Software Engineering Institute

Agile Software Teams: How They Engage with Systems Engineering on DoD Acquisition Programs

Eileen Wrubel
Suzanne Miller
Mary Ann Lapham
Timothy A. Chick

Contributing Authors from the Agile Collaboration Group:

Deborah Brey	Portia Crowe
Kenneth Nidiffer	Jennifer C. Walker
Robert W. Boardman	Philip Matuzic
Richard Carlson	Cynthia Molin

July 2014

TECHNICAL NOTE
CMU/SEI-2014-TN-013

Software Solutions Division

http://www.sei.cmu.edu

Carnegie Mellon University

Table of Contents

List of Figures

List of Tables

Acknowledgments

The Software Engineering Institute (SEI) author team would like to express our appreciation to all those who took time out of their busy schedules to complete our research surveys or allow us to interview them. Their contributions were invaluable to this process.

Thanks also go to our Agile Collaboration Group members, with whom we socialized and shaped this project, and who provided excellent discussion and feedback on our approach and findings.

For the first time, we reached out to members of our Collaboration Group to participate as contributing authors. Sincerest gratitude goes to these systems engineers and Agile practitioners, who synthesized their extensive experience with the results of our research effort:

- Deborah Brey, Boeing Defense Systems
- Phil Matuzic, Senior Scientist, Boeing Space and Intelligence Systems
- Dick Carlson, Agile and Lean Education Associates
- Jennifer C. Walker, Raytheon Missile Systems
- Robert W. Boardman, Raytheon Integrated Defense Systems
- Cynthia (Cindy) Molin, Raytheon Missile Systems
- Portia Crowe, U.S. Army, Program Executive Office C3T

Many of our community members provided us with data, content, editorial review, and thoughtful insight. We extend special thanks to the following people:

- Rob Frisch, AFMC, 578 SMXS
- Jack Supp, NAVAIR
- Tom Tschuor, Level III PM, PCI Strategic Management
- Carmen S. Graver, CSM, PMI-ACP, Marine Corps Systems Command
- Matthew R. Kennedy, PhD, Defense Acquisition University
- John McLoughlin, Lockheed Martin
- David R. Webb, Senior Technical Program Manager, 309th Software Maintenance Group, Hill AFB, Utah
- Kenneth Nidiffer, Dr. Sc., SEI
- John Robert, SEI
- Sarah Sheard, PhD, SEI

Finally, we would be remiss if we did not recognize the contributions of our unfailing and patient editor, Gerald Miller.

Executive Summary

In 2009, the Air Force tasked the Software Engineering Institute (SEI) to assess the state of the practice of Agile development in government acquisitions. Our original report, *Considerations for Using Agile in DoD Acquisition*, debunked the myth that Agile practices are incompatible with Department of Defense (DoD) acquisition policy and practices [Lapham 2010]. Through the next several years we continued our exploration of using Agile in regulated settings, such as the DoD. We have delivered papers, presentations, colloquia, e-learning courses, and program consulting on various aspects of adopting Agile in DoD and other regulated settings.

In 2013, William Broadus, a professor at Defense Acquisition University (DAU), agreed with our assessment when he stated, "There are no direct policy or practice issues that would preclude or limit the use of Agile methods within the DoD" [Broadus 2013].

As operational tempos increase and programs fight to drive down the length of time required to field capability to the warfighter, more and more members of the DoD acquisition community are turning their attention toward Agile or other iterative development methods. However, the DoD 5000 series and associated guidance still present a system-oriented perspective on acquisition, but leave unaddressed the next step regarding how to leverage iterative software development methods within the greater context of the program's systems engineering [Kennedy 2011]. This gap is not exclusive to Agile, however. In 2006, an NDIA task group reported that a key issue in software intensive systems acquisition was that "fundamental system[s] engineering decisions are made without full participation of software engineering" [NDIA 2006]. Its 2010 follow-up report, *Top Software Engineering Issues Within Department of Defense and Defense Industry,* revisited the issue and concluded that DoD policies had been amended to "imply software engagement, but are not explicit" [NDIA 2010].

The purpose of this report is to identify interactions that are taking place on DoD programs between Agile software development teams and their systems engineering counterparts in the development of software-reliant systems. We do not champion a particular development method, but rather explore the ways in which Agile software development teams are engaging systems engineers and associated stakeholders, to identify factors that will help the DoD benefit from Agile methods and barriers to achieving those benefits.

To explore these issues, we conducted surveys and interviews with Agile practitioners on a variety of DoD programs. We also conducted a literature search to gain a broader understanding of these issues in the commercial sector. Every acquisition category (ACAT) was represented, with programs from business and IT systems to weapon systems, and personnel from contract developers to organic DoD capabilities.

We envisioned three different approaches to systems engineering interacting with or being a part of Agile teams:

1. Agile software teams interacting with *traditional systems engineering*
2. systems engineers acting *as Agile team members*
3. systems engineers *applying Agile methods to their own work*

Our interviews uncovered active instances of each of these approaches and documented the features of these interactions that produced successful collaboration among software, systems engineering, and program office teams. Respondents were frank about challenges, growing pains, and factors that enabled success and created goodwill and trust among stakeholders. We present case-based models of each of the three interaction approaches and then delve into the patterns associated with challenges and successes across our interviews.

Agile software development is not a silver bullet to fix what ails DoD acquisitions, and it may not be appropriate in every situation. Our objective is to demonstrate what is happening with real Agile practitioners *today* as they engage with systems engineers on real programs. We hope to give acquisition decision makers at any level insight that enables them to assess the environment associated with their own programs and make determinations about whether Agile approaches may be appropriate and viable, as well as a better understanding of what it will take to enable success with them.

Abstract

This technical note (TN), part of an ongoing Software Engineering Institute (SEI) series on Agile in the Department of Defense (DoD), addresses key issues that occur when Agile software teams engage with systems engineering functions in the development and acquisition of software-reliant systems. Published acquisition guidance still largely focuses on a system perspective, and fundamental differences exist between systems engineering and software engineering approaches. Those differences are compounded when Agile becomes a part of the mix, rather than adhering to more traditional "waterfall"-based development lifecycles. For this TN, the SEI gathered more data from users of Agile methods in the DoD and delved deeper into the existing body of knowledge about Agile and systems engineering before addressing them. Topics considered here include various interaction models for integrating systems engineering functions with Agile engineering teams, automation, insight and oversight, training, the role of Agile advocates/sponsors and coaches, the use of pilot programs, stakeholder involvement, requirements evolution, verification and validation activities, and the means by which Agile teams align their increments with program milestones. This TN offers insight into how systems engineers and Agile software engineers can better collaborate when taking advantage of Agile as they deliver incremental mission capability.

1 Introduction

The Software Engineering Institute (SEI) is conducting a multiyear investigation into the adoption of Agile methods in Department of Defense (DoD) acquisition programs and other regulated settings. This report addresses some of the key issues that occur when Agile software teams engage with systems engineering functions in the development and acquisition of software-reliant systems.

Previous technical notes on related Agile topics are available and include

- *Considerations for Using Agile in DoD Acquisition* (CMU/SEI 2010-TN-002)
- *Agile Methods: Selected DoD Management and Acquisition Concerns* (CMU/SEI-2011-TN-002)
- *A Closer Look at 804: A Summary of Considerations for DoD Program Managers* (CMU/SEI-2011-SR-015)
- *DoD Information Assurance and Agile: Challenges and Recommendations Gathered Through Interviews with Agile Program Managers and DoD Accreditation Reviewers* (CMU/SEI 2012-TN-024)
- *Parallel Worlds: Agile and Waterfall Differences and Similarities* (CMU/SEI-2013-TN-021)
- *Agile Methods and Request for Change (RFC): Observations from DoD Acquisition Programs* (CMU/SEI-2012-023) (in security and policy review)
- *Selected DoD Acquisitions: Requirements Development and Management* (in management review)

  In addition, several podcasts and blog posts from the SEI have addressed additional topics. They can all be found on the Acquisition Research page of the SEI website.[1]

1.1 Why Focus on Agile and Systems Engineering?

The DoD 5000 series and associated guidance present a systems engineering-oriented perspective on acquisition and development, yet there is little guidance available about leveraging iterative or Agile methods in the context of *systems engineering.* Despite such measures as the congressional requirement via Section 804 of the National Defense Authorization Act for Fiscal Year 2010 that the secretary of defense establish "a new acquisition process for IT systems" [NDAA 2010], which yielded the emergent Agile IT Acquisition Lifecycle and the Business Capability Lifecycle (BCL) [DoD 2011a, DoD 2011b], systems engineering guidance is startlingly bare on the topic of leveraging Agile software approaches [Kennedy 2011].

Kennedy illustrates the problem as shown in Figure 1.

1 http://www.sei.cmu.edu/acquisition/research

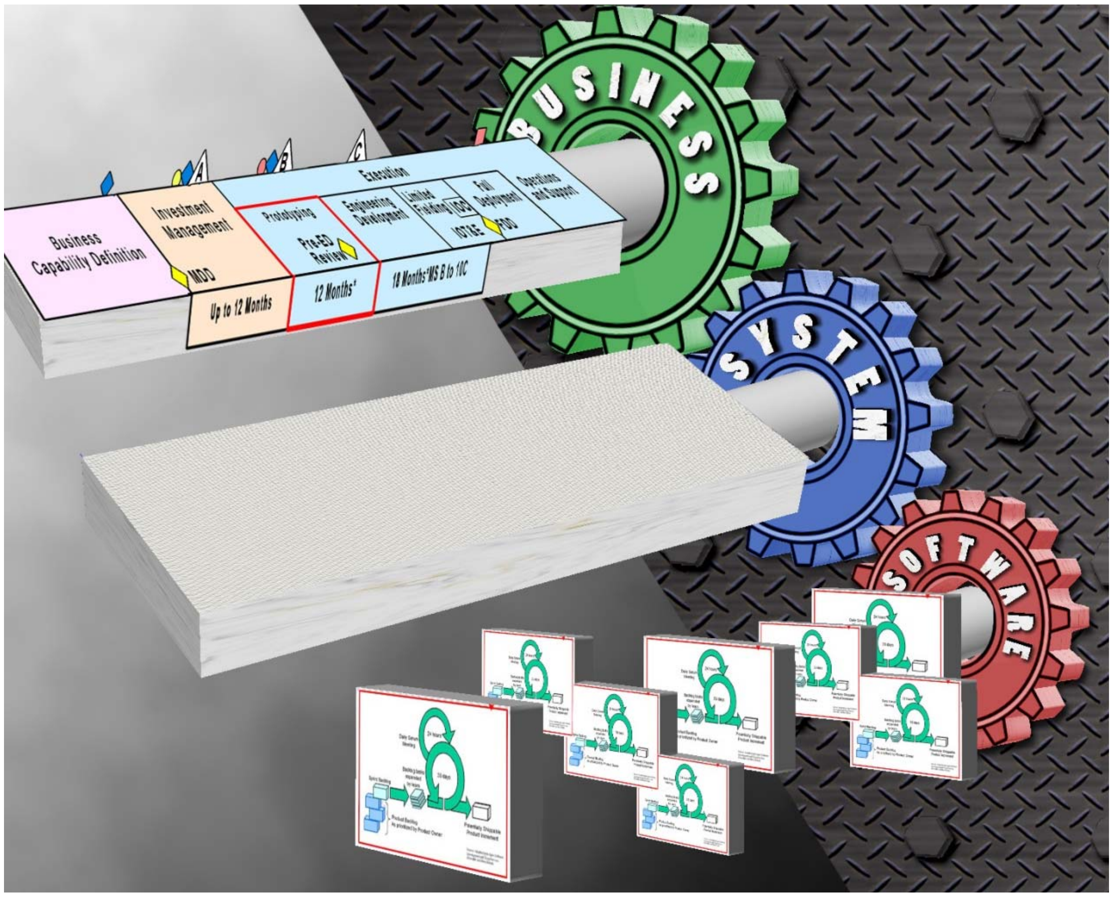

Figure 1: Business, System, and Software Agility: Missing System Aspect Framework [Kennedy 2013]

The "Business" gear demonstrates an incremental acquisition approach under the BCL, and the "Software" gear illustrates the use of Scrum or other iterative methods for developing software. The "System" gear, which links the developed software to the overall mission capability sought by the acquisition, is blank.

Our research efforts have thus turned toward understanding how Agile software development teams are interacting with systems engineering functions in the development and sustainment of DoD programs, given the dearth of guidance available regarding systems engineering. Agile practitioners, systems engineering teams with which they interface, and the program managers who execute programs have to "make up" the interactions as they go. This technical note reveals several observable interaction patterns, and the opportunities and challenges presented by employing Agile without solid system-level guidance for leveraging it.

1.2 Differences Between Software and Systems Engineering Approaches

The systems engineer has to perform activities across multiple engineering domains, and those domains are linked together by systems thinking and theory based on physical laws and the culture of systems. As a result, these domains often exhibit a common systems engineering architectural decomposition approach with respect to their workflow model and development models. In general, engineers for those domains perform their roles based on engineering processes, tools, and techniques that followed a functional decomposition approach.

Software engineers—based on the essential characteristics of software (see Section 1.2.2) and the use of software to interface, interconnect, and/or control components at different levels of the work breakdown structure (WBS)—do use architectural frameworks that facilitate functional decomposition ("is a part of" engineering approach) but also employ other types. For example, a software engineer uses architecture approaches that facilitate "used by" and "control by" constructs to handle software functional, nonfunctional, and interface requirements. In addition, software should adhere to but is not constrained by physical laws. As Brooks pointed out, "software is unlike other forms of engineering as other forms of engineering are like unto themselves" [Brooks 1975].

1.2.1 A Source of Tension on DoD Programs

Tension and disconnects between software and systems engineering functions are not new. Grady Campbell wrote in 2004 that "Systems engineering and software engineering need to overcome a conceptual incompatibility (physical versus informational views of a system)" and that systems engineering decisions can create or contribute to software risk if they "prematurely over-constrain software engineering choices" or "inadequately communicate information, including unknowns and uncertainties, needed for effective software engineering" [Campbell 2004]. In 2006, a National Defense Industrial Association (NDIA) task group reported that a fundamental issue in software-intensive systems acquisition was that "fundamental systems engineering decisions are made without full participation of software engineering" [NDIA 2006]. Their 2010 follow-up report, *Top Software Engineering Issues Within Department of Defense and Defense Industry*, revisited the issue and concluded that DoD policies had been amended to "*imply* software engagement, but are not explicit" [NDIA 2010] (emphasis added). A 2013 case study on agility and traditional systems engineering in large defense projects from around the world found that a primary cause of post-design requirements change was "keeping the software and hardware engineers … out of the decision making process" [Asan 2013]. As of this writing, the SEI currently has DoD-sponsored work in pre-publication review[2] addressing the problem of the tendency among many complex software-reliant systems acquisition programs to treat software as a 'specialty engineering' function.

Historically in DoD programs, the system is decomposed from the system level down to subsystem behavior, and a work breakdown structure is developed for the program based on this decomposition. Hardware-focused views are not appropriate for software, and some systems engineers, and most systems engineering standards, have not yet adopted an integrated view of these historical views and more recent information views. Thus software has typically been constrained within this WBS structure, even though a software-based decomposition of the system may not be well aligned with the hardware decomposition. Software engineers translate systems requirements to software requirements to logical design (architecture) in a significantly different way than other types of engineers translate systems requirements in their domains. In complex software-reliant systems, software components often interact with multiple hardware components at different levels of the system architecture. This traversing across physical architectural elements is less common in other engineering domains like mechanical engineering. An important difference between software engineering and other kinds of engineering (e.g., electrical, mechanical, fluid) is that

2 Korzec, Keith & Merendino, Thomas. *The Fallacy of Treating Software as a Specialty Engineering Discipline* (in pre-publication review).

software interfaces cross multiple levels of the decomposition, whereas hardware engineering tends to be more separable. The approach needed to architect effective software (crossing those multiple levels of decomposition) often causes confusion and communication disconnects among other types of engineering and program management functions on a program.

1.2.2 Essential Properties of Software

There is often a fundamental communication barrier between software engineering and other forms of engineering because the essential properties of software differentiate it from other kinds of engineering domains [Brooks 1975]:

- software complexity—Software, for its size, is more complex for the effort and the expense to develop it than other types of engineered components. Stated another way, on large projects, as the size of the software grows, software econometric research indicates that the expense and effort to develop it grow significantly (see also Reifer) [Reifer 2013].
- software malleability—Software is the most easily changed element in a system since no "remanufacturing" of completed hardware components is required. As a result, 40 to 60 percent of the cost for the software acquisition lifecycle is in sustainment and evolution.
- software invisibility—Software has no physical properties and cannot be physically observed independently of the hardware on which it runs.
- software conformity—Software must conform to exacting specifications in the representation of each part, in the interfaces to other internal parts, and in the connections to the environment in which it operates. This makes software different from other disciplines that have tolerances, generally resulting in software engineering functions subdividing their work into small chunks or computer software units that are worked on by small teams.

These essential properties of software cause software engineers to approach requirements development and design differently from other forms of engineering. Rather than tying the derivation of their requirements to a hardware component in a one-to-one fashion, they look across the hardware components for common interfaces, data management protocols, algorithms, and other mechanisms that make the interaction among hardware components more effective. So the software architecture rarely provides a one-to-one mapping from software component to hardware component.

As a result, the different approaches used by software and other disciplines lead to communication issues and system decomposition conflicts independent of the use of Agile methods. Additional confusion arises because Agile methods are new enough to many DoD environments that the culture change they embody has not proceeded very far [Lapham 2011].

1.3 Envisioning Systems Engineering Coexisting with Agile

Two key facets of systems engineering in software-reliant systems have been identified that help us to understand why systems engineering is an important player in programs adopting Agile methods:

- the **product** side of systems engineering: Systems engineering has a key role in *transforming the artifacts* that communicate the intent of the system as understanding of the system evolves.
- the **service** side of systems engineering: Systems engineering has an equally important role in *communicating and coordinating important information* about the evolving knowledge of the system among the many stakeholders, including technical staff, end users, and management. Systems engineers also have a strong conflict resolution role when inevitable technical and programmatic conflicts arise among stakeholders [Garcia-Miller 2010].

When we separately analyze these two sides of systems engineering, different possibilities of how the systems engineering community might take advantage of Agile methods emerge.

On the product side, the incremental, iterative approach with heavy user involvement common to all Agile methods could be leveraged to increase the speed of development of key requirement and design artifacts that are needed to implement different mission or system threads. Some methods like acceptance test-driven development could be incorporated into the activities of systems engineering to increase the connection between the two sides of the typical systems engineering "V" lifecycle [Hendrickson 2008].

On the service side, at the scale of a program that requires a separate systems engineering function, the coordination, communication, and conflict resolution services that systems engineering provides could translate into a product owner surrogate role, a Scrum of Scrums facilitator role, or other specialty roles that show up in scaling approaches such as the Scaled Agile Framework (SAFe) [Leffingwell 2007].

While we inquired of Agile practitioners in the DoD regarding both the product and service facets of systems engineering, it was the service facet—communicating and coordinating important information—that evoked the majority of responses regarding opportunities for, or barriers to, positive and successful interactions between software Agile practitioners and systems engineering functions. We will discuss patterns of interaction as observed in both the product and service facets in our analysis of observations in Section 3.

1.4 Software and Systems Engineering Interactions

We envisioned three different approaches to systems engineering interacting with or being a part of Agile teams:

1. Agile software teams interacting with *traditional systems engineering*
2. systems engineers acting *as Agile team members*
3. systems engineers *applying Agile methods to their own work*

We discovered programs with interactions displayed in each of these paradigms, and we will discuss these approaches and real-life acquisition success stories that display them. Multiple practitioners described their engineering teams or acquisition programs moving along these approaches as a continuum. As software teams demonstrated or continued to demonstrate success with Agile methods, systems engineering teams and leaders got engaged with the software processes. As those activities were successful, leaders and managers might consider applying some Agile methods to part of the systems engineering process.

This technical note will also briefly describe what we envision as a potential next position along the continuum, which involves scaling Agile efforts across the organization, using models such as SAFe.

1.5 Methods

We conducted a literature search, surveys, and interviews with Agile practitioners across the DoD and asked detailed questions about Agile software team interfaces with systems engineering. We included questions about the nature of the interactions between teams, relative to the three cases noted above, as well as questions about the challenges and opportunities they faced in being Agile in the highly regulated, document-driven DoD acquisition environment.

A detailed description of the research approaches and the program and respondent demographics is discussed in Section 7.

We also, for the first time, engaged contributing authors from our Agile Collaboration Group to support the development of this technical note. Our contributing author team featured personnel from both DoD offices and contractor development organizations, whose experiences spanned programs of varying types and sizes with a variety of government customers.

1.6 Audience for This Technical Note

The audiences for this report are

- senior DoD acquisition decision and policy makers, to advise them on the practicality and viability of encouraging the employment of Agile in their programs
- members of DoD program offices who may be challenged to undertake a software development acquisition with a contractor who will be using Agile
- software development teams (both government and contractor) who are using or contemplating the use of Agile to execute software development on a DoD program
- systems engineers (both government and contractor) who will be engaging with Agile software development teams in the execution of system development on a DoD program

This report is not intended to provide tutelage on basic Agile terms and definitions. See the technical note *Parallel Worlds: Agile and Waterfall Differences and Similarities* for basic information about Agile terms and practices [Palmquist 2013].

1.7 Organization of This Technical Note

Section 2 describes the three interaction cases in more detail and documents examples of each case that we encountered in practice. It also devotes attention to a brief discussion of enterprise Agile models such as SAFe.

Section 3 documents our interview and survey findings. We identify patterns of successes and challenges experienced by Agile software teams in their interactions with systems engineering functions.

Section 4 provides additional discussion on the role of policy and formal guidance, and their interpretation, in the success of Agile software efforts. A case study is included.

Section 5 briefly explores the implications of contract type and structure on Agile software projects.

Section 6 provides a retrospective in the words of our interview respondents.

Section 7 describes the research approach in detail, including the response demographics of the programs represented in our data.

Section 8 provides an overall summary of this technical note.

2 Systems Engineering Interactions with Agile Software Development

In analyzing data from our interviews, the following three cases represent the types of interaction between systems engineering functions and Agile software development teams, as discussed in Section 1:

- Agile software teams interacting with traditional systems engineering
- systems engineers acting as Agile team members
- systems engineers applying Agile methods to their own work

We know that the integration of Agile can scale across the enterprise under some newer, promising model frameworks, and so we also describe systems and software engineers working within a scaled Agile framework such as SAFe.

Throughout the interviews, multiple respondents described a journey of projects and programs traveling along a continuum represented by these cases.

The following subsections describe some of the observations from the literature survey and the interviews, as well as experiences from our author team, in relation to each of these cases. Each subsection provides a case description from a program described by our interview respondents or professional collaborators. These cases contain varying levels of detail as reported by the practitioners regarding execution and challenges.

2.1 Agile Software Teams Interacting with Traditional Systems Engineering

Several subcases were observed among Agile software teams that were interacting with traditional systems engineering.

First, there was the subcase where an Agile software team translated its outputs and project management artifacts into the forms expected by the systems engineering team, essentially doing "covert Agile" without the knowledge or explicit buy-in of the systems engineers. These interviewees saw cost, schedule, and quality improvement benefits from using Agile methods. However, they also saw a higher overhead to translate their results into forms expected by the systems engineering function, rendering the benefits, in terms of cost and schedule, not as high as might otherwise have been achieved in a more open Agile environment. In at least one case, however, the interviewee believed that the improved end user interaction was appreciated by systems engineers as well as the early timing and speed with which testable increments of software were provided by the software team, even as the team covertly practiced Agile.

Second, there was the subcase where an Agile software team negotiated its deliverables with the systems engineering team in such a way as to permit "deliveries" to the systems engineering team after several iterations (an Agile release). In this case, the systems engineering team knew the software team was using Agile methods, and accepted its methodology, but did not participate actively in it as team members.

One respondent reported functioning as a translator between Agile software teams and a systems engineering function housed in a government program office. The example in the next section describes the efforts reported by one interviewee engaged in covert Agile: the software team was engaged in Agile, but the systems engineering function behaved according to a very traditional paradigm.

Example: Interacting with Traditional Systems Engineering: Covert Agile

Program Z is involved in acquiring both new IT systems and evolving legacy systems. They are non-ACAT systems. The two Agile projects discussed here involve one legacy rehosting onto a different platform with little change in the functional requirements and one new project involving mobile applications that is not yet a program of record. The acquisition mechanism is to use an organic government organization for the system development. The only part of the developing organization using Agile is the software development team. A specific cost center on the task order is used to bill hours related to story point implementation. The new system team is small, with only three developers. The rehosting project has nine software staff working in three-week sprints. The software teams follow Scrum practices with fairly high fidelity, including all members of the team participating in all activities to get the system ready for demonstration, including documentation and testing. "Hit by a bus"[3] documentation is the priority for documentation, and squadron-level required documentation is produced, as well as customer-stipulated documentation. In all cases, the team tries to produce document artifacts as much as possible directly from work in process artifacts. They have also automated as much of the testing, integration, and peer review activities as possible. The Agile advocate and development manager translates from Agile activities and outcomes to traditional acquisition activities for the systems program office (SPO), which is the seat of systems engineering activities. The use of Scrum for the software development team was approved by the squadron director when it was briefed to him after an initial pilot success. The Agile advocate had used Agile methods successfully in industry and had a prior good relationship with the squadron director, which promoted some trust.

We call this example "covert" mode because the Agile advocate doesn't advertise to the SPO that the practices he is using or encouraging them to use come from Agile methods. He just suggested ideas for development team's process—"How about a product demonstration every three weeks so you can judge progress?" "How about prioritizing our requirements backlog so we're working on the most important things for the project?" "Can you answer some questions our developers have?" Over time, the SPO has started becoming more of a product owner without being called out in that role explicitly, and they have a positive perception of the practices being used, especially the iteration demonstrations every three weeks. Development team members have also become more interactive with the end users so that they have a better sense of prioritization of stakeholder needs.

Once a release goes into certification and accreditation activities (C&A), this team's primary interaction with it is to perform defect fixes, for which time is budgeted in their three-week iterations. C&A and other mandatory external testing have not been brought in to the Agile activity stream, which necessitates the translation function by the advocate.

[3] "Hit by a bus" frequently refers to the minimum level of documentation required to carry on the technical work if the current responsible party were suddenly and permanently no longer available to the team.

Key success factors in this example are

- the facility of the Agile advocate to translate from Agile terminology to traditional acquisition terminology
- the early production of working prototypes that have allowed the SPO to make early course corrections
- the smallness of the teams
- the support of the squadron director
- the closeness of the SPO to the operational users that it supports [Asan 2013]

2.2 Systems Engineers Acting as Agile Team Members

In the cases where systems engineers acted as Agile team members, the primary roles that we observed they took were either a product owner role or a systems architect role. In the case where the systems engineering team assigned someone as a product owner, there were variations in how that role was performed. In some cases, the systems engineer acted as a surrogate for the operational user (particularly where the operational user was unavailable, or the operational users reflected a large, diverse population). In others, system engineers assumed more of a product management team lead role, where they convened a team of end users or end user representatives, then synthesized their prioritizations for the requirements backlog, and ensured that questions were answered that were asked by the software development team.

A significant challenge noted by interviewees in incorporating a systems engineering team member as a product owner was availability. Systems engineers have their own set of artifacts and communications in addition to interacting with the software team, and rarely could a systems engineer be dedicated full time as a product owner to an Agile software team. In at least one case, as the development progressed, the systems engineers became more engaged as they saw the positive benefits (primarily around interface definition and anticipating architectural issues) that *they* accrued when interacting actively with the Agile team. A systems engineer interacting actively with the Agile software team as a systems (and sometimes software) architect also generally had the issue of availability. In this case, defining the kinds of questions the architect would be expected to answer and getting agreement on turnaround times was one way we have seen that made a part-time architect a more productive Agile team member.

Example: Systems Engineers Participating in Agile Software Development

Several programs of varying sizes within the U.S. Marine Corps (USMC) are engaged in Agile software development. The SEI team was fortunate to witness a briefing (approved for public release) on the USMC's pilot activities in this area, which demonstrated how the Agile process melded with traditional systems engineering activities and reviews. Graphic representations from that briefing are reproduced here with permission. They demonstrate an active working example of our second case: traditional systems engineering functions engaging as an Agile software engineering team member.

The USMC teams' disciplined Agile process includes planning and reviews at the sprint and release levels, aligned with the objectives of reviews in the traditional systems engineering process. Table 1 provides some definition of the Agile reviews presented in Figure 3, Figure 4, and Figure

5, mapping the Agile reviews to their analogs in the systems engineering process. Figure 2 demonstrates supporting documentation that is developed by the Agile teams, and the traceability provided throughout the development process. Figure 3, Figure 4, and Figure 5 collectively demonstrate how Agile sprints and releases deliver capability in a given increment of the program, and how they interact with the systems engineering process.

Table 1: Agile Reviews and Traditional Reviews[4]

Technical Reviews in the Agile Process	**Traditional Analogous Systems Engineering Technical Review**
Initial Release Planning Review (IRPR) - Focused on Initial Release and corresponding Sprints Infrastructure Review (IR) - Proposed HW Infrastructure - Estimated Virtualized Resource Pool	Systems Requirements Review 2 (SRR2) (See Figure 4) Systems Functional Review (SFR) (Incremental PDRs will be conducted at the Sprint levels)
Release Planning Reviews (RPR) - Oversight will be delegated to the Agile Review Board - Focused on follow-on Release and corresponding Sprints	Systems Functional Review (SFR) - Subsequent Release SFR
Sprint Planning/Reviews[5]	Sprint Preliminary Design Review (S)PDR** **Incrementally conducted with each Sprint
Daily Build/Test/Integration Sprint Demonstration Review*** *** Completed products are demonstrated to the product owner	Critical Design Review (CDR) N/A
Release Demonstration	Integration Readiness Review (IRR) Test Readiness Review (TRR)
Sprint and Release Retrospectives - Assessment opportunity to determine what went well and what did not for Sprint/Releases	Continuous Process Improvement (CPI)
Systems Verification Review (SVR)	Systems Verification Review (SVR)
Operational Test Readiness Review (OTRR)	Operational Test Readiness Review (OTRR)

The program's capability development document and system subsystem specification data are used by the team to develop the product backlog (Figure 4), which is a collection of requirements and features desired in the end product, maintained in both list and story form. The product backlog feeds release backlogs, which in turn feed the sprint backlogs shown in Figure 5. The grouping of requirements and stories into releases is carried out by assessing factors such as risk, size and complexity, and dependencies on other aspects of system development (e.g., the availability of test harnesses or interfaces owned by a third party). Sprint-level backlogs are developed from the release backlogs during sprint planning for each sprint. The teams manage the evolution, refinement, and development of requirements through the use of the program's requirements tracea-

[4] Adapted from Graver, Carmen & Greeley, Les. United States Marine Corps Agile Pilot Program Lessons Learned (MC-Agile). Briefing. February 2013. Unpublished.

[5] Sprint planning meetings occur at the *beginning* of the sprint for purposes of defining "what done means" for that sprint. Sprint reviews occur at the end of the sprint, to assess the progress against the agreed-upon parameters of the sprint: "did it get done?"

bility matrix (RTM). The design documentation (including decisions and rationale) and test plans produced in each sprint also feed the RTM (Figure 2). Thus the RTM gives all stakeholders continuous visibility into the evolution of requirements, their implementation status, and associated tests, demonstrating the linkages between the developed software, tests, system functions, and mission needs. Although an RTM is not a traditional Agile communication vehicle, it is one of the accommodations to the traditional acquisition process that is useful to the larger team.

The USMC Agile pilot team emphasized that all disciplines must be involved early and up front, sharing a common program vision. They use the concept of a "sprint zero," or a planning sprint (Figure 2). In sprint zero, the software team, systems engineers, and other stakeholders engage in infrastructure planning and the development of the product backlog. The systems engineering plan (SEP) and the test and evaluation management plan (TEMP) (documents from the traditional systems engineering process) provide inputs into this process and the development of the RTM. Sprint zero is also used to develop an Agile "annex" to the systems engineering management plan (SEMP). The SEMP itself may be updated at each sprint.

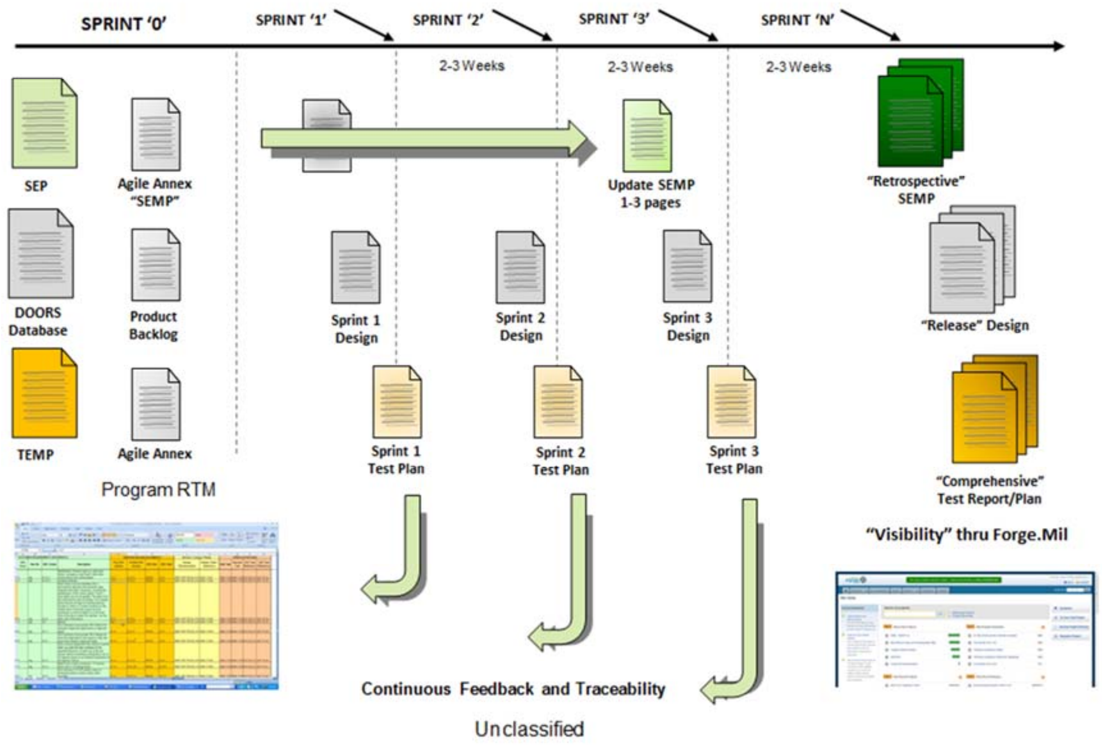

Figure 2: Supporting Documentation and Traceability[6]

Figure 3 demonstrates Agile sprint planning and execution, with the systems engineering organization's technical review process overlaid on top of the Agile sprints. The system requirements review (SRR), system functional review (SFR), and infrastructure review (IR) are combined into a release review conducted by the Technical Review Board. The software development process in-

6 Graver, Carmen & Greeley, Les. United States Marine Corps Agile Pilot Program Lessons Learned (MC-Agile). Briefing. February 2013. Unpublished.

cludes incremental PDRs conducted at the start of each sprint (an approach we discuss at length in *Agile Methods and Request for Change (RFC): Observations from DoD Acquisition Programs*[7]). The Agile Review Board (a subset of the Technical Review Board) approves user stories (especially acceptance criteria) and architecture for each sprint. The intent of CDRs is met with daily builds, demos, and other practices in each sprint, but formal CDRs are not conducted. These incremental approaches to design review are a means to align Agile software engineering with the program's PDR and CDR milestones. Systems engineering representatives take part in the sprint planning and reviews and software demonstrations at the end of each sprint; all reviews are "collaborative conversations."[8]

A specified group of completed sprints together constitutes a release, and multiple specified releases constitute a delivered increment. Figure 4 shows how the sprint-level PDRs fit in to the overall development of an increment, and Figure 5 illustrates sprint-level PDRs taking place within the context of a release. At the end of a sprint, a sprint memo (seen in the lower half of Figure 5) is produced that summarizes the contents of the sprint against the original plan. Sprint planning constitutes agreement among the parties on "the definition of done"; the sprint memo assesses the sprint against those criteria. Systems engineers thus have constant visibility into the state of software development. As the graphics show, release-level planning guides the more granular planning activities and execution of the sprints that comprise the release. Likewise, what is developed and discovered in each sprint can produce updates to not only the activities of follow-on sprints, but to the release plans for the current release and to the product and release backlogs.

7 Lapham, Mary Ann, Michael Bandor, and Eileen Wrubel. *Agile Methods and Request for Change (RFC): Observations from DOD Acquisition Programs* (CMU/SEI-2013-TN-031). http://resources.sei.cmu.edu/library/asset-view.cfm?assetid=77732

8 Graver, Carmen & Greeley, Les. United States Marine Corps Agile Pilot Program Lessons Learned (MC-Agile). Briefing. February 2013. Unpublished.

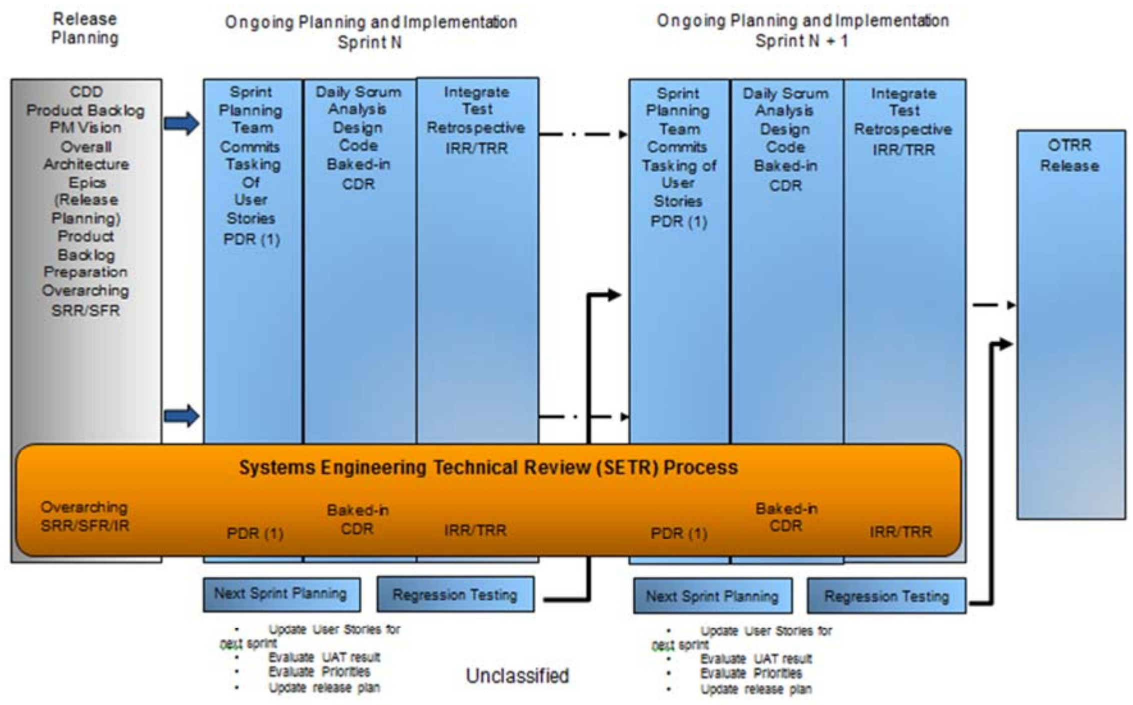
Release Planning
Ongoing Planning and Implementation Sprint N
Ongoing Planning and Implementation Sprint N + 1
CDD Product Backlog PM Vision Overall Architecture Epics (Release Planning) Product Backlog Preparation Overarching SRR/SFR
Sprint Planning Team Commits Tasking Of User Stories PDR (1)
Daily Scrum Analysis Design Code Baked-in CDR
Integrate Test Retrospective IRR/TRR
Sprint Planning Team Commits Tasking of User Stories PDR (1)
Daily Scrum Analysis Design Code Baked-in CDR
Integrate Test Retrospective IRR/TRR
OTRR Release
Systems Engineering Technical Review (SETR) Process
Overarching SRR/SFR/IR
PDR (1)
Baked-in CDR
IRR/TRR
PDR (1)
Baked-in CDR
IRR/TRR
Next Sprint Planning
Regression Testing
Next Sprint Planning
Regression Testing
Update User Stories for next sprint
Evaluate UAT result
Evaluate Priorities
Update release plan
Unclassified
Update User Stories for next sprint
Evaluate UAT result
Evaluate Priorities
Update release plan

Figure 3: Marine Corps (MC)-Agile Increment 1[9]

[9] Ibid

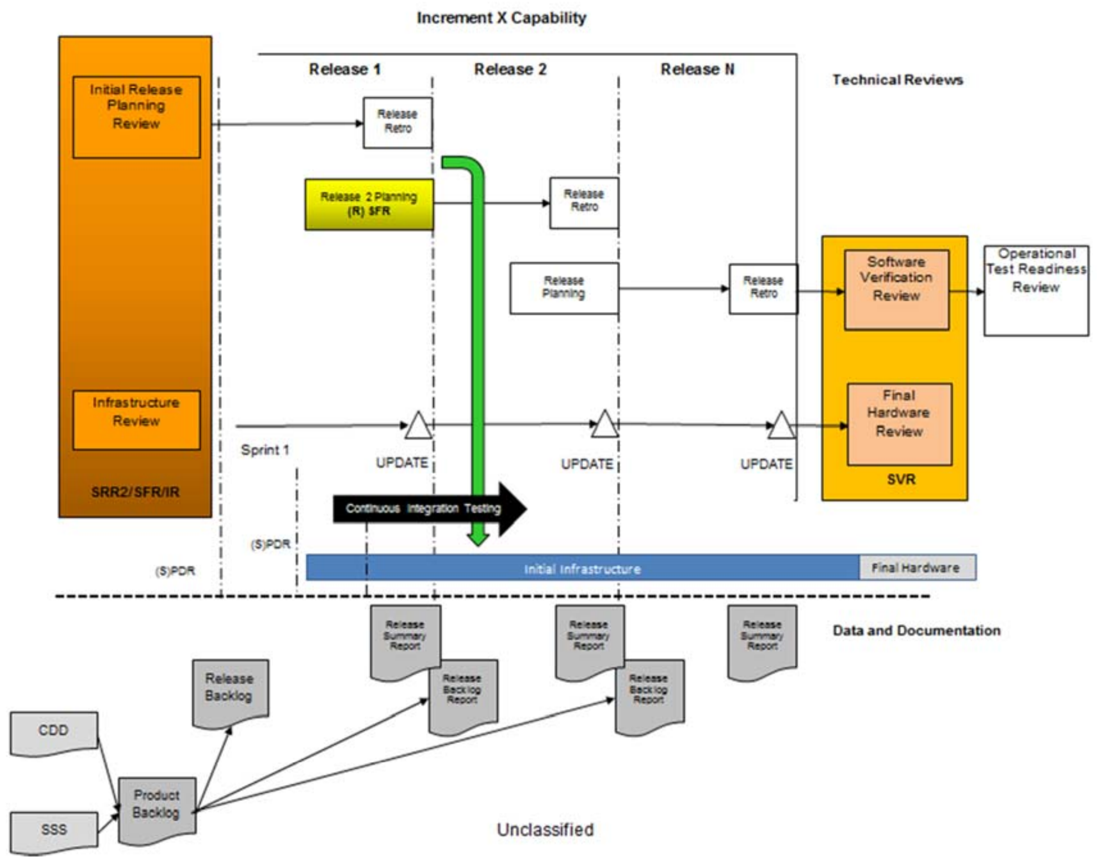

Figure 4: MC-Agile Development Process[10]

The Agile teams engage in continuous testing throughout the software development process, automated as much as possible. In Figure 3, we see that regression testing occurs during each sprint. Regression testing results then inform the backlog and subsequent sprints: problems discovered during regression tests are introduced into the backlog and prioritized for future sprints. Figure 5 illustrates automated unit testing being carried out throughout the sprints. With each planned release, a TRR is conducted, leveraging the results of the continuous testing. IRs are carried out as part of release planning, linking hardware and software planning. Figure 4 shows continuous integration testing throughout system development, leading toward a software verification review (SVR) and final hardware review as part of the SVR. Continuous integration ensures that there are no surprises at the integration with final hardware, the SVR, and the subsequent operational test readiness review (OTRR). Throughout system development, "**all** team members have access to collaborative tools and environments"[11]: systems engineers and testers are always aware of the status of integration and tests throughout software development. The team makes use of Forge.mil to provide visibility, in addition to using physical information radiators.

[10] Graver, Carmen & Greeley, Les. United States Marine Corps Agile Pilot Program Lessons Learned (MC-Agile). Briefing. February 2013. Unpublished.

[11] Ibid (emphasis in original)

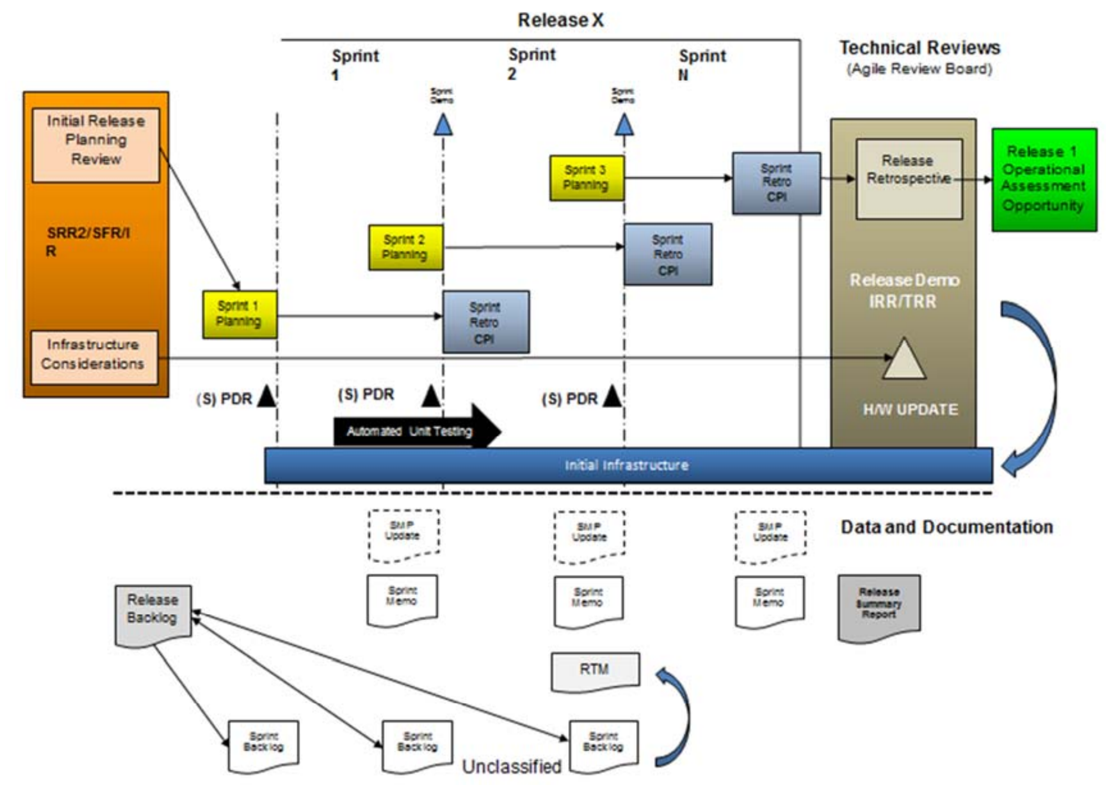

Figure 5: Individual Release Process[12]

Throughout system development, the linkages between software and systems engineering functions are present and visible as systems engineers take part in developing the product vision, defining the systems engineering process, and including Agile process links, release planning, demonstrations and reviews, and test planning and execution. The use of the RTM is a key communication strategy to both comply with acquisition guidance and to ensure that the trajectory of the release will result in a product that supports the end users of the system.

The team provided a series of lessons learned from the engagements on which their Agile processes were piloted:

> *Training at all levels is **absolutely vital***
>
> *It's OK to **raise issues***
>
> *It's OK to **change your process***
>
> *Involve **all** disciplines up front and early*
>
> *Develop a **common** program vision*
>
> *"Technical reviews" need to be **collaborative** conversations*
>
> *All documentation must be **value added***
>
> *Ensure that **all** team members have access to collaborative tools and environments*
>
> ***Share** lessons across programs*[13]

[12] Ibid

This example illustrates a thoughtful tailoring of traditional systems engineering and acquisition practice to create a productive environment for incrementally delivering value to stakeholders. Many Agile hallmarks pepper the lessons learned: involving all disciplines, engaging collaboratively at reviews, ensuring visibility to all parties, and ensuring that all documentation represents a value add. Notably, the publications of the lessons learned indicate that the processes were adapted over time as the teams learned more about the interactions and the execution of the processes.

2.3 Systems Engineers Applying Agile Methods to Their Own Work

In the case of systems engineers applying Agile methods to their own work, the biggest issue is the translation of "working software," a fundamental tenet of Agile methods when applied to software, to an equivalent for systems engineering. This case was more often seen in IT settings where there was no significant hardware development component, although there is at least one case of Agile systems engineering methods being applied across system, hardware, and software tasks.

Earlier we introduced the concept of the dual aspect of systems engineering—artifact transformation and communication (including coordination and conflict resolution services). In this case, it was mostly the Agile management methods like Scrum that were the method of expressing Agile principles, and when the systems engineering team went beyond artifact transformation in applying Agile principles, we saw a Lean method, kanban, as a method of choice. Kanban is a lean method deriving from the Toyota manufacturing system that emphasizes pulling tasks from a backlog rather than pushing tasks out to the team via plans and schedules. Some Agile frameworks designed to support larger scale projects are explicitly including kanban as one of the methods used (see our discussion of the Scaled Agile Framework in Section 2.4) [Reinertsen 2009, Leffingwell 2013].

Scrum, the most frequently applied Agile management method, can be adapted to the artifact transformation aspect of systems engineering more readily than the communication and similar functions, which are more easily adapted to kanban. Determining the "working product" (analog of working software) is a challenge when traditional systems engineering in its early phases focuses strongly on producing *documents* that guide implementation. In cases in regulated settings where relief from producing requirements, architecture, and interface documentation has not been granted, then those documents become working products. However, most systems engineering functions also produce various levels of breadboard and brassboard prototypes for hardware, and simulators and emulators that support early software development on hardware that is not yet designed. Applying Agile approaches to these types of working products is a natural fit into the systems engineering arena. Concepts like the product backlog, stories (in this case sometimes more technical or architecture stories than user stories), explicit iteration/sprint planning, short iterations that lead to a release of working prototype, and iteration demonstrations and reviews can

[13] Graver, Carmen & Greeley, Les. United States Marine Corps Agile Pilot Program Lessons Learned (MC-Agile). Briefing. February 2013. Unpublished.

provide productive enablers for the development of prototypes and the like. However, these activities are not as important to success as mindful application of Agile principles that focus on user/engineer collaboration, learning versus "shutting down" requirements or design constraints too soon, and empowering a cross-functional team to self-organize. If these principles are not expressed in the activities chosen, then many of the benefits achieved by Agile software teams will not be seen in the systems engineering environment.

Even when documents like requirements specifications, architecture descriptions, or interface control documents were the "product" focus, we saw examples of Scrum-type methods being used to produce them. In at least one case, the systems engineers interacting closely with end users early in the specification process produced a level of trust that provided benefit throughout the entire development cycle. On the other hand, in another case a change in management resulted in a program manager being assigned who did not understand or support Agile methods. In that case, surrogates had to be substituted by the contractor because end users were no longer authorized to provide the ongoing support for the document development processes to which they had previously contributed. Visible erosion of the trust between the development team and stakeholders was experienced, resulting in documentation products that required later rework, primarily due to insufficient understanding of the end user's perspective and a return to a lower level of detail for change authorization.

Example: Systems Engineering Teams Using Agile in Their Own Work

This example of a systems engineering group using Agile practices for its own work comes out of a complex federal IT domain.

The project was one from a large federal agency focused on enabling the sharing of research data across multiple university, federal agency, and private organizations. Ten different contracts were used to evolve six different applications, with multiple universities, federally funded research and development centers (FFRDCs), and industry contractors involved in one or more of them. Agile software methodologies had been used for the project, and FIST (fast, inexpensive, simple, tiny) guided the business aspect [Ward 2010]. In analyzing the way the business and software aspects were working, the chief engineer of the project observed the following key elements:

- incremental development
- iterative development
- short timelines
- retrospectives (lessons learned)
- empowered/self-organizing/self-managing teams
- prioritized product backlog for requirements
- small teams
- time boxing
- Lean initiatives
- prototyping
- continuous user involvement
- co-located teams

These elements were the ones that were used to guide the creation of an Agile framework for systems engineering, similar to that shown in Figure 6:

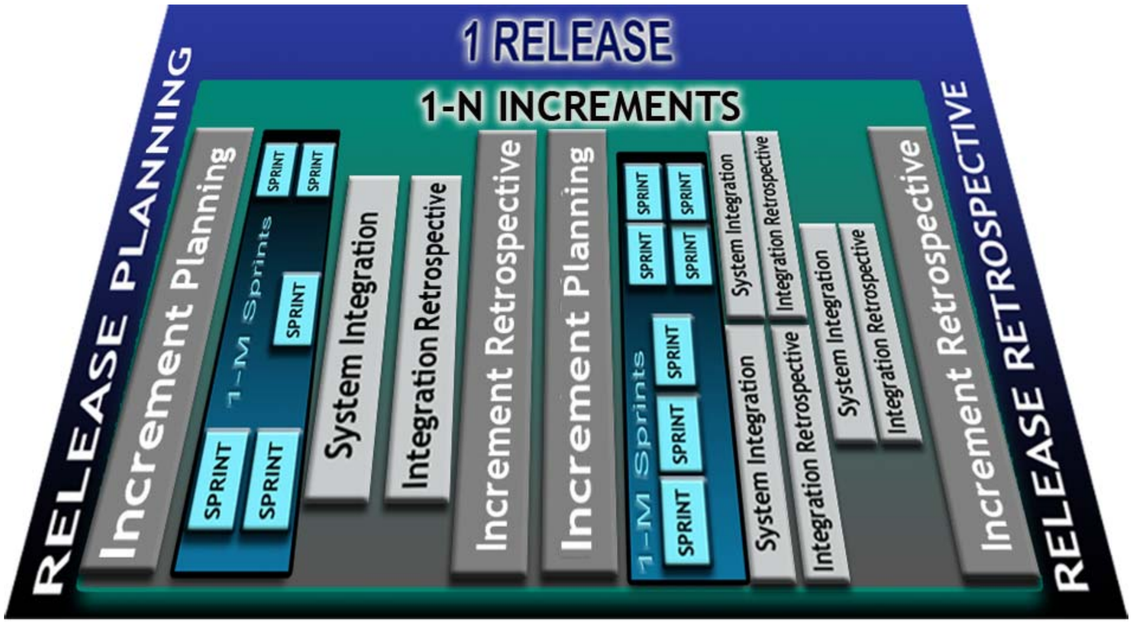

Figure 6: Agile Systems Engineering Framework Example [Kennedy 2013]

Note that the iterative development activities occur within each increment, and that each increment includes explicit system integration activities prior to the next increment, as well as increment and integration retrospectives that capture both technical and process lessons to be incorporated into the next increment.

Figure 7 shows the kinds of activities that could be included (this example is not from the federal IT project) as systems engineering activities, objectives, and input/exit criteria. In the systems engineering case, the product is not "working software" in most cases. The equivalent of working software is that the systems engineering work product under development (an interface specification, for example) meets a certain set of observable exit criteria upon completion of each iteration or sprint [Kennedy 2013].

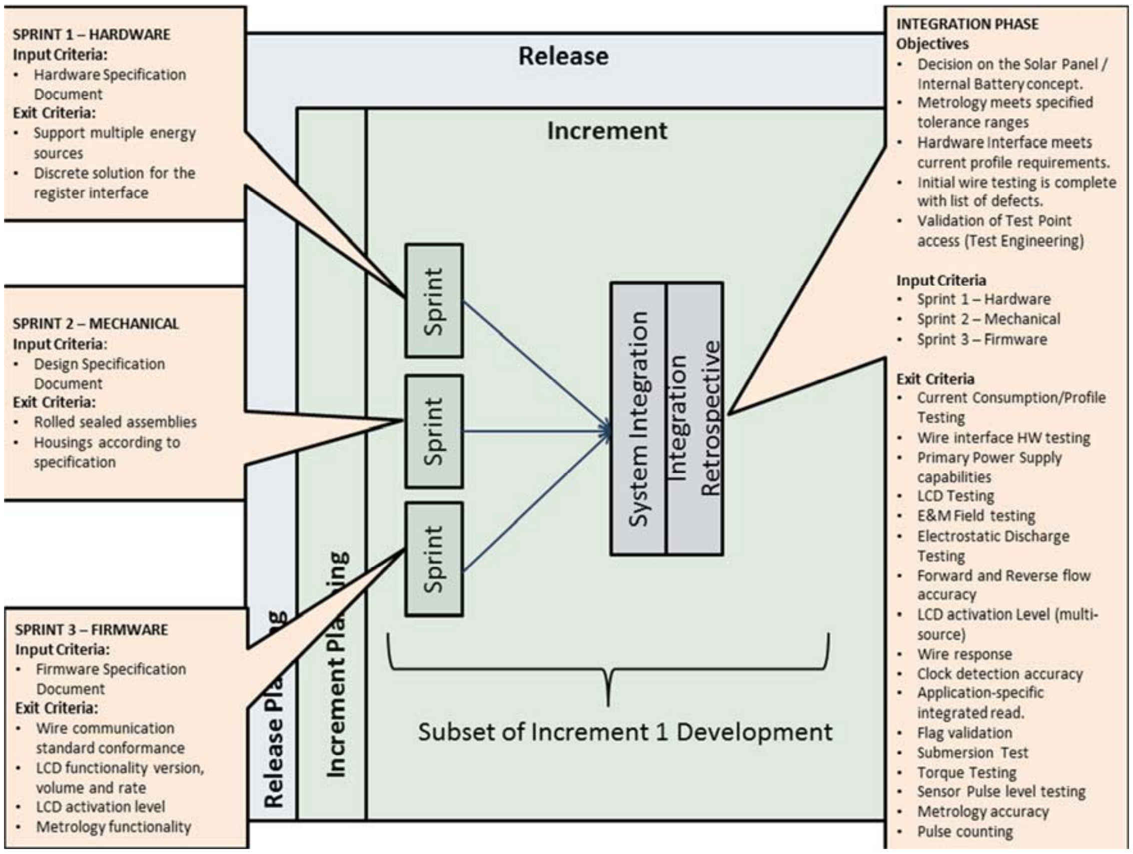
SPRINT 1 – HARDWARE
Input Criteria:
• Hardware Specification Document
Exit Criteria:
• Support multiple energy sources
• Discrete solution for the register interface
SPRINT 2 – MECHANICAL
Input Criteria:
• Design Specification Document
Exit Criteria:
• Rolled sealed assemblies
• Housings according to specification
SPRINT 3 – FIRMWARE
Input Criteria:
• Firmware Specification Document
Exit Criteria:
• Wire communication standard conformance
• LCD functionality version, volume and rate
• LCD activation level
• Metrology functionality
Release
Increment
Release Planning
Increment Planning
Sprint
Sprint
Sprint
System Integration
Integration Retrospective
Subset of Increment 1 Development
INTEGRATION PHASE
Objectives
• Decision on the Solar Panel / Internal Battery concept.
• Metrology meets specified tolerance ranges
• Hardware Interface meets current profile requirements.
• Initial wire testing is complete with list of defects.
• Validation of Test Point access (Test Engineering)
Input Criteria
• Sprint 1 – Hardware
• Sprint 2 – Mechanical
• Sprint 3 – Firmware
Exit Criteria
• Current Consumption/Profile Testing
• Wire interface HW testing
• Primary Power Supply capabilities
• LCD Testing
• E&M Field testing
• Electrostatic Discharge Testing
• Forward and Reverse flow accuracy
• LCD activation Level (multi-source)
• Wire response
• Clock detection accuracy
• Application-specific integrated read.
• Flag validation
• Submersion Test
• Torque Testing
• Sensor Pulse level testing
• Metrology accuracy
• Pulse counting

Figure 7: Notional Set of Systems Engineering Objectives, Input Criteria, and Exit Criteria for Systems Engineering Work Products [Kennedy 2013]

The federal IT setting didn't include all the hardware design and manufacturing shown in Figure 7 but was responsible for the overarching integration of the developers and contracts, and thus had to find ways to express objectives and acceptance criteria that were both meaningful and actionable. As soon as feasible, systems engineering moved to a prototype or demo system that allowed the product owners and other stakeholders to see working products, even if they were prototypes. Early, operations staff and independent testers were not enthused with the demonstrations but soon saw advantages when their feedback from one iteration often showed up just one or two iterations later in the development cycle.

Scrum practices were the ones used most heavily, since they primarily focus on technical management as opposed to technical execution. This made those practices, as well as some of those from the Scaled Agile Framework, particularly useful to the systems engineering function. A practice that is not classically "Agile" but was very useful in the project was to allow project leads for various components to drop out of the weekly Scrum of Scrums meeting once their component had been successfully delivered and deployed . This led both to a robust definition of "done" on the part of the systems engineer and a drive to achieve completion on the part of the developer.

The adoption of Agile practices was achieved primarily through on-the-job activities. The chief systems engineer was the primary Agile advocate, and he used planning and retrospective (lessons learned) activities already planned in the project to reinforce Agile concepts and practices, as well as providing informal communications as needed throughout the project.

2.4 One Example of an Agile Framework Adaptable to Support Systems Engineering: SAFe (Scaled Agile Framework)

While we did not encounter respondents whose organizations are currently operating in this manner, it is worthwhile to mention that Agile can and does scale beyond the individual team and across the program and the portfolio (organization level). This section provides the reader with high-level information and resources regarding the extension of Agile beyond the individual team or teams.

There are several approaches to scaling Agile methods to account for multiple teams and to account for software teams operating within larger system settings. Craig Larman suggests a systems thinking, bottom-up approach to scaling [Larman 2008]. Scott Ambler has developed an approach that synergizes Agile team activities with the Rational Unified Process, called Disciplined Agile Delivery [Ambler 2012]. Dean Leffingwell and colleagues have developed the Scaled Agile Framework (SAFe) as a way of connecting the team-level Agile activities to the larger engineering activities and the business/operations [Leffingwell 2007]. Other approaches continue to emerge from the community, some more focused on software-only systems, and some taking more account of systems engineering. Leffingwell's SAFe provides an example of how the thinking within the Agile community has shifted in recent years to embrace systems engineering concepts such as emphasizing architecture throughout the conception, development, and evolution of a system.

Figure 8: provides the overarching SAFe diagram that is a high-level summary of the key elements of the SAFe framework [Leffingwell 2013].

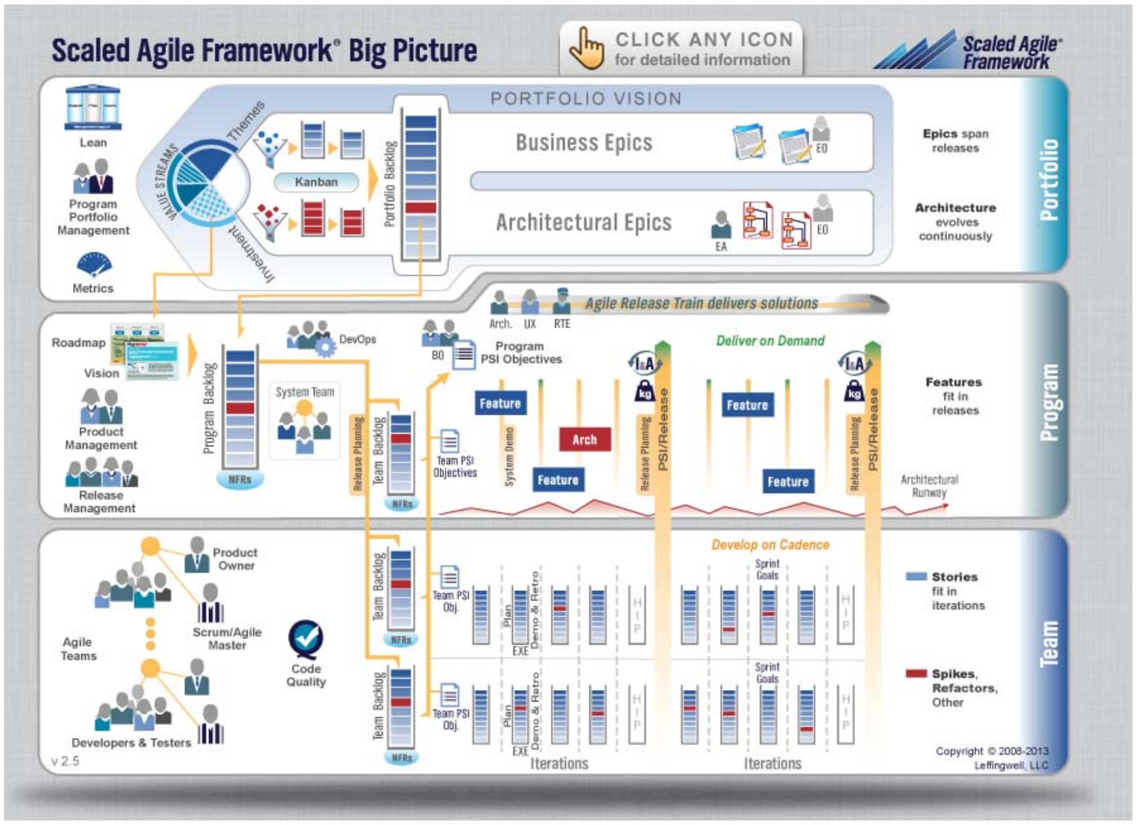

Figure 8: Scaled Agile Framework Big Picture [Used with Permission]

Note that the framework deals with two layers above the team level:

- program, where releases from the operational or business roadmap are planned and executed
- portfolio, where the organization’s business and operational needs are prioritized and processed to be amenable to inclusion on the organization’s product roadmap

Also note that the portfolio level uses kanban, a Lean systems technique, as the preferred approach for breaking down needs into prioritizable elements that can be processed into either business epics (functionality focused) or architectural epics (quality attribute focused). Although explicit systems engineering functions are not called out on the diagram per se, the inclusion of the specific role of system architect (one of the Agile release train roles) and explicit inclusion of architectural runways that inform and are fed by the Agile teams are both touch points where an Agile-aware systems engineering team could productively engage with Agile practitioners.

Although the main focus of SAFe is software systems development and evolution, systems engineering teams that are adopting Agile principles may find concepts within SAFe that resonate with the systems engineering view of applying science to economically useful products and projects, as well as with the systems engineer’s emphasis on architecture as the essential communication medium for technical decision making among development stakeholders.

As more experience is gained with SAFe and other scaling frameworks for Agile, more touch points and adaptations to enable productive collaboration between Agile software development teams and systems engineering teams can be expected.

3 Interview Observations and Findings

This section presents results from our team interviews with Agile practitioners. We reviewed our interview notes and observations to identify common topics and themes. Many of the topics are related to each other, and the findings overlap topic areas. Some topics focus on success enablers, others on challenges to adoption of Agile approaches.

3.1 Automation

Software development, build, and test automation can benefit software projects developed under any methodology—these benefits are not restricted to Agile projects, and their use and outputs can help program management teams gain valuable insight into the progress and quality of software development [Levinson 2013]. The rapid sprint cycles, continuous integration, and principle of "minimum necessary" documentation under Agile place significant emphasis on automation efforts to facilitate the execution of software development and communication and status monitoring within and across teams.

Success Observation: Automation Can Help to Harmonize Agile Projects with Traditional Constructs

Respondents who had the budget and resources to support automation (to the extent practical) of testing, integration, and other activities were able to take advantage of automation tools to streamline the development of documentation deliverables required under contracts or other agreements. Even in cases where the systems engineering function was not engaged in the Agile software development processes, respondents who made extensive use of automation reported that they were able to produce documentation from their Agile workflow that satisfied traceability and other communication requirements, for both program offices and systems engineers. Even if these documents go beyond the minimal required documentation favored under Agile methods, automation supported generating them as much as possible from work-in-progress artifacts in a manner that minimized the amount of additional work required to produce the artifacts; in other words, the team was able to "maximize the amount of work not done" in the production of additional documents [Agile Alliance 2001]. One "key to Scrum is to instrument as much as possible" throughout the development process to support test, documentation, analytics, and situational awareness.[14]

We have previously written about how achieving harmony between Agile approaches and the approaches favored under the DoD 5000 series may require "hybrid" approaches [Lapham 2010, 2011]. The use of instrumentation and automation environments under Agile to support information requirements for programs can be one way of achieving some of this harmony when the existing contract or agreement is constructed under a waterfall or traditional model.

[14] Quote from respondent interview.

Challenge Observation: Lack of Automation Can Hinder Communication and Insight

On the other hand, some respondents reported difficulty in securing "buy-in" or financial resources to support the implementation of automation. This hampered the ability of Agile teams to engage in cost- and resource-effective testing and generated frustration not only about engineering effort that could have been more effectively utilized with automation in place. These respondents also indicated that communication and transparency between software and system teams (and program offices) was hampered. Automated testing situations, for example, would allow software teams to more readily provide systems engineers with insight into the quality of software that will be delivered and integrated into the system. Note that any development approach can take good advantage of automation. However, common Agile methods focus on continuous integration and build testing to ensure that the software being produced is always working, providing incremental confidence that the demonstration at the end of the sprint will be successful. As systems become more complex, achieving that confidence without automated testing becomes more and more difficult and time consuming, eroding some of the benefits achievable in Agile settings with appropriate automation.

3.2 Insight/Oversight

A frequent objection to the use of Agile methods on DoD programs is that the "just enough" documentation philosophy espoused by Agile will result in a lack of insight into the progress of the development project. Our respondents reported that under Agile they had access to more data for managing and reporting within and across teams more frequently and with greater transparency than they would have under traditional methods.

Success Observation: Agile Can Increase Opportunities for Meaningful Insight

Our observation through these interviews is that Agile projects offer even more opportunity for visibility into technical progress than is generally available under traditional engineering methods, due to the frequent delivery of software and the use of automated environments that provide easy access to status of coding and testing efforts. Sometimes the level of insight is actually overwhelming to acquisition professionals who are not accustomed to continuous involvement in the development activities. That insight frequently takes a different form than the many contract data requirements lists (CDRLs), reports, and presentations/reviews that are long ingrained in DoD acquisitions.

One respondent engaged in Agile in both the software and systems engineering aspects of a complex multi-developer program with dozens of stakeholder organizations reported that the level of insight achieved under the Agile implementation was actually far superior to that which would have been available under other methods. Data was available for each sprint and increment of every element of the program and could be "rolled up" to a level suitable for executive and customer reporting. At the same time, the breakdown of the software and system according to the various release roadmaps and backlogs enabled detailed, far-reaching visibility into areas of the program that were affected by a change in a related interface. The Agile project data provided such granularity that the engineering leadership team was able to determine the sunk costs of all previously implemented capabilities related to the interface change and to report the economic cost of the change. The lead systems engineer also was able to look across the various elements of the system and determine with great clarity what additions would need to be made to various backlogs to accommodate the change in the previously implemented capabilities and what to-be-implemented capabilities in existing backlogs would be affected by the change. The lead systems engineer reported that the Agile implementation actually improved accountability throughout the software and systems engineering functions due to dramatically improved visibility and expectation setting. Note that although the level of planning of roadmaps on this project was more detailed than is typical on smaller, software-only Agile projects, there was still understanding and acknowledgment by the systems engineering and program management teams that these were notional until development activities either supported or contradicted the direction. Learning was explicitly supported throughout the development cycle.

Respondents also demonstrated that engaging systems engineering and program office personnel in sprint planning meetings, various interim reviews, and demonstrations, in addition to providing them with access to collaborative tools, consistently gave these stakeholders timely, meaningful insight into the progress and quality of the software project—regardless of whether systems engineers were participating in the Agile software development activities. The teams use interim reviews and demonstrations as "surveillance points": decision opportunities where key stakeholders can observe the functional demonstrations and review other activities conducted and work product completed to enable decisions related to design (form, fit, and function). The demonstration of "no kidding, working software" provides direct insight that gives systems engineering teams confidence in the quality and schedule commitments made by the software teams and simplifies planning and execution in systems engineering. Granular, up-to-date information on progress and quality is always readily available. In one case we learned of a program whose government counterparts were active participants in the various software sprint and release reviews and acting as product owners, while at the same time interfacing with traditional systems engineering teams that did not participate in the Agile reviews.

Challenge Observation: Program Offices and Systems Engineers Must Commit to Understanding "New" Ways of Gaining Insight

Some respondents indicated that they had trouble getting program office teams to exercise the constant insight opportunities that were available. One team reported that program office staff had been given accounts on the Agile team's collaboration tools so that status of sprints and iterations, backlog items, and test activity could be made available on an up-to-the minute basis. The developer later discovered that no one from the program office had ever logged in to the system. This was likely caused by a combination of a lack of training or understanding about how insight could be obtained in a timely, low-overhead manner (see Section 3.3), a lack of sponsorship to set expectations about how communication would be maintained between the developer and the program office team (see Section 3.4), or even a lack of willingness to engage (see Section 3.6). The constant, systemic delivery of production-ready code by Agile software teams over short iterations and the use of metrics and tools such as burn-down and cumulative flow diagrams offer up frequent, regular windows to monitor the progress and quality of the system under development, but stakeholders have to understand and actually exercise those insight opportunities to derive value from them.

3.3 Training

Our respondents generally reported that the experience and knowledge all participants possess regarding Agile methods play a significant role in the successful deployment of these techniques in acquisition programs.

Success Observation: Repeated, Targeted Training of Stakeholders Improves Insight, Oversight, and Communication

Teams that provide systems engineers, government program office personnel, and other stakeholders with Agile-specific training on a *recurring* basis have reported success with communicating and expectation setting when Agile is employed on a project, both with government program teams and with systems engineering functions and other technical teams, even when systems engineering teams are employing traditional systems engineering models.

Multiple respondents reported developing short team and executive courses, and even a "boot camp," to educate program office leadership and functional staff (e.g., engineering, contracts, and finance) and systems engineers and other stakeholders about the Agile methods and processes employed by the development organization. These training sessions in many cases were applied *inside* the developer's organization as well, to ensure that systems engineering, test, and other staff members were aware of and understood the Agile approach and to prepare for necessary organizational change. Systems engineering stakeholders were able to emerge from the training with a good understanding of the role they needed to play to interface with the Agile software team as appropriate, and government participants were able to understand various ways to communicate and interpret data on the project and the new and different ways they could gain insight and execute oversight on a program using Agile.

Those respondents who developed and delivered training reported the greatest success coming from *multiple* offerings on a regular basis. Agile represents a significant paradigm shift for most acquisition stakeholders, and by engaging personnel in the material repeatedly, coaches and trainers were able to both introduce the material and reinforce the concepts, creating a solid understanding of the techniques, processes, metrics, deliverables, and team member roles [Lapham 2010, 2011]. When this repetition follows a paradigm such as the SEI's Adoption Commitment Curve (contact, awareness, understanding, trial use, etc.), participants can approach implementation with more confidence [Garcia 2006].

Continually offering repeat training rather than a "once and done" approach also helps Agile practitioners combat lack of continuity that occurs due to the frequent rotation of government personnel in and out of acquisition program offices. We also commonly observe senior government lead engineers and systems engineers being loaned to different programs to provide short-term coverage for staff vacancies. With Agile representing such a significant departure from traditional approaches, having appropriately scoped training available for incoming staff can minimize miscommunication and misunderstandings. One respondent indicated that during her tenure on a program, no fewer than four government program managers had been involved. A team that had been successfully delivering software via Agile methods, and had built a good battle rhythm with the corresponding program office, was suddenly cut off at the knees with the arrival of a new program manager and new senior leadership who did not understand how the Agile deliveries and reporting were executed. The software program experienced a very painful pendulum shift back to a forced waterfall delivery model as a result.

Challenge Observation: Stove-Piped Functional Training Prevents Full Understanding of Agile Initiatives

Government program office staff training for the various career fields involved in acquisitions (e.g., contracting, finance) is functionally oriented or even stove piped. Respondents indicated that strict capability-based training for government personnel has hampered the ability of many program office team members to conceptualize the changes in how contracts and delivery orders must be structured to most effectively engage Agile development teams. As one respondent indicated, “This is not just a technical problem.” For example, financial managers need to understand the tempo at which the Agile team works, and how the software team engages in estimation and earning value for work completed. These challenges are not insurmountable but require commitment to resolve. A Defense Acquisition University professor concurs: “The concepts of Agile are based upon sound practices for software development and therefore are not new in nature. This drives a demand for training for all the government program office as appropriate for their role” [Broadus 2013].

For one example of the importance of cross-functional, role-based training, consider the development of contracts. Several respondents indicated that government teams had specifically requested “Agile” techniques be employed on software efforts yet wrote contracts and CDRL structures that significantly constrained or even negated the employment of Agile techniques. Warranted contract officers duly and in good faith developed contracts for what they *thought* the system wanted to acquire in terms of software, based on their understanding of traditional waterfall-like implementations. For example, a complex, extremely detailed software requirements specification (SRS) would be put on the contract, constraining the Agile team’s ability to develop a backlog, assign work to sprints, and explore high-risk or high-value (or both) requirements in early iterations to inform subsequent requirements analysis and design and development activities. Misunderstandings such as these impact the system-level contract specifications and deliverables as well. If the contracting officer has a better understanding of how the Agile development contractor is actually proposing to work and the means by which the team conducts reviews and provides metrics, then the contracting officer will be much better positioned to develop a contract vehicle of an appropriate type that protects government interests and yet still allows Agile teams the ability to incrementally innovate and evolve the software to meet its mission needs. (Section 5 provides further discussion specifically about contracts and agreements.)

3.4 Role of Sponsors, Advocates, and Coaches

In a white paper developed at the 2011 NDIA Agile Scrum Workshop, a diverse group of practitioners and researchers wrote that “DoD does not need an Agile Champion. It needs an Agile Champion in every organization” [NDIA 2011]. Sponsors, advocates, and coaches for Agile can be instrumental to teams undergoing change to meet operator needs and variations in processes and to ensure that culture can change to support Agile methods, processes, and techniques. Our respondents provided feedback on the role of advocacy in facilitating interactions between software and systems engineering functions.

Success Observation: Agile Advocacy Can Facilitate Improved Systems Engineering Engagement

In organizations where there was clear advocacy by leadership, our respondents indicated that development teams openly used Agile methods and processes not only in software development but also in some cases in systems engineering and other areas of the lifecycle. When coaches established and delivered training on Agile (see Section 3.3), systems engineering process and software development became better intertwined and decisions were made together instead of separately, as systems engineers either acknowledged or began to participate in the Agile software activities. This allowed for time efficiencies and consensus building that prevented miscommunication, rework, and unnecessary extra documentation. The interaction of teams when there was advocacy and coaching representatives was drastically different than when teams had no clear direction or support. Collaboration of cross-functional teams occurred openly and frequently. Guidelines on documentation and synchronization of the team goals were done frequently during sprints with expectations set via planning meetings and memos that documented the agreed-to definitions of "done."

Moore and colleagues wrote of the Army's efforts on the "Battle Command (BC) collapse" Agile systems engineering strategy, dedicated to removing separation from the information systems toward a consolidated product line, that

> *Innovation is emergent and dynamic and BC realized that it is typically a bottom-up approach in which people involvement is critical.… To overcome cultural challenges, we worked with leadership for buy-in of the agile process … and encouraged every member of the team to participate in sprint reviews and creation of the environment. Through this process, we found innovation came naturally and was accepted more openly. The rapid and aggressive approach also brought a higher number of risks than a traditional process, so we had to adjust our tolerance for acceptance and balance it with value to our users.* [Moore 2011]

Leadership support and advocacy was necessary at all levels to "communicate the strategy, the plan, and seek feedback at every level" to achieve the desired outcomes [Moore 2011].

Success Observation: Advocacy Is Necessary for Facilitating Cultural Change

Multiple respondents lamented that communication and culture shifts toward Agile represented significant problems, echoing a constant refrain from our ongoing research [Lapham 2010, 2011]. Aragon and colleagues wrote that Agile advocacy by a "visible, empowered champion" is *required* in order to effect these cultural changes [NDIA 2011]. Our interviews indicated that indeed advocacy (and coaching) was associated with more successful communication and cultural transition than programs that did not have this type of support. Leadership advocacy sets expectations for communication and collaboration within and across organizations and provides support that allows Agile practitioners to explore program-specific tailoring of the processes and documents required by acquisition regulations (see Section 4). Strong advocacy also, in the words of one participant, "lets [Agile practitioners] learn the lessons" from engaging in innovation, to improve both their Agile practice and the software product they create.

Consultants and coaches were also cast as the advocate to the group. Efficiencies were found by advocates and coaches such as combining design reviews and creating shorter meetings but meeting more frequently with focus areas. Our respondents indicated that some test organizations have adopted Agile and can be seen as advocates, even offering Agile workshops.

Challenge Observation: When There's No Advocacy, "Your Ability to Influence Change Ends at Your Sphere of Influence."[15]

With no Agile advocate in place to lead the charge, organizations (both program office and contractor/system developer organization alike) seemed to not want to adopt Agile as a method to deliver frequent iterations of capabilities. Instead, teams tended to do a partial Agile or "Agile-like" approach in small, easier-to-adopt functional elements without leadership oversight. These efforts originated in the software teams and may or may not have been visible to systems engineers and program offices as "Agile" efforts. Our respondents indicated that these areas frequently included constant integration and user feedback in product prototypes. This lack of advocacy can ultimately hinder the effectiveness of an Agile team, if they cannot get support for executing their methods and tailoring technical and documentation deliverables to meet program goals.

Challenge Observation: Agile Coaching in the Absence of Advocacy

On a balanced front, where coaching is present but advocacy is not reported, respondents said that teams evolved, innovated, and adopted their own Agile needs based on their project or program. Indeed, self-organizing teams are a bedrock of Agile methods [Agile Alliance 2001]. Such respondents also mostly indicated that they need more top-down support and advocacy. Our interview data reflects that these types of teams performed Agile-like methods but did not necessarily call them "Agile" (what is sometimes referred to as "covert Agile"). These types of teams also related that without leadership buy-in, "you are up against a brick wall." Organizational change literature confirms that grassroots adoption of new practices can only progress beyond the local adopting group if sponsorship at the larger organizational level is achieved [Garcia 2006].

3.5 Pilot Programs Demonstrate Success

Success Observation: Pilot Programs Assist with Developing Agile Leadership and Technical Team Support

Many respondents reported that systems engineering leadership carefully observed the results of smaller pilot efforts on software-centric programs. Pilot projects were initiated in a variety of ways:

- upon the initiative of software teams who specifically wanted to explore Agile, and may have experienced "pushback"
- upon the initiative of engineers experienced in Agile on other programs
- upon the initiative of teams that witnessed successful pilots on other programs
- at the request of teams experiencing difficulties, who sought out Agile coaches in their organizations
- at the direction of government leadership, as part of rescue or recovery efforts for projects in jeopardy

[15] Quote from interview respondent.

Respondents who engaged in piloting before broader rollouts indicated that demonstrated cost and schedule savings and the repeated demonstration of functional code often made believers out of previously skeptical systems engineering teams. One respondent described winning over systems engineers by consistently demonstrating "no kidding, working software." Successful pilot project teams facilitated systems engineering involvement in the Agile process, often through attendance at key reviews or by participating as product owners. Organizations that engaged in pilot activities tended to report that systems engineers were engaging on their Agile software teams and in some cases applying Agile techniques to their own work.

Respondents consistently reported that the demonstration of cost, schedule, quality, insight, or predictability improvements via pilot projects also garnered positive feedback from government program offices, which made it easier to secure leadership buy-in and sponsorship for expanding the use of Agile on future efforts. Respondents reported the following demonstrated results on various Agile pilots, which spurred further action and support for agile methods:[16]

- 2-3 times performance improvement, at 50% of the cost
- 12-month functionality/capability pilot achieved in 6 months
- challenged to execute an 18-month project in 6 months; fell short of 6 months but well ahead of 18, with results and performance such that leadership deemed the approach a success
- 18-month schedule savings on a large program

Respondents did not indicate any challenges or obstructions that arose as the result of piloting Agile efforts.

3.6 Stakeholder Involvement

Agile methods value "customer collaboration over contract negotiation," relying heavily on user, or user surrogate, participation throughout development [Agile Alliance 2001]. Turner observed that "[systems engineers] are often isolated from the customers because their customers are considered fully represented by the pre-defined requirements and operational concepts" [Turner 2007]. Collaboration between end users and customers in DoD acquisitions presents further challenges due to the many stakeholders in a system that may have conflicting needs [Lapham 2010]. This set of interviews demonstrates that the accessibility of end users and other stakeholders continues to present challenges for Agile teams. While some individual respondents indicated greater levels of success with end user collaboration, the overarching patterns mostly described challenges that Agile practitioners continue to face.

Challenge Observation: Continuous Collaboration with End Users May Not Be Easy to Achieve

Respondents observed that operational system end users are often difficult to access, which further isolates systems engineers from customer requirements; operational system users

- "are not used to" being continuously engaged in development efforts

[16] These data are as reported by the interview respondents from internal program or contractor data. As such, the respondents wished to keep the programs/organizations anonymous.

- may have little to no training on the type of role they might be expected to play in an Agile setting
- may lack leadership support to prioritize their time to facilitate their participation in the face of their operational responsibilities
- may not be available on a consistent basis due to operations tempo, funding, geographic dispersion, and other constraints

Most respondents—both government agency and contractor development organizations—mitigate this challenge by engaging user surrogates such as retired operators to provide constant collaboration, and then engaging with the software's or system's intended operators at less frequent opportunities such as pre-planned technical interchanges or system user conferences or events.

One respondent reported leveraging government program office staff to serve in the role of product owner for the system (though without using the Agile terminology). This team was successful with engaging this government subject-matter expert to reach out to various stakeholders (various "customer" systems) and develop a consistent voice. Then the person filling the product owner role was leveraged as a surrogate for the external stakeholders.

Several respondents noted that end users became more engaged over time as they saw the benefit of collaborating with the development team, whether through the program office interface or directly. However, as we saw in at least one case, that involvement can be fragile, especially if a new program office official or operational leadership rotates in who does not understand or support Agile methods (see Section 4.4).

Challenge Observation: Multiple, Diverse Stakeholder Groups Bring Conflicting Requirements, Notably Regarding System Security and Accreditation

Rarely is there a single end user for a DoD or federal system. Even with aircraft, consider the pilots, various crew members, refueling crews, aircraft maintainers, and other users, all of whom have legitimate needs when it comes to the features of the end system. Clearly establishing, prioritizing, and deconflicting these requirements happens constantly throughout the development of a system. How can Agile practitioners constantly collaborate with such a diverse group of stakeholders? As noted above, it is likely to be infeasible to have constant representation from every identified user throughout the life of the development cycle for a number of reasons. Product owners in systems engineering functions and government program offices have been reported to help manage these challenges.

In our discussions with Agile software practitioners, the most frequently cited case of conflict coming out of system requirements is a clash between continually changing security policy requirements and operational capability requirements as defined by system users. Entering into an independent certification and accreditation (C&A) process is often a "black box" into which program teams have very little visibility—and often wait in a queue for their turn. Problems are kicked back out of the process, and typically negotiations or system modifications ensue. Practitioners we interviewed indicated that security requirements are constantly changing (due to changing nature of threats) and that changes required by the affected agencies may directly conflict with operational capabilities required by the system.[17]

The frustrations reported regarding the navigation of independent C&A processes are not unique to Agile efforts. However, some Agile teams have reported success in engaging early in the development cycle with representatives from the independent C&A agency that will be responsible for the certification. The C&A process is conducted independently by the certification agency, but a representative engages on a regular basis with the Agile team to understand the work being developed and identify any potential issues prior to the C&A formal evaluation. C&A is treated as a stakeholder requiring ongoing collaboration. C&A package documentation is developed, and technical configuration audits and system hardening[18] are performed concurrently throughout system development. We interviewed one C&A practitioner at a recent conference who was very excited about the possibilities of working with Agile teams throughout their process and intends to engage collaboratively to as great an extent as she can manage. We look forward to experience reports on such pilots.

Challenge Observation: Disengaged Stakeholders Inhibit Progress

Many respondents reported having difficulty accessing not only system end users but also the government program office staff. As noted in Section 3.2, one respondent reported that a government program office was provided with full access to the development team's online collaboration toolkit to facilitate insight into the progress of the software development effort. The software team then discovered that the program office representatives missed out on this opportunity to maintain consistent insight, as they did not log in to the system for the entire duration of the development effort. This behavior likely stemmed from a lack of training (see Section 3.3) and awareness and a lack of cultural adaptation to accommodate the blending of the Agile techniques with the rest of the program.

Another respondent reported that the government program office customer to whom the software team reported refused to engage in any kind of training on the methods or approaches being used by the contractor's Agile software and systems engineering teams. They did not want to understand how the contractor was working, which meant that they could not understand how to effectively gauge progress and quality from the contractor's reporting and documentation. This represents a lost opportunity to gain valuable and timely insight for the duration of the software development activities. Many teams have demonstrated the ability to deliver short, highly targeted training to government personnel, which facilitates understanding of the development methods and insight approaches available with Agile software development. See Section 3.3 for further information on the impact of recurring training in this environment.

[17] See our technical note *DoD Information Assurance and Agile: Challenges and Recommendations Gathered Through Interviews with Agile Program Managers and DoD Accreditation Reviewers* for more information on Agile challenges related to information assurance [Bellomo 2012].

[18] System hardening is "the process of securely configuring computer systems, to eliminate as many security risks as possible" [Berkeley Security 2013].

3.7 Defining and Evolving Requirements

"One of the characteristics of traditional project management, and by implication much of traditional systems engineering, is the assumption by all stakeholders that foreknowledge is perfect" [Turner 2007]. In a talk titled *The Illusion of Certainty,* Campbell characterized the issue of requirements uncertainty in acquisition programs: "Traditional acquisition practice relies on certainty in requirements.… Uncertainty is unavoidable but seen as a weakness to be eliminated" [Campbell 2010]. Traditional, waterfall-based processes assume (or require) a linear relationship between the identification of system requirements and system design that remains a demonstrable assumption in many systems engineering models: requirements must complete before design can commence. In reality, system requirements evolve even as the system is being constructed. Technology refresh for commercial components is rapid. New threats and vulnerabilities emerge and do not wait for systems to catch up to them. Kennedy wrote that "Previous experience shows that changes within an SIS [software-intensive system] are inevitable, whether or not there are changes in requirements or technology" [Kennedy 2011].

Our upcoming technical note, *Selected DoD Acquisitions: Requirements Development and Management* (in management review), deals with this topic in detail.

Challenge Observation: Preferences for Detailed Up-Front Requirements vs. an Evolving Backlog Remains a Source of Conflict

Agile software practitioners still report a tendency in many acquisition programs to create detailed software requirements at the inception of the program, when the system is decomposed and the initial WBS developed. (Indeed, this is a common objection of software engineers on DoD projects regardless of the software development methods employed, as noted in Section 1.2.1.) Agile software teams are then accountable to systems engineers to develop within a rigid construct that inhibits the discovery and refinement of the software requirements. One Agile practitioner explained that his efforts at risk-based sprint and release planning were frustrated by "an SRS you can practically compile," that brought with it a cumbersome change control process. When detailed requirements are placed on contract at the beginning of a program, they are typically accompanied by change control boards and engineering change proposal (ECP) processes that many Agile practitioners report to be cumbersome, time-consuming, and expensive to complete. Often these result from the type of contract vehicle employed (for example, when firm fixed-price contracts are employed, development contractors resist changes to requirements that will cost them additional effort to address). What Campbell referred to as "due effort" to resolve requirements uncertainty, Agile practitioners address through risk-based iterations and continuous collaboration to refine requirements and design [Campbell 2010]. Agile practitioners accept that the unknowns at the inception of a program are a natural and expected phenomenon, rather than treating those uncertainties as weak points.

Several respondents have indicated that changes to methods of putting work on contract can help alleviate some of these problems. One example is putting epics on contract, where an "epic" is a collection of user stories. The program office and the Agile developer must agree on the size (using an agreed-to sizing methodology) of the epic and completion criteria (e.g., "what *done* means"). The epic covers a certain number of sprints or iterations, thus bounding both the size of the work and the time frame to completion. Agile practitioners enjoy the flexibility within such a structure to work down from the high-level capability requirements to evolve the functional requirements and design for the system, with the support of user feedback.

3.8 Verification and Validation

Verification and validation activities within the DoD environment continue to be both a source of opportunity and of frustration. While Agile teams generally prefer to execute much of the functional testing (especially unit and regression testing) of the software within the confines of the sprint, various independent test and validation functions are necessary within the constructs of DoD projects. Software must be subject to integration testing with hardware or with other systems and interfaces and must be assessed for information assurance purposes and fitness for deployment on operational networks. Interactions between programs and these independent test functions have historically been reported as sources of bottlenecks, independent of the software development methods employed.

Success Observation: Integrating Test and Validation Functions and Personnel into Agile Teams Provides Significant Insight Useful to Acceptance Testing and Operational Testing

In respondent settings where test and validation (T&E) personnel were included explicitly as members of the development team, they reported that many of the T&E personnel found significant benefit in developing acceptance criteria for stories and creating acceptance test fragments for portions of the system being completed. They also gained significant insight into the architecture, quality attributes, and functioning of the system, all of which translated into better test readiness when independent test activities had to be undertaken.

We observed a dichotomy of philosophies from test community members we have interacted with in terms of the benefit of interacting directly in the development tasks. Some testers embraced the opportunity to work early with the requirements and subsequent development. In traditional settings, testers frequently clamor for more insight into requirements during initial systems engineering phases so that they can assess the testability of the requirements. As part of the development team, they have transparent access to evolving requirements, design, and implementation strategies and many appreciate that level of granular insight. Others saw getting involved with the development team directly as violating their need for independence by allowing the developer to "develop to the test." One Agile method, test-driven development, actually takes the perspective that developing to a test is an effective and efficient way of helping developers to build only what is needed to complete a feature, rather than building additional, un-required functionality based on developer preferences or interests [Shalloway 2012].

Challenge Observation: Integrating Test Personnel into Development Teams Has Both Organizational and Skill Challenges

One of the general Agile practices when forming development teams is to keep the core team focused on individuals who can actually implement the designs. Systems engineers are generally seen as architects, who are known to be a scarce resource and may not be expected to participate in coding activities, or product owners, who are explicitly not expected to implement. T&E personnel are expected to perform their normal duties of creating testing artifacts and helping to execute test activities during an iteration or sprint, but if they are core team members, they are also expected to provide implementation support. Depending on the context and skill level of the T&E personnel, implementation support may be outside their comfort or skill level. Some respondents solved this by making T&E personnel extended team members, or by having periodic technical interchanges focused on testing issues, as well as explicit hardening iterations where T&E personnel provided feedback on the acceptance criteria, definition of done for the release, and insight into areas they felt were at risk for passing independent evaluation.

Organizational issues can also prevent direct involvement of T&E staff on development teams. Some programs use specific test laboratories where test personnel are located there rather than co-located with development personnel. Although development T&E (DT&E) personnel are often organizationally aligned with development personnel, if the developer is a contractor and the DT&E personnel are government personnel, there can be additional organizational challenges if agreements have not been made contractually as to the way the DT&E and development staff will work. Operational T&E personnel were not mentioned in the interviews as being part of the Agile teams, and those who did include DT&E personnel on their teams saw this separation as one of the ways to preserve the needed independence of test functions in relation to development.

Success Observation: While Certification and Accreditation Activities Continue to Frustrate Agile Practitioners, There Are Pilot Successes to Build From

As noted in Section 3.6, many of our respondents expressed frustration with their interaction with independent C&A functions, as a phase gate forced them into more of a "water-Scrum-fall" as they experienced long wait times to complete the C&A processes. Compounding the problem is the constant evolution of security threats to data and networks—C&A activities are viewed by many software developers as impossible moving targets. Resolving this situation requires diligence and commitment from Agile teams, systems engineers, and the certification and accreditation authority, but there exist multiple precedents for this collaboration.

As we previously discussed, we have reports from our interviews regarding regular collaboration with willing C&A authorities. The C&A process is conducted independently by the C&A agency, but a representative engages on a regular basis with the Agile team to understand the work being developed and identify any potential issues. C&A is treated as a stakeholder requiring ongoing collaboration. C&A package documentation is developed and technical configuration audits and system hardening are performed concurrently throughout system development.

One respondent reported working closely with the C&A authorities and other stakeholders on several interface and data hosting requirements early on in their Agile process. As the software began to take shape, the parties were able to assess what risks were associated with each of various implementation alternatives and *which entities were willing or able to accept certain risks*. This resulted in the C&A team, the systems engineers, the development team on the other end of the interface, and the software development team identifying a modification to the design that obviated the potential C&A risk entirely.

In a published case study, the Army's BC collapse team reported engaging the relevant C&A authority early on in its 30-day sprint cycles to execute "concurrent planning and execution of security accreditation and training modules earlier than traditional waterfall processes [that] allowed us to provide the system to a beta unit for feedback much earlier than anticipated" [Moore 2011].

3.9 Aligning the Program Roadmap with Agile Releases

Defense acquisitions operate within a statutory and regulatory framework that requires the completion of certain phase gates on the programs. Technical and other reviews are inserted into the process according to policy guidance and serve as a gating function for continuing with the development of the project. The use of these reviews and milestones as strict phase gates on requirements, design, and test runs antithetical to the small, iterative approach employed by Agile teams in order to produce working code at the end of each small iteration or sprint. Agile development teams have worked with programs in a number of ways to attempt to tailor the existing milestones and reviews to more closely align with the manner in which Agile teams deliver software functionality.

Success Observation: Tailoring of Program Milestones Allows Flexibility of Development While Maintaining Alignment with Program Objectives

Many respondents engaged in new system development reported being granted the flexibility of tailoring program milestones such as the preliminary design review (PDR) and critical design review (CDR) by engaging systems engineers and program offices in sprint and iteration reviews. They are thus able to treat activities like PDR and CDR as “capstone” reviews and Agile surveillance points. Because the systems engineers and government program offices are engaged in the iteration-level reviews, surprises do not emerge at the larger (contractually required) milestone reviews. Tailored milestone reviews have been used to demonstrate detailed “prototypes” or to define and review highly critical architecture elements of the system to various stakeholders. These iterative reviews are successful when there is a high level of trust between the government program office, the systems engineers, and the software development team, allowing a relinquishment of the control typically exercised at PDR and CDR activities.

In at least one case, a contractor was expressly granted relief from CDR by a paragraph included in the contract on the subject of “iterative lifecycle development.” This team engages in iterative design reviews, and the associated program office is reported to be planning to execute an ACAT 1D program in the future with alternative life-cycle language that will not require CDR.

Several respondents reported negotiating for requirements (expressed as capabilities to evolve under the Agile paradigm) and then time-boxing their Agile increments against CDR targets. This allows program offices to plan against the CDR milestone but still gives flexibility to the Agile team to define increments and processes and evolve the software toward the desired capabilities.

Some program leadership, however, will not allow tailoring of program milestones. In these cases, Agile practitioners report to being stuck in water-Scrum-fall situations where they can be Agile up to a point before hitting a phase gate. Program leadership in these cases is not supportive of conducting the “early and often” iteration-level reviews preferred by Agile software teams. Resistance to tailoring of program milestones is discussed further in Section 4.

3.10 Summary

As observed in Section 1.3, the responses we received focused on the service facet of systems engineering, involving communication and coordinating important information. Our interviews demonstrated a wide array of characteristics of successful implementation when leveraging Agile principles such as continuous automation/integration and continuous collaboration with multiple kinds of stakeholders. Combined with a commitment to training and cultural change on the part of the acquirer, developer, and systems engineering team, these principles can offer powerful opportunities to expand insight and oversight on the software development in broad yet noninvasive ways.

Pockets of resistance remain, in which existing practices, policies, and trust have not caught up or kept pace with the rapid delivery capabilities associated with Agile. In situations where there is little management support, a lack of training, or a lack of willingness to explore legitimate tailoring options for acquisition milestones, the use of Agile methods may lose some of its efficiency due to the need to rework additional deliverables or wait for the completion of water-Scrum-fall cycles.

The next section of this report discusses the role of existing regulation and policy as it limits or encourages the use of Agile development methods.

4 Impact of Policy and Guidance on Agile Software Implementation

This section examines how acquisition policy and guidance sometimes raise artificial barriers to the implementation of Agile software and systems engineering practices in both DoD organizations and for DoD contractors.

4.1 DoD Policy

While DoD policy does not specifically prohibit Agile software and systems engineering methods, neither does it make Agile methods easy to implement. No examples in the current 5000 series (2008 version) highlight the use of Agile methods to satisfy acquisition requirements. This is most clearly visible in the pervasive use of classic waterfall-style technical design reviews in describing (and funding) developmental progress for DoD programs [Lapham 2010]. Kennedy and Broadus have also written about this challenge [Broadus 2013, Kennedy 2013].

DoD 5000.02 discusses the requirement to conduct preliminary and CDRs *at the system level* before major milestone reviews, but the mandatory conduct of software-centric preliminary and CDRs is not specified [OSD(AT&L) 2008]. However, it is only in lower echelon commands where the broader spectrum of systems and software engineering technical reviews is examined. This is the condition that has allowed things like incremental PDR and CDR events to be feasible in acquisitions that traditionally perform software PDRs and CDRs.

4.2 DoD Regulations

DoD regulations implement DoD policy and are best examined at the Echelon 2 level, that is, at the budgetary level that exists just below the Pentagon. It is at this level that higher level policy and regulations within DoD are put into actionable documents for use by the various services. Different services respond to these regulations in different ways. The Air Force, for example, is in the process of updating one of its acquisition pamphlets and is planning to include an explicit chapter on use of Lean and Agile approaches. In federal agencies such as the Department of Homeland Security, directives from its CIO have been used to incentivize the use of Agile methods without explicitly reframing their policy guidance.

4.3 Tailoring Program Approaches to DoD Policy and Regulation

Our interview respondents consistently indicated that they felt that resistance to the employment of Agile, or to the tailoring of reviews and documentation, did not stem from requirements laid forth in the 5000 series. Rather, they indicated that as the guidance became more "local" or granular (making its way down from the policy level to the implementation level within the program office), program office staff continued to make more and more conservative interpretations of what is allowed under the policy. Thus while researchers and practitioners have repeatedly argued that the 5000 series does *not* explicitly exclude the use of Agile methods [Lapham 2011, NDIA 2011], these more conservative approaches taken at lower echelons seem to stem from a lack of awareness or even fear of tailoring the activities in a manner that deviates from waterfall-based activities. Several interviewees agreed with the characterization of acquisition staff as wanting to take the “safe” path when constructing and implementing an acquisition strategy, which leads to the more conservative interpretations of higher level guidance.

4.4 DoD Case Study – Project A

The remainder of this section details the process followed by a major DoD software development effort herein referred to as Project A, to propose a technical review schema that would permit the use of Scrum/Agile while maintaining the technical oversight desired by the program executive office (PEO).

While examining Project A as a case study, keep in mind that Scrum/Agile was implemented very late in the spiral development of the program. The majority of the software implemented from the complete design had previously been completed using waterfall methods. System and software architectures, mission threads, development tools, and other project components had been established for more than five years before the program adopted the Scrum/Agile model.

Key to any DoD program following CMMI-DEV[19] guidelines in a traditional way is the establishment of the software development plan (SDP) and the SEMP/requirements development plan. These plans are expected to include a discussion of the formal technical reviews required for successful product development. Additionally, most DoD contracts will invoke locally prepared standards for conducting formal reviews. In the Project A case, this standard is reflected in the Technical Review Manual (TRM) for the PEO. At the time Project A chose to adopt Scrum/Agile, the TRM followed the classic waterfall development approach for the software reviews, outlining entrance and exit criteria as well as boilerplate review content, for the software specification review, software PDR, software CDR, and test readiness review to be conducted for each software delivery. A common theme for each of these reviews is that they served as “gates” to proceed to the next phase of development.

In order to succeed, Project A recognized that it must break down the traditional TRM approach and implement a novel approach for complying with the *intent* of the TRM while maintaining a Scrum/Agile development framework. Early in the proposal development phase, Project A developed a schema for mapping the entrance and exit criteria and artifacts required by the TRM into new reviews that would enable Scrum/Agile to flourish. This schema was briefed to the government program management office (PMO) with the specific intent of providing it with the ammunition to carry the battle to the stewards of the TRM.

Not only were technical reviews used by the PMO to approve entry into the next phase of development, but also successful completion of those reviews within the planned fiscal quarter was used by budget controllers as a first-order assessment of program health. Complete elimination of PDR, CDR, and other review activities would not support existing budgetary documents and would therefore need to be retained.

[19] CMMI®-DEV, or Capability Maturity Model® Integration for Development, is a framework of engineering and management practices for system and software development that have been successfully used in multiple industrial and government contract settings [Chrissis 2011]. The model itself is accompanied by evaluation methods that can result in an indexed rating of achievement that is often used as an economic gate for bidding on a proposal.

Project A adopted many of the suggestions from Lapham in defining the following technical reviews in both the SDP and the SEMP [Lapham 2011]:

1. **System Design Review (SDR):** The system design review (SDR) is an informal review conducted to establish the vision and roadmap for the contract. The SDR also covers program of record (POR) dependencies.
2. **Product Backlog Review (PBR):** The product backlog review (PBR) is an informal review conducted to establish the system-level product backlog as a baseline for measuring growth within the software release. Entrance criteria and review items from traditional SSR, PDR, and CDR events are allocated to the PBR based on Project A implementation of Agile. The review, conducted in two phases early in the development cycle, includes
    a. identification of the proposed allocated baseline (new capability or POR) to the elements
    b. system threat description
    c. a review of the architecture of the system identifying where the ensembles fit within the overall system capabilities and the roles they play relative to the overall system requirements. This overview is necessary to ensure that all attendees have the same perspective on the context and role of product features.
    d. a review of POR dependencies
3. **Incremental Design Review (IDR):** An incremental design review (IDR) is an informal review to allow management to demonstrate closure on capability development during a development cycle. They are scheduled every two months (approximately) and are used to incrementally address the PEO TRM criteria. These reviews provide insight into the progress made across Scrum teams and show convergence of design artifacts leading to delivery of capabilities in the upcoming release. IDRs will include
    a. review of action item status from previous IDRs or capstone tech review[20]
    b. updates to cost, schedule, and performance risks
    c. review of results of current risk analyses and risk mitigation steps
    d. system capability progress (features completed, etc.)
    e. summary of new safety impacts from recent sprint cycles where requirements have been developed
    f. software measures including cost (plan vs. actual), schedule (plan vs. actual), software size estimates (including ESLOC,[21] DSLOC[22] for release), and quality (defect data)
    g. objective quality evidence, updated by integrated product teams (IPT)s and independent verification and validation for products developed during the prior two sprint cycles
4. **Software Specification Review (SSR):** The software specification review (SSR) is a formal review that can be held as a capstone event at the end of requirements development for the

[20] In this case study, capstone tech review refers to a review with the customer held at the end of a development cycle focused on architecture and design.

[21] ESLOC = equivalent source line of code

[22] DSLOC= delivered source lines of code

release, or at a designated point between the product baseline review and the software PDR. For a capstone event conducted after all sprints have completed, as planned for Project A development cycles, the review is an executive summary of the IDRs relative to the software requirements and interfaces.

5. **Software Preliminary Design Review (S-PDR):** The software preliminary design review (S-PDR) is a formal review of development progress conducted at about the one-third point of the development for the release. The scope of the S-PDR is expanded to include a PDR for each release in the development cycles and will include critical PORs. The S-PDR, as part of the PDR, provides the certification team with an in-phase assessment leading to mission readiness assessment for Project A. Since the review will be conducted before all design is completed, the product backlog will document the features for which design activities remain. A statement of finding and memo to the PM and certifying official will be completed.
6. **Software Critical Design Review (S-CDR):** The software critical design review (S-CDR) is a formal review of development progress conducted at about the two-thirds point of the development cycle for the release. The scope of the S-CDR is expanded to include a CDR for each release in the development cycles and will include critical PORs. The S-CDR, as part of the CDR, provides the certification team with an in-phase assessment of the functionality leading to mission readiness assessment for Project A. Since the review will be conducted before all design is completed, the product backlog will document the features for which design activities remain. A statement of finding and memo to the PM and certifying official will be completed.

Table 2 is an excerpt of the schema proposed that demonstrated complete mapping of the PEO TRM to the proposed tailorings.

Table 2: Mapping Schema Example from TRM to Project A Scrum/Agile

Technical Review Manual (TRM)	Program A	Program A Development Cycles
Documentation is complete for software requirements (SWR), software processes, and tools. If an incremental approach is being used, the documentation must be complete for the increment.	All applicable element architecture description documents (EADDs)/ component architecture description documents (CADDs) are under configuration control by the Architecture Review Board (ARB) (in accordance with applicable program directives and work instructions) and contain the allocation of higher level requirements from element/component specifications to components or ensembles/ software configuration items (SCI).	EADDs/CADDs under configuration management (CM) by the ARB
Software metrics have been collected and are ready for presentation	The SWR Cross Product Team (CPT) has reviewed and analyzed the commitments for the release documented in the Software End-to-End Display and created a Release Vision Document that has been internally agreed to.	Product backlog review (PBR)
Software-related item performance specifications validated	Allocated functional, nonfunctional, and design constraints have been entered into the SRSs and placed in the DOORS database.	Preliminary at PBR 1. Updated at subsequent IDRs
SCI performance specifications validated	Traceability between SWR specification and parent element specifica-	Software ensemble requirements are derived from the sprint and

Technical Review Manual (TRM)	Program A	Program A Development Cycles
	tions has been completed.	product backlogs
Cost, schedule, and performance risk identified, quantified, and prioritized		Preliminary at PBR-1 updated at subsequent IDRs
Software RMP established and risk analysis performed		Preliminary at PBR-1 updated at subsequent IDRs
Systems Engineering: i. Common operating environment (COE) performance analysis performed ii. Functional architecture for embedded modeling & simulation (M&S) developed iii. Functional architecture reviewed for system safety-critical functions		PBR

With the foundation of this TRM-to-Project A technical review strategy, as shown in Table 3, the program was able to start development with the full support of the PMO.

Subsequent to the first PDR (for the first of two releases), the government Technical Authority caucused with the PMO to provide feedback on the value of conducting these capstone reviews and proposed eliminating the remaining PDR, CDRs, and SSRs in lieu of a slightly expanded, quarterly IDR.

Table 3: Project A Technical Review Strategy

Review (per Major Release)	Agile
Software Specification Review (SSR)	Apportioned to PBR and IDRs with capstone SSR event per formal delivery
Internal SSR (ISSR)	Allowed per SWI; N/A to Agile
Software Preliminary Design Review (S-PDR)	Apportioned to PBR and IDRs with capstone PDR event per formal delivery
Software Critical Design Review (S-CDR)	Apportioned to PBR and DRs with capstone CDR event per formal delivery
Test Readiness Review (TRR)	No impact for Project A - Continuous integration per sprint not currently planned
Phase Completion Reviews	Feature completion checklist

4.5 Conclusions

Conservative interpretations of acquisition guidance still err on the side of assuming that Agile is incompatible with DoD regulations, but we have repeatedly demonstrated that this assumption is inaccurate. While researchers and practitioners have repeatedly argued that the 5000 series does *not* explicitly exclude the use of Agile methods [Lapham 2011, NDIA 2011], these more conservative approaches taken at lower echelons seem to stem from a lack of awareness or even fear of tailoring the activities in a manner that deviates from waterfall-based activities. Our respondents concurred with this characterization.

Early and frequent interaction is the key to gaining buy-in from DoD stakeholders. As demonstrated in the Project A case discussion, any plan for adopting Scrum/Agile must gain approval not only from the PMO but also from the technical oversight community for the program. Since no two programs are alike, implementation of Scrum/Agile will undoubtedly be different from program to program. While subsequent programs may not have to deal with the novelty of

Scrum/Agile to the DoD technical community, they will certainly need to plan on the "That's not the way we did it on Project X" factor. The systems engineering community is a key stakeholder in the technical reviews cycle, so getting involvement and support is a key element to consider when trying to implement Agile methods in software settings that include systems engineering (essentially with any medium to large project).

The next section of this report briefly discusses our observations of ways in which contract language and structure impact Agile software development and systems engineering.

5 Contracts

The contracting or agreement structure utilized by a program or project has an effect on how Agile the project can be and significantly affects software and systems engineering processes, documentation, and deliverables. We asked respondents to characterize the contracting situation on their programs and discuss pain points or benefits they saw to those approaches, especially as they regarded interfaces to systems engineering components.

Analysis of our survey and interview responses indicates that currently many software development teams appear to be using existing (previously in-place) contractual structures while implementing Agile to get improved productivity or visibility for their own purposes. They are providing data and documentation deliverables in the format and frequency required under the contract, which may involve translation of data and the development of additional documentation that represents little to no value-add to the software development aspects of the project. One respondent indicated (as discussed in Section 3.1) that automation activities were very valuable in this regard—while documentation required under the contract was "above and beyond" the documentation necessary to actually execute the software development, careful automation on the project allowed the developer to automate the generation of most of the contractually required documentation products at a level of detail satisfactory to the government customer.

This section contains a summary of our observations on contract structure within the scope of our discussions on Agile software teams and systems engineering. Future research is planned to specifically assess the viability of various contract types and structures for use in Agile settings.

5.1 "Agile" on Contract?

In most cases we have observed in the course of researching this report, Agile is not inherently being requested by the customer. This isn't necessarily a bad thing—as we mentioned in Section 3.3, if a program asks for "Agile" without understanding what that means, miscommunication and misalignment of expectations are likely results. Contracting and other acquisition processes have not caught up to the new methods that software developers use to execute those processes.

Some teams have been able to get support or direction from their PMO and contracting officers to achieve the next stage of adoption by ensuring that iterative development with frequent demonstration is included in the contract, or that the contract language supports Agile principles and process flows. These projects have seen improved agility because of the overall program structure included Agile from the beginning. These projects also exhibited a pattern of more involved stakeholders who have been educated regarding Agile principles and provide "top cover" for Agile use.

5.2 Organic Software Development Organizations

We define "organic" software development as that which is executed by a software development entity that exists within a government organization. Examples of organizations that house organic development capabilities include, but are not limited to, the Air Force's Ogden Air Logistics

Complex, the Army's Aviation and Missile Research Development and Engineering Center Software Engineering Directorate, and the Naval Air Warfare Center Aircraft Division.

While organic development organizations do not have formal contracts with acquirers, a memorandum of agreement (MOA) or memorandum of understanding (MOU) is generally put in place between the acquirer and the developer, with a statement of work (SOW) or statement of objectives (SOO) negotiated between the parties. Respondents indicated that in these situations more flexibility generally exists in the development of the agreements than in situations involving external contractors, but there was no pattern in our data to suggest that arrangements with organic development organizations are any more or less successful in establishing positive linkages with systems engineering functions than those arrangements that involve external development contractors. Agile software engineering efforts did not appear to be any more or less successful when the developer represented an organic capability. Respondents from, or who dealt with, organic development organizations reported challenges and opportunities similar to those reported by developers and acquirers in contract situations. They faced the same challenges with regard to deliverables and milestones as their contractor counterparts. We address some of these issues briefly in the next section.

5.3 Effect of Contract Type and Structure on Agility

Interview respondents for contract development teams (those development teams that are external entities such as civilian contractors, FFRDCs,[23] UARCs,[24] etc.) reported using a variety of contract structures including indefinite delivery/indefinite quantity (IDIQ), cost plus award fee (CPAF), cost plus incentive fee (CPIF), and firm fixed price (FFP). While there was no direct corollary between the contract type and the apparent success of Agile engineering methods, most interview respondents indicated that it was more the specific language, artifacts, and oversight required within the contract that were the key factors into their ability to become more agile.

Several respondents indicated a preference for IDIQ arrangements with CPAF, CPIF, or time and materials (T&M)-based task orders as a means to streamlining the time, effort, and cost required to place capability requirements on contract and begin development. By putting "undefined epics" of varying size (small, medium, or large) on contract, two of our respondents were able to rapidly engage with Agile software developers to deliver capability. It was noted that this was particularly beneficial in the case of user interface (UI) development, in which this strategy enabled rapid engagement with user representatives to fine-tune the UI as it was developed, rather than engaging in formal ECPs.

While IDIQ arrangements are popular for getting capability on contract and providing transparency, the key factor for enabling agility is the deliverable and reporting structure called out in the contract, and not the specific contract type. Agile is being used, with positive results, on contracts ranging from FFP to straight T&M. Multiple respondents reported using different means of earn-

[23] FFRDC: Federally Funded Research and Development Center (e.g., MITRE, the RAND Corporation, MIT Lincoln Labs, Aerospace Corporation)

[24] UARC: University Affiliated Research Center (e.g., Johns Hopkins University Applied Physics Laboratory, Georgia Tech Research Institute, Stevens Institute Systems Engineering Research Center)

ing value based on completion of stories or iterations, as a means of successfully meeting earned value management reporting requirements.

Milestones and CDRLs are also components of the Agile contract. Not many respondents identified that they had to meet milestones such as technology readiness assessments; however, many were required to adhere to the review cycle that is traditional with waterfall-oriented contracts as defined by data item descriptions—SDR, PDR, CDR. Many teams were, as discussed in Section 3.9, able to get concurrence to tailor these reviews with the Agile components that provide visibility into the project via product demonstrations.

Many of the respondents, however, indicated that Agile was not integrated into the contract in any way. This pattern indicates that there could be more for program office teams to learn in regard to how to structure a contract—and modify program office culture—to leverage the promise of Agile/iterative development methods. Again, this is intended to be the subject of a future report.

5.4 Effect of Stakeholder Engagement

Most respondents indicated that stakeholder engagement and understanding of the Agile process was a key factor in the success of the contract. This can stem from good language in the contract, to a trusting relationship with the PMO, to support from management, as well as to the availability and engagement of end user representatives across the development cycle.

5.5 Exercising Oversight

Not all respondents provided information on the method of oversight that was employed. Several interviewees expressed concern with the traditional measurements required by earned value management (EVM) and finding ways to support EVM in relation to the execution of Agile projects. Some expressed success in statusing EVM at the epic level, sizing backlog via story points, and earning value for iterations when the iterations are completed. A couple of interviewees identified providing status on the contract at the epics level, which seemed to provide enough flexibility to enable agility but also to provide the customer with enough insight into the scope of capability that was being implemented. We have previously addressed the use of metrics in monitoring Agile software efforts on DoD projects [Lapham 2011] and will do so in greater detail in a future technical note.

The next section of this report contains a brief retrospective from our interviews.

6 A Retrospective as Part of Our Interviews

One of the most important aspects of an Agile implementation is the use of the continuous improvement vehicle of retrospectives, or lessons learned. Our interview approach included an opportunity for retrospective on the topic at hand and the respondent's experiences. In the interviews conducted, there were a variety of answers to the question "If you could change one thing about Agile interaction with systems engineering, what would it be?"

Several themes surfaced to include

- improved systems engineering discipline understanding of what Agile methods entail
- improved communicating and reporting of progress for Agile teams
- alignment of Agile with policies and processes
- systems engineering process adaptation for Agile
- cultural change necessary for the DoD to fully realize the promise of Agile

For teams integrating systems engineers into the software team's Agile execution, in many cases the systems engineers lacked a basic understanding of how Agile would change the way they completed their assignments. Respondents concluded that there is a major paradigm shift needed for the systems engineers to understand. One suggestion was to target an accomplished Agile software developer to become a systems engineer. This engineer could be assigned to the systems engineering organization and provide leadership on how to be an Agile engineer.

With respect to program progress reporting, respondents indicated a desire to improve the alignment of DoD program reporting requirements with the data available to an Agile development team. The pace of execution and task completion could be better communicated using some of the metrics available through an Agile method, such as cumulative flow diagrams. Reporting program progress and success using traditional methods is not necessarily providing an accurate glimpse of the true status. Earned value management inherently penalizes the iterative nature of Agile requirements discovery, if value is "earned" is based on specific preset requirements rather than on the delivery of capability in a sprint or even a release. Multiple respondents have indicated that up-front negotiation of "what does *done* mean?" for each sprint can give Agile software teams the ability to report earned value in a meaningful way.

As we dug deeper into the tasking of the systems engineers on an Agile development team, many respondents voiced the opinion that the method of preparing technical data such as SRS should be re-evaluated. One suggestion was to move toward user stories as a means to complete the requirements definition. Another recommendation was to envision the systems engineers executing the systems engineering "V" iteratively throughout each sprint.

The final category interviewees felt should be changed is a better alignment of Agile methods with approved DoD program management policies such as earned value management, and DoD 5000, and making the 5000 series "more Agile-friendly." Currently, trying to remain compliant with the policies and allowing the flexibility of Agile development are perceived as being at odds with each other. Attempting to iteratively develop solutions and continue to prepare for heavily document-driven waterfall milestones such as PDR and CDR are in conflict if the acquirer cannot work with the developer to envision an appropriate tailoring strategy. Several respondents suggested publishing a revision of these policies to fully and explicitly embrace the flexibility of Agile development. [25]

[25] A new draft of DODI 5000.02 is reported to address these issues. At the time of this publication, that document was still unpublished.

7 Research Approaches and Program Demographics

This report uses grounded theory, a qualitative research approach prevalent in the investigation of practice adoption, as the foundational research approach for accomplishing this work.[26] It is based on a literature search, surveys, and interviews conducted with Agile practitioners in the DoD and federal agency acquisition environment. We interviewed and surveyed both government employees and practitioners employed in the commercial sector to work on government-funded programs. This section describes the data collection via survey and interview, and summary characteristics of the programs our respondents support. Since this is a practitioner-focused report, we have omitted notes on the generation of the issue categories via grounded theory methods.

7.1 Online Survey

An online survey was distributed to members of the SEI's Agile Collaboration Group. Other potential respondents were identified through professional interactions, conferences, and the collaboration group members. The survey was conducted anonymously, and no identifying information about the 16 respondents or acquisition programs was collected. Respondents were permitted to opt out of answering any individual questions.

The next few subsections contain snapshots of the demographic data associated with the respondents' program responses.

7.1.1 Lifecycle

Respondents were asked to categorize the development projects as "new" system development or "sustainment activities" that could include corrective maintenance, tech refresh, and the inclusion of new capability in a fielded system.

- new: 5
- sustainment: 6
- legacy upgrade: 1
- other: 4

Respondents who selected "other" reported they were working with multiple programs in different life-cycle phases or were engaged in a combination of new and sustainment development.

7.1.2 System Type

Respondents were asked to characterize the type of system for which the software was being developed.

- embedded software in a weapon system: 3
- other embedded system: 2

[26] For those interested in grounded theory as a research methodology used in software engineering practice adoption, see Adolph [Adolph 2011].

- space systems: 1
- IT/business systems: 2
- C4ISR/C2: 2
- other: 6

Respondents who indicated “other” said they were working on training systems, modeling and simulation systems, and multiple kinds of systems and engaging in operational test.

7.1.3 Type of Development Organization

Respondents were asked to classify the body performing the software engineering tasks as either contractors, organic government agency teams, or other approaches.

- organic (government employees): 2
- contractor-only: 6
- combination organic/contractor: 7
- other: 1

7.1.4 Contract Type/Structure

When a respondent indicated that nonorganic development teams were in place, we asked about the structure of the contracts.

- 1 reported using CPAF contracts
- 5 reported using CPIF contracts
- 3 reported using FFP contracts
- 1 reported using “all except service contracts”

7.2 Interviews

The SEI research team conducted 10 interviews (in person and via teleconference) with 12 respondents (1 interview featured multiple respondents). Responders were volunteer participants identified via their participation in the SEI’s Agile Collaboration Group and other professional interactions. Respondents reported experience on more than 20 unique programs. Many respondents provided data on their primary current work but reported anecdotes about their experiences on prior projects. Respondents were permitted to opt out of answering any individual questions. Each respondent was assigned a unique identifier, and identifying information about respondents was not stored with the notes on their responses.

The next few subsections contain snapshots of the demographic data associated with the respondents’ program responses. We expect that there was some degree of overlap between survey and interview respondents.

7.2.1 Lifecycle

Respondents were asked to categorize the development projects as “new” system development or “sustainment activities” that could include corrective maintenance, tech refresh, and the inclusion of new capability in a fielded system.

- new: 5 programs
- sustainment: 7 programs
- Multiple respondents indicated a "combination of both" new development and sustainment development in many instances when discussing multiple programs.

7.2.2 System Type

Respondents were asked to characterize the type of system for which the software was being developed.

- embedded software in a weapon system: 5
- IT systems: 2
- C4ISR/C2: 3
- other: 3, one of which was a classified, unidentified system
- One joint program was identified in one of the categories noted above.

7.2.3 Acquisition Category (ACAT)

Respondents were asked to identify the ACAT level of the programs in question. Most respondents chose not to answer this question.

- ACAT I: 2
- ACAT II: 1
- ACAT III: 1

7.2.4 Type of Development Organization

Respondents were asked to classify the body performing the software engineering tasks as either contractors, organic government agency teams, or other approaches.

- organic (government employees): 3
- contractor-only: 3
- other: 4

Respondents who indicated "other" discussed a variety of models including FFRDC support, hybrid government–contractor teams, and programs in which different capabilities were developed by different kinds of teams (e.g., a government team developed one software component, and a contractor team developed another).

7.2.5 Contract Type/Structure

When a respondent indicated that nonorganic development teams were in place, we asked about the structure of the contracts.

- 4 respondents indicated leveraging IDIQ vehicles
- 5 reported using cost-based contracts (i.e., cost plus fixed-fee, CPIF, or T&M)
- 3 reported using FFP contracts (notably 2 of those did report migrating over time to other types of vehicles based on limitations they associated with the FFP model)

7.2.6 Raw Questionnaire

The template questionnaire for these interviews is included in this report as Appendix A, Interview and Survey Questions.

8 Summary

Agile software development techniques are gaining a foothold within DoD programs. The DoD's acquisition guidance is, however, largely focused from a systems engineering orientation derived from a hardware-centric product model. As such, Agile software practitioners often experience tension in integrating their work practices and products with those of systems engineers operating on non-Agile teams.

We envisioned three cases of Agile interactions with systems engineering teams:

- Agile software teams interacting with traditional systems engineering
- systems engineers acting as Agile team members
- systems engineers applying Agile methods to their own work

Our interviews identified successful examples of each of these three cases in current programs, and we have documented case studies relevant to them. Throughout our discussions, it was apparent that the greatest opportunities for successful Agile implementations occurred when systems engineering functions were at least aware of and engaged with the Agile processes.

Through surveys and interviews we identified successes and challenges in nine key areas:

- automation
- insight/oversight
- training
- role of sponsors, advocates, and coaches
- pilot programs
- stakeholder involvement
- defining and evolving requirements
- verification and validation
- aligning the program roadmap with Agile increments

Successful deployment of Agile methods and successful interactions with systems engineering teams generally relied upon a combination of factors from the above areas. For example, automation of many software development activities provided enhanced oversight for program offices, which in turn improved trust in the software team and the methods and encouraged improved stakeholder collaboration. Likewise, challenges in any of the above areas tended to ripple outward: if a software team did not receive leadership and budgetary support to pursue appropriate automation techniques, this might both result in increased time to produce contract-required documentation and degrade the capability of the software team to effectively communicate with systems engineers and testers regarding the development status and quality of software code.

While DoD policy and guidance do not prohibit the use of Agile development methods, we find that in many cases acquisition teams are not comfortable with pursuing legitimate tailoring options to align the Agile methods and work products with the program's roadmap and milestones.

Resistance to this alignment tends to occur, by report of our interviewees, at more local levels. Improved training for program office teams, continued reporting of pilot project successes, and the reported revision of DoDI 5000.02 may help program office teams to be more comfortable with the tailoring options legitimately available under DoD policy.

As part of our analysis, we briefly explored the contract vehicles under which Agile practitioners were operating and found a wide variety in place. Few contracts specifically requested Agile methods (and those that did, misunderstood the terminology), but we did gain insight into what may constitute Agile-friendly contract provisions. Future work is planned to analyze this topic in depth.

We provided a retrospective that offered, in their own words, our respondents' wish lists for improving the state of Agile practice within the DoD and also provided detailed demographic breakdowns of our survey and interview respondents to demonstrate the cross-section they provided of the DoD acquisition landscape.

The research conducted for this report provided insights into areas on our future research agenda. Deeper research is under way regarding the appropriate use of metrics for reporting and managing programs employing Agile software development, and on DoD contract vehicles and provisions that support the use of these methods. Additional reports and technical notes will follow on these topics in the near future.

We hope that Agile software practitioners will find in this report potential solutions to challenges they are experiencing and opportunities to improve communications with program offices, systems engineers, and other stakeholders. As more government programs see the potential benefits of leveraging Agile, we hope that practitioners will continue to share their experiences with us and with each other.

Appendix A Interview and Survey Questions

The following question template was used to guide interviews with Agile practitioners. A subset of the template was used to conduct the anonymous online survey.

1. What type of system does this program acquire?
    - Weapon system
    - Space
    - IT
    - C4ISR
    - Other/ describe

2. What best describes the program's current lifecycle phase?
    - New system development
    - Sustainment
    - If in sustainment, is this a legacy system?

3. What is your annual budget?

4. Can you describe the project/program in terms of:
    - Approximate size (e.g. SLOC, ESLOC, etc.)
    - Relative complexity (H, M, L)
    - Duration (deliverable cycle and age of product)
    - Requirements stability (stable, frequent changes, infrequent changes?)

5. Which teams are using Agile methods?
    - Software
    - Systems engineering
    - Both
    - Other/explain

6. Is someone within the program office performing the role of Agile advocate/sponsor? If so, what do they do?

7. What best describes your role/experience regarding Agile interfacing with Systems Engineering on this program?
    - Systems engineering practitioner
    - Agilist who interacts with systems engineering teams
    - Systems engineering manager
    - Agile consultant
    - Government program manager

- Agile advocate/sponsor
- Specialty/other/specify (e.g. system test, supplier)

8. What is the average length of an Agile iteration/sprint on this project?

9. How did Agile methods come to be used on this effort?
   - Pilot program (directed by a sponsor)
   - Pilot program (initiated by software team)
   - Software team was already experienced with Agile methods/already an “Agile shop”
   - Don’t know
   - Other/specify

10. How large is your Agile development team (headcount)?

11. Is the development team a contract team, or an organic government shop?
    - Contractor
    - Organic government
    - Combination government/contractor
    - Other/describe

    If the answer to the above is “contractor” or “combination”, then what kind of contract vehicle is in use?
    - Fixed-fee
    - Cost-plus incentive fee
    - Cost-plus award fee
    - Other/misc/describe

12. Who are the program’s major stakeholders?

13. What other non-agile external teams do you interface/integrate with? (NOT systems engineering functions)

14. How are your team’s interfaces with systems engineering functions defined?

15. Can you describe a successful systems engineering activity in an Agile setting?
    - There exists successful protection of the Agile dev team from traditional systems engineering
    - Getting systems engineers to “do”/adopt Agile
    - Getting better interaction/engagement between a systems engineering team and an agile development team
    - SE actively participate as a team member on an Agile Dev team

  - Additional characterization (please describe)

16. What opportunities have your agile and systems engineering teams faced in interacting with each other in each of the following phases?
    1. Sys Eng traditional lifecycle:
    2. Requirements & analysis
    3. Architecture & Design
    4. Implementation
    5. Integration
    6. System Testing
    7. Acceptance/operational testing
    8. Deployment
    9. Sustainment
    10. Retirement/disposal

17. What user experience models really work in your experience, in…. ?
    - Product visioning
    - Product roadmap
    - Release planning
    - Sprint planning
    - Sprint execution
    - Sprint review/demo
    - Release review/demo
    - Deployment
    - Sustainment
    - Retirement/disposal

18. What approach do you use to support collaboration of systems engineers with the customer, internal system developers, and external parties?

19. How do you align your program roadmap and system architecture with Agile increments?

20. How are Agile processes incorporated into contractually binding documents?

21. How are standard milestones criteria in the acquisition lifecycle in systems engineering (e.g. CDR) met?

22. If you are program office staff (disregard if otherwise), how has your approach to government oversight changed, if at all) with the involvement of an Agile team?)

23. How do existing federal and/or local policies/regulations support success in integrating Agile and Systems Engineering?

24. For agile and systems engineering to gracefully cohabit within the DoD world, would policy modifications improve the situation/reduce friction? If so, what and how?

25. If you could change one thing about agile interaction with systems engineering, what would it be?

Appendix B Cultural Expectations

Table 4 is reproduced from our 2011 report, *Agile Methods: Selected DoD Management and Acquisition Concerns* [Lapham 2011]. It lists expectations of cultural elements expected in development and acquisition organizations on DoD programs, with the adoption of Agile methods. This data is based on observations of actual programs adopting Agile.

Table 4: Comparison of Agile and Traditional DoD Cultural Elements [Lapham 2011]

	Agile DoD	**Traditional DoD**
Organizational Structure	Flexible and adaptive structures Self-organizing teams Collocated teams or strong communication mechanisms when teams are distributed	Formal structures that are difficult to change Hierarchical, command-and-control-based teams Integrated product teams that have formal responsibilities
Leadership Style	Facilitative leadership Leader as champion and team advocate	Leader as keeper of vision Leader as primary source of authority to act
Rewards System	Team is focus of reward systems Sometimes team itself recognizes individuals	Individual is focus of the reward system
Communications & Decision Making	Daily stand-up meetings, Frequent retrospectives to improve practices Information radiators to communicate critical project information Evocative documents to feed conversation “Just enough” documentation, highly dependent on product context	Top-down communication structures dominate External regulations, policies, and procedures drive the focus of work Indirect communications, like documented activities and processes, dominate over face-to-face dialogue Traditional, representational documents used by the PMO throughout the development lifecycle to oversee the progress of the developer PMO oversight tools focused on demonstrating compliance vs. achieving insight into progress
Staffing Model	Cross-functional teams including all roles across the lifecycle throughout the life span of the project Includes an Agile advocate or coach who explicitly attends to the team’s process	Uses traditional life-cycle model with separate teams, particularly for development and testing Different roles are active at different defined points in the lifecycle and are not substantively involved except at those times

References

URLs are valid as of the publication date of this document.

[Adolph 2011]
Adolph, Steve, Hall, Wendy, & Kruchten, Philippe. "Using Grounded Theory to Study the Experience of Software Development." *Journal of Empirical Software Engineering 16*, 4 (2011): 487-513.

[Agile Alliance 2001]
Agile Alliance. *History: The Agile Manifesto*. http://agilemanifesto.org/history.html (2001).

[Ambler 2012]
Ambler, Scott. *Disciplined Agile Delivery: A Practitioner's Guide to Agile Software Delivery in the Enterprise.* IBM Press, 2012.

[Asan 2013]
Asan, Emrah & Bilgen, Semih. "Agility Problems in Traditional Systems Engineering—A Case Study." *Complex Systems Design and Management* (2013): 53-71.
http://link.springer.com/chapter/10.1007%2F978-3-642-34404-6_4#page-2

[Bellomo 2012]
Bellomo, Stephany; & Woody, Carol. *DoD Information Assurance and Agile: Challenges and Recommendations Gathered Through Interviews with Agile Program Managers and DoD Accreditation Reviewers* (CMU/SEI-2012-TN-024). Software Engineering Institute, Carnegie Mellon University, 2012. http://resources.sei.cmu.edu/library/asset-view.cfm?AssetID=34083

[Berkeley Security 2013]
Berkeley Security. *Recommended Resources for System Hardening.*
https://security.berkeley.edu/node/143?destination=node/143 (2013).

[Broadus 2013]
Broadus, William. "The Challenges of Being Agile in DoD." *Defense AT&L* (January-February 2013): 5-9. http://www.dau.mil/pubscats/ATL%20Docs/Jan_Feb_2013/Broadus.pdf

[Brooks 1975]
Brooks, Fred. *The Mythical Man-Month: Essays on Software Engineering*. Addison Wesley, 1975.

[Campbell 2004]
Campbell, Grady. *SIS Acquisition: Reconsidering the Role of Systems Engineering in DoD Software Problems.* Software Engineering Institute, Carnegie Mellon University, 2004.
http://resources.sei.cmu.edu/library/asset-view.cfm?assetid=19270

[Campbell 2010]
Campbell, Grady. *The Illusion of Certainty.* Software Engineering Institute, Carnegie Mellon University, 2010. http://resources.sei.cmu.edu/library/asset-view.cfm?assetid=20918

[Chrissis 2011]
Mary Beth Chrissis, Mary Beth, Konrad, Michael D., & Shrum, Sandra. *CMMI for Development: Guidelines for Process Integration and Product Improvement*, 3rd ed. Addison-Wesley Professional, 2011.

[DoD 2011a]
Department of Defense. DTM 11-009, Directive-Type Memorandum (DTM) 11-009, *Acquisition Policy for Defense Business Systems (DBS)*. DoD, 23 June 2011.
https://dap.dau.mil/policy/Documents/2011/DTM%2011-009.pdf

[DoD 2011b]
Department of Defense. *A New Approach for Delivering Information Technology Capabilities in the Department of Defense*. DoD, November 2010.
https://acc.dau.mil/adl/en-US/412545/file/54776/New%20Acquisition%20Process_OSD%2013744-10%20-%20804%20Report%20to%20Congress%202.pdf

[Garcia 2006]
Garcia, Suzanne & Turner, Richard. *CMMI Survival Guide: Just Enough Process Improvement*. Addison-Wesley, 2006.

[Garcia-Miller 2010]
Garcia-Miller, Suzanne. "Treating Systems Engineering as a Service," 115-122. *CMMI for Services: Guidelines for Superior Service*. Pearson Education, 2010.

[Hendrickson 2008]
Hendrickson, Elisabeth. *Driving Development with Tests: ATDD and TDD*. Test Obsessed, 2008. http://www.testobsessed.com

[Kennedy 2011]
Kennedy, Matthew. "An Agile Systems Engineering Process: The Missing Link?" *CrossTalk* (May-June 2011): 16-20. http://www.crosstalkonline.org/storage/issue-archives/2011/201105/201105-Kennedy.pdf

[Kennedy 2013]
Kennedy, Matthew & Umphress, David. "Case Study: Applying Agile Software Methods to Systems Engineering." *The Journal of Cyber Security & Information Systems 1,* 3 (June 2013): 12-21. https://www.thecsiac.com/sites/default/files/journal_files/CSIAC_V1N3_Web.pdf

[Lapham 2010]
Lapham, Mary Ann, Williams, Ray, Hammons, Charles (Bud), Burton, Daniel, & Schenker, Alfred. *Considerations for Using Agile in DoD Acquisition* (CMU/SEI-2010-TN-002). Software Engineering Institute, Carnegie Mellon University, 2010.
http://resources.sei.cmu.edu/library/asset-view.cfm?assetID=9273

[Lapham 2011]
Lapham, Mary Ann, Garcia-Miller, Suzanne, Adams, Lorraine, Brown, Nanette, Hackemack, Bart, Hammons, Charles (Bud), Levine, Linda, & Schenker, Alfred. *Agile Methods: Selected DoD Management and Acquisition Concerns* (CMU/SEI-2011-TN-002). Software Engineering Institute, Carnegie Mellon University, 2011. http://resources.sei.cmu.edu/library/asset-view.cfm?AssetID=9769

[Larman 2008]
Larman, Craig and Vodde, Bas. *Scaling Lean and Agile Development: Thinking and Organizational Tools for Large-Scale Scrum*. Addison-Wesley Professional, 2008.

[Leffingwell 2007]
Leffingwell, Dean. *Scaling Software Agility: Best Practices for Large Enterprises*. Addison-Wesley Professional, 2007.

[Leffingwell 2013]
Leffingwell, Dean. *Scaled Agile Framework*. http://www.scaledagileframework.com (2013).

[Levinson 2013]
Levinson, Harry & Librizzi, Richard. *Using Software Development Tools and Practices in Acquisition* (CMU/SEI-2013-TN-017). Software Engineering Institute, Carnegie Mellon University, 2013. http://resources.sei.cmu.edu/library/asset-view.cfm?AssetID=72893

[Moore 2011]
Moore, David M., Crowe, Portia, & Cloutier, Robert. "Driving Major Change: The Balance Between Methods and People." *CrossTalk* (July-August 2011): 11-14.
http://www.crosstalkonline.org/storage/issue-archives/2011/201107/201107-Moore.pdf

[NDAA 2010]
National Defense Authorization Act for Fiscal Year 2010 (Public Law 111-84, Stat. 2190). GPO, 2010.

[NDIA 2006]
National Defense Industrial Association Systems Engineering Division Task Group. *Top Software Engineering Issues Within Department of Defense and Defense Industry*. NDIA, September 2006. http://www.ndia.org/Divisions/Divisions/SystemsEngineering/Documents/Studies/NDIA_Top_SW_Issues_2006_Report_v5a_final.pdf

[NDIA 2010]
NDIA. *Top Software Engineering Issues Within Department of Defense and Defense Industry*. September 2010.
http://www.ndia.org/Divisions/Divisions/SystemsEngineering/Documents/Studies/NDIA%20Top%20SW%20Issues%202010%20Report%20v5a%20final.pdf

[NDIA 2011]
NDIA. "White Paper of Practitioner's Concerns, Submitted for Policy Makers' Consideration." *Proceedings of 2011 NDIA Scrum Workshop*. Baltimore, MD, Nov. 2011. AFEI, November 2011.

http://www.ndia.org/Divisions/Divisions/C4ISR/Documents/Event%202750%20Agile%20Scrum%20Workshop%20Outcomes/NDIA%20Whitepaper-Event%202750%20Final.pdf

[OSD(AT&L) 2008]
Office of the Under Secretary of Defense (Acquisition, Technology & Logistics). Operation of the Defense Acquisition System: DoDI 5000.02. December 8, 2008.
http://www.acq.osd.mil/asda/docs/dod_instruction_operation_of_the_defense_acquisition_system.pdf

[Palmquist 2013]
Palmquist, Steven; Lapham, Mary Ann; Garcia-Miller, Suzanne; Chick, Timothy; & Ozkaya, Ipek. *Parallel Worlds: Agile and Waterfall Differences and Similarities* (CMU/SEI-2013-TN-021). Software Engineering Institute, Carnegie Mellon University, 2013.
http://resources.sei.cmu.edu/library/asset-view.cfm?AssetID=62901

[Reifer 2013]
The Impact of Software Size on Productivity. ISBSG, Capers Jones, and Reifer Consultants LLC, Sep. 2013. http://www.isbsg.com/collections/analysis-reports

[Reinertsen 2009]
Reinertsen, Donald. *The Principles of Product Development Flow: Second Generation Lean Product Development*. Celeritas Publishing, 2009.

[Shalloway 2012]
Shalloway, Alan, Bain, Scott, Pugh, Ken, & Kolsky, Amir. *Essential Skills for the Agile Developer*. Addison-Wesley, 2012.

[Turner 2007]
Turner, Richard. “Toward Agile Systems Engineering Processes.” *CrossTalk* (April 2007): 11-15.
http://www.crosstalkonline.org/storage/issue-archives/2007/200704/200704-Turner.pdf

[Ward 2010]
Ward, Dan. *The FIST Manifesto*. *Defense AT&L Magazine* (Nov./Dec. 2010): 31-32.
http://www.dau.mil/pubscats/ATL%20Docs/Nov-Dec10/The%20FIST%20Manifesto.pdf

REPORT DOCUMENTATION PAGE			*Form Approved OMB No. 0704-0188*
Public reporting burden for this collection of information is estimated to average 1 hour per response, including the time for reviewing instructions, searching existing data sources, gathering and maintaining the data needed, and completing and reviewing the collection of information. Send comments regarding this burden estimate or any other aspect of this collection of information, including suggestions for reducing this burden, to Washington Headquarters Services, Directorate for information Operations and Reports, 1215 Jefferson Davis Highway, Suite 1204, Arlington, VA 22202-4302, and to the Office of Management and Budget, Paperwork Reduction Project (0704-0188), Washington, DC 20503.			
1. AGENCY USE ONLY (Leave Blank)	2. REPORT DATE July 2014		3. REPORT TYPE AND DATES COVERED Final
4. TITLE AND SUBTITLE Agile Software Teams: How They Engage with Systems Engineering on DoD Acquisition Programs			5. FUNDING NUMBERS FA8721-05-C-0003
6. AUTHOR(S) Eileen Wrubel, Suzanne Miller, Mary Ann Lapham, Timothy A. Chick			
7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES) Software Engineering Institute, Carnegie Mellon University, Pittsburgh, PA 15213			8. PERFORMING ORGANIZATION REPORT NUMBER CMU/SEI-2014-TN-013
9. SPONSORING/MONITORING AGENCY NAME(S) AND ADDRESS(ES) AFLCMC/PZE/Hanscom, Enterprise Acquisition Division, 20 Schilling Circle, Building 1305, Hanscom AFB, MA 01731-2116			10. SPONSORING/MONITORING AGENCY REPORT NUMBER n/a
11. SUPPLEMENTARY NOTES			
12A DISTRIBUTION/AVAILABILITY STATEMENT Unclassified/Unlimited, DTIC, NTIS			12B DISTRIBUTION CODE
13. ABSTRACT (MAXIMUM 200 WORDS) This technical note (TN), part of an ongoing Software Engineering Institute (SEI) series on Agile in the Department of Defense (DoD), addresses key issues that occur when Agile software teams engage with systems engineering functions in the development and acquisition of software-reliant systems. Published acquisition guidance still largely focuses on a system perspective, and fundamental differences exist between systems engineering and software engineering approaches. Those differences are compounded when Agile becomes a part of the mix, rather than adhering to more traditional "waterfall"-based development lifecycles. For this TN, the SEI gathered more data from users of Agile methods in the DoD and delved deeper into the existing body of knowledge about Agile and systems engineering before addressing them. Topics considered here include various interaction models for integrating systems engineering functions with Agile engineering teams, automation, insight and oversight, training, the role of Agile advocates/sponsors and coaches, the use of pilot programs, stakeholder involvement, requirements evolution, verification and validation activities, and the means by which Agile teams align their increments with program milestones. This TN offers insight into how systems engineers and Agile software engineers can better collaborate when taking advantage of Agile as they deliver incremental mission capability.			
14. SUBJECT TERMS Agile, systems engineering, acquisition, development			15. NUMBER OF PAGES 79
16. PRICE CODE			
17. SECURITY CLASSIFICATION OF REPORT Unclassified	18. SECURITY CLASSIFICATION OF THIS PAGE Unclassified	19. SECURITY CLASSIFICATION OF ABSTRACT Unclassified	20. LIMITATION OF ABSTRACT UL

NSN 7540-01-280-5500

Standard Form 298 (Rev. 2-89) Prescribed by ANSI Std. Z39-18
298-102