 15: Overview of Agile Metrics Best Practices


Identify key metrics based on the program's Agile framework
Ensure metrics align with and prioritize organization-wide goals and objectives
Establish and validate metrics early and align with incentives
Establish management commitment
Commit to data-driven decision making
Communicate performance information frequently and efficiently

Source: GAO. | GAO-20-590G

Table 17: Summary of Agile Metrics Best Practices

Agile metrics best practice	Summary
Identify key metrics based on the program's Agile framework	• Metrics should be tailored based on a program's needs. • Different metrics are important for technical management, program management, and Agile methods. • Metrics should be tailored based on the intended audience.
Ensure metrics align with and prioritize organization-wide goals and objectives	• Connections between strategic goals and objectives should be traceable to Agile artifacts such as the road map and backlog. • Metrics facilitate feedback and communication between internal and external customers.

Agile metrics best practice	Summary
Establish and validate metrics early and align with incentives	• Metrics should motivate desired behaviors and emphasize a greater focus on results for the team rather than the individual. • Metrics can be used to measure team performance, product quality and performance, and the team's adherence to Agile development best practices.
Establish management commitment	• Management should ensure that the processes for measuring performance are established, reflect an Agile approach, and are used consistently over time. • Management must be committed to balance periodic program-wide health assessments with monitoring progress made to deploy capabilities.
Commit to data-driven decision making	• Metrics are designed to support specific decisions that need to be made at different levels of the organization. • Performance goals should be assessed frequently to match the Agile development cadence. • Metrics for performance monitoring should be identified in the contract. • Metrics should be captured using automated tools, whenever possible.
Communicate performance information frequently and efficiently	• Agile program management and software development tools are used to capture and display Agile metrics in real time.

Source: GAO. | GAO-20-590G

Identify key metrics based on the program's Agile framework

Each software development program should select and tailor its metrics according to the program's chosen Agile framework. Additionally, different types of software development will need a tailored approach. For example, customizing commercial software requires a different approach than developing custom software for specialized hardware. Metrics should also be transparent. For example, the program has a clearly stated goal or objective with a metric that clearly conveys to the Agile team what data to gather and to the customer what the metric means.

General categories of metrics include:

- Technical management (e.g., testing and integration)
- Program management (e.g., cost, schedule, and performance)
- Agile methods (e.g., collaboration or continuous improvement)

In addition to the general categories of metrics, there are different metrics for the organization, program, and team levels.

In designing performance metrics, organizations should ensure that the metrics have the key attributes of a successful metric. Specifically, metrics should be quantifiable, meaningful (e.g., have targets for tracking progress, be clearly defined, and be linked to organization priorities), repeatable and consistent, and actionable (e.g., be able to be used to make decisions). We have previously reported on the importance of ensuring that metrics reflect these attributes.[95] Without meaningful, clear, and actionable metrics, management will not have the information they need to evaluate program performance.

In addition, Agile developers and managers should tailor metrics to their intended audience. For example, developers should convey meaningful information that addresses customers specifically. Some metrics may be powerful measures for the team to evaluate its performance, but they may not be of interest to the customer and do not need to be shared with them, while others may address specific customer questions. If a program is not aligning metrics with customer questions, they may not need the data to evaluate program performance.

Although the set of metrics used to measure program performance can vary for different programs, metrics such as lead and cycle time are frequently used for all Agile programs. Lead time measures how long it takes to move from the identification of a capability or feature to when that capability or feature is to be released into the production environment. Cycle time is the time it takes from starting to work on a feature to getting it into production.

Other frequently-used metrics include how often a feature is delivered and its value. Value can be determined by measuring specific benefits derived from that feature, such as increased productivity, or measuring the use of a new feature delivered to a customer.

[95]See, for example, *Information Security: Concerted Effort Needed to Improve Federal Performance Measures,* GAO-09-617 (Washington, D.C.; Sept. 14, 2009).

Case study 13: Identify key metrics based on Agile framework, from *Immigration Benefits System,* GAO-16-467

In July 2016, GAO reported that the U.S. Citizenship and Immigration Services (USCIS) Electronic Immigration System (ELIS), the case management component of the Transformation Program, partially met the key Agile practice of monitoring and reporting on program performance through the collection of reliable metrics. GAO found some metrics were reliable and addressed their intended purpose. For example, the program provided evidence of collecting reliable metrics associated with code quality. However, other metrics were either unreliable or were not collected. For example, the program did not monitor internal USCIS user satisfaction with USCIS ELIS. Therefore, it could not measure the level of satisfaction of adjudicators or others using the system to facilitate the processing of applications.

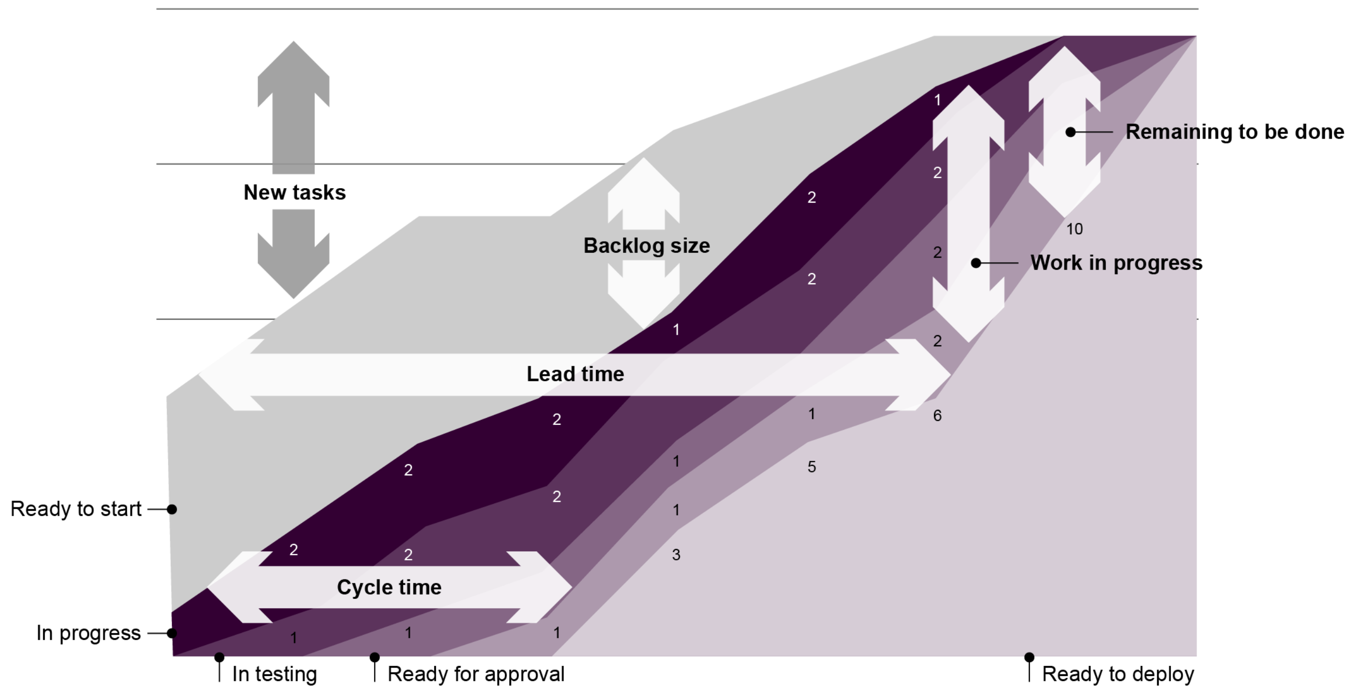
GAO reported that USCIS ELIS calculated production defect/incident metrics, automated code scanning results, code issue counts, and code development metrics to gauge the quality of code delivered during a sprint. These metrics were included as part of a monthly status report and used for high-level planning. The results of measurements associated with these metrics identified underlying challenges the program was facing with product quality. For example, production metrics showed that the rate in which issues (e.g., defects, incidents, bugs) were found exceeded the rate the issues could be closed. Such metrics may indicate a quality issue somewhere in the development process; however, the use of the metrics allows the program to identify such concerns and take steps to address them.

GAO also determined that USCIS ELIS did not measure internal user satisfaction. Officials from the Quality Assurance Team (USCIS staff responsible for the collection of program metrics) stated that they monitored issues raised by adjudicators and adjudicator representatives during program reviews and retrospectives. Further, the Chief of the Capability Delivery Division stated that the operational test agent obtained internal user feedback on USCIS ELIS. However, the Chief of the Office of Transformation Coordination explained that incident management (e.g., reporting defects or issues by the field and service centers) and operational test agent reports were not proven to be a useful tool for obtaining internal user feedback. As such, the Chief stated that the Office of Transformation Coordination was developing a method for capturing internal user satisfaction. Program officials did not elaborate on the steps the program was planning to take to collect internal user satisfaction or provide a time frame for collecting such metrics. By not establishing metrics to obtain user feedback, GAO reported that the program limited its understanding of the value being delivered with each software release.

GAO, *Immigration Benefits System: U.S. Citizenship and Immigration Services Can Improve Program Management,* GAO-16-467 (Washington, D.C.: July 15, 2016).

A cumulative flow diagram is an analytical tool that allows teams to visualize their effort and program's progress. The graph is built from different colored bands representing different tasks and shows how tasks mount up over time and their distribution along the stages of the process. Ideally, the cumulative flow diagram will show the bands rising evenly, except for the completed tasks, which should be getting taller. Figure 16 contains an example of a cumulative flow diagram. It shows five phases: ready to start, in progress, in testing, ready for approval, and remaining to be done. This example shows that there is no bottleneck as work ready to deploy expands over time.

Figure 16: Example of a Cumulative Flow Diagram



Source: © 2020 CC PACE "How to Read a Cumulative Flow Diagram". | GAO-20-590G

Lead time measures the time required for a feature in the backlog to move into production. Cycle time reports the time after work starts on a story before its goes into production. Development teams strive for lead and cycle times to be short.

Ensure metrics align with organization-wide goals and objectives

Aligning program metrics with organization-wide goals and objectives reinforces the connection between long-term strategic goals and day-to-day activities. We previously reported that organizations that have successfully adopted performance metrics ensured that the metrics were tied to program goals and demonstrated the degree to which the desired results were achieved, limited the metrics to a few that were considered essential for producing data for decision making, covered multiple priorities, and provided useful information for decision making.[96] In an Agile methodology, these connections should be traced from the road map through releases and items in the prioritized backlog, such as in the epics and user stories. If the metrics do not allow traceability from the road map through the releases and prioritized backlog, the organization may not have the right information to make decisions about prioritization and potential re-planning.

An organization should also define and organize the goals, objectives, and performance information that are appropriate to the managerial responsibilities and controls at each level of the organization. An organized structure will increase the usefulness of performance information collected by decision makers at each level by helping to ensure that metrics are aligned with management goals. Further, this alignment will reinforce the connection between strategic goals and the day-to-day activities of the development team. In addition to providing insights to the development team, Agile metrics can be tailored to convey the developers' progress and achievements to internal and external customers. This can facilitate feedback and communication between both entities.

Establish and validate metrics early and align with incentives

Early in the process, the Agile team should establish and validate the appropriate metrics to ensure they are in place to use to monitor and evaluate the team from the beginning. These metrics should be aligned with incentives for the team and be monitored at the organization, program, and team levels. Incentives will help ensure that the teams are appropriately rewarded for achieving the desired goals.[97] If metrics are not aligned with incentives, then the teams may not feel appropriately rewarded for achieving program goals.

Having incentives is particularly important in an Agile environment, as reward and incentive structures are based on team, rather than individual, accomplishments. At the same time, the Agile team should make sure that the value delivered by each metric exceeds the effort to collect the

[96]GAO, *Managing for Results: Enhancing Agency Use of Performance Information for Management Decision Making*, GAO-05-927 (Washington, D.C.: Sept. 9, 2005); *Information Security: Concerted Effort Needed to Improve Federal Performance Measures*, GAO-09-617 (Washington, D.C.: Sept. 14, 2009); and *Managing for Results: Government-wide Actions Needed to Improve Agencies' Use of Performance Information in Decision Making*, GAO-18-609SP (Washington, D.C.: Sept. 5, 2018).

[97]As mentioned in chapter 3, incentives may differ between government and contractor staff due to contract requirements and forms of recognition available.

data because if the effort to collect data to support a metric is too extensive, the metric may not deliver enough value to justify its collection.

In addition to Waterfall development metrics, various Agile frameworks are associated with metrics that can help determine the status of software development efforts at the team level. Examples of these metrics include:

- velocity (volume of work accomplished in a specific period of time by a given team)
- features or user stories delivered[98]
- number of defects or bugs
- cumulative flow
- customer satisfaction
- time required for full regression test
- time required to restore service after outage

For example, velocity is a metric that quantifies the work developers can deliver in each iteration. Velocity is reported in story points and can be captured using a type of chart called a burn up or burn down chart. A team can use historical velocity data from a previous iteration as it plans future work. Note that this metric is specific to a team and therefore cannot be used for comparison across teams.

Other effective measures of team performance are the number of user stories completed in an iteration and whether any were carried over to the next iteration. Some metrics measure the flow of work over time through the use of cumulative flow diagrams or by reporting the number of features delivered in each iteration or release. Other metrics are associated with product quality and performance. An example of a metric associated with product quality is the number of defects identified after deploying a product into the production environment. Various tests at different development stages also help ensure a quality product. A program may also capture metrics that measure a team's adherence to Agile software development best practices. Some of these metrics are described in chapter 6, which discusses the execution of contractual obligations.

[98]A further elaboration of this metric may consider user stories or story points committed versus user stories or story points accepted.

Agile in Action 5: Health radars

In October 2014, GAO met with Agile Transformation, a consulting company that offers tools and coaching to Agile programs, to discuss the AgilityHealth® platform, which is a tool for continuous measurement and growth platform used to provide companies visibility into the performance and health of their program teams, product lines, and portfolios. According to those interviewed, the assessments help evaluate maturity, performance, and delivery of outcomes on an individual, team, program, or organization level. One way to collect and review this data is through a "health radar". Each radar provides a comprehensive picture of a program and team over time and can indicate whether an Agile implementation is progressing as planned. Documentation provided shows that the radars are shaped like a wheel and delineate metrics into three levels: key areas are labeled on the outer most edge and then are divided into drivers in the second level and each driver is then divided into success metrics. For example, one driver may be "Manage changing business priorities," with the following associated metrics: existence of single backlogs to manage work for each portfolio/program/team; business customer engagement and ownership of managing their backlog ranking; and continuous backlog refinement processes that manage the addition, removal, re-ranking, slicing, or renaming of user stories. Each metric is associated with a set of questions based on a maturity scale to be answered by Agile team members. The result is an AgilityHealth radar score for that objective. The Agile Transformation Program Office said that the assessment is typically performed at a release retrospective, perhaps once a quarter. The meetings are facilitated by a certified AgilityHealth facilitator, who explains the questions associated with each metric and helps ensure the assessment is objective.

Agile Transformation said that teams can use the TeamHealth Assessment to review its strengths, improvements, and impediments and then build a growth plan with the most important areas it wants to improve in the next quarter. They showed us their secure portal where these results are benchmarked against the results of other teams and industry for comparison. AgilityHealth's assessments are one tool that can be used to provide Agile programs a consistent way to measure health and performance of teams, product lines, and portfolios, and a holistic view for how the program is performing.

Establish management commitment

The commitment of an organization's managers to establishing effective performance metrics and using performance information to inform program decisions is critical to program success. Management should ensure that the processes for measuring performance are established and used consistently over time, including establishing procedures, monitoring the establishment and use of performance metrics, and taking the necessary corrective actions. Management should also perform health assessments to ensure that adequate resources, including people, funding, and tools are provided so performance management and evaluation activities can be implemented appropriately at various levels. Management can also issue guidance or procedures for programs using Agile methodologies. Guidance or procedures can include the metrics used to evaluate the program and help ensure that the necessary tools are in place to support automation and Agile program management and reporting.

Management commitment to using performance metrics is critical when adopting and using performance information for program decisions and evaluation. Managers demonstrating their willingness and ability to make decisions and manage programs on the basis of results and inspiring others to embrace such a model are important indicators of management’s commitment. For example, if management determines that a program is not achieving its intended results in a timely manner, management should take steps to identify changes that will help the program better achieve its intended results. If management does not demonstrate a commitment to use performance metrics, others may not embrace metrics as useful.

At an organization level, management should allow programs to tailor metrics to ensure that they meet organization needs while also limiting unnecessary work on the part of the program. For example, organizations might consider calling for programs to establish a dashboard that can provide management with real-time updates on a program’s progress and success. Regardless of the tailored set of metrics used by a program, organization management needs to have information to hold an Agile program accountable.

Management must also be committed to balancing periodic program-wide health assessments with monitoring the progress made in deploying capabilities during each release. Agile cadence enables frequent, regular performance review meetings to discuss progress made toward achieving the desired results. Staff from different levels of the organization should be involved in performance review meetings to assess a program’s progress and results and to discuss any issues or concerns raised. Involving staff from different levels helps to ensure that decisions can be made efficiently with a view toward course correction if necessary. To achieve this, the feedback loop needs to be short.

Case study 14: Frequent performance reviews, from *TSA Modernization,* GAO-18-46

In October 2017, GAO reported that the Transportation Security Administration's Technology Infrastructure Modernization (TIM) program management office conducted frequent and regular performance reviews. Specifically, program management officials monitored TIM's performance and progress during weekly program status review meetings and in periodic Agile reviews that were conducted at the end of each release. The program used an automated tool to track and maintain a complete list of all corrective actions that had been identified and monitored these actions during weekly program status reviews. The periodic Agile reviews included officials from the development teams and program stakeholders. The reviews focused on, among other things, velocity, progress, and product quality. They also included the status of key activities and risks impacting cost, schedule, and performance. TSA had documented processes for the program's Agile milestone reviews, such as conducting workshops at the end of the release cycle to perform a system demonstration, reviewing qualitative metrics, and promoting continuous quality improvement.

However, GAO reported that, while the program management office used performance metrics, the program had not established thresholds or targets for acceptable performance levels for these metrics. Program officials said that they planned to develop targets based on the capacity of work that development teams are expected to complete in a release, but the program had developed three releases and continued to lack performance thresholds and targets. GAO reported that until program officials established performance thresholds and targets, oversight bodies may lack important information to ensure the program is meeting acceptable performance levels.

GAO, *TSA Modernization: Use of Sound Program Management and Oversight Practices is Needed to Avoid Repeating Past Problems,* GAO-18-46 (Washington, D.C.: October 17, 2017).

Commit to data-driven decision making

An organization realizes the benefits of collecting performance information when management commits to using the information to make decisions aimed at improving results. Since the success of an Agile software development program is measured in the value delivered to the customer, metrics should be designed to support specific decisions that need to be made at different levels of the organization. There are many dimensions of the software development program that inform how valuable the software is to the customer and how efficiently the work is being completed. Decision makers, developers, and customers need to have insight into the people, processes, technology, quality, and cost, schedule, and performance of the program to determine the value the program is delivering. The actual metrics used to evaluate performance depend on the specific circumstances of the program, such as type of development program, the maturity of its Agile adoption, the program team, and the size and complexity of the program. Metrics typically fall into three categories: technical, performance measurement, and process improvement.

Frequent assessment of performance goals across different program dimensions allows management to determine whether Agile development activities contribute to organization goals as planned. Furthermore, these metrics reviews should match the cadence of the development process in order to provide timely feedback or take the necessary corrective actions. To help guide such reviews, organization or program management should establish target values for critical metrics. For example, a program should have established expectations for how long it should take from the time of program launch to its deployment of minimum viable product (or base functionality). Similar target values should have been established for deploying high-priority functionality to production and fixing software bugs found in production. Product quality and customer satisfaction should be monitored throughout the development life cycle. Given the frequent interaction with customers, changing priorities should be monitored as well. If the metric review schedule does not match the cadence of the development process, then management may not be able to provide timely feedback to take the necessary corrective actions in order to maximize the value of delivered software.

Much of the software development work conducted for the federal government is conducted by contractors. With regard to monitoring contractors' performance, the requirements captured in the contract will form the basis for performance monitoring. Examples of metrics could include software size, development effort, schedule, requirements definition and stability, staffing, progress, computer resource utilization, and number of working capabilities delivered and in operation. Contracts should be formulated in a way that allows flexibility for implementation and, at the same time, provides meaningful information to decision makers. If contracts are not formulated to capture the requirements to align with Agile processes, decision makers may not have the meaningful information they need to manage development.

With respect to overall program performance, a program may rely on earned value management reporting generally applied to conventional development efforts to gain insight into the costs associated with delaying work or missing a milestone. More details on applying EVM to Agile programs are provided in chapter 7. Additionally, a program may estimate the cost of technical debt and time and effort necessary to repay the debt. The program may also measure and monitor the frequency of releases as well as product delivery and progress. Earned value management has been used successfully to monitor progress in a variety of environments. Nevertheless, some Agile practitioners may feel that EVM does not capture the information that they seek to help manage their programs and they prefer to rely on other metrics. Notably, teams rely on burn up and burn down charts to communicate progress during iterations, and the backlog across iterations and within releases to track and measure value. As mentioned in chapter 7, a work breakdown structure tied to a program's Agile structure can help implement EVM reporting and ensure the program collects metrics to measure overall program performance. Without collecting metrics for overall program performance, organizations will not have a good understanding of the cost and time required to achieve a valuable product.

Further, metrics should be captured, to the greatest extent possible, by automated tools already in use by a program, such as Agile program management suites, version control systems, testing, or continuous integration pipelines. Programs should use automated tools, as they capture a variety of metrics that can be a starting point before additional resources are committed to developing other metrics. Automated tools and the availability of data may also enable programs to use advanced analytics to determine their status. The data collected should be evaluated for its completeness, comprehensiveness, and correctness to ensure that it is suitable for its intended purpose. Otherwise data can mislead decision makers instead of accurately informing them about the program's status.

Testing is an area where automated tools are critical for providing instant feedback to developers. Automated testing can support unit and regression testing, as well as static code analysis. An automated approach to code testing can reveal defects early in the development process. Our prior work has emphasized the importance of monitoring and using data from automated testing to inform program decision making.[99] An absence of automated testing or an over-reliance on manual testing can be an indicator of an organization that is still maturing in the adoption of Agile practices. (See chapter 3.)

Data obtained from automated tools will not be sufficient to inform all aspects of program performance. For example, data related to team dynamics and other organization behaviors will also need to be captured

[99]Because Agile operates differently from previous approaches, earned value management applied to Agile programs leverages different artifacts to measure progress. These are treated in more detail in chapter 7.

using tools other than those used in software development. Accordingly, this data should be augmented with data from other sources, such as periodic surveys or questionnaires, to provide a complete view. Without data collected by using both automated tools and other data collection processes, decision makers may not be able to determine if the program is delivering its desired value and outcomes.

Communicate performance information frequently and efficiently

Agile software development methods employ short delivery time frames for deploying usable features to the customers. The short time frames require that progress be tracked daily and be made visible to all stakeholders at all levels of the organization to enable feedback as quickly as possible. The relevance, reliability, and timeliness of metrics help mitigate Agile adoption and program execution risks.

Agile program management and software development tools provide capabilities for capturing and displaying key Agile metrics that can help enable frequent and efficient communication of performance information. These tools can greatly facilitate access to and dissemination of performance metrics. Co-located teams can also display the information using whiteboards or other means of visual communication that don't rely on software tools. These "information radiators"—highly visible and easily accessible physical or electronic displays of information—can improve communication of performance information among staff and stakeholders. Such improvements in information dissemination can facilitate better use of performance information.

Frequently reporting performance information allows decision makers to take action in a timely manner to make improvements or corrective actions. For example, providing frequent data on the number of defects found versus the number of defects addressed can help identify and address issues that may be rooted in architectural or code-based decisions. However, while performance information should be reported frequently, it should also be reliable and traceable back to requirements so that decision makers are aware of its value.

Case study 15: Reporting reliable metrics to management, from *Immigration Benefits System,* GAO-16-467

In July 2016, GAO reported that the U.S. Citizenship and Immigration Services (USCIS) Electronic Immigration System (ELIS), the case management component of the Transformation Program, lacked traceability between its reporting and planning metrics which miscommunicated performance to management. USCIS ELIS was reporting the scope of each release in the form of sub-features to be delivered within each release. The program identified the planned number of sub-features to be developed in each release and updated this number to reflect the actual number of sub-features developed. Based on review of the backlogs for releases 6.1, 6.2, and 7.1, GAO found the program had not fully documented if it was delivering the sub-features it had intended to deliver in each release. The backlogs provided to GAO in March 2016 included a field termed “traceability,” which mapped a user story to a supporting sub-feature and/or feature. According to this field:

- Six of the nine sub-features were not developed or were not clearly traceable to the backlog for release 6.1.
- The one sub-feature associated with release 6.2 was not developed or was not clearly traceable to the backlog.
- Nineteen of the 28 sub-features were not developed or were not clearly traceable to the backlog for release 7.1.

GAO reported that, in a written response, the Business Integration Division of the Office of Transformation Coordination recognized issues in traceability of user stories to sub-features. This division stated that the process that was used to verify the number of sub-features implemented against planned was based on verbal confirmation from the product owner. The division subsequently determined that this process was not effective since it relied solely on the review of the user stories and was not as exact and reliable as expected. As a result, the division stated that there could be sub-features that were reported as implemented by the product owner but that would not show any associated user stories because they were not directly mapped to the sub-feature in the software management tool. The lack of traceability between scope metrics reported by the program and the release backlogs indicated a level of unreliability in reporting on scope. The continual need for additional effort after delivery of a sub-feature raised additional concerns regarding the extent to which the program had effectively forecasted future work in its cost and schedule projections. The division noted that requirements traceability is critical to avoid scope creep and to demonstrate that the user stories implemented addressed mission needs.

GAO, *Immigration Benefits System: U.S. Citizenship and Immigration Services Can Improve Program Management,* GAO-16-467 (Washington, D.C.: July 15, 2016).

Automated tools and dashboards with current information can be used to provide real-time input into oversight and decision making. Under the right circumstances, automated dashboards have the potential to help management view data consistently across programs. For these tools to be useful, the information displayed must be carefully reviewed. An example of such a tool is a visible burn up or burn down chart; a tool to track the progress to the program's completion. In a burn up chart, the horizontal axis represents time, while the vertical axis tracks progress measured in story points. Burn-up charts show how past iterations reveal cumulative story points completed since the beginning of the program. In combination with the product vision and road map, such information can inform management decisions about resources and funds by tracking the progress of the development program.[100] A burn up chart can track progress for releases or iterations. Burn down charts can be used in a similar fashion to help the team track progress toward requirements. After analyzing historical data, the team can project minimum, average, and maximum velocities to estimates when it will complete all the story points.

Similarly, developers can create dashboards for customers to encourage feedback so the team can address issues and concerns early. Without automated tools, management may not have access to data that allows them to assess all programs consistently and quickly.

Best Practices Checklist: Agile Metrics

1. Identify key metrics based on the program's Agile framework
   - Metrics are tailored based on the program's needs
   - The metrics support their intended use:
     - technical management
     - program management
     - Agile methods
   - Metrics are tailored based on the intended audience
2. Ensure metrics align with and prioritize organization-wide goals and objectives
   - Connections between strategic goals and objectives are traceable to Agile artifacts such as the road map and backlog
   - Metrics facilitate feedback and communication between internal and external customers
3. Establish and validate metrics early and align with incentives
   - Metrics should motivate desired behaviors and emphasize a greater focus on results for the team rather than the individual

[100]Another Agile tool—the burn down chart—represents the remaining work (on the vertical axis) over time (on the horizontal axis).

- Metrics can be used to measure team performance, product quality and performance, and the team’s adherence to Agile development best practices

4. Establish management commitment
   - Management has ensured that the processes for measuring performance are established, reflect an Agile approach, and consistently used over time
   - Management is committed to balance periodic program-wide health assessments with monitoring progress made to deploy capabilities
5. Commit to data-driven decision making
   - Metrics are designed to support specific decisions that need to be made at different level of the organization
   - Performance goals are frequently assessed to match the Agile development cadence
   - Metrics for performance monitoring are identified in the contract
   - Metrics are captured using automated tools, whenever possible
6. Communicate performance information frequently and efficiently
   - Agile program management and software development tools are used to capture and display Agile metrics in real time

Appendix I: Scope and Methodology

Our objective was to identify and describe Agile software development practices, key challenges that agencies face in applying these practices, and best practices for Agile adoption, execution, and program control and monitoring. This guide provides a brief overview and background of Agile software development practices and the challenges faced by federal agencies as they acquire and manage IT systems and transition to and manage Agile software development. In addition, the guide lays out some of the risks to Agile adoption that face organizations, programs, or teams; providing Agile adoption, execution, and control best practices. This guide is not meant to encompass all aspects of software development or program management.

To develop these best practices, we reviewed information from a variety of sources related to Agile adoption and compiled a draft of leading practices commonly mentioned across these different sources.[1] We also convened a working group of knowledgeable specialists that met with us between August 2016 and August 2019 to review and discuss these best practices. We established the composition of the working group by contacting our cost and schedule working group members to identify those specialists with Agile program management expertise. We also sent letters of inquiry to both the General Services Administration and the Chief Information Officer's Council to identify additional specialists with Agile technical expertise. The group expanded over time through referrals from group members and inquiries to GAO throughout the course of our audits on Agile programs.

The group met at GAO headquarters, both in person and via telephone, three times a year. The meetings were open to all with interest and technical expertise in Agile (e.g., developers), as well as program managers and organization executives. Meeting members were from government organizations, private companies, independent consultant groups, trade industry groups, and academia from around the world.

[1]See, for example, Booz Allen Hamilton, *Agile Playbook, Version 2.0* (Washington, D.C.: June 2016); California Department of Technology, California Project Management Office, *Understanding Agile, Version 1.0* (California: Dec. 5, 2016); National Association of State Chief Information Officers and Accenture, *Agile IT Delivery: Imperatives for Government Success* (Washington, D.C.: 2017); Office of Management and Budget, U.S. Digital Services, *Playbook* (version pulled on Dec. 22, 2017*); TechFAR: Handbook for Procuring Digital Services Using Agile Processes* (version pulled on Mar. 8, 2018); Project Management Institute, Inc. *Agile Practice Guide*, 2017; and Software Engineering Institute, *The Readiness & Fit Analysis: Is Your Organization Ready for Agile?* (Pittsburgh, PA: Apr. 2014). A complete list of references is included at the end of this guide.

Prior to each meeting, we sent an agenda to the working group mailing list of approximately 400 knowledgeable specialists, and received feedback and discussion on agenda items through in-person discussion, telephone participants, and email. In-person, the meetings provided an open forum for the working group and all discussion and opposing views and were documented and archived. We used information from these discussions and analysis of literature to inform the information in this guide.

We identified best practices in the areas of Agile adoption, execution, and program control and monitoring and best practices to establish Agile metrics. For each set of best practices, we reviewed available information and discussed practices and terminology with our working group of knowledgeable specialists. We then developed draft Agile guide chapters and asked the working group to review the chapters and provide feedback, both during meetings and by email.

To develop this exposure draft, we asked for comments on sections of the original draft and vetted each comment received on whether it was (1) actionable, (2) within scope, (3) technically correct, and (4) feasible. During development of the exposure draft we received and vetted 912 comments from our working group. We made appropriate changes throughout the guide to reflect these comments.

To supplement the information included in the guide's contents, we presented case studies and Agile in action cases as examples. Case studies were taken from GAO reports and highlight problems typically associated with a specific Agile practice. These examples were chosen to augment key points and lessons learned that are discussed in the guide. Agile in Action excerpts feature practices adopted by programs and organizations we interviewed that we believe illustrate Agile key practices executed in an exemplary or innovative way. The difference between a case study and an Agile in action example is that the Agile in action examples are not based on published GAO reports, but rather our research, interviews, and self-reporting entities. For more information on case studies and Agile in action examples, see appendix VII.

Consistent with our methodology for best practice guides, this public exposure draft is released for 12 months for input and feedback from all who are interested. Please click on this link https://tell.gao.gov/agileguide to provide us with comments on the Guide.

Appendix II: Key Terms

The terms and definitions provided in this appendix are intended for this guide. These terms can be both contextually and organizationally dependent.

Acceptance criteria: These criteria by which a work item (usually a user story) is judged to be successful or not; either “all or nothing”, it is “done”, or “not done”. Acceptance criteria are developed to identify when the user story has been completed and meet the preset standards for quality and production readiness.

Acceptance testing: Formal testing conducted to determine whether or not a user story satisfies its acceptance criteria in preparation for the customer to accept or reject it.

Affinity estimation: An estimating technique used to quickly estimate for release planning a large number of user stories and story points. It is often used when a project has just started and has a backlog that has not been estimated yet. It gives new programs an idea of how to scale user stories and helps communicate that information to stakeholders. This is related to a group estimation technique known as wide-band Delphi from traditional planning.

Agile: An umbrella term for a variety of best practices in software development. Agile software development supports the practice of shorter software delivery. Specifically, Agile calls for the delivery of software requirements in small and manageable predetermined increments based on an “inspect and adapt” approach where the requirements change frequently and software is released in increments. More a philosophy than a methodology, Agile emphasizes early and continuous software delivery, fast feedback cycles, rhythmic delivery cadence, the use of collaborative teams, and measuring progress in terms of working software. There are many specific methodologies that fall under this category, including Scrum, eXtreme Programming, and Kanban.

Architecture: a set of values and practices that support the active evolution of the planning, designing, and constructing of a system. The approach evolves over time, while simultaneously supporting the needs of current customers.

- **Enterprise architecture** is the conceptual model of principles and practices to guide organizations through the structure, operation, information, process, and technology changes necessary to execute and achieve their current and future strategies and objectives. These practices use the various aspects of an enterprise to identify, motivate, and achieve the necessary changes.

- **Functional architecture** is the infrastructure and road map used to fully address the needs of the system in the present and in the future.
- **System architecture** is the conceptual model that defines the structure, behavior, and views of a system, organized in a way that supports reasoning for its structures and behaviors.

Backlog: The backlog is a list of features, user stories, and tasks to be addressed by the team, program or portfolio and is ordered from the highest priority to the lowest priority. A backlog includes both functional and non-functional work, including technical team-generated user stories, features, or epics. If new requirements or defects are discovered, they are added to the backlog. A backlog can occur at varying levels; for example, a product backlog is a high-level backlog that contains all the requirements for the entire program, and an iteration backlog includes a list of user stories intended for that iteration.

Backlog refinement: The process for keeping the backlog updated by adding detail and revisiting the order and estimates assigned to work that teams agree to be necessary. This allows details to emerge as knowledge increases through feedback and learning cycles. This is also called “backlog grooming.”

Business manager: A person who uses program management techniques and Agile principles to deliver business value. This person is responsible for removing impediments, stimulating empowerment, collaboration and communication, and makes decisions that ensure a sustainable pace.

Business sponsor: Owns the business case for a program and is responsible for the business solution. The sponsor is usually the most senior person on the program and typically allows the program to progress without interference; generally only getting involved with escalated issues.

Burn-down chart: A visual tool displaying progress via a simple line chart representing the remaining work (vertical axis) over time (horizontal axis). It shows where the team stands regarding completing the tasks that comprise the backlog items. Related to the burn-up chart, except burn-down charts display remaining work instead of work accomplished.

Burn-up chart: A visual tool displaying progress via a simple line chart representing work accomplished (vertical axis) over time (horizontal axis). Burn-up charts are also typically used at the release and iteration levels. They are related to the burn-down chart except they display accomplished work instead of remaining work.

Cadence: The rhythm and predictability that a team enjoys by delivering in consistent time boxes.

Capacity: The quantity of resources available to perform useful work.

Champion: Spreads Agile principles and continually makes adjustments to Agile practices that suit the environment for successful outcomes. Their goal is to assist with Agile adoption and transformation and influence others, regarding the Agile process.

Coding standards: An agreed upon approach for programming style, practices, and methods. Coding standards keep the code consistent and comprehensible for the entire team to read and refactor. The concept is that code that looks the same encourages collective ownership.

Collective code ownership: A software development principle popularized by eXtreme Programming. Its principle is that all contributors to a given codebase have access to and are jointly responsible for the code in its entirety. Collective code ownership, as the name suggests, is the explicit convention that “every” team member is not only allowed, but has a positive duty, to make changes to “any” code file as necessary: to complete a development task, to repair a defect, or to improve the code’s overall structure.

Complexity point: Units of measure used to estimate development work in terms of complexity but not effort.

Continuous delivery: Continuous delivery is one of the principles of the Agile Manifesto. Continuous delivery builds on continuous integration by taking the step of orchestrating multiple builds, coordinating different levels of automated testing, and moving the code into a production environment in a process that is as automated as possible.

Continuous deployment: Continuous deployment builds on continuous delivery and is a software delivery practice in which the release process is fully automated in order to have changes promoted to the production environment with little or no human intervention.

Continuous integration: Teams practicing continuous integration seek two objectives: to minimize the duration and effort required by “each” integration episode and to be able to deliver at any moment a product version suitable for release. In practice, this dual objective requires an integration procedure that is reproducible at the very least, and mostly automated. This is achieved through version control tools, team policies and conventions, and tools specifically designed to help achieve continuous integration.

Could have: Refers to those features that are not critical for the program. While these features have a higher priority than nice to have features, they do not need to be delivered as part of the core capabilities. (See also: should have, must have, and nice to have.)

Cross-functional team: A team that is made up of people who have a mix skills and ability to define, build, and test ideas into a working product.

Customer: Synonymous with business sponsor because the customer is ultimately the user of the solution. The customer is an integral part of the development and has specific responsibilities depending on the Agile methods used. The customer wants continuous improvement of products and services.

Daily standup meeting: A brief, daily communication and planning forum where the developers and other relevant stakeholders evaluate the health and progress of the iteration. Attendees also discuss any impediments to their planned progress.

Definition of done: A predefined set of criteria that must be met before a work item is considered to be complete. This set of criteria serves as a checklist that is used to check each work item for completeness and used as the work item’s artifact.

DevOps: An extension of Agile that includes operations and all other functions that support the application development life cycle to increase efficiency, consistency, quality, and sustainability.

Epic: A large user story that can span an entire release or multiple releases. An epic is progressively refined into features and then into smaller user stories that are at the appropriate level for daily work tasks and are captured in the backlog. It is useful as a placeholder to keep track of and prioritize larger ideas.

Evolutionary development: The evolutionary strategy develops a system in builds, but differs from the incremental strategy in acknowledging that the customer need is not fully understood and all requirements cannot be defined up front. In this strategy, customer needs and system requirements are partially defined up front, then are refined in each succeeding build.

eXtreme programming (XP): A software development approach based on the values of communication, simplicity, feedback, and respect. Some of XP's core practices are: test-driven development, refactoring, pair programming, collective ownership, continuous integration, coding standards, and sustainable pace.

Feature: A functional or non-functional distinguishing characteristic of a system that, can be an enhancement to an existing system. Features are a customer-understandable, customer-valued piece of functionality that serves as a building block for prioritization, planning, estimation, and reporting.

Framework: A collection of values, principles, practices, and rules that form the foundation for development.

Function point: A unit of measure for functional size that looks at the logical view of the software code accounting for external inputs, external outputs, external inquiries, external interface files, and internal logical files.

Integration testing: The phase in software testing in which individual software modules are combined and tested as a group. It typically occurs after unit testing and before validation or acceptance testing. Organizations without continuous integration/continuous development (CI/CD) need integration testing at the end of iterations, but those with CI/CD do not.

Iteration: A predefined, time boxed and recurring period of time in which working software is created. Instead of relying on extensive planning and design, an iteration relies on rework informed by customer feedback.

Kanban: The term "Kanban" is Japanese and derived from roots that translate to "visual board". Kanban's focus is to optimize the throughput of work by visualizing the flow of work through the process, limiting work in progress, and explicitly identifying policies for the flow of work. Kanban has distinct differences from other popular Agile methodologies, primarily the fact that it is not based on time boxed iterations, but rather allows for continuous prioritization and delivery of work.

Kanban board: Unlike a task board, the Kanban board is not reset at the beginning of each iteration; its columns represent the different processing states of a unit of value, which is generally (but not necessarily) equated with a user story; each column may have associated with it a work-in-progress limit. The priority is to clear current work-in-progress, and team members will “swarm” to help those working on the item blocking the flow of the work.

Kanban method: An approach to continuous improvement that relies on visualizing the current system of work scheduling, managing flow as the primary measure of performance, and whole-system optimization. As a process improvement approach, it does not prescribe any particular practices. Agile teams employing a Kanban method may deemphasize the use of iterations, effort estimates and velocity as a primary measure of progress; rely on measures of lead time or cycle time instead of velocity; and replace the task board with a “Kanban board.”

Minimum viable product: The simplest version of a product that can be released. A minimally viable product should have enough value that it is still usable, demonstrates future benefit early on to retain customer buy in, and provides a feedback loop to help guide future development.

MoSCoW: A prioritization technique used to reach a common understanding with stakeholders on the importance placed on the delivery of each requirement, it is also known as MoSCoW prioritization or MoSCoW analysis. MoSCoW is an acronym for, must have features, should have features, could have features, and will not have features.

Must haves: Those features that are critical for a program; these are the features that must be delivered as part of the requirements. In addition to must have features, there are also should have, could have, and nice to have features.

Nice to have: Those features that are not critical for the program’s success. These are the features that are developed if there is enough time or money to develop them.

Pair programming: Two developers working side-by-side to develop code and how may frequently switch roles to complete tasks. This method of programming provides a real-time code review, allowing one developer to think ahead while the other thinks about the work at hand, and it supports cross-training. The concept can also be extended to pair designing and pair unit testing to provide real-time peer reviews. Pair programming is a fundamental part of XP.

Peer inspections: A form of code review performed by a peer that occurs after the code is complete to ensure consistency.

Product: A tangible item produced to create specific value to satisfy a want or requirement.

Product owner: The “voice of the customer,” the person who is accountable for ensuring business value is delivered by creating customer-centric items (typically user stories), ordering them, and maintaining them in the backlog. The product owner defines acceptance criteria for user stories. In Scrum, the product owner is the sole person/entity responsible for managing the backlog. The product owner’s duties typically include clearly expressing the backlog items, prioritizing the backlog items to reflect goals and missions, keeping the backlog visible to all, optimizing the value of development work, ensuring that the developers fully understand the backlog items, and deciding when a feature is “done.” A product owner should be available to the team within a reasonable time for both decision making and empowerment.

Program: The result of a development effort. In the context of this guide, a program can also be called a project or can refer to multiple projects managed as one program.

Quality attribute: A factor that specifies the degree of an attribute that affects the quality that the system or software must possess, such as performance, modifiability, or usability.

Refactoring: Refactoring involves modifying code to improve performance, efficiency, readability, or simplicity without affecting functionality. It is done after automated regression tests are written to ensure that existing functionality has not actually been affected with the modifications. Generally considered part of the normal development process, refactoring improves software longevity, adaptability, and maintainability over time.

Regression testing: A type of software testing that verifies that software that was previously developed and tested still performs correctly after it was changed or interfaced with other software. These changes may include software enhancements, patches, configuration changes, etc. During regression testing, new software bugs or regressions may be discovered.

Release: A planning segment of requirements (typically captured as features or user stories in the backlog) that deploys needed capabilities. The release is a time boxed event that consists of a set number of iterations that are determined by the program. The release plan is where different sets of usable functionality or products are scheduled to be delivered to the customer.

Requirement: A condition or capability needed by a customer to solve a problem or achieve an objective.

Requirements scrub: See backlog refinement.

Retrospective: A team meeting that occurs at the end of every iteration to review lessons learned and to discuss how the team can improve the process and team dynamics. The retrospective is an integral part of Agile planning and process and product improvement; typically a retrospective occurs at the end of every iteration or release. During each retrospective, the team explores ways to improve how they communicate, collaborate, problem solve, and resolve conflict in an effort to improve their own performance.

Road map: A high level plan that outlines a set of releases and the associated features. The road map is intended to be continuously revised as the plan evolves. It can also be used in Waterfall development programs, but typically a different term would be used. (See related terms in appendix III.)

Scrum: Scrum is a framework for developing and sustaining complex products. See appendix V for a brief description of Scrum and other Agile methods.

Should have: Those features that are not critical for a program and do not need to be delivered as part of the requirements. However, these features are higher priority than the could have or nice to have features and could significantly improve the capability of the program.

Solution: Products, systems, or services delivered to the business sponsor that provide value and achieve goals. A specific way of satisfying one or more needs in a context.

Sprint: See Iteration.

Stakeholder: Anyone who has an interest in the program. Specifically, parties that may be effected by a decision made by or about the program, or that could influence the implementation of the program's decisions. Stakeholder engagement is a key part of corporate social responsibility and for achieving the program's vision. A group or individual with a relationship to a program change, a program need, or the solution can be considered a stakeholder.

Story board: A wall chart (or digital equivalent) with markers (cards, sticky notes, etc.) used to track user stories' progress for each iteration. For example, the board may be divided into "to do", "in progress", "done", etc. and the movement of the markers across the board indicates a particular user story's progress. One goal of the story board may also be to recognize the order and the dependencies of the user stories in representing end-to-end functionality for the customer.

Story map: A visual technique to prioritize user stories by creating a "map" of customers, their activities, and the user stories needed to implement the required functionality.

Story point: A unit of measure for expressing the overall size of a user story, feature, or other piece of work in the backlog. The number of story points associated with a user story represents the complexity of the user story relative to other user stories in the backlog. There is no set formula for estimating the size of a user story, rather, a story point estimate is an amalgamation of the amount of effort involved in developing the feature, the complexity of developing it, and the risk inherent in it.

Sustainable pace: A management workload philosophy that is a part of the XP Agile method. (See appendix V for a brief description of the XP method.) It refers to a manageable, constant workload negotiated between the team and management so that the team will not be overextended. Sustainable pace is crucial when using velocity to estimate how much work a team is able to complete during an iteration.

Team facilitator: A person who has the explicit role of conducting a meeting and provides indirect or unobtrusive assistance, guidance, and supervision. Their primary focus is creating a process that helps the group achieve the intent of the meeting and takes little part in the discussions on the meeting's topics.

Technical debt: The obligation that a software organization incurs when it chooses a design or construction approach that is expedient in the short term but increases complexity and is more costly in the long term.

Test driven development: A software development process that relies on the repetition of a very short development cycle with unit testing. For example, first the developer writes an (initially failing) automated test case that defines a desired improvement or new function, then produces the minimum amount of code to pass that test, and finally refactors the new code to acceptable standards.

Theme: A group of user stories that share a common attribute, and for convenience they are grouped together and may span programs. A theme may be broken down into sub-themes, which are more likely to be product-specific. They can be used to drive strategic alignment and communicate a direction.

Time box: A time box is a previously agreed-upon period of time during which a person or a team works steadily towards completing a product. Rather than allow work to continue until the product is completed and evaluating the time taken, the time box approach consists of stopping work when the time limit is reached and evaluating what was accomplished. For example, in Scrum, the daily scrum is a 15 minute time boxed event. This means that the daily scrum should take up to, but no longer than, 15 minutes to complete. Time boxed iterations are typically associated with Scrum and XP.

Unit testing: Software testing in which individual units of source code, sets of one or more computer program modules together with associated control data, usage procedures, and operating procedures are tested to determine whether they are fit for use. This is the smallest testable increment in software development.

User story: A high-level requirement definition written in everyday or business language; it is a communication tool written by or for customers to guide developers though it can also be written by developers to express non-functional requirements such as security, performance, or quality. User stories are not vehicles to capture complex system requirements on their own. Rather, full system requirements consist of a body of user stories. User stories are used in all levels of Agile planning and execution. An individual user story captures the “who”, “what”, and “why” of a requirement in a simple, concise way, and can be limited in detail by what can be hand-written on a small paper notecard (also called "story").

Velocity: Velocity measures the amount of work a team can deliver each iteration. Commonly, this is measured as story points accomplished per iteration. For example, if a team completed 100 story points during an iteration, the velocity for the team would be 100. Velocity is a team-specific abstract metric and should not be compared across teams as a measure of relative productivity.

Verification and validation testing: Independent procedures that are used together for checking whether the program meets the requirements and specifications; that is, that it fulfills its intended purpose.

Vision: The highest level of Agile planning, the purpose for the program that is strategic in nature. The vision represents a shared understanding of the mission and objectives, capability gaps, expected behavior, and final outcomes to be addressed. The vision should be consistent over the life of the program unless business needs change significantly.

Appendix III: Related Terms

Agile terms can be specific to an individual program where they were used; even within the same organization. Prior to an audit, it is imperative that auditors understand the terms that each program uses. Table 18 highlights the terms that we have chosen to use in this guide and synonyms that we found in use in Agile developments. This list is not all inclusive, but intended to be a starting point to help bridge any misunderstandings caused by using different terms.

Table 18: Terms Used In This Guide and Related Terms

Term used in this guide	Related terms
Backlog	Inventory, feature list
Backlog refinement	Backlog grooming, backlog pruning
Daily stand up meeting	Daily Scrum
Epic	High-level requirement, theme
Feature	Capability, requirement
Iteration	Sprint
Kanban	Enterprise services planning
Minimal viable product	Minimally Sufficient Product, Minimal Marketable Feature
Must haves	Key Performance Parameters
Program	Project
Release	Product Increment
Retrospective	Lessons learned
Road map	Project vision, vision statement, Acquisition Program Baseline, Integrated Master Plan
Story board	Task board, Kanban board, progress board, story map
Team facilitator	Scrum master
Theme	Related user stories
User story	Story, product backlog item
Velocity	Capacity

Source: GAO. | GAO-20-590G

Appendix IV: Auditor's Key Questions and Effects

At the beginning of an audit, auditors should collect documentation and familiarize themselves with organizational, programmatic, and team specific Agile practices. Once they are familiar with the data collected, the following questions can be used as a starting place when reviewing Agile practices. They are not intended to represent a comprehensive set of questions that will be appropriate for every organization, program, or team. Prior to interviewing or discussing these questions within an organization, program, or team, we recommend that auditors discuss and come to a consensus on common terminology. For each best practice, this appendix also describes potential effects if organizations, programs, or teams are not fully implementing a best practice.

Chapter 3: Agile Adoption Best Practices

Best practice: Team composition supports Agile methods

Key considerations and questions

1. Agile teams are self-organizing
   - How does the organization typically form teams?
   - How did the team form (e.g., assigned by a manager, self-selected by employees)?
   - What is the team composition? Expertise mix?
   - Do team members have cross-functional skills allowing them to perform all of the work rather than a single specialty?
   - Is the team integrated with the program office, and able to enlist specialists such as designers, contract specialists, etc., as needed?
   - Are teams stable across iterations?
   - Is the team provided the latitude to collectively own the whole product and decide how work will be accomplished?
   - What allowances are made to ensure the team has adequate resources and time to complete the work effectively?

- Are all team roles defined and filled with the appropriate expertise?

2. The role of the product owner is defined to support Agile methods
   - Has a product owner been identified? Is there one person who serves as the product owner per team?
   - Is the product owner responsible for working with one team or multiple teams? If multiple, will this impact their availability to each team?
   - Is the product owner empowered with the ability to prioritize work in the backlog?
   - Is the product owner responsible for defining acceptance criteria and deciding whether those criteria have been met?
   - How does the product owner engage stakeholders and the developers to ensure work priorities align with stakeholder requirements?
   - Is the product owner available to the team when needed? Are there guidelines about product owner response rates?
   - Does the product owner continually interact with the team to discuss the success of the team throughout the process?
   - Is the product owner empowered to approve completed work?

Likely effects if criteria are not fully met

1. If the teams are not self-organizing or self-managing, the teams will likely be inefficient, causing program cost and schedule slips.
2. If a team does not have the requisite skill sets, it will likely be reliant on other teams that may have other responsibilities, thus delaying progress on the product.
3. Frequently shifting resources within a team, or between teams, can undo learning and shift team dynamics and skills, thereby diminishing the team's ability to meet commitments.
4. If there is not a clearly identified product owner who is the authoritative customer representative, who manages requirements prioritization, communicates operational concepts, and provides continual feedback, the developers may not be sure which features are priorities if they receive conflicting information, resulting in delays to delivering high priority features and deployment of the overall system.
5. If the product owner is not a dedicated resource, the developers may find that person unavailable to answer questions when needed. If

questions are not addressed in a timely manner, the developers may make assumptions in order to continue with development to meet commitments. If these assumptions do not match the expectations of the product owner, significant rework may be necessary. This can slow the development process.

6. A product owner must be empowered to make decisions about program development. Without such responsibility, the development process can slow down due to waiting on others with competing responsibilities to consider and respond on behalf of the business.
7. Without maintaining contact with both the developers and the customers, a product owner may not be able to represent what the customer priorities are and they may misrepresent them to the developers. This could result in a decreased value from the system if the wrong features are given priority in the backlog or cause schedule delays if critical features were not developed.

Best practice: Work is prioritized to maximize value for the customer

Key considerations and questions

1. Agile teams use user stories to define work
   - Is there a standard structure used to write user stories? (e.g., elements that should be included in a standard user story?)
   - Who writes the user stories and how are they managed? Can anybody write a user story?
   - How does the product owner ensure that user stories are independent? Negotiable? Valuable? Estimable? Small? Testable?
   - How do the user stories reflect acceptance criteria and do they define what "done" is?
   - How and when are user stories reevaluated based on organizational needs and return on investment?
2. Agile teams estimate the relative complexity of user stories
   - How does the team estimate user story complexity? (For example, what techniques and metrics are used for estimating?)

- Does the team consider potential factors that can increase the complexity of the work when sizing the work?
- What techniques does the team use, such as affinity estimation, to help identify the factors that could affect the complexity of a user story?
- Who is involved in estimating and at what level does estimating take place?
- Does the size estimation use prior estimates to inform future estimates?
- Is the size estimate refined over time?
- Is acceptance criteria well-defined and consistent for user stories?
- Does the team 'lock' sizing estimates once an iteration begins so the team can examine variances between estimated and actual work accomplished?
- Is there a method in place to evaluate the success of these estimates?
- Have the teams been meeting their commitments for each iteration/release?

3. Requirements are prioritized in a backlog based on value
    - Is the product owner considering value when prioritizing the backlog?
    - Is there a shared understanding of value among the team, program, and organization?
    - Is the team working from a prioritized backlog to provide frequent software deliveries?
    - What approaches are used to prioritize the backlog: the must-have, should-have, could-have, would like to have, MoSCoW, etc.?
    - Is the value of the work accomplished tracked and monitored?
    - Does the program track feature usage statistics or customer satisfaction? Is the team assessing value expected versus value delivered?
    - Does the product owner reevaluate requirements frequently to reprioritize as necessary?

Likely effects if criteria are not fully met

1. Establishing a common structure for the user story helps ensure consistency across teams and can help prevent delays when product owners work with multiple teams or teams are reorganized.
2. If teams are not using relative estimation to compare current size and work estimates to historical completed work, the team may underestimate, or overestimate the complexity and time necessary to complete the user story.
3. Well-defined acceptance criteria can help teams estimate a user story's complexity. Less well-defined user stories will carry more risk and uncertainty around size estimates.
4. If teams are not estimating user stories consistently, the teams may be committing to too much work, leading to user stories lasting more than one iteration and team burnout.
5. A lack of traceability between different levels of backlogs and program planning artifacts could lead to overlooking user stories or features that are critical to the program due to their high value to the customer or key dependencies that those user stories or features might have with other aspects of the system.
6. A lack of understanding or insight into the methods used to measure value for user stories could cause a disconnect between the customer and developers and allow delivery of features that do not maximize value.
7. Without clearly prioritizing work, the developers could work on features that are not "must haves" to the customer, resulting in the delivery of features that may not be used and might contribute to schedule and cost overruns.

Best practice: Repeatable processes are in place

Key considerations and questions

1. Agile program employs continuous integration
   - How frequently is the software integrated?
   - How does the team ensure that software handoffs between the various stages of development and testing are performed in a reliable, dependable manner?

- Are functional and non-functional requirements tested at each stage of the continuous integration process?
- Is the scope of the automated testing tracked and monitored based on established expectations?

2. Mechanisms are in place to ensure the quality of code being developed
   - How does the team incorporate manual coding in concert with automated processes to manage the code base?
   - What mechanisms are in place to alleviate factors that contribute to negative impacts on code quality, such as time constraints and unsustainable pace of development, or undisciplined coders?
   - What processes are in place to manage "technical debt"?
   - What assurance methods are incorporated in code development to ensure the integrity of manual coding, pair programming, refactoring, peer review?
3. Agile teams meet daily to review progress and discuss impediments
   - Is the team holding a standup meeting every day and if so, who leads it?
   - Who attends the standup meetings?
   - Are all members of the team present and actively involved in the standup meetings?
   - What are the objectives of the daily standup and how do they help the team plan and execute work?
4. Agile teams perform end-iteration demonstrations
   - Is the team holding a review/demo at the end of every iteration?
   - Who attends the end-iteration demos?
   - Do all stakeholders attend the demonstration? For example, does the product owner(s) attend the demos?
   - Is the software depicted in a realistic setting?
   - How are the demos accepted?
   - Is the team demonstrating every completed user story at the demo?
5. Agile teams perform end-iteration retrospectives
   - Is the team holding retrospectives at the end of each iteration?

- Who attends the retrospective? Does the product owner attend the retrospective with the team? Are all members of the team present and actively participating in the meeting?
- How are action items from the retrospective implemented?
- How are implemented tasks from the retrospective managed?
- What is the average time to fully implement tasks identified in the retrospective?

Likely effects if criteria are not fully met

1. Without continuous integration using automation, reliable, dependable software handoffs may not occur.
2. Without automated build and testing tools, the program may experience challenges in delivering the product on time and may have a limited assurance of product quality.
3. The accumulation of deficiencies over time, technical debt, can present obstacles to an Agile program if not properly managed. For example, as a code base grows, additional functions will rely on the deficient code, causing a degradation in overall system performance. Moreover, as the interest incurred on technical debt continues to rise, teams will devote more time to cleaning up errors instead of producing new features.
4. Without the daily standup meetings, team members may not be held accountable for their work, duplication of work could occur, or work may not get accomplished because of a lack of communication and understanding of who is doing what for the program.
5. Without daily standup meetings, the team might also not identify impediments which may result in rework or schedule delays.
6. If used as a status update by management instead of focusing on progress and impediments, the meetings could last too long.
7. If end-iteration demonstrations are not performed, the team may not be able to identify portions of the software that need improvement or modifications to provide the anticipated functionality.
8. If retrospective meetings are not held at the end of each iteration, the team may not reflect on or improve the efficiency and effectiveness of its work processes, thereby impacting the timely delivery of a high-quality product.

Best practice: Staff are appropriately trained in Agile methods

Key considerations and questions

1. Program staff are trained in Agile methods
   - Has the program developed a strategic approach that establishes priorities and leverages investments in training and development to achieve results?
     - Does the program have training goals and related performance measures that are consistent with its overall goals and culture?
     - How does the program determine the skills and competencies its workforce needs to achieve current, emerging, and future goals and identify gaps that training and development strategies can help address?
     - How does the program identify the appropriate level of investment to provide for training and development efforts and prioritize funding so that the most important training needs are addressed first?
     - What measures does the program use in assessing the contributions that training and development efforts make toward individual mastery of learning and achieving program goals?
     - How does the organization incorporate employees' individual developmental goals in its planning processes?
     - How does the program integrate the need for continuous and lifelong learning into its planning processes?
   - Are all members of the Agile team and all stakeholders in the program receiving appropriate training?
   - Does the training in specific Agile methods include Agile policy and procedures?
   - How does the organization track and monitor training requirements for all team members?

- Under what circumstances is refresher training conducted, such as on the use of new programming languages, applications, compliance requirements, coding, or security standards?

2. Developers and other supporting team members have the appropriate technical expertise
   - How does the program ensure immediate access to specialized expertise, including contracting, architecture, database administration, development, quality assurance, operations (if applicable), information security, risk analysis, and business systems analysis, that may be required to aid existing teams?
   - How did the program identify the technical expertise needed to successfully meet program goals?
   - How did the program assess the existing expertise of Agile team members?
   - How were gaps addressed, if any?
   - Does the program define requirements for contractor personnel to be provided in contractor proposals?
   - How is the program evaluating the qualifications of the contractor to perform the work when evaluating proposals?

Likely effects if criteria are not fully met

1. Without training, there might be a lack of common understanding in the program about the Agile methods to be used.
2. Without effective training based on a strategic human capital analysis, the program will be challenged in helping to ensure that the required capabilities and mission value will be delivered in a timely and cost-effective manner.
3. An Agile team needs to have all the appropriate technical expertise, or it could be delayed in completing its work while waiting on input from an expert outside of the team.
4. If individual team members are not proficient in the skills necessary to complete the work, then the quality of the product being developed may suffer, requiring substantial re-work.

Best practice: Technical environment enables Agile development

Key considerations and questions

1. System design supports iterative delivery
   - How has the program established an architecture that allows for incremental delivery and loose coupling?
   - How does the design architecture support delivery of iterations that can be seamlessly inserted into the operational environment?
   - How does the program manage staff assignments distributed across multiple locations to facilitate iterative delivery and loosely coupled architecture?
   - How does the program manage frequent testing and reviews to ensure that newly-developed components are properly integrated with existing components?
2. Technical and program tools support Agile
   - What tools are being used to support Agile software development?
   - Are tools used organization-wide, program-specific, team-specific, or a combination?
   - Do both government and contractor personnel, involved in the Agile development effort, have access to the same data?
   - How is the program working to ensure that both government and contractor personnel have access to the same data?
   - How is the program setting up internal controls to restrict access rights for Agile-support tools to ensure the proper access across government and contractor personnel?
   - How is program management working to align their program management tools with Agile principles and practices?
   - How frequently is software integrated and tested?
   - How are automated tools used to support integration and testing of software?
   - Are the tools integrated into the program's technology environment (e.g., automated regression testing suites and

continuous integration support tools) and is access available to all team members and stakeholders?

Likely effects if criteria are not fully met

1. Not allowing time up front to consider system requirements can increase future complexity, re-work, and unnecessary investment.
2. If the program does not consider the system architecture during its initial planning and instead relies on building out the architecture as code is developed, the architecture may not support the needs of the system when fully operational and may require a complete technical refresh.
3. If software design and architecture are not loosely coupled, changes to individual pieces of the system may require a significant amount of testing of the entire system, slowing the pace of development and delivery of the product.
4. If technical and program tools are not consistently available to those members of the team requiring access, then the productivity of developers may suffer and result in increased costs for development.
5. Without automated tools, the program risks inconsistent implementation of processes across teams, which may negatively affect product delivery and understanding of the program's progress.
6. Large programs not using automated tracking tools could miss key dependencies between user stories and features.

Best practice: Program controls are compatible with Agile

Key considerations and questions

1. Critical features are defined and incorporated in development
    - Has the program identified mission, architectural, and safety-critical components and dependencies?
    - How often does the program revisit these components to validate their importance?
    - At what point in a program's life cycle are these components defined? During an initial iteration before any software development begins?
    - How does the program strategy account for mission and safety criticality along with dependencies? Is the strategy adequate or is the program increasing its risk?

- In determining the criticality of software, how does the program evaluate and prioritize the relative value of work to ensure that each iteration delivers the most business value?

2. Non-functional requirements are defined and incorporated in development
    - How are non-functional requirements for a program identified? Where are these requirements defined?
    - How does the program consider and implement security requirements throughout the development?
3. Agile teams maintain a sustainable development pace
    - Does management work with teams to prioritize user stories, establish an agreed upon definition of done, and develop a mutual commitment on the work to be accomplished for each iteration?
    - How does management encourage teams to maintain a consistent development pace that can be sustained indefinitely?
    - Does the program track velocity or other metrics to evaluate pace?
    - How does velocity or sustainable pace factor into iteration and release planning? Into iteration/release review or retrospective?
    - Does the program monitor the teams to ensure a consistent pace is being achieved on a team by team basis? If so, how and how often?

Likely effects if criteria are not fully met

1. Without clearly identifying mission and system critical architecture features, the program risks developing these features after other software is in place and facing substantial rework and integration challenges and unnecessarily increasing the cost and time to deliver all critical features.
2. If critical business requirements are not prioritized appropriately, software may not provide the required functionality.
3. Lack of communication between the product owners and developers regarding features' priorities risks the development of noncritical software in place of critical software and lower customer satisfaction with the completed product.
4. Teams overlooking nonfunctional requirements may develop a system that does not comply with current federal standards (e.g., cybersecurity standards for IT programs), causing unnecessary risks

to business operations and resulting in the software not becoming operational until these components have been addressed.

5. If the teams are not working at a sustainable pace, there is a risk of burnout, which can cause delays in the program.
6. Without establishing a consistent pace, the program cannot reliably use historical metrics, such as team velocity, to estimate future efforts required in product development.

Best practice: Organization activities support Agile methods

Key considerations and questions

1. Organization has established appropriate life-cycle activities
    - Is there a documented process for acquisition?
    - Is there a documented process for software development?
    - Are programs allowed to deviate from the documented processes if pursuing Agile software development? If so, under what conditions?
    - Do organization acquisition policy and guidance allow for changing requirements?
    - Do organization acquisition policy and guidance allow for frequently delivered software in small deployments?
    - Do organization activities support technical reviews occurring throughout development that are tailored to the cadence of Agile software development?
    - Do the program's structure and support mechanisms foster a strong relationship between customers and the developers?
    - How is success being measured for Agile programs including any benefits such as shortened timeframes and higher quality software being delivered?
    - How is the organization encouraging more frequent collaboration between the customer and developers, and more frequent delivery of incremental software?
    - Has the organization developed policies and procedures allowing requirements to change throughout the program's life cycle?

- Early in a program's life cycle, are requirements defined at a high enough level that the program can modify the requirements as needed to reflect a better understanding of needs?
- Has the organization specified policy and procedures regarding the speed with which changes can be approved?

2. Goals and objectives are clearly aligned
    - Has the organization and/or component developed a strategic plan for IT that aligns with the overall objectives of the organization and/or component strategic plan?
        - Is IT consulted by management to identify technology that is creating opportunities that the business can turn into enterprise benefits?
        - Are members of IT management actively helping to realize the enterprise goals?
        - Is there accountability for achieving enterprise goals to determine executive commitment to the goals?
    - Have the goals for the program been defined?
    - Were program goals approved and agreed to by all relevant stakeholders in accordance with agency and/or component acquisition policy?
    - Do program goals logically trace back to the IT strategic plan and business strategic plan in turn?
    - Do the technical goals of the program (e.g., software and hardware) align with the organization's software-related goals?
    - Is the organization collecting objective measures and clearly communicating feature and capability achievements to the entire organization?
    - How does the organization ensure that goals are clear but not static, and that the Agile implementation allows for rapid response to changes in either the external or internal environment?
    - How does the organization allow for goals that are not clear? How does the organization effectively and routinely communicate program goals?

Likely effects if criteria are not fully met

1. If programs are unable to tailor life cycle activities, then the organization's oversight process could negatively affect the cadence established by the Agile team, resulting in less predictable development efforts.

2. If collaboration is not occurring regularly, then priorities regarding requirements will not be known and the result may not meet the program's vision or customer's needs.
3. Where changing requirements are not understood or defined at an organizational level, the adoption and full realization of the benefits from Agile methods will be hard to achieve.
4. If the organization's goals are not clear or do not adequately reflect stakeholder concerns and mission needs, then lower-level decision making may be misaligned with the organization's focus. This misalignment can, in turn, erode trust and often results in overbearing governance and bureaucracy, leading to delays.
5. If the organization's software-specific needs are not considered to be part of the larger program goals, then the implementation of software applications may not fulfill minimum requirements established by the organization or by the federal government.
6. If approved program goals do not align with both the IT and business goals, then lower level decision making runs the risk of being misaligned with the organization's focus.

Best practice: Organization culture supports Agile methods

Key considerations and questions

1. Sponsorship for Agile development cascades throughout the organization
   - Who is/are the sponsor(s) for Agile software development?
     - Do sponsors have sufficient authority to manage execution of the transition within the overall goals established for the transition group?
     - Are the responsibility and accountability defined for each sponsor and level of management in transitioning to Agile?
   - Do all sponsors within the organization and IT agree on and accept the goals and definition of success for the transition to Agile?

- Do sponsors adhere to Agile software development commitments documented in organizational policy?
- How were sponsors selected? Why do sponsors believe in and support a transition to Agile software development (e.g., flexibility demonstrated by a team adhering to a Scrum framework)?
- Does sponsorship cascade to the overall life-cycle management process including those involved in certification and accreditation, or operational test and evaluation?
- Is there guidance in place at the organization, encouraging employees and groups to adopt Agile methods?
- What indicators have been considered regarding a program readiness to adopt Agile? For example, are requirements flexible, is there an established process in place to further define the requirements over time, etc.?
- Are laws, policies, and guidance available to facilitate the adoption of Agile?

2. Sponsors understand Agile development
    - How familiar are sponsors with the Agile process in place within the organization?
        - Is each sponsor aware of the roles and responsibilities of other sponsors?
    - How familiar are sponsors with the values and principles of Agile?
    - Can sponsors speak to how the values and principles of Agile are reflected in the adapted organizational processes?
    - Do sponsors accept accountability for results?
    - Are sponsors committed to applying the organization's Agile framework consistently across the organization?
    - Are sponsors aware and in touch with Agile methods and practices applied at the program and team levels of the organization?
    - Do organizational policies require sponsors and senior stakeholders to be fully educated regarding Agile values and principles?
3. Organization culture supports Agile
    - How are teams physically structured (co-located or split across geographic areas)?

  - Are all members of a team co-located (business representative/product owner, developers, testers, etc.) or are only some co-located?
  - If not co-located, how are team members communicating? How often?
  - If not co-located, why not?
- Does organizational policy support co-location and promote the need for face-to-face conversation?
- Are all team members, including the product owner, immediately accessible to answer questions, as required?
- How does the organization promote trust between the enterprise and the customer organization? Examples include conducting a joint workshop that focuses on the effort, but provides opportunities for working together across organizational boundaries.
- How is the organization promoting awareness of long-term goals of the system to ensure that Agile teams can operate effectively with greater autonomy?
- Does the organization have a process and terminology in place to facilitate communication practices and encourage transparency, availability of team message boards, collaborative workspaces, etc.?
- Does the organization encourage communities of practice to promote strong interactions in a healthy climate of trust?
- How does the organization implement inspection and adaption to continue to learn and adapt from feedback? Inspection and adaption might take the form of a more formal meeting, such as a retrospective, or may only require an informal set of discussions among sponsors.
- What data are collected during the transition to Agile to facilitate and support senior stakeholder adaptation and decision-making?
- What modifications to policies and processes have been adopted to reflect Agile practices and policies? For example, how are modifications made to policies and processes, such as systems engineering life cycle documentation, to address Agile development methods?

4. Incentives and rewards aligned to Agile methods

- How does the organization evaluate employees for traditional programs? Is the evaluation process for an Agile program different?
- Are appropriate organizational entities, such as human resources or employee unions, involved to establish an organizational goal to align incentives and rewards with their Agile values and principles?
- Are rewards tied to results (e.g., working software) and not the outputs (e.g., ancillary documents) of an Agile process?
- Has the organization developed specific criteria or refined the process for evaluation of employees associated with an Agile program?
    - What metrics does the organization collect and measure when evaluating individual or team performance for an Agile program?
- Who participates in performance reviews and how actively are they involved in the day-to-day operations of an Agile program?
- Do organizational incentives and rewards promote and recognize teams or individuals?
- What are some examples of incentives and rewards available to teams?

Likely effects if criteria are not fully met

1. Without high-level encouragement, Agile implementation might become a paperwork exercise, leading to a failure to complete software development.
2. Without encouragement and commitment from upper-level management, Agile teams may not appropriately collaborate with business owners when they are unsure about the importance of certain functionality, causing confusion that ultimately can result in a poor product. Thus, functionality developed using a process that does not embrace an Agile mindset might require heavy investment in the post deployment correction of errors or functionality enhancements to meet customer needs.
3. Without sponsorship from senior stakeholders and the presence of an Agile champion, or multiple champions, the organization may not embrace the transition to Agile, which can lead to inconsistent Agile practices and lackluster results.
4. While having a clearly defined policy for Agile programs can be effective in many cases, using a policy or mandate to force adherence

to Agile principles does not produce the healthy adoption of new practices. For example, putting policies in place too early, before the appropriate transition mechanisms are solidified, may lead to basic compliance but without consideration for the organization's culture and mindset that should occur during a successful Agile transition.

5. If sponsors are unable to effectively differentiate between a Waterfall and an Agile implementation, they may hamper or impede the effective adoption of Agile principles, leading to a breakdown in processes.
6. If all team members, including the product owner, are not immediately accessible to answer questions, team work may be delayed.
7. If appropriate organizational entities, such as human resources, are not considered, changes to incentive and reward systems might be slow and ineffective, preventing team cohesion and unity, and restricting productivity.
8. Since the federal acquisition environment is built on strong oversight, traditional acquisition can often result in adversarial relationships between the acquirers and the developers. In an Agile environment, a climate of trust, built by shared experiences in which all parties feel respected and accepted, is needed so that the program team can achieve its fullest potential.
9. If an Agile supportive environment is not in place, then team and program operations might not have the resources necessary to be successful, thus impeding delivery of the product and not meeting agreed-upon goals for cost, schedule, and performance.
10. Changes to incentive and reward systems might be slow and ineffective, preventing team cohesion and unity, and restricting productivity unless there is active involvement from the appropriate organization entities, such as human resources and employee unions.
11. If organizational incentives are not structured to promote improved team performance, competiveness or a lack of respect among team members might increase, impacting team behavior, productivity, and output.

Best practice: Organization acquisition policies and procedures support Agile methods

Key considerations and questions

1. Guidance is appropriate for Agile acquisition strategies
    - Does the organizational acquisition policy and guidance require the contract structure and acquisition strategy to be aligned to support Agile methods of software development?
    - What policy and guidance does the program use to analyze the risks, benefits, and costs before entering into any contract?
    - Are contracts structured to allow for the implementation of Agile principles, frequent interim deliverables, product demonstrations, changing requirements, etc.?
    - Do the contract structure and acquisition strategy allow for interim demonstration and delivery between official releases?
    - Does the contract specify delivery cadence and how product demonstrations will be used to solicit customer feedback?
    - Does the contract structure allow the government team, in coordination with the product owner, enough flexibility to adjust feature priority and delivery schedule as the program evolves?
    - What mechanisms are in place in the contract and acquisition strategy to allow for close collaboration between the developers and stakeholders in order for everyone to agree on what features have the highest priority?
    - Does the contract language reflect Agile principles such as enabling incremental and frequent progress reviews at key points?
    - Do contract oversight mechanisms align with Agile practices?
    - From a contract oversight perspective, are the expectations of reviewers and oversight personnel set appropriately to ensure Agile principles can be effectively employed?

Likely effects if criteria are not fully met

1. If an acquisition strategy and contract structure do not allow for interim delivery and product demonstrations, then the organization may lose opportunities to obtain information and face challenges in adjusting requirements to meet changing customer needs. This may negatively impact continuous delivery of software.
2. If the organization does not adjust its oversight process to account for Agile methods, then the contractors' productivity may decrease.

Chapter 5: Agile Requirements Management Best Practices

Best practice: Elicit and prioritize requirements

Key considerations and questions

1. How does the process to elicit customer needs, expectations, and constraints incorporate customer feedback? Does the process use incorporate surveys, forums, and other mediums to brainstorm the needs of the organization?
2. Does the process to elicit requirements reflect an iterative process?
3. Are requirements defined at various levels? If so, is there a different approach to eliciting customer needs, expectations, and constraints and a different process for prioritization decisions for each level?
4. Does the review cycle allow a customer to observe the system and communicate additional functionality or modifications to existing functionality?
5. Does the program have a process in place to field customer suggestions, via testing, demonstrations, or other means?
6. Does the product owner proactively solicit and prioritize input from customers to inform future requirements?
7. How does the program identify non-functional requirements? Is the process to identify non-functional requirements iterative and on-going?

8. How does the program capture non-functional requirements? For example, one option is to define each discrete requirement as a separate user story that traces to a non-functional feature such as architecture.
9. How does the team test non-functional requirements?

Likely effects if criteria are not fully met

1. If there is not a strong commitment to ongoing elicitation and refinement of requirements, the delivered software may not meet the changing needs of the customer or address the evolving technical landscape.
2. If the product owner does not capture feedback from reviews for consideration, there is no historical record of proposed requirements or modifications for reference. The lack of a documented change control process could hinder the decision maker's insight into the true value of delivered features.
3. Agile tends to emphasize customer-facing requirements. However, when the focus on customer functionality becomes exclusive, the underlying system (or non-functional) requirements can go unnoticed.

Best practice: Refine and discover requirements

Key considerations and questions

1. How does the program discover and refine requirements?
2. Does the program use visualization tools to discover and refine requirements?
3. What process does the program use to incorporate lessons learned into requirements and their prioritization?

Likely effects if criteria are not fully met

1. If Agile programs do not learn to discover and refine requirements throughout the development process, a program may miss an opportunity to incorporate newly discovered requirements or eliminate requirements previously thought to be essential, which could create a disconnect between deployed software and the customer's needs.
2. Without ensuring full prioritization of current and future features and user stories, a program could be at risk of delivering functionality that is not aligned with the greatest needs of the customers.

Best practice: Ensure requirements are complete, feasible, and verifiable

Key considerations and questions

1. How do the team(s) and the product owner develop a shared understanding of the definition of done?
2. How does the team establish acceptance criteria?
3. How does the team determine when a requirement is adequately defined or ready for work to begin?

Likely effects if criteria are not fully met

1. Without having clear criteria and an established definition of done allows uncertainty into the development process.
2. Without clear definitions for ready, acceptance, and done, the team may be working inefficiently and on requirements that are not high priority.

Best practice: Balance customer needs and constraints

Key considerations and questions

1. What process does the product owner use to calculate the value of work and ensure user stories are being developed based on relative value? For instance, does the product owner value high-risk work early in a release to mitigate risk, or determine value based on resource availability, etc.?
2. How does the product owner balance customer needs and constraints when determining the value of work?
3. What additional information is collected in the backlog documentation to articulate relative value, details about the work, estimates for time, and priority ranking?
4. How do the product owner and team work together to refine the backlog priority?
5. How are customer suggestions considered in the backlog review and refinement?

Likely effects if criteria are not fully met

1. If the product owner does not consider the relative value of the work, all of the user stories can end up being developed just prior to deployment. Often this is a sign that the product owner is not prioritizing the requirements and is developing functionality that is not immediately necessary.
2. The practice of developing each and every user story can lead to problems if funding is reduced mid-iteration, mid-release, or mid-program, or other external factors impede the progress of the development work.
3. If the product owner does not consider the relative value of work, the team may develop functionality that is not immediately necessary to meet customer needs.
4. If the highest value requirements are not completed first, the customer may be left without necessary functionality.

Best practice: Test and validate the system as it is being developed

Key considerations and questions

1. How are continuous integration and automated testing incorporated in the Agile environment?
2. What process is used to validate the user story: a user story demonstration or a review at the end of each iteration?
3. How do customers participate in the review process to observe functionality and whether it meets the intended purpose or requires further refinement?

Likely effects if criteria are not fully met

1. If customers are not involved in the review and acceptance process for software functionality, the software may not meet the intended purpose required by the customer.

Best practice: Manage and refine requirements

Key considerations and questions	1. How does the Agile program manage refining requirements? 2. What process does the product owner use to manage requirements and maximize the value of software delivered?
Likely effects if criteria are not fully met	1. If the requirements refinement process is too inflexible, it becomes a change prevention process and user needs will not be adequately incorporated into the program, making it less useful to customers than intended. 2. If the refinement process is too flexible, then boundless development can occur and the organization may not receive the full value that it requires.

Best practice: Maintain traceability in requirements decomposition

Key considerations and questions	1. How does the program maintain traceability from source requirements to lower level requirements and then from those lower level requirements back to the source requirements? 2. Is a traceability matrix or road map used to trace requirements? 3. If automated tools are used, are discrete fields included to trace high level requirements to user stories?
Likely effects if criteria are not fully met	1. Without tracing a user story back to high level requirements, a program cannot justify whether it is meeting the commitments made to various oversight bodies, and in turn, cannot establish that the work is contributing to the goals of the program and thereby providing value.

Best practice: Ensure work is contributing to the completion of requirements

Key considerations and questions

1. How does the team assure they are working on tasks that directly contribute to the completion of user stories committed for the current iteration?
2. Is the product owner ensuring that user stories contribute to the commitments made to oversight bodies?
3. What mechanism, such as a management plan or program road map, etc., is used to lay out capabilities or features for development in a timeline?

Likely effects if criteria are not fully met

1. If work performed is not associated with the user story commitments for an iteration, there may be a misalignment between the requirements and work, and it can present a risk for the program.
2. If the schedule of programs and phases and the scope of each program are defined and committed to in advance, there should be alignment between the user stories being developed and the scope of a specific program.

Chapter 6: Agile and Contracting Best Practices

Best practice: Tailor contract structure and inputs to align with Agile practices

Key considerations and questions

1. Modular contracting
    - Does the contract structure support small, frequent releases?

- Does the acquisition strategy avoid any potential lags between when the government defines its requirements and when the contractor delivers a workable solution?
- Does each program acquisition reflect individual increments with a life cycle and scope such that they can be delivered independently?

2. Enable flexibility in the contract's requirements
   - Does the contract structure provide sufficient structure to achieve desired mission outcomes?
   - Does the contract structure offer flexibility for refinement of software requirements within the agreed-on scope of the system?
   - Do the contracting strategies support the short development and delivery timelines that Agile requires?
   - Does the statement of objectives/statement of work (SOO/SOW) include a purpose, scope, period of performance, location for conducting the work, background, performance standards (the required results), and any identified operating constraints?
   - Does the SOO/SOW include the product vision, strategic themes, an initial road map, and an initial backlog of features and capabilities?
   - Does the SOO/SOW establish performance standards for the expected accomplishment level required by the government to meet contract requirements?
   - Are the performance standards measurable and structured to enable performance assessments?
3. Contract structure and type
   - Has the program office clearly delineated to the contracting officer whether the contract intends to procure goods or services?
   - Do the solicitation and resulting contract clearly delineate the responsibilities of the contractor to ensure that federal employees oversee and make the final decisions regarding the disposition of the requirements?

Likely effects if criteria are not fully met

1. If each program is not separable, then the government may need to acquire future programs, which could be costly and burdensome.
2. If performance standards are not measurable and structured to enable performance assessments, the government may not be able to assess the expected accomplishments.

3. To follow the FAR and ensure that the contractor doesn't perform inherently governmental functions, the organization should carefully delineate responsibilities in the contract to ensure that the government clearly has decision making authority regarding the final product.
4. If the contract does not provide sufficient structure to achieve the desired mission outcomes, while offering flexibility for adaption of software requirements within the agreed-on scope of the system, it may not support an Agile development approach. A lack of structure and flexibility increases the likelihood of disruption and delays.

Best practice: Incorporate Agile metrics, tools, and lessons learned from retrospectives during the contract oversight process

Key considerations and questions

1. Contract data requirements rely on Agile metrics
   - Do the contract data requirements list align with Agile metrics to reflect the different processes and artifacts used in Agile?
   - Do the quantity and type of contract data requirements established in the contract account for the program environment?
2. Data from Agile artifacts enables contract oversight
   - Does the program collect data from the program's releases, features, and capabilities to enable contract oversight to hold contractors accountable for producing quality deliverables?
   - Do the work elements collected allow the program to measure whether a user story is "done"?
   - Does the program collect metrics throughout the Agile development life cycle to monitor the contracted development effort?
3. Conduct retrospectives to continually improve based on lessons learned

- Does the program require retrospective reviews where stakeholders to interact with the developers?

4. Contract oversight reviews align with the program's Agile cadence
   - Do contract oversight reviews align with the program's Agile methods and cadence?
   - Does the contract allow for contractual gate reviews to be tailored in order to successfully align the contract requirements with the functional requirements?
   - Are reviews tied to the program's Agile cadence for completing releases?

Likely effects if criteria are not fully met

1. If the contract data requirements list does not account for the Agile development program environment, the program may miss the opportunity to collect data about the quality of its software products.
2. If the program does not collect Agile metrics for technical management, program management, and Agile methods, the government may not have the right information for effective contract oversight and will not be able to hold contractors accountable for producing high quality deliverables.
3. If reviews for the program are not tailored to align with the program's Agile cadence, the review structure could impede progress and cause delays.

Best practice: Integrate the program office and the developers

Key considerations and questions

1. Train program office, acquisition, and contracting personnel
   - Do the acquisition team and developers have a common understanding of Agile techniques so that an acquisition strategy can be properly structured to establish a development cadence?
   - Is there a close partnership between the developers, program managers, customers, and contractors?
   - Does the program have a dedicated onsite contracting team trained in Agile implementation?

- Have contracting personnel been trained to enable an Agile mindset?
- Have clear roles been established for contract oversight and management?
- Has management adopted a role of mentor, fostering an environment of trust and open communication?

2. Identify clear roles
    - Are the product owner and contracting officer’s representative (COR) working closely to align the program’s business and technical requirements?
    - Are the COR and the product owner both government employees?
    - Has a designated product owner been identified and are they empowered to make decisions quickly and to prioritize requirements within the scope of the road map?
    - Are all personnel familiar with the distinction between contract and functional requirements that are part of the Agile development process?
    - Does the contract have a dedicated contracting officer who works closely with the product owner to align roles and responsibilities?
    - Are the contracting officer, the product owner, and any government developers working closely to develop an effective acquisition strategy?
3. Awareness of the contract’s scope
    - If additional requirements are identified after contract award, is there enough time on the contract to complete the additional work or can these requirements be substituted for currently-identified features?
    - Is the contract structure such that it can be modified should new work be identified as higher priority to accomplish goals outside the scope of the current contract?
    - Is the product owner empowered to prioritize among system requirements within the scope of the product vision, and is this documented in the contract?
    - Do persons in all roles understand the Agile process for the program?

Likely effects if criteria are not fully met

1. Without properly trained program office personnel, including contracting personnel, staff may not have the skills to assist the program in making business decisions and trade-offs that come with the implementation of an Agile effort.
2. Without a dedicated onsite contracting team, who are trained in Agile implementation and are able to assess the impact Agile cadences have on the program's acquisition strategy, the program may suffer delays due to a lack of close partnership between the program and the developers.
3. If management does not foster an environment of trust, the product owner may not feel empowered to make decisions.
4. Roles must be clearly defined and carried out in order to prevent bottlenecks and ensure that rapid feedback channels are clearly established from the start of development.
5. Typically, both the COR and product owner should be government employees so that they can be empowered to make day-to-day decisions for the development effort. If the product owner is not a government employee, the product owner may not be empowered to make day-to-day decisions for the development effort, causing development delays.
6. If the contracting officer and the program office do not understand the distinction between contract and functional requirements, then all compliance and security requirements may not be included.
7. Lack of authority and involvement by the product owner can result in bottlenecks in the contracting process.

Chapter 7: Agile and Program Controls

Detailed best practice checklists are found in the companion guides; the GAO Cost Guide (GAO-20-195G) and the GAO Schedule Guide (GAO-16-89G).

Chapter 8: Agile Metrics Best Practices

Best practice: Identify key metrics based on the program's Agile framework

Key considerations and questions

1. How does the organization consider metrics and determine which metrics are appropriate for the chosen software approach?
2. Do the metrics address technical management, program management, and Agile methods?
3. How does the organization identify and delineate metrics for each level, organization, program, and team?
4. How does the organization ensure that metrics are quantifiable, meaningful, repeatable and consistent, and actionable?
5. Are Agile developers and managers conveying meaningful information to address customer concerns?
6. How does the program delineate between metrics needed for the team to measure performance, and metrics needed for the customer?
7. With what frequency does the program collect metrics?
8. How does the program measure the value of a specific metric?

Likely effects if criteria are not fully met

1. Without meaningful, clear, and actionable metrics, management may not have the information needed to evaluate program performance.
2. If a program is not aligning metrics with customer questions, it may not have the data needed to evaluate program performance.
3. Not establishing metrics to obtain user feedback limits a program's understanding of the value delivered with each software release.

Best practice: Ensure metrics align with and prioritize organization-wide goals and objectives

Key considerations and questions

1. Are metrics tied to program goals? Is the program able to measure the success of the program goals from the collected metrics?
2. Are metrics identified and tracked that are used to impact decision making?
3. Do the metrics allow traceability from the road map through releases and items in the product backlog?
4. Has the organization defined the goals, objectives, and performance information appropriate to managerial responsibilities and controls at each level of the organization?
5. Have Agile metrics been tailored to allow the organization to convey progress and achievements to internal and external customers?

Likely effects if criteria are not fully met

1. If the metrics do not allow traceability from the road map through the releases and backlog, the organization may not have the right information to make decisions about prioritization and potential re-planning.
2. If the organization does not adopt an organized structure to collect performance information at each level of the organization, the metrics may not align with management goals.
3. If Agile metrics are tailored to reflect developers' progress and achievements to internal and external customers, it can facilitate feedback and communication between both entities.

Best practice: Establish and validate metrics early and align with incentives

Key considerations and questions

1. Are metrics established at the start of the program?
2. Are metrics aligned with incentives?
3. Are metrics monitored at the organization, program, and team levels?
4. Are reward and incentive structures based on team accomplishments?
5. How is the Agile team determining the value of each metric in relationship to the cost of collecting the supporting data?
6. Are metrics collected to measure the flow of work over time, such as features delivered in each iteration?
7. Is the team collecting metrics associated with product quality, such as the number of defects identified after a product deploys?
8. Is the team capturing metrics that measure adherence to Agile software development best practices?

Likely effects if criteria are not fully met

1. If metrics are not aligned with incentives, then the teams may not feel appropriately rewarded for achieving program goals.
2. If the effort to collect data to support a metric is too extensive, the metric may not deliver enough value to justify its collection.

Best practice: Establish management commitment

Key considerations and questions

1. Has management established procedures to collect metrics consistently over time?
2. Is management monitoring the performance metrics, and using them to inform corrective actions?

3. Is management working to ensure that metrics are in place to support automation and Agile program management and reporting?
4. How is management supporting programs' abilities to tailor metrics to ensure that they meet organization needs while also limiting unnecessary work on the part of the program?
5. How is management balancing periodic program-wide health assessments with monitoring the progress made in deploying capabilities during each release?
6. During performance review meetings, are staff from different levels of the organization involved?

Likely effects if criteria are not fully met

1. If management does not demonstrate a commitment to use performance metrics, others may not embrace metrics as useful.
2. If forced to report Waterfall development-based metrics, such reporting will not only impede Agile adoption and execution, but also will not provide accurate insight into the software development process.
3. If program officials do not establish performance thresholds and targets, oversight bodies may lack information to ensure the program is meeting acceptable performance levels.

Best practice: Commit to data-driven decision making

Key considerations and questions

1. Are metrics designed to support specific decisions that need to be made at different levels of the organization?
2. Does the contract structure achieve desired mission outcomes?
3. Is the program collecting technical, performance measurement, and process improvement metrics?
4. Does the organization capture metrics that allow it to determine whether Agile development activities contribute to organization goals as planned?
5. Do metric reviews match the cadence of the program?
6. Are target values established for critical metrics?

7. Are contracts formulated in such a way that they allow flexibility for implementation and provide meaningful information to decision makers? For example, are metrics such as software size, development effort, schedule, staffing, progress, etc. collected?
8. How are product quality and customer satisfaction monitored throughout the development cycle?
9. How are changing priorities monitored throughout the development cycle?
10. How are metrics considered in the requirements when formulating the contract?
11. How does the program collect metrics to gain insight into the costs associated with delaying work or missing a milestone?
12. How does the program estimate the cost of technical debt and the time and effort necessary to repay the debt?
13. How does the program measure and monitor the frequency of releases, the product delivery, and program progress? For instance, burn-up, and burn-down charts may be used to communicate progress.
14. Does the program use automated tools to capture metrics?
15. How does the program evaluate data for its completeness, comprehensiveness, and correctness to ensure that it is suitable for its intended purpose?
16. Does the program use automated tools for testing?
17. Is the program collecting necessary data that cannot be captured using automated tools, such as data related to team dynamics or other organizational behaviors?

Likely effects if criteria are not fully met

1. If the metric review schedule does not match the cadence of the development process, then management may not be able to provide timely feedback to take necessary corrective actions in order to maximize the value of delivered software.
2. If contracts are not formulated to capture the requirements to align with Agile processes, the decision makers may not have the meaningful information they need to manage development.
3. Without collecting metrics for overall program performance, organizations will not have a good understanding of the cost and time required to achieve a valuable product.

4. The data collected should be evaluated for its completeness, comprehensiveness, and correctness to ensure that it is suitable for its intended purpose. Otherwise data can mislead decision makers instead of accurately informing them about the program's status.
5. Without data collected using both automated tools and other data collection processes decision makers may not be able to determine if the program is delivering its desired value and outcomes.

Best practice: Communicate performance information frequently and efficiently

Key considerations and questions

1. How are metrics used to track Agile programs daily?
2. How is performance information communicated frequently and efficiently?
3. What tools are used to facilitate access to and dissemination of performance metrics?
4. Does the program have access to automated tools and dashboards to provide real-time input into oversight and decision making?
5. Does management have tools that allows it to view data consistently across programs?

Likely effects if criteria are not fully met

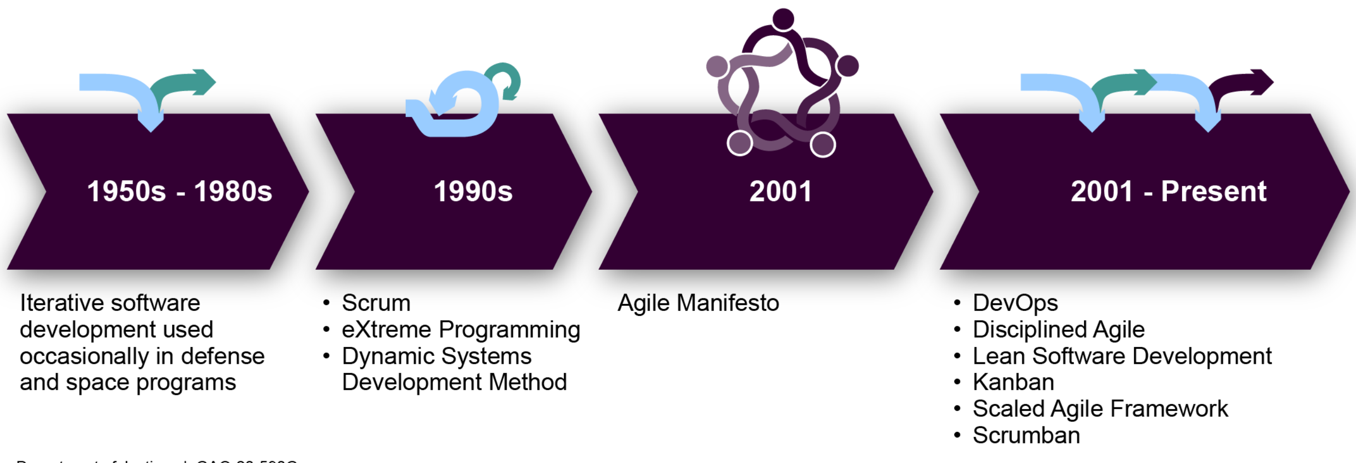
1. If metrics are not relevant, reliable, and timely, they cannot help mitigate Agile adoption and program execution risks.
2. Without tools to facilitate frequent information dissemination, decision makers may not have access to performance information and may not be able to take action in a timely manner to make improvements or corrective actions.
3. Miscommunicating performance information prevents staff and stakeholders from making necessary improvements or corrective actions in a timely manner can, contribute to program execution risks.
4. Without automated tools, management may not have access to data that allows it to assess all programs consistently and quickly.

Appendix V: Common Agile Frameworks

This appendix provides details on the most common Agile development frameworks that are mentioned in chapter 1. Each highlighted framework includes an overview, a brief discussion of the typical structure, and unique principles of the framework.

The *Agile Manifesto* was published in 2001; however, several frameworks that preceded it may have influenced the manifesto. Figure 19 provides a timeline showing the evolution of Agile development in the United States. For example, prior to 2001, some versions of incremental software development were being used, and in the 1990s, several Agile frameworks were published, most notably the presentation of Scrum in 1995. After the issuance of the *Agile Manifesto,* frameworks such as Kanban began incorporating the principles from lean manufacturing, which further supplemented Agile principles. Agile frameworks continue to evolve, giving developers a wide array of options for tailoring their development approach. Frameworks included in this appendix are: those commonly used according to literature;[1] frameworks used on federal programs GAO previously reported on; and those recommended to be included by experts. Although we are referring to these frameworks as "Agile frameworks," this is a loose term encompassing Agile-related frameworks, some which may not adhere to all Agile principles. The frameworks in figure 17 are discussed in this appendix.

Figure 17: Timeline of Agile Development



Source: Department of Justice. | GAO-20-590G

[1]CollabNet VERSIONONE, COLLAB.NET, VERSIONONE.COM, *14th Annual State of Agile Report,* (Atlanta, GA: May 26, 2020).

DevOps

Overview

DevOps methods combine both development and operations. Prior to DevOps, a typical Agile team would have been responsible for the software from requirement to deployment, with an operations team being responsible for the support of the software after the deployment. DevOps reduces the barrier between development and operations by combining them, thus delivering software quickly and ensuring its high quality by using the same team. The rationale is that, if the developers are also responsible for support, they may have more of an incentive to create reliable code.

Structure

In DevOps, the development and operations teams collaborate: the developers may also be responsible for operation, or there could be two separate teams that have open communication. Regardless of the particular configuration, teams should be made to feel ownership of the entire software life cycle.

Principles

The driving force of DevOps is to create frequent, small releases.[2] In order to do this, DevOps teams frequently adopt several of the principles listed in table 19.

Table 19: DevOps Principles

Principle	Description
Automation of processes	DevOps teams try to release software as frequently as possible, which requires automated testing and development (continuous integration and continuous delivery).
Standardized environment	Many issues of interoperability arise when new code does not work in the operations environment. Since the DevOps team develops the software and troubleshoots bugs in the operations environment, the developers become more familiar with this environment. Standardizing the environment helps with these interoperability issues.
Microservices	In order to push frequent releases, the DevOps team uses an architecture comprised of microservices: small, decoupled components that ideally work independently of the other software components.
Monitoring	Since DevOps teams are responsible for support and operations, the teams should be monitoring the operational software. The frequent releases can help the team isolate and pinpoint which software update has an issue.

Source: GAO analysis of Booz Allen Hamilton information. | GAO-20-590G

[2]This is in accordance with the third principle in the *Agile Manifesto*, "Deliver working software frequently, from a couple of weeks to a couple of months, with a preference to the shorter timescale." © 2001-2020 *Agile Manifesto* authors https://agilemanifesto.org.

Disciplined Agile

Overview

The Disciplined Agile (DA) framework scales Agile methods with the intent of addressing the full IT product delivery process from program initiation to deployment into production. DA is a hybrid process that adopts and tailors strategies from a number of frameworks. Specifically, DA adopts strategies from Scrum, Extreme Programming (XP), Agile Modeling (AM), Agile Unified Process (AUP), and Kanban. DAD is goal-driven, emphasizing the delivery life cycle and how a product can provide a solution (rather than being simply an independent product).

Structure

The primary roles of a DA team are described in table 20.

Table 20: Disciplined Agile Roles and Responsibilities

Role	Responsibilities
Stakeholder	Provides requirements, either as part of the team or through a team representative, in order to inform user stories. Also responsible for ensuring that developed products satisfy all appropriate requirements following iterations and tests, thus preparing the products for release. Stakeholders include four distinct sub-groups: customers (who actually use the system), principals (decision makers who pay for the system), partners (who make the system work in the environment with other existing systems), and insiders (developers).
Product owner	Clarifies details and maintains list of work items that the team needs to implement. Represents work of Agile team to stakeholder community.
Team member	Performs analysis, testing, evaluation, design, programming, planning, estimation, and many more activities throughout the program.
Team lead	Facilitates communication and empowers team members to self-optimize their processes. Ensures that the team has the resources needed.
Architecture owner	Makes system architecture decisions for the team and ensures that the solution is integrated and tested on a regular basis. The individual in the team lead role on smaller Agile teams may also fill the role of architecture owner.

Source: GAO analysis of DOJ and FAA information. | GAO-20-590G

In addition to these primary roles, DA identifies secondary roles for specialist, independent tester, domain expert, technical expert, and integrator. These secondary roles are not required for every team and are often used when a program scales larger and may only be needed for a short period of time.

Principles

Key principles for DA are shown in table 21.

Table 21: Disciplined Agile Principles

Principle	Description
People first	DA defines primary roles for a specific team, as described in table 20
A hybrid framework	DA uses strategies and principles from many different methods, such as Scrum, XP, Kanban, and more.
A full delivery life cycle	The process supports the full life cycle, from planning to release. A DA program can choose from four different life cycles and tailor each to support their program.
Goal driven	DA emphasizes that a program is flexible, easy to scale, and lays out general goals and various solutions including any pros/cons. The program uses this information to pick the solution that works best.
Enterprise aware	A DA team works within an organization, follows the organization’s guidance, and leverages existing assets.

Source: GAO analysis of DOJ and FAA information. | GAO-20-590G

Dynamic Systems Development Method

Overview

Created in 1994, the Dynamic Systems Development Method (DSDM) brings control and quality to software development by focusing on transparency and communication. The framework, which can be used to scale Agile for larger programs with multiple teams, is intended to be used for management of the full life cycle of a program.

Structure

DSDM has a defined 4-phase process that covers the entire life cycle of a program: feasibility, foundations, evolutionary development, and deployment. The team is sorted by areas of interest: business, technical, and management. The roles to support these areas of interest are categorized into program, development, and supporting roles. Each specific role has defined responsibilities within the DSDM process. For example, the technical coordinator provides technical leadership at the program level.

DSDM promotes certain practices—such as facilitated workshops, prioritizing work, and modelling, among others—to facilitate the program process and align with DSDM principles. Specifically, facilitated workshops are used to help a team reach consensus on the requirements for a deliverable. In addition to prioritizing the work, DSDM uses the MoSCoW technique, in which work is categorized as must have, should have, could have, or won’t have this time. This triage allows the team to focus on the highest priority work. Finally, DSDM promotes modelling (visual representation of a program) as a way to increase communication within the team.

Principles

There are eight principles in DSDM, described in table 22.

Table 22: Dynamic Systems Development Method Principles

Principle	Description
Focus on the business need	The team understands the business needs and priorities. There is continuous business commitment throughout development.
Deliver on time	Teams time box their work in iterations, allowing them to always deliver on time while flexing the scope of features. With iterations, the team should be able to have a predictable delivery.
Collaborate	The entire team—including stakeholders and business representatives—collaborate for better understanding and shared ownership of a program. This is supported by empowering team members to make decisions in areas they represent.
Never compromise quality	The level of quality is agreed on before development starts with acceptance criteria. Testing is integrated throughout development, done early and continuously.
Build incrementally from firm foundations	Understand the business problem and plan the proposed solution. Teams should design an overarching solution first in order to lay a firm foundation and build the solution from this foundation, with increments providing for feedback and routine re-assessment of the program.
Develop iteratively	Iterative development allows for timely feedback through frequent demonstrations and reviews.
Communicate continuously and clearly	DSDM encourages informal, face-to-face communication and daily standups. Additional modelling, prototyping, and workshops can increase communication throughout the team.
Demonstrate control	In order to demonstrate control, the program manager should measure and report plans and progress. The program manager should be assessing the program according to the business needs.

Source: GAO analysis of DSDM Consortium information. | GAO-20-590G

eXtreme Programming

Overview

eXtreme Programming (XP) advocates frequent releases in short, iterative development cycles. This approach promotes team productivity and introduces checkpoints where various customer/stakeholder requirements can be introduced, refined, and adopted. Kent Beck originally developed XP while working for Chrysler Corporation in 1996;

he published and expanded on the method in eXtreme Programming Explained.[3]

Structure

XP does not prescribe formal roles and responsibilities to teams; instead, it relies on teams that are self-organized, cross-functional, and include the customer. XP has several best practices, including: small releases, simple design, and pair programming, among others.

Like other Agile frameworks, XP attempts to reduce the cost of changes in requirements by having multiple short development iterations with feedback loops to continually refine customer requirements, rather than one single long cycle. This approach focuses on coding and helps to ensure that team members have a complete understanding of business requirements early in the process.

Activities performed during every XP software development cycle include planning, managing, designing, coding, and testing, which are further described in table 23.

Table 23: eXtreme Programming Activities

Activity	Description
Planning	Involves writing user stories, release planning, and dividing programs into small iterations.
Managing	Teams operate in an open work space at a sustainable pace, participate in standup meetings, and continually measure their velocity.
Designing	Teams focus on keeping the design simple, only adding functionality when needed, and refactoring, among other things.
Coding	In XP, all code is produced using pair programming, meaning two developers create the code together, with the intent to increase the quality of the code. In addition, unit tests are written first, standards are used for all code, and new code is integrated often. XP also practices the idea of collective ownership, meaning all team members have a responsibility for the code base and can make changes to improve it.
Testing	All code should have a unit test, and the code must pass all unit tests before it is released. Acceptance tests are run frequently and all test results are published.

Source: GAO analysis of DOJ and Agile Alliance information. | GAO-20-590G

Principles

Table 24 shows the five key values embraced by XP to guide how team members, program managers, and stakeholders interact and collaborate to ensure product quality. When employed by teams, these values (communication, simplicity, feedback, courage, and respect) can help

[3]Kent Beck and Cynthia Andres, *eXtreme Programming Explained: Embrace Change* (Boston: Addison-Wesley Professional, 2004).

them to achieve clear coordination and feedback throughout the development process.

Table 24: eXtreme Programming Values

Value	Explanation
Communication	System requirements are effectively communicated from the customer to the team. XP builds rapidly and passes along institutional knowledge among members of the development team in an effort to give one another consistent information. XP advocates sharing among customers and designers to improve the design and construct the system.
Simplicity	XP emphasizes starting with the simplest possible solution and building functionality on it later. To achieve this goal, XP strives to do only what is asked for, and nothing more, in order to maximize value. Simplicity in code also contributes to reduced maintenance, as the code can be easily understood by the maintainers.
Feedback	Teams obtain system feedback through periodic integration and unit testing that is intended to catch problems before the product is released. Teams help ensure that the software meets customer needs by conducting acceptance testing and incorporating feedback.
Courage	Programmers are encouraged to throw away portions of low quality code they have worked on to ensure what they deliver high quality. Improved code can lead to better results and remove impediments to effective development. XP programmers are also urged to accurately report progress, develop reasonable estimates, and adapt to changes when they happen.
Respect	Team members are expected to be respectful to one another and to value the expertise of their customers, who participate in the development effort. Program managers and executives respect team members' responsibility and appropriate authority over their own work.

Source: GAO analysis of DOJ and Agile Alliance information. | GAO-20-590G

Kanban

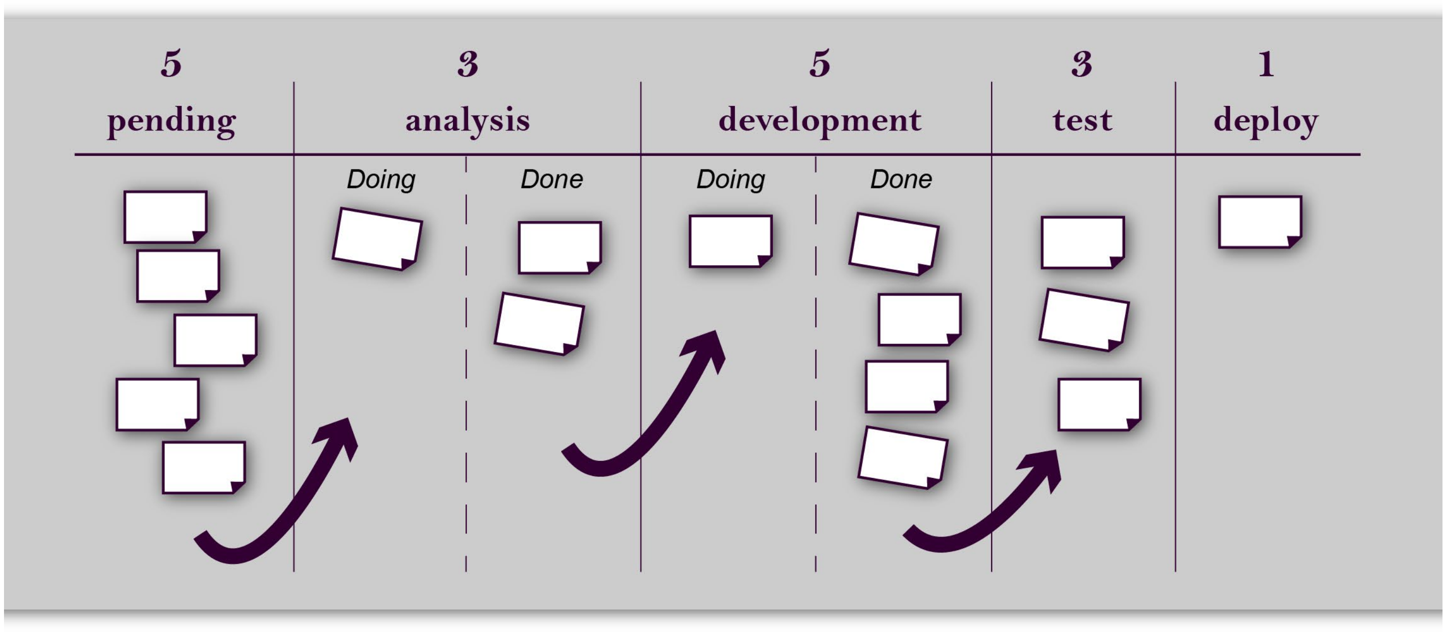
Overview

Kanban seeks to alleviate the bottlenecks in Waterfall development by limiting "in-progress" work in order to efficiently and effectively design and deliver products to customers. Limiting work-in-progress prevents a team from committing to too much work. Since new work should not be started until the current work has been completed, bottlenecks blocking the completion of work should become more visible in the process. This framework focuses on the flow of work and was inspired by lean manufacturing. Kanban is still used in manufacturing, as well as other applications; this section focuses on Kanban for software development.

Structure

There are no prescribed roles in Kanban, allowing for maximum team flexibility so that members can work on each other's artifacts easily. Teams use a Kanban board to keep track of their work, which can be either physical or virtual. A Kanban board maintains a clear, visual representation of the work through various stages of development. An example of a typical board is shown in figure 18.

Figure 18: Kanban Board



Source: Department of Justice. | GAO-20-590G

A Kanban board displays work using sticky notes. The numbers at the top of each column are the limits on the number of work items allowed per column. As a task is completed, the related notes are physically moved to the next stage so that completed and remaining work can be seen. Having a board to review provides a summary of where the team needs to focus its efforts.

Principles

Kanban is based on three basic principles: visualize what you do today (workflows), limit the amount of work-in-progress, and focus on flow (backlog prioritization). These Kanban principles are intended to be responsive to changes that often occur during a demonstration. Having a short cycle time helps ensure that customers provide feedback to the team on a regular basis, resulting in delivery of desired software features faster than traditional methods. In addition, Kanban promotes having user stories that are all similar in size in order to limit in-process work so that it is both manageable and predictable.

Lean Software Development

Overview

Lean software development combines lean manufacturing and IT principles to streamline software development. Although there is no single lean software development process, the structure, principles, and practices further explained in table 25 stem from the book *Lean Software Development* by Mary and Tom Poppendieck.[4]

Lean and Agile are related philosophies. More specifically, Lean can be characterized as related to, but not a subset of, Agile. Many of the lean practices and principles can be mapped to Agile methods, such as speed and customer engagement.

Structure

There is no formal team structure according to Lean principles.

Principles

Lean software development is organized around seven key principles that are aligned closely with those found in Lean manufacturing, as shown in table 25.

Table 25: Lean Software Development Principles

Principle	Description
Eliminate waste	Recognize waste, create nothing but value, and keep the code simple.
Amplify learning	Try different ideas, maintain a culture of constant improvement, and teach problem-solving methods.
Deliver fast	Deliver solutions in small iterations, focus on cycle time, release early and often, and follow the just-in-time ideology.
Defer commitment	Make irreversible decisions at the last responsible moment (when the customer better realizes their need), break dependencies between components, and maintain options for as long as possible.
Empower the team	Train team leaders and supervisors, move responsibility and decision making to the lowest possible level, and instill a “find good people and let them do their own job” approach.
Build integrity in	Synchronize effort, automate testing and routines, and refactor to avoid code duplication.
Optimize the whole	Focus on value to the customer, deliver a complete product with input from all stakeholders, and find and eliminate all defects.

Source: GAO analysis of DOJ and Addison-Wesley Professional information. | GAO-20-590G

[4]Mary and Tom Poppendieck, *Lean Software Development: An Agile Toolkit* (Boston, Massachusetts: Addison-Wesley Professional, 2003).

These principles guide lean software development by emphasizing limiting any “waste” that teams create (e.g., duplicate code, re-iteration of working components, and extensive documentation of activities beyond what is required) to achieve a streamlined, efficient program outcome.

There are also 22 practices or tools to implement lean software development practices. Among them are eliminating waste and focusing on value by using value stream mapping, amplify learning via feedback from iterations, and deliver as fast as possible with pull systems and queuing theory.

Scaled Agile Framework

Overview

The Scaled Agile Framework (SAFe) is a governance model used to align and collaborate product delivery for modest-to-large numbers of Agile software development teams. The framework provides guidance for roles, inputs, and processes for teams, programs, large solutions, and portfolios. It is also intended to provide a scalable and flexible governance framework that defines roles, artifacts, and processes for Agile software development across all levels of an organization.

Structure

There are four different configurations of SAFe: essential, large solution, portfolio, and full. These configurations allow for different scales of teams to adopt SAFe, depending on the size and complexity of the product. These levels allow teams to perform iterative processes using Agile frameworks such as Scrum, XP, Lean, or others to develop features to be used by a larger program that conforms to the overarching portfolio vision within an enterprise. SAFe uses many of the same tools as other Agile methods, such as backlogs, development teams, and time boxed iterations.

Depending on the scale, the framework is divided into different levels, each with its own responsibilities and processes that connect the different levels. Development teams in SAFe align with the selected framework and are advised to embrace the traditional “cross-functional team” mentality. At the program level, these Agile teams come together to create a “release train” that reflects specific roles and responsibilities, as shown in table 26.

Table 26: Scaled Agile Framework Roles and Responsibilities

Role	Responsibilities
Scrum master	Facilitates meetings, removes impediments, and maintains the team's focus.
Product owner	Owns the team backlog and prioritizes work. Also acts as the customer for developer questions and collaborates with Product Management to plan and deliver solutions.
Development team	Has three to nine individual contributors, covering all the roles needed to build an increment of value for an iteration.
Release Train Engineer	Facilitates program-level execution, removes impediments, performs risk and dependency management, and fosters continuous improvement.
Product management	Responsible for identifying items to be added to the program backlog, prioritizing the backlog, and interfacing with product owners to confirm alignment between the software components and enterprise goals. Also responsible for the vision, road map, and new features in the program backlog.
System architect/engineer	Focuses on stakeholder needs and ensuring that the solution is designed to cater to these needs while delivering functionality across various features, components, and the larger solution.
Business owners	Responsible for the business outcomes of the product.

Source: GAO analysis of DOJ, FAA, and Scaled Agile Inc. information. | GAO-20-590G

Principles

SAFe[5] has ten framework principles, outlined in table 27, that can be tailored to suit a program's requirements.

Table 27: Scaled Agile Framework Principles

Principle	Description
Take an economic view	Decisions are made within the proper economic context. Strategies for incremental delivery are developed and communicated. A framework is created that takes into account risk, different types of cost, and decentralized decision making.
Apply systems thinking	Systems thinking solutions development takes a holistic view, incorporating both the system and the environment, taking into account people, management, and processes.
Assume variability; preserve options	Variability is neither good nor bad in SAFe. Multiple options should be considered, and these options should be maintained for as long as possible. Learning should be encouraged, even if it results in mistakes.
Build incrementally with fast, integrated learning cycles	Develop the system incrementally in order to determine technical feasibility, establish usability, and gain customer feedback, among other benefits. Value is delivered at each increment, and uncertainty is reduced as more is learned.
Base milestones on objective evaluation of working systems	Milestones with SAFe are based on demonstrating working software. These milestones allow stakeholders to frequently evaluate the software.
Visualize and limit work in progress, reduce batch size, and manage queue length	Limiting work-in-progress helps ensure that teams are not overloaded with work, while visualizing work-in-progress allows for easy identification of bottlenecks. Another way to limit work-in-progress is to decrease batch size (batch being the requirements, design, code, tests, etc.), so more work can flow through the process. This is typically accomplished by increasing automation and infrastructure.

[5]As of May 2020, this guide refers to SAFe v5.0.

Principle	Description
Apply cadence, synchronize with cross-domain planning	Cadence provides a rhythmic pattern and a consistent routine to development. Synchronization allows the teams to align with a common goal and is enabled by events like release planning, where all stakeholders participate in planning the next increment.
Unlock the intrinsic motivation of knowledge workers	Since knowledge workers understand more about the technical aspects of their work than their manager, the manager's role is to motivate teams rather than direct their work. Motivation should stem from innovation and engagement rather than threats, intimidation, or fear. Managers provide workers with a larger vision, which guides them to autonomously perform daily tasks. Managers support teams during disagreements (where appropriate) by helping them to negotiate and problem solve, among other things.
Decentralize decision-making	Strategy decisions that are infrequent, long lasting, and provide significant economies of scale can be centralized while all other decisions can be decentralized in order to reduce delays.
Organize around value	The organization's structure with SAFe should be driven by value flow instead of traditional silos. This allows the organization to more quickly adapt to changes in the value flow.

Source: GAO analysis of DOJ, FAA, and Scaled Agile Inc. information. | GAO-20-590G

Scrum

Overview

Scrum, the most widely used framework for Agile software development, seeks to address complex problems while delivering high-value products frequently and effectively. Originating from a 1986 text by Hirotaka Takeuchi and Ikujiro Nonaka titled, "The New New Product Development Game," the method was first referred to as "Scrum" by Ken Schwaber and Jeff Sutherland in the early 1990s to emphasize a holistic approach using multiple, overlapping phases.[6] Schwaber and Sutherland authored the *Scrum Guide,* which details the methodology.[7] Scrum relies heavily on the concept of "Scrum teams" that are responsible for producing working software in increments often referred to as a "sprint." Each sprint is a short, time boxed iteration that is intended to provide distinct, consistent, and incremental progress of prioritized software features.

Structure

The Scrum framework is centered on Scrum teams where members fill specific roles and responsibilities. These members are responsible for various tasks, including developing Agile artifacts. Each team contains members that fit into one of these three main roles, as shown in table 28.

[6]Takeuchi, Hirotaka and Ikujiro Nonaka. "The New New Product Development Game." Harvard Business Review 64, no. 1 (January–February 1986).

[7]Ken Schwaber and Jeff Sutherland, *The Scrum Guide:™ The Definitive Guide to Scrum: The Rules of the Game* (Creative Commons, 2017).

Table 28: Scrum Team Structure

Role	Responsibility
Product owner	Represents stakeholders.
Development team	The group that carries out software coding, implementation, testing, and development.
Scrum master	Responsible for making sure Scrum theory, practices, and rules are adhered to by the development team.

Source: GAO analysis of DOJ, Booz Allen Hamilton, and The Scrum Guide information. | GAO-20-590G

With Scrum, teams are self-organizing and choose how best to accomplish their work, rather than being directed by management. Teams are also cross-functional, meaning they include members who have the capabilities to achieve the work without depending on someone outside the team. This model optimizes flexibility, creativity, and productivity and seeks to eliminate the need for a traditional program manager since each team supervises itself.

During sprint planning meetings, the team determines the type of work to be done, prepares the sprint backlog (ordered list of tasks to be accomplished during the sprint), and communicates expected responsibilities between team members. Teams meet daily during each sprint for a brief status update. Each sprint is intended to produce, among other things, completed increments of software features that are ultimately built into the final product solution.

The sprint backlog is a subset of the most important features from the overall product backlog. Teams decompose these requirements into user stories that describe what the customer wants. The software developed during the sprint should satisfy those needs in order for a user story to be considered complete.

A burn-down chart is a public display of the remaining work in the sprint backlog. The team updates the burn-down chart daily to keep everyone informed of the status of tasks.

Principles

Scrum is founded on three pillars that uphold the process. Table 29 outlines the three pillars.

Table 29: Scrum Principles

Principle	Description
Transparency	A common standard and understanding must be shared in order for the process to be visible. For example, the definition of done documents a common definition between developers and product owners.
Inspection	Artifacts are frequently inspected to detect any issues, but this inspection should not get in the way of work.
Adaptation	Adjustments should be made as soon as possible. Recurring events like sprint planning meetings and retrospectives provide additional refinements and updates.

Source: GAO analysis of DOJ, Booz Allen Hamilton, and *The Scrum Guide* information. | GAO-20-590G

Scrumban

Overview

Scrumban combines both Scrum and Kanban, typically by using the Scrum team structure with Kanban process principles. Scrumban is seen as being more flexible than Scrum, but more structured than Kanban.

Structure

Similar to Scrum, Scrumban uses iterative planning, requirements prioritization, and structured teams. From Kanban, Scrumban uses the pull system, work-in-progress limits, and work visualization (Kanban board).

Principles

Scrumban relies on the principles of Scrum and Kanban, as discussed in the previous sections.

Appendix VI: Debunking Agile Myths

Myth 1: Agile does not require any documentation

The adaptive and iterative nature of Agile places less emphasis on the need for documentation when compared to Waterfall development methods, but that does not mean that no documentation is required. A Waterfall development results in detailed documentation at the end of each phase and the program requirements are not expected to change much over time. However, elements of the program continuously evolve as additional information becomes available and customer needs are further defined. As a result, Agile programs use an appropriate level of documentation at the end of pre-defined time boxed periods in the Agile development cadence (e.g., the iteration, release, or other major milestone as defined by the program). In addition, in some cases, an Agile approach might replace more formal documentation with information embedded in program tools.

Myth 2: Agile does not require planning

As with any approach, planning is a vital aspect that will greatly diminish the effectiveness of a successful implementation if not done appropriately. Waterfall development conducts extensive planning upfront, while Agile spreads planning activities (e.g., what specific functionality will be delivered when) more evenly throughout the program life cycle. High-level planning is completed at the beginning of an Agile program and is continuously elaborated on throughout the program as new information becomes available. Continuous planning allows a program to start much more quickly and make adjustment to the customers' needs as new information becomes available.

Myth 3: Agile does not require any oversight

Within an Agile approach, the team members working on the program have autonomy over decisions about how to meet the needs of the customer. However, most government organizations will find it challenging to allow teams complete autonomy due to reporting and accountability requirements. As a result, organizations transitioning to Agile may need to modify their governance practices. This includes incorporating clearly defined parameters (also called guard rails) within which the team is free to make decisions and a clearly defined, fast-moving governance process to make decisions that are outside the team's control.

Myth 4: Agile works only in co-located environments

For any program, it is almost always better off if its participants are co-located. Frequent human interaction is a necessary element of Agile, but it is also necessary when employing Waterfall development methods. Furthermore, a lack of co-location can be a serious impediment if a program is poorly managed. However, distributed programs can still succeed. As is true for any program type, distribution calls for careful management and awareness what needs to be executed differently when some team members are not in the same location. For example, there are many tools available that allow for close communication between team members who are distributed throughout various locations.

Myth 5: Agile only works for small programs with a single team

An Agile development team consists of small, cross-functional groups that collaborate throughout the development process. This approach can be equally effective on small programs and larger efforts working to develop complex systems, since Agile teams typically “divide and conquer.” For larger programs, this means that teams can be organized and focus on separate components of system functionality and/or technical architecture.

For Agile programs of all sizes, but especially for the large and complex programs, continuous integration of developed components on a daily, if not more frequent, basis is a critical success factor. More specifically, teams need to check in and test newly-developed code against the larger solution within a production-like environment. In an Agile program with typically short development iterations, parallel development efforts, and frequent delivery of functionality, teams must integrate their work often to detect and resolve errors as quickly as possible, with the ultimate goal of being able to deploy at any time. If teams delay integration to just-prior-to-release, they will likely run out of time to adequately perform testing, address defects, and prepare the infrastructure. Teams should ensure that they have the right automated build and test tools, and the appropriate processes in place to support continuous integration.

Myth 6: Using any Agile framework will automatically result in program success

Deciding to use an Agile framework should occur on a program-by-program basis. Agile is not necessarily the solution for all programs. For example, not all programs will have flexible requirements, allowing trade-offs to occur between scope with schedule and costs. With every software development effort, learning to deal with issues as they arise is the key to reducing the risks of failure.

Myth 7: Agile requires a lot of rework

While Agile emphasizes that only near-term work is planned in detail (such as just the next iteration), programs still define their overall goal in a vision and typically plan the releases needed to satisfy the vision. This plan could change or end early, but still provides a high-level view of the work to be accomplished for the entire duration of the program. Additionally, while the team self-organizes its own work, it must still be aware of any dependencies with other teams or resources.

Myth 8: Agile does not require an architecture

Agile does not mean cobbling together an IT system with little or no design or architectural thinking. Agile stresses simplifying upfront design, not eliminating upfront design. The *Agile Manifesto* states that "Continuous attention to technical excellence and good design enables agility."[1] Furthermore, many Agile frameworks provide the tools and techniques for the team to produce high-quality code. Many of the best practices discussed in previous chapters are aimed specifically at ensuring that the quality of the product being delivered is fit for the purpose. Agile stresses simple, upfront design to focus on the foundation and general structure of the software. For example, Agile developers avoid building software features that may or may not be needed and instead build for the current need and receive feedback in the iterative delivery of software to the client. However, that does not mean that Agile teams do not need high-level architecture to succeed. Rather, Agile systems strive to keep their architecture simple and only add complexity when it is needed.

Myth 9: Agile does not require risk analysis

As Agile teams are self-organizing and its iterative process is viewed as a way to mitigate the inherent risk in developing complex software programs, a perception can develop that explicit risk management practices are unnecessary. All programs face risk and uncertainty, whose likelihood and potential impact should be examined. For example, effective practices for Agile include developing initial plans at a high level and updating these frequently as more is learned about the program. While Agile emphasizes that teams will uncover risk via early and frequent delivery of software, the potential impact of some issues, such as technical debt or team size, should be considered earlier rather than later.

Myth 10: A schedule baseline cannot be reliably developed or used for an Agile software development effort

A central tenet of Agile is to welcome change. As part of this, teams practice rolling wave planning, a technique where only near-term work is planned in detail. This helps to minimize the cost of changing plans, but frequent changes can appear to be in conflict with the concept of adhering to a baseline. Another key principle is that working software should be the primary measure of progress, so schedule trends displayed in burn-down/burn-up charts are seen as lagging indicators. However, welcoming change does not mean that software is developed and delivered in an undisciplined or ad hoc manner. A key principle of Agile is that the highest priority is to satisfy the customer through early and continuous delivery of usable software, and teams typically develop and

[1]©2001-2020 Agile Manifesto authors https://agilemanifesto.org.

deliver working software to the customer in time boxed iterations. These iterations are guided by the vision, which establishes a high-level definition of the cost, schedule, and scope goals for the program and provides a basis for specifying expected outcomes for iterations. These must have features identify the program's schedule baseline and, as a result, developers have the ability to demonstrate the value provided by features developed at the end of releases and how those features tie to the program vision. A baseline should be created and approved in concert with a rolling wave planning process, and it should contain enough detail to enable a collaborative agreement between product owners and developers without making schedule updates overly frequent or cumbersome. As the schedule is updated with actual data and revisions are made, updates can be documented in progress records through various Agile metrics and a schedule narrative. Schedule trends showing deviations from the baseline can be used to understand the need for changes, whether to program execution or to the baseline itself, which can be updated only if it is no longer a realistic portrayal of program execution. This helps ensure that the baseline provides a good basis for measuring and understanding progress and maintaining accountability.

Myth 11: Earned value management is not compatible with Agile Programs

Since Agile development is dynamic, developers claimed that earned value management (EVM) is not well suited as a measurement tool in an Agile environment. However, EVM is an important management tool that provides performance measurement information for a program. In the past, recommendations were made to eliminate EVM for Agile programs because it was not fluid enough to implement effectively. While EVM tracks program performance to a fixed point in time, using an Agile approach does not preclude the need for a disciplined approach for performance measurement processes. This is especially true for government Agile programs. While scope is flexible for an iteration, often scope is not flexible for the overall program. When the scope is not flexible, as assumed for Agile programs, then additional expenditures and time may be needed to meet all requirements. A tailored EVM approach, as discussed in chapter 7, can leverage EVM's benefits for Agile programs. Additionally, EVM is not tied to any specific development methodology and does not prevent the use of other risk management techniques like those used in Agile development. Furthermore, Agile development can be used to incrementally deliver functionality to the customer, while EVM provides a standard method for measuring progress.

Appendix VII: Background for Case Studies and Agile in Actions

Case studies

Case studies used in this guide were taken from GAO reports and highlight problems and successes typically associated with Agile practices. These particular examples were chosen to augment key points and lessons learned that are discussed in the guide. Agile in action examples feature practices adopted by programs and organizations we interviewed that we believe illustrate Agile key practices executed in an exemplary or innovative way. The difference between a case study and an Agile in action example is that the Agile in action examples are not based on published GAO reports, but rather on our research, interviews, and by self-reporting entities.

The material in the guide's 15 case studies was drawn from the eight GAO reports described in this appendix. The material in the guide's five Agile in action examples were drawn from six site visits GAO made to various organizations. Table 30 shows the relationship between published GAO reports and case studies and the chapters in which the reports are cited. The table is arranged by the order in which the case study appears in the guide. Following the table, paragraphs describe the reports used (listed in the same order as listed in the table).

Table 30 shows the relationship between the case studies, GAO report, and the chapters in which the organizations are cited. The table is arranged by the order in which the case studies appear in the guide. Following the table, paragraphs describe the organizations visited.

Table 30: Case Studies Drawn from GAO Reports Used In this Guide

Case Study	GAO report #	Chapter
1	GAO-20-146: *Space Command and Control*	2
2	GAO-18-184: *Defense Management*	3
3, 13, 15	GAO-16-467: *Immigration Benefits System*	3, 8
4, 5, 6, 10, 12	GAO-20-213: *Agile Software Development*	3, 5
7	GAO-19-136: *DOD Space Acquisitions*	3
8	GAO-19-164: *FEMA Grants Modernization*	4
9, 14	GAO-18-46: *TSA Modernization*	5, 8
11	GAO-20-170SP: *Homeland Security Acquisitions*	5

Source: GAO. | GAO-20-590G

Case Study 1: From *Space Command and Control: Comprehensive Planning and Oversight Could Help DOD Acquire Critical Capabilities and Address Challenges,* GAO-20-146, October 30, 2019.

Since the early 1980s, the Air Force has been working to modernize and consolidate its space command and control systems. The past three programs to attempt this have ended up significantly behind schedule and over budget. They also left key capabilities undelivered.

This report describes the status of DOD's newest efforts to develop space command and control capabilities and identifies challenges the Air Force faces in bringing them to fruition.

We found the Space C2 program is facing a number of challenges and unknowns, from management issues to technical complexity. Additionally, DOD officials have not yet determined what level of detail is appropriate for acquisition planning documentation for Agile software programs. They are also not certain about the best way to provide oversight of these programs but are considering using assessments by external experts. These knowledge gaps run counter to DOD and industry best practices for acquisition and put the program at risk of not meeting mission objectives. Additionally, software integration and cybersecurity challenges exist, further complicating program development. The Air Force has efforts underway to mitigate some of these challenges in the near term, but, until the program develops a comprehensive acquisition strategy to more formally plan the program, it is too early to determine whether these efforts will help to ensure long-term program success.

GAO reported its findings on October 30, 2019 in *Space Command and Control: Comprehensive Planning and Oversight Could Help DOD Acquire Critical Capabilities and Address Challenges,* GAO-20-146.

Case Study 2: From *Defense Management: DOD Needs to Take Additional Actions to Promote Department-Wide Collaboration,* GAO-18-194, February 28, 2018.

Although the Department of Defense (DOD) maintains military forces with unparalleled capabilities, it continues to confront organizational and management challenges that hinder collaboration and integration across the department. To address these challenges, section 911 of the *National Defense Authorization Act* (NDAA) for Fiscal Year 2017 directed the Secretary of Defense to issue an organizational strategy that identifies critical objective which span multiple functional boundaries and that would benefit from the use of cross-functional teams.

This report evaluates the extent to which DOD, in accordance with statutory requirements and leading practices, has developed and issued an organizational strategy, established Secretary of Defense-empowered cross-functional teams, and provided associated training for Office of the Secretary of Defense leaders. We found that DOD has implemented some of the statutory requirements outlined in section 911 of the NDAA to address organizational challenges but could do more to promote department-wide collaboration. Specifically, DOD established one cross-functional team to address the backlog on security clearances and developed draft guidance for cross-functional teams that addresses six of seven required statutory elements and incorporates five of eight leading practices that GAO has identified for effective cross-functional teams. Fully incorporating all statutory elements and leading practices will help the teams consistently and effectively address DOD's strategic objectives.

GAO reported its findings on February 28, 2018 in *Defense Management: DOD Needs to Take Additional Actions to Promote Department-Wide Collaboration,* GAO-18-194.

Case Studies 3, 13, 15 From *Immigration Benefits System: U.S. Citizenship and Immigration Services Can Improve Program Management,* GAO-16-467, July 7, 2016.

Each year, the U.S. Citizenship and Immigration Service (USCIS) processes millions of applications for persons seeking to study, work, visit, or live in the United States, and for persons seeking to become a U.S. citizen. In 2006, USCIS began the Transformation Program to enable electronic adjudication and case management tools that would allow users to apply and track their applications online. In 2012, to address performance concerns, USCIS changed its acquisition strategy to improve system development.

In May 2015, GAO reported that USCIS expected the program to cost up to $3.1 billion and be fully operational by March 2019. This includes more than $475 million that was invested in the initial version of the program's key case management component, USCIS's Electronic Immigration System (USCIS ELIS), which has since been decommissioned. This report evaluates the extent to which the program is using information technology program management leading practices.

We found software development and systems integration and testing for USCIS ELIS have not consistently been managed in line with the program's policies and guidance or with leading practices. Regarding software development, the Transformation Program has produced some software increments, but is not consistently following its own guidance and leading practices. The software development model (Agile) adopted by the USCIS Transformation Program in 2012 includes practices aimed at continuous, incremental release of segments of software. Important practices for Agile defined in program policies, guidance, and leading practices include ensuring that the software meets expectations prior to being deployed, teams adhere to development principles, and development outcomes are defined.

We also found the Transformation Program has established an environment that allows for effective systems integration and testing and has planned for and performed some system testing. However, the program needs to improve its approach to system testing to help ensure that USCIS ELIS meets its intended goals and is consistent with agency guidance and leading practices. Among other things, the program needs to improve testing of the software code that comprises USCIS ELIS and ensure its approaches to interoperability and end user testing, respectively, meet leading practices. Collectively, these limitations have contributed to issues with USCIS ELIS after new software is released into production.

GAO reported its findings on July 7, 2016 in *Immigration Benefits System: U.S. Citizenship and Immigration Services Can Improve Program Management,* GAO-16-467.

Case Study 4, 5, 6, 10, 12: From *Agile Software Development: DHS Has Made Progress in Implementing Leading Practices, but Needs to take Additional Actions,* GAO-20-213, June 1, 2020.

Many of the Department of Homeland Security's (DHS) major acquisition programs have taken longer than expected to develop or failed to deliver the desired value. In April 2016, to help improve the department's IT acquisition, and management, DHS identified Agile software development as the preferred approach for all of its IT programs and projects. This resulted in five Agile pilot programs. Each pilot program was overseen by a component integrated program team. Collectively, the first pilot programs were also overseen and supported by a DHS integrated program team. In April 2016, the department issued an Agile instruction, which identified Agile software development as the preferred approach for all DHS programs and projects that are to deliver an IT, or embedded-IT capability. The department also set an expectation for its component Chief Information Officers (CIO) to develop plans to increase the use of Agile development and justify any major IT programs that did not intend to use Agile development practices. Many DHS programs were already using Agile or similar incremental development methods before the department identified it as the preferred approach.

GAO found that DHS has addressed four of nine leading practices for adoption Agile software development. For example, the department has modified its acquisition policies to support Agile development methods. However, it needs to take additional steps to, among other things, ensure all staff are appropriately trained and establish expectations for tracking software code quality. By fully addressing leading practices, DHS can reduce the risk of continued problems in developing and acquiring current, as well as, future IT systems.

GAO reported its findings on June 1, 2020 in *Agile Software Development: DHS Has Made Progress in Implementing Leading Practices, but Needs to take Additional Actions,* GAO-20-213.

Case Studies 9, 14: From *TSA Modernization: Use of Sound Program Management and Oversight Practices Is Needed to Avoid Repeating Past Problems,* GAO-18-46, October 17, 2017.

TSA conducts security threat assessment screening and credentialing activities for millions of workers and travelers in the maritime, surface, and aviation transportation industries that are seeking access to transportation systems. In 2008, TSA initiated the TIM program to enhance the sophistication of its security threat assessments and to improve the capacity of its supporting systems. However, the program experienced significant cost and schedule overruns, and performance issues, and was suspended in January 2015 while TSA established a new strategy. The program was rebaselined in September 2016 and is estimated to cost approximately $1.27 billion and be fully operational by 2021 (about $639 million more and 6 years later than originally planned).

We were asked to review the TIM program's new strategy. This report determined, among other things, the extent to which TSA implemented selected key practices for transitioning to Agile software development for the program. We found the program only fully implemented two of six leading practices necessary to ensure successful Agile adoption. Specifically, the Department of Homeland Security (DHS) and TSA leadership fully committed to adopt Agile and TSA provided Agile training. Nonetheless, the program had not defined key roles and responsibilities, prioritized system requirements, or implemented automated capabilities that are essential to ensuring effective adoption of Agile.

GAO reported its findings on October 17, 2017 in *TSA Modernization: Use of Sound Program Management and Oversight Practices is Needed to Avoid Repeating Past Problems,* GAO-18-46.

Case Study 7: From *DOD Space Acquisitions: Including Users Early and Often in Software Development Could Benefit Programs,* GAO-19-136, March 18, 2019.

Over the next 5 years, DOD plans to spend over $65 billion on its space system acquisitions portfolio, including many systems that rely on software for key capabilities. However, software-intensive space systems have had a history of significant schedule delays and billions of dollars in cost growth.

Senate and House reports accompanying the *National Defense Authorization Act for Fiscal Year 2017* contained provisions for GAO to review challenges in software-intensive DOD space programs. This report addresses, among other things, (1) the extent to which these programs have involved users; and (2) what software-specific management challenges, if any, programs faced.

We found actual program efforts to involve users and obtain and incorporate feedback were often unsuccessful. This was due, in part, to the lack of specific guidance on user involvement and feedback. Although DOD policies state that users should be involved and provide feedback on software development projects, they do not provide specific guidance on the timing, frequency, and documentation of such efforts. In selected instances, the lack of user involvement has contributed to systems that were later found to be operationally unsuitable.

The programs we reviewed also faced software-specific challenges in using commercial software, applying outdated software tools, having limited knowledge, and training in newer software development techniques. For example, programs using commercial software often underestimated the effort required to integrate such software into an overall system. Secondly, selected programs relied on obsolete software tools that they were accustomed to using but which industry had since replaced. Finally, we found that two of the reviewed programs lacked knowledge of more modern software development approaches. DOD has acknowledged these challenges and has efforts underway to address each of them.

GAO reported its findings on March 18, 2019 in *DOD Space Acquisitions: Including Users Early and Often in Software Development Could Benefit Programs,* GAO-19-136.

Case Study 8: From *FEMA Grants Modernization: Improvements Needed to Strengthen Program Management and Cybersecurity,* GAO-19-164, April 9, 2019.

The Federal Emergency Management Agency (FEMA), a component of DHS, annually awards billions of dollars in grants to help communities prepare for, mitigate the effects of, and recover from major disasters. However, FEMA's complex IT environment supporting grants management consists of many disparate systems. In 2008, the agency attempted to modernize these systems but experienced significant challenges. In 2015, FEMA initiated a new endeavor (the GMM program) aimed at streamlining and modernizing the grants management IT environment.

GAO was asked to review the GMM program. We found GMM's initial May 2017 cost estimate no longer reflected current assumptions about the program. FEMA officials stated in December 2018 that they had completed a revised cost estimate, but it was undergoing departmental approval. We also found GMM's program schedule was inconsistent with leading practices; of particular concern was that the program's final delivery date of September 2020 was not informed by a realistic assessment of GMM development activities, and rather was determined by imposing an unsubstantiated delivery date.

GAO reported its findings on April 9, 2019 in *FEMA Grants Modernization: Improvements Needed to Strengthen Program Management and Cybersecurity,* GAO-19-164.

Case Study 11: From *Homeland Security Acquisitions: Outcomes Have Improved but Actions Needed to Enhance Oversight of Schedule Goals,* GAO-20-170SP, December 19, 2019.

Each year, the Department of Homeland Security (DHS) invests billions of dollars in a diverse portfolio of major acquisition programs to help execute its many critical missions. DHS plans to spend more than $10 billion on these programs in fiscal year 2020 alone. DHS's acquisition activities are on GAO's High Risk List, in part, because of management and funding issues. This report, GAO's fifth review, addresses the extent to which DHS's major acquisition programs are on track to meet their schedule and cost goals and current program baselines trace to key acquisition documents.

To help manage its multi-billions dollar acquisition investments, DHS has established policies and processes for acquisition management, requirements development, test and evaluation, and resource allocation. The department uses these policies and processes to deliver systems that are intended to close critical capability gaps, helping enable DHS to execute its missions and achieve its goals.

Traceability, which is called for in DHS policy and GAO scheduling best practices, helps ensure that program goals are aligned with program execution plans, and that a program's various stakeholders have an accurate and consistent understanding of those plans and goals.

Appendix I of this report presents individual assessments for each of the 29 programs we reviewed. Each assessments presents information current as of August 2019. They include standard elements, such as an image, a program description, and summaries of the program's progress in meeting cost and schedule goals, performance and testing activities, and program management-related issues, such as staffing.

GAO reported its findings on December 19, 2019 in *Homeland Security Acquisitions: Outcomes Have Improved but Actions Needed to Enhance Oversight of Schedule Goals,* GAO-20-170SP.

Agile in Actions

Agile in action examples were developed through various site visits made by GAO during the course of developing this guide. While they are not based on a previously published GAO report, they were developed by interviewing agency officials, reviewing documentation, and site visits to observe Agile being used. To verify that the information presented in these examples was complete, accurate, and up-to-date, we provided each organization with a draft version of our summary analysis.

Table 31: Agile in Action Drawn from GAO Interviews

Agile in Action	Agency/company visited	Chapter
1, 4	NNSA G2	5, 7
2	GSA (18F), US Air Force	6
3	DHS HQ	7
5	Agility Health	8

Source: GAO. | GAO-20-590G

Appendix VIII: Specialists Who Helped Develop this Guide

The list in this appendix names the knowledgeable specialists, with their organizations, who helped us develop this guide. The list includes the names of those who made significant contributions to the Agile Guide. They attended and participated in numerous working group meetings, provided text or graphics, submitted comments, and hosted research site visits.

Organization	Specialist
Agile Infusion, LLC	Bob Schatz
Agile Transformation, Inc.	Sally Elata
Artemis Consulting	Rohit Gupta
Boeing	Jonathan Kiser
	Jerry Starling
California Department of Technology	Jeffery Porcar
	Crystal Taylor
Census Bureau	Linda Flores-Baez
CGI Federal	Laura Bier
	Ed Canoles
ClearPlan, LLC	Robin Pulverenti
David Consulting Group	Mike Harris
Department of Defense	Lawrence Asch
	Harry Culclasure
Department of Education	Trey Wiesenburg
Department of Energy	Ty Deschamp
	Kim Hobson
	Tim Wynn
Department of Homeland Security	Katherine Mann
Department of Justice	Anthony Burley
Excella Consulting	Patrick McConnell
	Dane Weber
General Services Administration	Zachary Cohn
	Kendrick Daniel
	Ashley Owens
Humphrey's and Associates	Denise Jarvie
IBM	Myke Traver
Independent Consultant	Wendy Hilton
Intel	Sam Caldwell
	Leo Monford
Internal Revenue Service	Jerome Frese

Organization	Specialist
International Council on Systems Engineering (INCOSE)	Phyllis Marbach
Leidos	Phil Magrogan
Macro Solutions	Todd Hager
MITRE	Hassib Amiryar
	Tony Curington
National Archives and Records Administration	Sherli Nambiar
National Defense Industrial Association (NDIA)	Joe Fischetti
National Geospatial-Intelligence Agency	James Barclay
Northrop Grumman	Eugene Nkomba
National Science Foundation	Manik Naik
Office of Management and Budget	Jim Wade
Project Management Institute (PMI), Madrid Chapter	Mario Coquillant
Prometheus Consulting	Harold Affo
Scaled Agile team	Steve Mayner
Software Engineering Institute	Suzanne Miller
Sway Digital and Data	Eric Christoph
TekNirvana	Tarak Modi
TeraThink Corporation	Michael Staab
Treasury Department	Matthew Kennedy
United States Air Force	Michael You
United States Patent and Trade Office	Carol Eakins
	Victoria Figaro
	Kris Hillstrom
	John Owens
Vergys, LLC	Greg Mantel
Vidya, LLC	Neil Chaudhuri

Source: GAO. I GAO-20-590G

Appendix IX: GAO Contacts and Staff Acknowledgments

GAO Contacts

Timothy M. Persons, Ph.D., Managing Director, Science Technology Assessment and Analytics (STAA)/Chief Scientist, at (202) 512-6888 or personst@gao.gov

Carol Harris, Director, Information Technology and Cybersecurity (ITC), at (202) 512-4456 or harriscc@gao.gov

Other Leadership on this Project

Michael Holland, Assistant Director, ITC

Jennifer Leotta, Assistant Director, STAA

Key Contributors

Mat Bader, Senior Information Technology Analyst
Jenn Beddor, Senior Systems Engineer
Brian Bothwell, Assistant Director
Chris Businsky, Visual Communications Analyst
Juaná Collymore, Senior Operations Research Analyst
Alan Daigle, Information Technology Analyst
Tim DiNapoli, Director
Emile Ettedgui, Senior Operations Research Analyst
Nancy Glover, Senior Communications Analyst
Anna Irvine, Senior Operations Research Analyst
Karen Richey Mislick, Assistant Director
Amy Pereira, Senior Attorney
Martin Skorczynski, Assistant Director
Walter Vance, Senior Methodologist
Mary Weiland, Senior Operations Research Analyst

References

Agile Alliance. *Agile Glossary*. Retrieved March 5, 2020, from https://www.agilealliance.org/agile101/agile-glossary/.

Alleman, Glen B., and Henderson, Michael. “Making Agile Development Work in a Government Contracting Environment: Measuring Velocity with Earned Value.” Salt Lake City, Utah: Agile Development. (June 2003.)

Arell, Ray, Jens Coldeway, and Jorgen Heselberg. “Characteristics of Agile Organizations.” Agile Alliance. (2015.) Retrieved March 24, 2017, from https://www.agilealliance.org/characteristics-of-agile-organizations/.

Barclay, Jim, and Jon Ruark. “Top 10 Agile Questions to Ask as a Senior Manager”. National Geospatial-Intelligence Agency: July 25, 2016.

Bashir, Salma. “Team Facilitation.” (October 30, 2009.) Retrieved March 5, 2020, from https://www.slideshare.net/cococorina/team-facilitation.

Bellomo, Stephany, and Carol Woody. “DOD Information Assurance and Agile: Challenges and Recommendations Gathered Through Interviews with Agile Program Managers and DOD Accreditation Reviewers.” Pittsburgh, Pennsylvania: Carnegie Mellon Univeristy, Software Engineering Institute. (November 2012.)

Bier, L, and others. “Measuring Earned Value in an Agile World.” Binder Dijker Otte (BDO), Consultants to Government and Industry (CGI), & Deltek. (n.d.)

Booz Allen Hamilton. version 2.0. McLean, Virginia: Booz Allen Hamilton, June 2016. *Booz Allen Agile Playbook,*

California Department of Technology. *Understanding Agile,* version 1.0. Sacramento, California: Project Mangement Office, December 5, 2016.

Carnahan, Robin, Randy Hart, and Waldo Jaquith. “De-Risking custom technology projects.” General Services Administration. (August 5, 2019.) Retrieved September 3, 2019, from https://github.com/18F/technology-budgeting/blob/master/handbook.md#basic-principles-of-modern-software-design.

Carney, David, Suzanne Miller, and Mary Ann Lapham. “Agile Development in Government: Myths, Monsters, and Fables.” Pittsburg, Pennsylvania: Carnegie Mellon University, Software Engineering Institute. (September 2016.)

Clarios Technology. “What is a cumulative flow diagram?” (2016.) Retrieved December 31, 2019, from http://www.clariostechnology.com/productivity/blog/whatisacumulativeflowdiagram.

CollabNet and VersionOne. “9th Annual State of Agile Report.” (2015.)

——-. “12th Annual State of Agile Report.” (2018.)

——-. “13th Annual State of Agile Report.” (2019.)

——-. “14th Annual State of Agile Report.” (2020.)

Craddock, Andrew, and others. “The DSDM Agile Project Framework for Scrum.” DSDM Consortium. (2012.)

Dalton, Jeff. *A Guide to Scrum and CMMI*: Improving Agile Performance with CMMI. Pittsburg, Pennsylvania: Capability Maturity Model Integration Institute, January 18, 2017.

Davis, Christopher, W.H. *Agile Metrics in Action*. Shelter Island, New York: Manning Publications Co., 2015.

Defense Science Board. “Design and Acquisition of Software for Defense Systems”. Office of the Secretary of Defense. Washington, D.C: February 14, 2018.

Department of Homeland Security. “Department of Homeland Security Agile Acquisition Software Delivery Core Metrics.” Washington, D.C.: Department of Homeland Security. (May 22, 2017.)

Department of Justice. *DOJ Agile Guidance Document*. Washington, D.C.: November 6, 2015.

Derby, Esther. “Why Not Velocity as an Agile Metric?” (October 18, 2011.) Retrieved February 26, 2018, from https://www.estherderby.com/why-not-velocity-as-an-agile-metric/.

Donovan, Shaun. *Management and Oversight of Federal Information Technology,* Office of Management and Budget. Washington, D.C.: June 10, 2015.

Eisenberg, Robert and Ron Terbush. “Topics on Earned Value Management for Agile Development.” Gaithersburg, Maryland: September 17, 2013.

Federal Acquisition Institute. “Contracting Professionals Smart Guide.” (June 19, 2017.) Retrieved August 28, 2017, from https://www.fai.gov/drupal/resources/contracting-professionals-smart-guide.

——. “Contracting Professionals Smart Guide: Types of Contracts.” Retrieved August 28, 2017, from https://www.fai.gov/resources/contracting-professionals-smart-guide.

——. “Agile Acquisitions 101: The Means Behind the Magic.” April 22, 2015.

Federal Aviation Administration. *Federal Aviation Administration Agile Acquisition Principles and Practices*. Washington, D.C.: April 2016.

Foote, Steve, Justin F. Brunelle, and Tim Rice. *Building Agile Programs*. MITRE’s Software Engineering Technical Center: November 28, 2018.

Garcia, Suzanne, and Richard, Turner. *CMMI® Survival Guide: Just Enough Process Improvement.* Boston, Massachusetts: Pearson Education, Inc., 2007.

Glover, M. Tanner, and Debra Dennie. *How to be Agile with CMMI. CMMI–Agile Process Combo*. LMI Technology: January 27, 2017.

Gorans, Paul, and Philippe Kruchten. “A Guide to Critical Success Factors in Agile Delivery.” Washington, D.C.: IBM Center for the Business of Government. (January 16, 2014.)

Government Accountability Office. *Drive to Deliver Capabilities Faster Increases Importance of Program Knowledge and Consistent Data for Oversight.* GAO-20-439. Washington, D.C.: June 3, 2020.

——. *DHS Has Made Significant Progress in Implementing Leading Practices, but Needs to Take Additional Actions*. GAO-20-213. Washington, D.C.: June 1, 2020.

——. *Cost Estimating and Assessment Guide.* GAO-20-195G. Washington, D.C.: March 12, 2020.

——. *Outcomes Have Improved but Actions Needed to Enhance Oversight of Schedule Goals.* GAO-20-170SP. Washington, D.C.: December 19, 2019.

——. *Comprehensive Planning and Oversight Could Help DOD Acquire Critical Capabilities and Address Challenges.* GAO-20-146. Washington, D.C.: October 30, 2019.

——. *Agencies Need to Develop Modernization Plans for Critical Legacy Systems.* GAO-19-471. Washington, D.C.: June 11, 2019.

——. *Including Users Early and Often in Software Development Could Benefit Programs*. GAO-19-136. Washington, D.C.: March 18, 2019.

——. *Substantial Efforts Needed to Achieve Greater Progress on High-Risk Areas*. GAO-19-157SP. Washington, D.C.: March 6, 2019.

——. *Departments Need to Improve Chief Information Officers' Review and Approval of IT Budgets*. GAO-19-49. Washington, D.C.: November 13, 2018.

——. *Government-wide Actions Needed to Improve Agencies' Use of Performance Information in Decision Making*. GAO-18-609SP. Washington, D.C.: September 5, 2018.

——. *DOD Needs to Take Additional Action to Promote Department-Wide Collaboration.* GAO-18-194. Washington, D.C.: February 28, 2018.

——. *DOD Senior Leadership Has Not Fully Implemented Statutory Requirements to Promote Departmentwide Collaboration.* GAO-18-513. Washington, D.C.: June 25, 2018.

——. *Further Implementation of FITARA Related Recommendations Is Needed to Better Manage Acquisitions and Operations.* GAO-18-234T. Washington, D.C.: November 15, 2017.

——. *Agencies Need to Improve Certification of Incremental Development*. GAO-18-148. Washington, D.C.: November 7, 2017.

——. *Use of Sound Program Management and Oversight Practices Is Needed to Avoid Repeating Past Problems.* GAO-18-46. Washington, D.C.: October 17, 2017.

——. *Further Progress Made in Implementing the GPRA Modernization Act, but Additional Actions Needed to Address Pressing Governance Challenges.* GAO-17-775. Washington, D.C.: September 29, 2017.

——. *U.S. Citizenship and Immigration Services Can Improve Program Management.* GAO-16-467. Washington, D.C.: July 15, 2016.

——. *IRS Needs to Improve Its Processes for Prioritizing and Reporting Performance of Investments.* GAO-16-545. Washington, D.C.: June 29, 2016.

——. *DOD Has Taken Initial Steps to Formulate an Organizational Strategy, but These Efforts Are Not Complete.* GAO-17-523R. Washington, D.C.: June 23, 2017.

——. *Schedule Assessment Guide.* GAO-16-89G. Washington, D.C.: December 22, 2015.

——. *Better Informed Decision Making Needed on Transformation Program.* GAO-15-415. Washington, D.C.: May 18, 2015.

——. *Cost and Schedule Commitments Need to Be Established Earlier.* GAO-15-282. Washington, D.C.: February 26, 2015.

——. *Ineffective Planning and Oversight Practices Underscore the Need for Improved Contract Management.* GAO-14-694. Washington, D.C.: July 31, 2014.

——. *Agencies Need to Establish and Implement Incremental Development Policies.* GAO-14-361. Washington, D.C.: May 8, 2014.

——. *Effective Practices and Federal Challenges in Applying Agile Methods.* GAO-12-681. Washington, D.C.: July 27, 2012.

——. *Critical Factors Underlying Successful Major Acquisition.* GAO-12-7. Washington, D.C.: October 21, 2011.

——. *Action Needed to Improve Administration of the National Flood Insurance Program.* GAO-11-297. Washington, D.C.: June 9, 2011.

——. *Veterans Affairs Can Further Improve Its Development Process for Its New Education Benefits System.* GAO-11-115. Washington, D.C.: December 1, 2010.

——. *Management Improvements Are Essential to VA's Second Effort to Replace Its Outpatient Scheduling System.* GAO-10-579. Washington, D.C.: May 27, 2010.

——. *Concerted Effort Needed to Improve Federal Performance Measures.* GAO-09-617. Washington, D.C.: September 14, 2009.

——. *Lessons Learned for the Next Administration on Using Performance Information to Improve Results.* GAO-08-1026T. Washington, D.C.: June 24, 2008.

——. *Framework for Assessing the Acquisition Function at Federal Agencies.* GAO-05-218G. Washington, D.C.: September 1, 2005.

——. *Implementation Steps to Assist Mergers and Organizational Transformations.* GAO-03-669. Washington, D.C.: July 23, 2003.

——. *Better Acquisition Outcomes Are Possible if DOD Can Apply Lessons From F/A-22 Program.* GAO-03-654T. Washington, D.C.: April 11, 2003.

——. *1997 Government-wide Implementation Will be Uneven.* GGD-97-109. Washington, D.C.: June 2, 1997.

Hayes, Will. "Agile Metrics: Seven Categories." Software Engineering Institute blog. (September 22, 2014.) Retrieved June 2, 2020, from https://insights.sei.cmu.edu/sei_blog/2014/09/agile-metrics-seven-categories.html.

——. *Three Secrets to Successful Agile Metrics*. Software Engineering Institute, November 2017.

Hayes, Will, and others. "Agile Metrics: Progress Monitoring of Agile Contractors." Pittsburg, Pennsylvania: Carnegie Mellon University, Software Engineering Institute. (January 2014.)

Intelliware Development Inc. "7 Myths of Agile Development." Retrieved March 27, 2018, from http://www.intelliware.com/7-myths-agile-development/.

Jaikrishan, Vidarth. "Understanding Cumulative Flow Diagram." Zepel. Retrieved December 31, 2019, from https://zepel.io/agile/reports/cumulative-flow-diagram/.

Jordan, Andy. *Focus on the Right Stuff: Agile Metrics Matter*. CA Technologies, March 20, 2018.

Jordan, Joseph G., and Steven VanRoekel. *Contracting Guidance to Support Modular Development.* Office of Management and Budget. Washington, D.C.: June 14, 2012.

Kan, Stephen, H. *Metrics and Models in Software Quality Engineering.* Upper Saddle River, New Jersey: Pearson Education, Inc., 2003.

Kanban Tool. “Cumulative Flow Diagram.” *Get to know one of the most insightful Kanban metrics*. Retrieved December 31, 2019, from https://kanbantool.com/cumulative-flow-diagram.

Kundra, Vivek. *25 Point Implementation Plan to Reform Federal Information Technology Management*. Office of Management and Budget. Washington, D.C.: December 9, 2010.

Lapham, Mary Ann, and others. “Agile Methods: Selected DOD Management and Acquisition Concerns.” Pittsburg, Pennsylvania: Carnegie Mellon University, Software Engineering Institute. (October 2011.)

Lapham, Mary Ann, and others. “RFP Patterns and Techniques for Successful Agile Contracting.” Pittsburg, Pennsylvania; Carnegie Mellon University, Software Engineering Institute, (November 2016.)

Leffingwell, Dean. *Agile Software Requirements Lean Requirements Practices for Teams, Programs, and the Enterprise*. Boston, Massachusetts: Addison-Wesley, December 27, 2010.

List, Doc. “How to Get Started With Story Points Via Affinity Estimation (And Cheat Sheet).” (June 2, 2016.) Retrieved May 5, 2020, from https://agilevelocity.com/blogget-started-story-points-via-affinity-estimation-cheat-sheet/.

Lorell, Mark A., Julia F. Lowell, Obaid Younossi. *Evolutionary Acquisition Implementation Challenges for Defense Space Programs.* Santa Monica, California: Rand Publishing, August 1, 2006.

Magennis, Troy. *Moneyball for Software Projects: Agile Metrics for the Metrically Challenged*. Agile Alliance, 2014.

Mahmoud, Omar. “Agile Project Management Contorls.” The Barakah Consulting Group. SW and IT Cost IPT Conference. Arlington, VA: August 2015.

Marquis, Hank. “5 Steps to Transparent Metrics.” *Do-IT-Yourself* Guides. itSM Solutions. (April 1, 2008.) Retrieved May 19, 2020, from www.itsmsolutions.com/newsletters/DITYvol4iss14.htm.

McNally, Frank. “Enabling Acquisition Success for Agile Development.” Arlington, Virginia: ASI Government. (March 2014.)

McQuade, Michael J., and others. “Software is Never Done: Refactoring the Acquisition Code for Competitive Advantage.” Washington, D.C.: Department of Defense. (March 14, 2019.)

Measey, Peter. “The Top 10 Myths about Agile Development.” June 2015. Retrieved March 27, 2018, from http://www.computerweekly.com/opinion/The-top-10-myths-about-agile-development.

Miller, Suzanne. “The Readiness & Fit Analysis: Is Your Organization Ready for Agile?” Pittsburg, Pennsylvania: Carnegie Mellon University, Software Engineering Institute. (April 2014.)

——. “Is Your Organization Ready for Agile?–Part 5.” SEI Insights. Carnegie Mellon University. (June 23, 2014.) Retrieved July 15, 2016, from https://insights.sei.cmu.edu/sei_blog/2014/06/is-your-organization-ready-for-agile.html.

——. “Is Your Organizaion Ready for Agile?–Part 6.” *SEI Insights.* Carnegie Mellon University. (January 11, 2015.) Retrieved July 15, 2016, from https://insights.sei.cmu.edu/sei_blog/2015/01/is-your-organization-ready-for-agile-3.html.

——. “Is Your Organization Ready for Agile?–Part 7.” *SEI Insights*. Carnegie Mellon University. (April 25, 2016.) Retrieved July 15, 2016, from https://insights.sei.cmu.edu/sei_blog/2016/04/is-your-organization-ready-for-agile-4.html.

Miller, Suzanne, William Hayes, and Eileen Wrubel. “Agile in Government. Written testimony of Software Engineering Institute’s Agile in Government Team to House Ways and Means SSA Subcommittee.” Pittsburg, Pennsylvania: Carnegie Mellon University, Software Engineering Institute. (July 14, 2016.)

MITRE. “Agile Cost Estimation. Acquisition in the Digital Age.” Retrieved June 10, 2019, from https://aida.mitre.org/agile/agile-cost-estimation/.

Modigliani, Pete, and Su Chang. “Defense Agile Acquisition Guide:Tailoring DOD IT Acquisition Program Structures and Processes to Rapidly Deliver Capabilities.” McLean, Virginia: The MITRE Corporation. (March 2014.)

NASCIO (National Association of State Chief Information Officers) and Accenture Consulting. “Agile IT Delivery: Imperatives for Government Success.” (October 2, 2017.) Retrieved February 26, 2018, from https://www.nascio.org/resource-center/resources/agile-it-delivery-imperatives-for-government-success/.

National Defense Industrial Association (NDIA). “An Industry Practice Guide for Agile on Earned Value Management Programs.” Arlington, Virginia: NDIA. (March 31, 2017.)

——-. “An Industry Practice Guide for Agile on Earned Value Management Programs, Version 1.3.” Arlington, Virginia: NDIA. (May 26, 2019.)

Nicolette, David. *Software Development Metrics*. Shelter Island, New York: Manning Publications, Co., 2015.

Niven, Paul, R. and Ben Lamorte. *Objectives and Key Results: Driving Focus, Alignment and Engagement with OKRs.* Hoboken, New Jersey: Wiley, John & Sons, Inc., 2016.

Norton, Michael. *Agile Metrics: Velocity is Not the Goal*. Agile Alliance, 2013.

Project Management Institute, Inc. *Agile Practice Guide*, 2017.

——-. *A Guide to the Project Management Body of Knowledge (PMBOK® Guide)*, Fifth Edition, 2013. PMBOK is a trademark of the Project Management Institute, Inc.

Potomac Forum, Ltd. *Implementing and Managing Agile Development in Government. AGILE Development in Government Training Workshop IV.* Washington, D.C.: Potomac Forum, January 28, 2015.

Office of the Chief Information Officer; Office of the Chief Technology Officer. “Department of Homeland Security Delivery Metrics Playbook.” Washington, D.C.: Department of Homeland Security. (July 19, 2017.)

Office of Management and Budget. “Capital Programming Guide.” *Supplement to Office of Management and Budget Circular A-11*: Planning, Budgeting, and Acquisition of Capital Assets. Washington, D.C. (2017.)

——-. “Capital Programming Guide.” *Supplement to Office of Management and Budget Circular A-130: Managing Information as a Strategic Resource.* Washington, D.C. (2016.)

——-. “Request for Comments on Digital Services Playbook and TechFAR Handbook.” (August 21, 2014.) Retrieved March 23, 2020, from https://www.federalregister.gov/documents/2014/08/21/2014-19805/request-for-comments-on-digital-services-playbook-and-techfar-handbook.

——-. “TechFAR Handbook for Procurring Digital Services Using Agile Processes.” Washington, D.C.: U.S. Digital Services. (August 7, 2014.)

——-. “Contracting Guidance to Support Modular Development.” Washington, D.C. (June 14, 2012.)

——-. “IT Dashboard: IT Spending FY2011-2021.” Retrieved June 19, 2020, from https://myit-2021.itdashboard.gov/

Oltman, J. “Agile vs. Traditional: An Unncessary War.” *PM World Journal*, vol. II, Issue III. (March 2013.)

Palmquist, Steven M., and others. “Parallel Worlds: Agile and Waterfall Differences and Similarities.” Pittsburg, Pennsylvania: Carnegie Mellon University, Software Engineering Institute. (October 2013.)

Pinot, Avinish, and others. “Federal Aviation Administration Agile Acquisition Principles and Practices.” McLean, Virginia: The MITRE Corporation. (April 2016.)

Performance Assessments and Root Cause Analyses (PARCA). “Agile and Earned Value Management: A Program Manager’s Desk Guide.” Washington, D.C.: Department of Defense. (March 3, 2016.)

Rasmusson, Jonathan. “Agile In a Nutshell*.” Agile Myths*. Retrieved March 27, 2018, from http://www.agilenutshell.com/agile_myths#antiarchitecture.

Reinersten, Donald. *The Principles of Product Development Flow: Second Generation Lean Product Development.* Renoldo Beach, California: Celeritas Publishing, 2009.

——-. *Managing the Design Factory: A Product Developer’s Toolkit*. New York, New York: The Free Press, 1997.

Rodrigues, Alexandre. “Can We Measure Agile Performance with an Evolving Scope? *Agile Product Management & Software Engineering Excellence,* vol. 18, no. 1. Arlington, Massachusetts: Cutter Consortium. (May 22, 2017.)

Rubin, Kenneth, R. *Essential Scrum: A Practical Guide to the Most Popular Agile Process.* Upper Saddle River, New Jersey: October 2015.

Runyon, Tamara Sulaiman. “Agile EVM Information for Good Decision Making.” CollabNet, Inc. (2010.)

Sahota, Michael, and others. “Beyond Budgeting: a Proven Governance System Compatible with Agile Culture.” Agile Alliance: (2015.) Retrieved March 24, 2017 from https://www.agilealliance.org/beyond-budgeting-a-proven-governance-system-compatible-with-agile-culture/.

Scaled Agile. “Overview of the Scaled Agile Framework® for Lean Enterprises.” *SAFe® 4.6 Introduction*. Boulder, Colorado: Scaled Agile. (November 2018.)

——-. “Overview of the Scaled Agile Framework® for Lean Enterprises.” SAFe® 5.0. Boulder, Colorado: Scaled Agile. (December 2019.)

Schwaber, Ken, and Jeff Sutherland. “The Definitive Guide to Scrum: The Rules of the Game.” *The Scrum Guide*. Mountain View, California: Creative Commons. (November 2017.)

Scott, Tony, and Anne E. Rung. *Federal Source Code Policy: Achieving Efficiency, Transparency, and Innovation through Reusable and Open Source Software.* Office of Management and Budget. Washington, D.C.: August 8, 2016.

Section 809 Panel. *Report of the Advisory Panel on Streamlining and Codifying Acquisition Regulations,* vol. 1. Arlington, Virginia: January 2018.

——-. *Report of the Advisory Panel on Streamlining and Codifying Acquisition Regulations,* vol. 2. Arlington, Virginia: June 2018.

——-. *Report of the Advisory Panel on Streamlining and Codifying Acquisition Regulations,* vol. 3. Arlington, Virginia: January 2019.

Sidky, Admed. *Dr. Agile Training Videos*. Retrieved March 1, 2017, from http://www.dragile.com/videos.php#.

Sims, Chris. "Should Management Use Velocity as a Metric?" Agile Learning Labs. (August 27, 2013.) Retrieved February 26, 2018, from http://www.agilelearninglabs.com/2013/08/should-management-use-velocity-as-a-metric/.

Software Engineering Institute. *CMMI® for Development, Version 1.3*. Pittsburgh, Pennsylvania: November 2010.

Solomon, Paul J. "Agile Earned Value and the Technical Baseline" *Managing for Success*. The Data & Analysis Center for Software. (September 2009.)

——-. "Basing Earned Value on Technical Performance." CrossTalk. (January 2013.)

——-. Software Engineering Institute. "Using CMMI to Improve Earned Value Management." *Software Engineering Process Management*. Pittsburg, Pennsylvania: Carnegie Mellon University, Software Engineering Institute. (October 2002.)

——-. *Tutorial: Integrated Systems Engineering with Earned Value Management and Program Management, Contractually and Practically*. NDIA Systems Engineering Conference. Tampa, Florida: October 22, 2018.

Sterling, Chris. “Affinity Estimating: A How To.” Getting Agile. (July 4, 2008.) Retrieved May 4, 2020, from https://www.gettingagile.com/2008/07/04/affinity-estimating-a-how-to/.

Sulaiman, Tamara, Brent Barton, and Thomas Blackburn. “Agile EVM-Earned Value Management in Scrum Projects.” Agile 2006 Conference. Minneapolis, Minnesota: July 2006.

The PMI Agile Community of Practice Wiki. “Glossary.” Retrieved March 5, 2020, from http://agile-pm.pbworks.com/.

The U.S. Digital Service. “Digital Services Playbook.” Retrieved July 25, 2016, from https://playbook.cio.gov/.

The White House. “Fact Sheet: Improving and Simplifying Digital Services.” Office of the Press Secretary. (August 11, 2014.) Retrieved March 23, 2020, from https://obamawhitehouse.archives.gov/the-press-office/2014/08/11/fact-sheet-improving-and-simplifying-digital-services.

Van Doorem, Wouter, Geert Bouckaert, and John Halligan. *Performance Management in the Public Sector*. New York, New York: Routledge, 2010.

Wrubel, Eileen, and Jon Gross. “Contracting for Agile Software Development in the Department of Defense: An Introduction.” Pittsburg, Pennsylvania: Software Engineering Institute, Carnegie Mellon University. (August 2015.)