Software Engineering Institute
Carnegie Mellon University

Scaling Agile Methods for Department of Defense Programs

William Hayes
Mary Ann Lapham
Suzanne Miller
Eileen Wrubel
Peter Capell

December 2016

TECHNICAL NOTE
CMU/SEI-2016-TN-005

Software Solutions Division
[DISTRIBUTION STATEMENT A] This material has been approved for public release and unlimited distribution.

http://www.sei.cmu.edu

Copyright 2016 Carnegie Mellon University

This material is based upon work funded and supported by the Department of Defense under Contract No. FA8721-05-C-0003 with Carnegie Mellon University for the operation of the Software Engineering Institute, a federally funded research and development center.

Any opinions, findings and conclusions or recommendations expressed in this material are those of the author(s) and do not necessarily reflect the views of the United States Department of Defense.

References herein to any specific commercial product, process, or service by trade name, trade mark, manufacturer, or otherwise, does not necessarily constitute or imply its endorsement, recommendation, or favoring by Carnegie Mellon University or its Software Engineering Institute.

This report was prepared for the
SEI Administrative Agent
AFLCMC/PZM
20 Schilling Circle, Bldg 1305, 3rd floor
Hanscom AFB, MA 01731-2125

NO WARRANTY. THIS CARNEGIE MELLON UNIVERSITY AND SOFTWARE ENGINEERING INSTITUTE MATERIAL IS FURNISHED ON AN “AS-IS” BASIS. CARNEGIE MELLON UNIVERSITY MAKES NO WARRANTIES OF ANY KIND, EITHER EXPRESSED OR IMPLIED, AS TO ANY MATTER INCLUDING, BUT NOT LIMITED TO, WARRANTY OF FITNESS FOR PURPOSE OR MERCHANTABILITY, EXCLUSIVITY, OR RESULTS OBTAINED FROM USE OF THE MATERIAL. CARNEGIE MELLON UNIVERSITY DOES NOT MAKE ANY WARRANTY OF ANY KIND WITH RESPECT TO FREEDOM FROM PATENT, TRADEMARK, OR COPYRIGHT INFRINGEMENT.

[Distribution Statement A] This material has been approved for public release and unlimited distribution. Please see Copyright notice for non-US Government use and distribution.

Internal use:* Permission to reproduce this material and to prepare derivative works from this material for internal use is granted, provided the copyright and “No Warranty” statements are included with all reproductions and derivative works.

External use:* This material may be reproduced in its entirety, without modification, and freely distributed in written or electronic form without requesting formal permission. Permission is required for any other external and/or commercial use. Requests for permission should be directed to the Software Engineering Institute at permission@sei.cmu.edu.

* These restrictions do not apply to U.S. government entities.

DM-0003767

[DISTRIBUTION STATEMENT A] This material has been approved for public release and unlimited distribution.

Table of Contents

Acknowledgments **vii**

Executive Summary **ix**

Abstract **xi**

1 Introduction **1**
- 1.1 Background 1
- 1.2 Audience 2
- 1.3 Scope 2
- 1.4 Purpose 2

2 What is Scaling? **3**
- 2.1 Team Size 4
- 2.2 Specialization of Roles 5
- 2.3 Iteration Length 5
- 2.4 Synchronized Cadence 6
- 2.5 Release Definition 7
- 2.6 Batch Size 8
- 2.7 Product Owner Role 8
- 2.8 User Role 9

3 Cross-Cutting Themes **11**
- 3.1 Architecture 11
- 3.2 Stakeholders 12
- 3.3 Organizational Structure 13

4 Published Work Supporting Scaling **15**
- 4.1 Disciplined Agile Delivery (DAD) 15
- 4.2 The DSDM Agile Project Framework (DSDM) 18
- 4.3 Large Scale Scrum (LeSS) 21
- 4.4 Modular Framework for Scaling Scrum 23
- 4.5 Scaled Agile Framework (SAFe) 24

Appendix A SEI Publications on Agile Adoption **27**

Appendix B Follow-Up Questions to Authors **29**

Appendix C Elaborations on Disciplined Agile by Scott Ambler **32**

Appendix D Elaborations on DSDM by Steve Messenger **35**

Appendix E Elaborations on Large Scale Scrum by Craig Larman **38**

Appendix F Elaborations on Scaled Agile Framework by Dean Leffingwell **42**

References **45**

[DISTRIBUTION STATEMENT A] This material has been approved for public release and unlimited distribution.

[DISTRIBUTION STATEMENT A] This material has been approved for public release and unlimited distribution.

List of Figures

Figure 1: Example Goal Diagram from Disciplined Agile 16

Figure 2: Disciplined Agile IT Workflow (http://www.disciplinedagiledelivery.com/) 17

Figure 3: The DSDM Process (used with permission from DSDM.org) 18

Figure 4: DSDM Products 20

Figure 5: LeSS Framework (http://less.works/) 22

Figure 6: Modular Framework for Scaling Scrum (http://www.scruminc.com/scrum-scale-part-1/) 23

Figure 7: Scaled Agile Framework (http://scaledagileframework.com/) 25

[DISTRIBUTION STATEMENT A] This material has been approved for public release and unlimited distribution.

[DISTRIBUTION STATEMENT A] This material has been approved for public release and unlimited distribution.

List of Tables

Table 1: RFA Categories 30

[DISTRIBUTION STATEMENT A] This material has been approved for public release and unlimited distribution.

[DISTRIBUTION STATEMENT A] This material has been approved for public release and unlimited distribution.

Acknowledgments

The authors extend their sincere thanks to the five experts who generously gave their time for interviews. Their thought leadership in the field spurs innovation, and their devotion to their craft was apparent in what we heard. The five frameworks and the associated experts interviewed were

Disciplined Agile Delivery (DAD) – Scott Ambler

Dynamic Systems Development Method (DSDM) – Steve Messenger

Large Scale Scrum (LeSS) – Craig Larman

Modular Framework for Scaling Scrum – Jeff Sutherland

Scaled Agile Framework (SAFe) – Dean Leffingwell

[DISTRIBUTION STATEMENT A] This material has been approved for public release and unlimited distribution.

[DISTRIBUTION STATEMENT A] This material has been approved for public release and unlimited distribution.

Executive Summary

The prevalence of Agile methods in the software industry today is obvious. Every major defense contractor in the market can tell you about their approach to implementing the values and principles found in the Agile Manifesto [Beck 2001]. Published frameworks and methodologies are rapidly maturing, and a wave of associated terminology is part of the modern lexicon. We are seeing so-called Agilistas and other consultants feuding on Internet forums as well—with each one claiming to have the "true" answer for what Agile is and how to make it work in your organization.

The challenge now is to scale Agile to work in complex settings, with larger teams, larger systems, longer timelines, diverse operating environments, and multiple engineering disciplines. In this report, we discuss the dimensions of this scaling problem in detail and offer advice on cross-cutting themes that warrant your attention.

A number of published frameworks are available to you today. We have interviewed leaders in the market who invented these frameworks and oversee their evolution. Rather than comparing and contrasting them for strengths and weaknesses, this report describes each framework and provides graphics, references, and the advice of the authors.

[DISTRIBUTION STATEMENT A] This material has been approved for public release and unlimited distribution.

[DISTRIBUTION STATEMENT A] This material has been approved for public release and unlimited distribution.

Abstract

Most introductory discussions of Agile software development have focused on team management concepts and the implications of the Agile Manifesto for a single, small team. The focus now includes scaling these concepts for a variety of applications. The context in which Agile methods are employed drives important choices for how the work is done. Published frameworks and commercial training available in the market offer a variety of solutions for scaling Agile. This report addresses what is meant by scaling, contextual drivers for implementation choices, and the frameworks available for use today.

[DISTRIBUTION STATEMENT A] This material has been approved for public release and unlimited distribution.

[DISTRIBUTION STATEMENT A] This material has been approved for public release and unlimited distribution.

1 Introduction

Most introductory discussions of Agile software development have focused on team management concepts and the implications of the Agile Manifesto for a single (small) team. The focus now includes scaling these concepts for a variety of applications. The context in which you employ Agile methods drives important choices in how you work. Published frameworks and commercial training available in the market offer a variety of solutions for scaling Agile. This report addresses what is meant by scaling, contextual drivers for implementation choices, and the frameworks available for use today.

1.1 Background

Scaling is the term we often hear used to describe a class of implementation decisions for using Agile methods with a larger team. The well-known context of "seven plus-or-minus two" developers working on a self-directed Agile team forms a building block for many scaling discussions. Most authors on this subject focus on coordinating functions and communication paths among the teams—forming a team of teams. If you are concerned about a long-range roadmap for your product line, or a broader ecosystem in which your product must live, then the coordination of more people is not the only type of scaling you need to consider. Successful scaling may need to accommodate larger, more complex products, built and maintained using different engineering disciplines, external suppliers, and system integrators.

Context drives scaling decisions. For example, in the domain of embedded weapons systems, you are always obliged to view the system under development as a component of a larger system (or system of systems). In this context, you would certainly not limit your focus to software alone. We find that the engineering domains involved in building and maintaining a product, not just the magnitude of the effort, shape implementation decisions for Agile methods. As well, the diversity of the user base often dictates how you must identify requirements and validate the results of your work. In major defense programs (e.g., Acquisition Category, or ACAT 1 programs), the influence of major stakeholders and a formal "decision authority" are an overarching driver for many program decisions. Successful scaling approaches, then, represent a marriage between the beneficial performance of Agile methods and the constraints imposed by your environment.

Frameworks available in the market today are maturing quickly and provide a variety of choices. Like the Agile Manifesto, these frameworks are based on principles, and they vary widely in the specificity of the recommended approach. Most frameworks reference Scrum as a fundamental building block (e.g., the Modular Framework for Scaling Scrum and Large Scale Scrum). Some frameworks offer an evolutionary path from existing methodologies (e.g., Disciplined Agile Delivery may be more easily assimilated by those familiar with the Rational Unified Process; Dynamic Systems Development Method is said to be based on Rapid Application Development). Others provide specific ways to operationalize Lean concepts (e.g., the Scaled Agile Framework draws heavily on Donald Reinertsen's book *Principles of Product Development Flow* [Reinertsen 2009]). It is not the purpose of this report to advocate or recommend one framework over others; they are discussed here because their omission would make the report incomplete.

[DISTRIBUTION STATEMENT A] This material has been approved for public release and unlimited distribution.

1.2 Audience

We have found that many organizations engaged in software development are interested in frameworks for scaling Agile methods. The primary audience for this report consists of acquirers of major software-reliant systems in the government arena, particularly in the Department of Defense (DoD). While a more diverse set of readers will find interest in the report, the content is meant to serve the needs of government acquisition organizations—specifically, those program offices charged with initiation and oversight of contracts as well as those who oversee "government-organic" capability for systems and software engineering activities.

1.3 Scope

In this report, we explain fundamental concepts that drive the design of scaling frameworks, the contextual drivers that shape implementation, and widely known frameworks available today. This report does not provide instructions for scaling Agile methods to meet your specific needs, nor does it evaluate the frameworks available in the marketplace. Rather, the report informs readers who need to determine how Agile methods must be implemented in their context.

1.4 Purpose

This report will help you gain an understanding of important scaling concepts so you can recognize successful as well as flawed attempts to enact Agile methods on a larger scale. Contextual factors that drive implementation choices are discussed to help you relate scaling to your own situation. A variety of approaches available in the marketplace are described to give you an understanding of how others have solved the scaling challenge.

[DISTRIBUTION STATEMENT A] This material has been approved for public release and unlimited distribution.

2 What is Scaling?

Many people understand Agile concepts through the illustrations offered by widely adopted methods such as Scrum [Sutherland 1995]. These team-focused development processes embody patterns of Agile behavior and offer concrete implementation examples. If you want to achieve success with Agile methods in large-scale development efforts, you might be tempted to view the challenge as simply a matter of tailoring Scrum to work with larger groups of people. What we are learning from the experiences of major DoD programs is that this view is an oversimplification of the real work to be done. To illustrate the challenge of scaling, we offer the following example from a completely different domain.

If you develop products that involve moving fluids through pipes, the work attributed to an English mathematician named Peter Barlow probably governs some of the design choices you make. Barlow's formula helps us understand the relationship between the outside diameter of a pipe, its wall thickness, the internal pressure, and the tensile strength of the pipe materials. Using this formula, we can choose the pipe we use for plumbing in a home, calibrate a pump that circulates water in a swimming pool, or evaluate some key specifications for a hydroelectric plant.[1]

$$P = \frac{2St}{D}$$

where

P = pressure

S = allowable stress

t = wall thickness

D = outside diameter

This formula serves an invaluable role in scaling systems that move fluids and represents an encapsulation of important knowledge that is able to withstand the test of many applications over time. There are variants that account for temperature and other variables as well. One very important observation about this example is that the purpose of the formula is not to make larger versions of something that has been proven in the small. Rather, the formula captures a utility function that has been found to hold up under a wide variety of conditions.

If such formulas are available for scaling Agile methods, we would certainly want to understand the range of applications for which they've been validated. In fact, we do see a surprising level of consistency on some noteworthy attributes. Just as Barlow's formula helps to avoid catastrophic failure in the specific domain where it is applied, we seek to illustrate scaling principles that help you to avoid failure in your domain(s).

The sections that follow address selected attributes that we have found significant in successfully applying Agile methods in DoD programs. Like the parameters found in Barlow's formula, these attributes deserve attention as you architect the way your program will implement Agile processes. The attributes discussed include

- Team size
- Specialization of roles

[1] See https://en.wikipedia.org/wiki/Barlow's_formula

[DISTRIBUTION STATEMENT A] This material has been approved for public release and unlimited distribution.

- Iteration length
- Synchronized cadence
- Release definition
- Focus on batch size
- Product owner role
- User role

Similar to the variables in Barlow's formula, these represent key attributes of the solution space; they are not dimensions for growing the process to a larger version of what works in the small.

2.1 Team Size

The work of British anthropologist Robin Dunbar is frequently cited for the reported correlation between brain size and the size of the social groups in primates, which is suggestive of a biological limit on an otherwise social phenomenon [Dunbar 1992]. The implication is that there is some natural limit that governs our ability to sustain ongoing communication with larger and larger groups of people.

The motivation for establishing an appropriate team size can be attributed to the desire to achieve "communication saturation." Published work on this topic includes a reference in *The Patterns Handbook*, wherein Patricia Genualdi describes the work of the Pasteur research program, which produced a process and organizational pattern language aimed at improving productivity in software-development organizations [Rising 1998]. This research quantified the level of collaboration among defined roles in an organization by measuring the ratio of actual collaborations between roles and the possible role pairs. The findings showed that projects with a large number of roles tended to have low communication saturation and were also not highly productive [Harrison 1996].

Keeping teams small can enable every member of the team to have all of the information needed to effectively contribute to the work. When the work to be done is greater than what can be accomplished by a small group of people, the advice is to add teams, or to stage the work in iterations over time, rather than growing an ever-larger team. You would need to define the work in a way that lends itself to such a partitioning. This is a matter of scaling down the work scope, as much as it is a choice to scale up the team processes. The so-called Scrum-of-Scrum meetings serve to communicate details with a meta-level structure, summarizing important information that needs to be shared across teams—rather than inundating each individual with everything.

Scaling Agile practices successfully will require you to consider the importance of communication. As stated in one of the principles that accompany the Agile Manifesto, "The most efficient and effective method of conveying information to and within a development team is face-to-face conversation" [Beck 2001]. However, this does not mean that you must limit yourself to face-to-face conversation for all communication. Visually oriented methods for communicating detailed status are part of most Agile methods. The "Scrum board" we see at the team level, as well as

[DISTRIBUTION STATEMENT A] This material has been approved for public release and unlimited distribution.

other types of "information radiators"[2] provide ways to chunk information and make it available to people who need it.

2.2 Specialization of Roles

Though it doesn't specifically trace back to the four tenets of the Agile manifesto or one of the 12 accompanying principles, limiting the extent of specialization among team members is something most Agile proponents strive for. That is, you will be encouraged to cross-train engineering staff and move away from a team structure where people focus on only one specialty, such as design, development, or test. The ability to scale Agile methods to a variety of settings is enhanced by the availability of cross-trained team members. However, as the complexity of the system increases, the need for highly specialized engineering disciplines will likely increase as well. A balance of skills and knowledge must be managed. People often allude to this balance by reference to "T-shaped" individuals—indicating a preference for people who have deep knowledge in the area of their specialty, but also a broad-based skill set that enables contribution in a number of different areas. So-called specializing generalists are potentially useful in any context, but implementing Agile methods becomes easier when you have such staff.

Another goal associated with the formation of self-organizing teams, as is often discussed in Scrum training, is to limit the fracturing of people's time across multiple projects. Much has been written about the fallacy of multi-tasking and the deteriorating performance of people matrixed out to competing efforts [Reynolds 2011, Frick 2015]. Most Agile trainers will advise you to form teams that stay together and bring the work to them—rather than building and disbanding teams around projects as they come and go. Much of industry today is focused on maximizing utilization of assets, and this may represent one of your largest barriers to effectively scaling Agile methods in your context. The move toward specializing generalists offers an alternative to fracturing your most valuable assets (people) across competing efforts—thus degrading their performance.

Finally, there are a number of hallmark roles introduced with Agile methods, notably "Scrum master" and "product owner." Roles such as "system architect" are not found in Scrum or XP but we find this role, for example, in the Scaled Agile Framework. More generally, the advice you will receive from most Agile proponents is to avoid specialized roles—unless and until they become necessary. Some frameworks take a more assertive position on this matter, as they name roles to be filled, while others remain silent. Those who remain silent on specialized roles typically assume that the capability enabled by the role is fulfilled by one or more people on the team—no individual is assumed to be working on that activity to the exclusion of other types of work.

2.3 Iteration Length

Iteration length has received much attention among those adopting Agile practices. The challenge of implementing shorter iterations may be one of the things that challenges you most in scaling

[2] Cockburn offers a useful definition at http://alistair.cockburn.us/Information+radiator.

[DISTRIBUTION STATEMENT A] This material has been approved for public release and unlimited distribution.

Agile to build larger, more complex software systems. Like the team size topic discussed above, the advice you are likely to get from Agile coaches is to add iterations rather than planning longer iterations—to scale Agile methods. Iterations that last longer than three or four weeks can quickly become "mini-waterfalls." The Agile principle of simplicity ("the art of maximizing the amount of work not done is essential") will be fostered by the sense of urgency that comes with shorter iterations.

Those who think of Agile methods such as Scrum as only project management strategies fail to see the benefit of slicing work into small pieces so you can get feedback faster. Short iteration lengths speed up the rate of feedback (derived from potentially usable products, rather than models or documents that describe them). This enables you to more frequently assess the *value* of what you've delivered—an attribute many view to be more important than the *quantity* delivered. As well, with shorter iterations, you can recover more quickly from failure, re-thinking design choices (as appropriate) before their consequences are embedded throughout the system.

When you scale Agile methods to build larger, more complex systems, the length of iterations is not merely a project management (or communication) consideration. In designing more complex systems, there may be a greater engineering (or design) challenge to overcome to reap the benefit of short iterations. The discussion of batch size in Section 2.6 elaborates this point further.

2.4 Synchronized Cadence

Coordinating the contributions of multiple teams to a single product delivery cycle can be challenging no matter what development method you choose. If your teams use different iteration lengths, you will want to look for ways to synchronize the end points of iterations. If teams deliver product features or components (or slices of the architecture) at different times, the cascading effect of rework as integration occurs with each new arrival can lead to a chain of rework-driven cycles that amplify over time. Postponing integration while lagging teams finish their work may lead the early finishing teams to move onto other work—without the benefit of potentially course-correcting feedback from the work they've just completed.

The advice given by Donald Reinertsen in *The Principles of Product Development Flow* will lead you to structure iterations to align their endpoints as much as you can [Reinertsen 2009]. This means that, if you have teams using a mix of two- and three-week iterations, consider scheduling integration events every six weeks—sequencing work and lower level tests between these six-week milestones to maximize the work of individual teams. This approach allows different teams to synchronize their work every six weeks, and focus resolution of integration challenges at those times, rather than prolong their potential effect in a more disruptive manner.

A more desirable pattern is to have all teams working at the same cadence, building to the same code base. However, this pattern may not be feasible in all cases. Consider this, too, as you devise milestones and delivery schedules for suppliers. With the widespread adoption evident in industry today, your suppliers may be more Agile than you know. Formalisms in contracts and the structure of award fees may incentivize them to deliver in large batches with long timelines, when their internal cadence might offer opportunities for more frequent integration events—ones that lead to more rapid feedback and potential course corrections. Sometimes it takes a bit of creativity (and a

[DISTRIBUTION STATEMENT A] This material has been approved for public release and unlimited distribution.

trust-based relationship) to gain access to *breadboards* and *brassboards,*[3] rather than waiting to perform first article testing. To be fair, there will be system components that can't easily be pre-viewed or delivered iteratively.

2.5 Release Definition

At scale, most Agile development efforts are structured into a series of releases, each built up from a set number of iterations. Typically people set a release to contain four to six iterations, which often fit into a calendar quarter (e.g., four iterations of 3 weeks each, or six iterations of 2 weeks each lead to a 12-week cycle). In scaling Agile methods, synchronizing with business cycles (e.g., quarterly reporting requirements, or cadence driven by earned value management systems) is a useful consideration, as budget cycles and other external dependencies may follow these patterns.

Another cycle often discussed in planning for releases is the "concept-to-cash" cycle. You will find many Agile authors put emphasis on ways to shorten this cycle. This is in keeping with the third principle associated with the Agile manifesto: "Deliver working software frequently, from a couple of weeks to a couple of months, with a preference to the shorter timescale" [Beck 2001]. Most people believe that the user is able to provide the most relevant feedback on the value of the product. However, in some settings, the user may not be able to tolerate such frequent updates to the system—at least not changes that alter the workflow supported by the system. For example, it would not be practical to retrain all air traffic control personnel on new procedures each quarter. For this reason, when we scale Agile methods, the incremental cycles often referred to as releases do not necessarily get pushed out to the user.[4] The concept-to-cash cycle is better thought of as a concept-to-capability cycle in many settings. The phrase used in the Scaled Agile Framework (SAFe) that best reflects this thinking is "develop on cadence, deliver on demand."

For the largest programs funded by the DoD, especially in the realm of embedded systems, a multi-tiered testing regime is quite common. In many such programs, testing is first done by an individual engineer, then pieces of the product move to a coordinated integration event, then into a test lab supported by simulated operating environments and other forms of system-level developmental testing prior to a complex operational test involving the ecosystem in which the product must operate. In these settings, it may be easier to understand the output of a set of iterations as an engineering release rather than a delivery to a field operator. No matter what term you use, the importance of rapid feedback needs to be understood. Typically we see demos used to gain user feedback at the end of each iteration, and many will use a release-level demo as well. For some system capabilities, integration and some level of system test may be needed before a demonstration is possible. However, waiting until *all* capabilities can be demonstrated prior to seeking any feedback is a recipe for failure in scaling Agile.

---

[3] See *Communicating Project Management: The Integrated Vocabulary of Project Management and Systems Engineering* [Mooz 2003] for definitions of these terms.

[4] In the product owner section, we will have much more to say about who the "user" is in our contexts.

[DISTRIBUTION STATEMENT A] This material has been approved for public release and unlimited distribution.

2.6 Batch Size

If you've only read the glossy half-page brochure about Agile, or heard only the 15-minute version of the explanation, then your perception may be colored by an apparent objection to heavy weight, documentation-focused processes. A more complete telling of the story reveals that the long wait for feedback is a bigger problem than the heavy weight of documentation. A focus on work in small batches, prioritized by value, with rapid feedback from the user, is a more important priority for Agile. It is true that choosing to build comprehensive documentation (e.g., complete and detailed requirements, architecture, and design specifications) before writing any code is counter to the tenets of Agile. However, this is not because Agile methods forsake such documentation in favor of some sort of misguided preference for improvisation. The reason to avoid Big Design Upfront (BDUF) is that it delays the implementation of software that can be shown to a user for the valuable feedback they can provide. In building and sustaining major systems, there are needs for comprehensive documentation that do not necessarily derive from the BDUF tradition. Prematurely committing to details in the design can create an obstacle because the prospect of reworking the whole design to accommodate a change can lead development organizations to shy away from valuable user feedback.

If you think of requirements as perishable opportunities to give the users what they want—before they change their minds—it follows that, the sooner you get to capability, the more likely you will delight the customer. Rapid iterations are enabled by smaller batch size. When large systems are developed without iteration, you take on risk from the potential for early decisions to be overturned by later events. Of course such a risk will exist no matter how you structure the work. In contrast, if you move too rapidly to build components without due consideration for architecture and design, the risk of extensive rework is increased. Therefore a balance of competing risks is the goal, and controlling the batch size is one of the most powerful tools available.

2.7 Product Owner Role

The product owner role in Scrum is a pivotal element to successful implementation because it provides the development team with a single voice of the business. Even among teams not using Scrum, we are seeing a product owner role—or some variant of it—used to communicate priorities based on value for users. In DoD terms "the business" can include a diverse set of stakeholders (e.g., various military commands, operational users, sustainment organizations, information assurance specialists, and decision makers in the acquisition process).

In scaling Agile methods, the implementation of this concept may take on a variety of forms. You will find a broader discussion of related roles in Section 3.2, Stakeholders. However you implement it, the importance of the voice of the user in driving Agile development cannot be overstated. While a Scrum master helps enable the team's work by removing impediments and coaching people to focus their efforts, the product owner role is responsible for assuring that the effort is focused on building "the right things." The product owner's obligation is to maximize the return from the investment made by the people who do the technical work.

An area of acute difficulty for managing major government contracts is the challenge of adequately accounting for the needs of all stakeholders of the system. In this context, we have a multi-tiered (or multi-dimensional) "customer" population, each exerting an influence on the program. Not all of them are driven by technical performance requirements for the (to be) deployed

[DISTRIBUTION STATEMENT A] This material has been approved for public release and unlimited distribution.

system itself. The fiduciary obligations of some roles often compete with the obligations of other roles. Sometimes these influences seem to be in opposition with each other. Enabling the performance of Agile development teams often requires us to reconcile these drivers so developers hear a "single voice of the user." Commercial organizations are not immune to such complexities either and effective implementation of the product owner role is just as important in those settings.

If you are successful at scaling Agile methods, one of the reasons for your success will undoubtedly be your good choices in how this role is implemented. There are situations where a single person from the development organization plays this role; other times, a team of people fulfill the responsibilities. We are aware of organizations that have a "chief product owner" supported by a team of product owners, each assigned to a development team. Other implementations we have seen define a product owner team comprised of representatives from the government program office teaming with representatives from the development contractor organization(s). One simple recipe for success would seem unlikely, given the diversity of contexts.

2.8 User Role

Agile development relies on collaboration with the users of the system (or someone who represents the user base) in a way that many other development styles do not. Rather than basing all work on an "up-front" comprehensive requirements specification, you will engage the users to help refine the development team's understanding of what is needed—at a time when that discussion will have the most beneficial effect. The product owner role helps to make sure this communication happens and balances potentially competing priorities across different users, the market served by the company, and the roadmap of the product line.[5] The Agile development team must have access to a person who can adequately represent the needs of the user, if not to users themselves.

In building major systems for military and government use, the challenge of adequately representing the needs of a diverse user community is not a new issue. This challenge has always been difficult. Use of Agile methods brings a new element to this challenge in that the user has a greater role during the development process to help elaborate, clarify, or correct misunderstandings about requirements. Rather than working to capture a complete and detailed user-approved specification in advance, Agile methods set you up to engage the user with working system components (sometimes prototypes operating in a simulation environment, or an end product ready for the field) rather than documents describing them. As you will see in the Architecture section, it is not intended that the development team start with no requirements. The intent is to base detailed design choices on a firmer foundation, with things like user stories (that help explain why something needs to function one way or another) and working functionality (that allows the user to see the system behavior).

In scaling Agile methods, the challenge and opportunity can be further magnified because you can't rely on a signed requirements specification to defend yourself from the evolving preferences of a fickle user. You will need to establish a cooperative relationship with the user community. However, because you will tend to engage the user with a more concrete basis for discussion, you

[5] Reconciling these potentially competing concerns becomes more challenging in larger programs, where the difference between the product owner role and the user role becomes more apparent.

[DISTRIBUTION STATEMENT A] This material has been approved for public release and unlimited distribution.

have the opportunity to deliver functionality that is a higher fidelity representation of what the user needs (not just what they were able to articulate during the up-front requirements gathering activities).

[DISTRIBUTION STATEMENT A] This material has been approved for public release and unlimited distribution.

3 Cross-Cutting Themes

In Section 2, we discussed eight scaling factors to consider when implementing Agile methods in a DoD program setting. Experience has shown that how you deal with these eight topics will greatly influence your chance for success. In addition to revealing their importance, experience has shown that there are key boundary conditions to be understood as well as optimal settings to be sought for these important parameters.

In this section, we turn our attention to similarly important topics for scaling, but ones for which boundary conditions and optimal settings may not be as well understood. Mindfully specifying the architecture, effectively managing stakeholders, and understanding the influence of organizational structure will be necessary to successfully scale Agile methods to your setting.

3.1 Architecture

The first airline pilot to land at a new airport cannot specify the requirements for the landing strip as the plane descends from 35,000 feet on the day the airport opens. Someone has to think ahead and specify (and indeed *build*[6]) the runway. However, the exact locations of the coffee shops in the passenger terminal need not be decided for transportation to be realized. In fact, until the expected foot traffic pattern of busy passengers is understood, some shop owners may be reluctant to commit to a lease with the airport authority. In a similar manner, scaling Agile methods typically call for some governing principles about the "architectural runway" [Leffingwell 2007] that assures enabling decisions are made in time while constraining tradeoffs are postponed to the last responsible moment.

Building further on the airport analogy, the placement of chairs in the coffee shop, or the choice of regional food items on the menu, may be decisions that emerge from the early days of operation. In contrast, the counter heights and lighting levels in the shop will likely follow a well-understood standard. For builders of software-reliant systems, important human-computer interaction decisions may emerge while developing innovative solutions that enable unprecedented capabilities. However, if you deploy in an ecosystem of large interconnected systems, there will be interface standards you depend on for efficient and reliable communication with inter-dependent systems. These standards are typically known well in advance of implementing the new system capabilities.

Agile methods lead you to move away from making all the design decisions "up front." However, this does not necessarily mean that the pendulum swings all the way to making no decisions in advance. A clear vision for the product serves to focus effort, while balancing "just enough" decision making assures ripe opportunities will be harvested in a timely manner. As you scale to larger, more diverse settings, the mechanisms you employ to achieve the balance will need to respond to your context. A good discussion on the patterns seen in practice, in the context of quality

[6] As any aviator will tell you, a runway is much more than a strip of pavement. There are precise specifications for thickness, width, length, and the materials used—which depend on the type of aircraft expected.

[DISTRIBUTION STATEMENT A] This material has been approved for public release and unlimited distribution.

attribute requirements, is found in the Software Engineering Institute (SEI) report *Enabling Incremental Iterative Development at Scale: Quality Attribute Refinement and Allocation in Practice* [SEI 2015]. There you will find five typical patterns of allocating effort to the development of features versus the development of architecture. The five process patterns described by the authors of the SEI report are

1. "You Aren't Going to Need It (YAGNI)," which essentially dismisses architecture work as non-value-added. The authors point out the fallacy of this approach, debunking the implication that there is no design work. This approach drives expensive rework or obliges you to focus on refactoring the system between iterations or releases.
2. "Hardening Sprints" are used by some to define an iteration of work focused exclusively on bug-fixing and work to make the system robust, as defined by the architectural (or quality attribute) requirements. This iteration follows a set of iterations where features have been implemented.
3. "Iteration Zero" might be viewed as a Lean version of the specification-driven approach commonly associated with the waterfall lifecycle. The idea is to do just enough architecture to meet the needs for the set of iterations that follow, but architectural work is done before any features are implemented.
4. "Rework," described by the authors as more of an anti-pattern, is a process pattern wherein feature development comes to a screeching halt as technical debt becomes impossible to ignore [Nord 2012].
5. "Evolutionary/Runway" is the label for an approach where an ongoing effort to address architectural work is supported at a level that yields "just enough architecture" to support the feature implementation work underway at that time.

The results of the survey reported by the authors confirms that most organizations employ a mix of these patterns [SEI 2015].

3.2 Stakeholders

The role of key stakeholders and decision authorities is one of the hallmarks of the defense contracting environment. Major programs often exist within an ecosystem of 'programs of record,' and they may serve a complex user-base with divergent needs for system performance. Funding authority and content authority may not always be vested in the same government organization. As well, roadmaps for system modernization are sometimes at odds with funding. In such an environment, the emphasis on face-to-face engagement, and the assertion that "business people and developers must work together daily," as found in the principles of the Agile Manifesto [Beck 2001], may appear daunting.

If we view Agile as an intent to simply shed the web of fiduciary obligations that govern a major program, then it is unlikely to succeed in the government contracting arena. Rather than forsaking processes that enable planning, reviews, and approvals, you will find that intelligent application of Lean concepts guides "refactoring" of the practices in some cases. Some scaling methods offer a complement of roles, or examples of how such roles function in the enterprise.

As you consider scaling Agile methods to your context, you will need to consider the stakeholders with whom you presently cooperate—even if only to confirm that you will maintain the current

[DISTRIBUTION STATEMENT A] This material has been approved for public release and unlimited distribution.

engagement approach in some cases. Examples of potential changes in how stakeholders are engaged include the following:

- Milestone Decision Authority (MDA) required documentation may be produced incrementally (or on a different schedule). For example, waivers can be established early in the program for a roadmap of less formal "mini-milestone events" that culminate in a capstone event, with delivery of the final artifact.
- Financial reporting obligations can often be met with limited change to current practice. For example, Earned value management systems (EVMS) that accommodate Agile development methods have been described in the literature and continue to receive interest in the federal community.
- System users' evolving needs for the system may be sought and taken into consideration more often. For example, acceptance-test-driven approaches as well as frequent use of demos can focus input in more detail. Prudent balance in how you involve a divergent user community remains important, of course.
- Program management personnel may find a new cadence of work to schedule. For example, frequent planning events or system demos may require a new pattern of travel, VOIP, or teleconferences than in the past.

A more complete discussion of these and other examples can be found in the series of reports from the SEI Agile adoption research team. The list of publications to date is provided in Appendix A of this report.

3.3 Organizational Structure

Melvin Conway, in his 1968 paper *How Do Committees Invent*, coined an adage (known as Conway's law) which is summarized in the conclusion of that paper:

> *… organizations which design systems … are constrained to produce designs which are copies of the communication structures of these organizations* [Conway 1968].

This age-old adage offers important insight if you are working to scale Agile methods in a large enterprise. Most trainers of Agile teams will explain how to decompose products in terms of features (often further decomposed into user stories). Most large systems are traditionally built (and maintained) by teams organized around system components. This approach can present an obstacle if you don't reconcile the potentially competing views of the system in a coherent manner.

If you've ever worked in a matrixed organization, then you are probably familiar with the potential for tension between departments staffed with particular subject matter experts and the project management office (PMO) organized around elements of the customer base (or product line). First line managers in these companies sometimes find themselves with too much responsibility and too little authority. That is, they are charged with achieving goals that flow from one side of the organization, but need the authority, which is vested in the other side of the organization. Sometimes project managers aren't able to get adequate participation from certain departments, or those departments aren't able to get the project to adopt the best technical approach. When the two sides are mutually cooperative, this structure works well. However, the balance can easily be disrupted by unforeseen events (e.g., a dramatic shift in market strategy or unanticipated staff turnover).

[DISTRIBUTION STATEMENT A] This material has been approved for public release and unlimited distribution.

Where large, mature systems are being sustained and enhanced, the structure of ongoing work is conveniently organized by system components. Teams of engineers can be organized according to architectural layers of the system, or subsystems differentiated by the primary engineering discipline involved. The vast majority of software-reliant systems in military and government usage do not tend to be "green field," unprecedented systems, but rather reflect an evolution of existing systems or a new system that will operate in the larger context of a system of systems. Because of this fact, many organizations supporting these systems are organized (in harmony with Conway's law) according to the architecture of the system.

The architecture and design of large software-reliant systems are typically represented with a component-focused decomposition. The user's needs for system performance, however, are typically expressed in terms of features or capabilities of the system. In traditional waterfall-based programs, the process of "allocating" requirements to system components is typically a series of tradeoffs made as specifications are drafted to account for user needs (in features and capabilities). These user needs are met by individual system components—and often through interactions of those components. This approach to design is an endeavor to collect and adjudicate (typically before any code is written) the part to be played by each system component. In contrast, explanations of how Agile methods work typically focus primarily on the functional decomposition and place little to no emphasis on a component (or sub-system) decomposition.

Successful scaling of Agile to major DoD programs often involves a healthy mix of feature-oriented and component-oriented teams. Component teams are a natural structure for maintaining mature systems with loosely coupled architectures. Feature teams are a natural structure for building system functions in small rapid iterations—if you staff them with the complement of talents and specialties required. As mentioned in the discussion of architecture (above), there is a need for balance between stability and flexibility. A delicate balance will let you maintain the robustness of fielded components while rapidly adding new or varied paths of logic through them. Agile methods, scaled to meet these needs, must help you address the engineering demands you face, not just the project management cadence of short iterations.

[DISTRIBUTION STATEMENT A] This material has been approved for public release and unlimited distribution.

4 Published Work Supporting Scaling

Publications and commercial training that help you apply Agile concepts in practice are plentiful. A rapidly growing community of practice exists, with mature products (including training, tools, and work aids) to support adoption of Agile principles using patterns and practices cataloged in reference books and well-curated web content. We had the good fortune to interview five of the thought leaders in this space, who each generously gave us a one-hour interview. In addition, we followed up with a written communication requesting specific information mapped to topics of interest in our transition research. In the five subsections below, we cover each of the frameworks.

It is important to understand that the inclusion of a particular framework in this report does not reflect an endorsement or any other special acknowledgement for the good work of the author(s)—just as omission of a particular framework or author from this discussion does not reflect an unfavorable view of their good work.

The five interviews (in alphabetical order) are

- Disciplined Agile Delivery (DAD); interviewee Scott Ambler
- Dynamic Systems Development Method (DSDM); interviewee Steve Messenger
- Large Scale Scrum (LeSS); interviewee Craig Larman
- Modular Framework for Scaling Scrum; interviewee Jeff Sutherland
- Scaled Agile Framework (SAFe); interviewee Dean Leffingwell

4.1 Disciplined Agile Delivery (DAD)

To describe DAD as a scaling framework may not tell the whole story—though the authors clearly intend to support scaling. Ambler and Lines, in the central book on this topic [Ambler 2012], position DAD as a foundation that extends and integrates other Agile methods to provide a more robust solution for implementing Agile. The emphasis on being "enterprise aware" appears to be more of a focus on doing Agile well, than on scaling practices, per se. The authors do offer an explicit discussion on scaling, listing specific tactical scaling factors to be considered [Ambler 2012, pg. 22], including

- geographical distribution
- team size
- regulatory compliance
- domain complexity
- technical complexity
- organizational distribution
- organizational complexity
- enterprise discipline

[DISTRIBUTION STATEMENT A] This material has been approved for public release and unlimited distribution.

One of the main points of emphasis for DAD is that it explicitly addresses initiation of a product to its release, not just the "construction" phase,[7] as the authors describe it. In addition, the emphasis of the authors is on IT solutions, rather than embedded systems or software-reliant product development—though clearly, the information found in DAD can inform those contexts.

Scott Ambler explained that the fundamental strategy was to provide a decision framework anchored to a set of process goals (or capabilities). It is presumed that the user is making process decisions by selecting among alternative practices or supplying their own, to meet those goals. The authors are emphatic about their intent to avoid prescribing particular methods and also make their case for avoiding particular practices. Ambler provided the graphic below as an example of a goal diagram—illustrating the choices discussed for "Address Changing Stakeholder Needs."

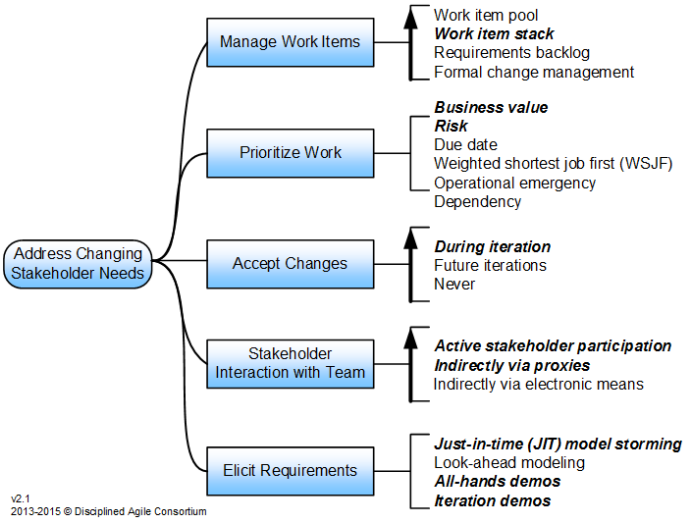

Figure 1: Example Goal Diagram from Disciplined Agile

In releasing version 2.0, now simply called *Disciplined Agile*, the focus is on scaling Agile strategically to address the full workflow of an IT department. The key organizing graphical depiction associated with the framework is reproduced Figure 2 (with permission; URL valid as of publication date).

[7] Readers familiar with the Rational Unified Process may recognize this terminology. The RUP phases include Inception, Elaboration, Construction, and Transition. DAD simplifies this to support three phases: Inception, Construction, and Transition.

[DISTRIBUTION STATEMENT A] This material has been approved for public release and unlimited distribution.

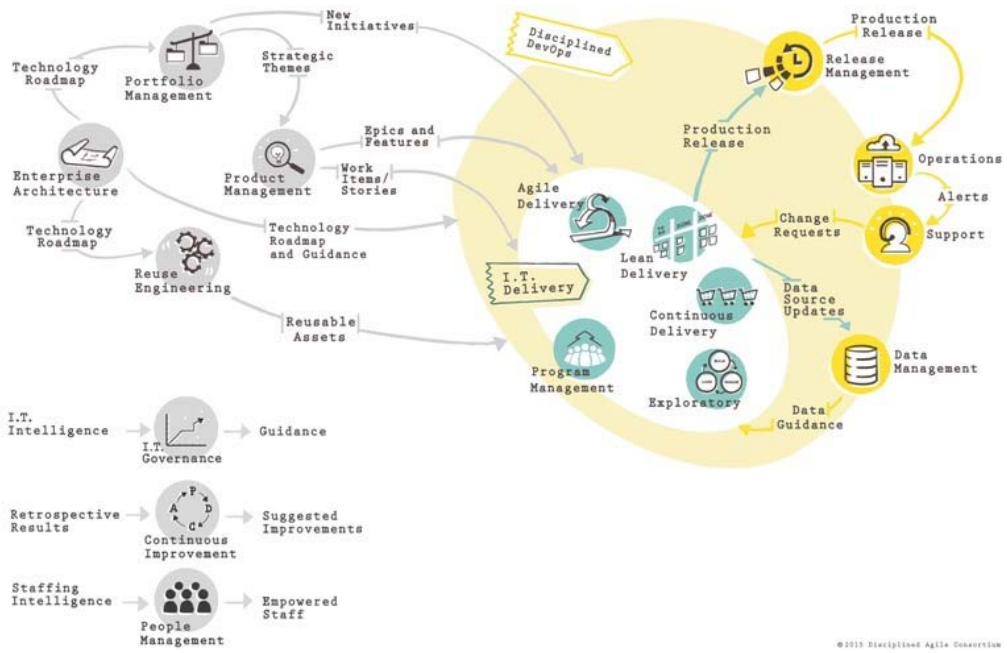

Figure 2: Disciplined Agile IT Workflow (http://www.disciplinedagiledelivery.com/)

As a resource for planning to tactically scale Agile methods in your setting, the book by Ambler and Lines offers a large collection of examples and options to consider [Ambler 2012]. The authors discuss their experiences working with larger enterprises and their focus extends well beyond the team level. In particular, the book contains a number of large tables that discuss the potential benefits and limitations of practices you might consider adopting. In addition, there are survey results and articles published on the Internet, which the authors reference throughout the book.

During our interview, Ambler observed that sub-optimal choices are sometimes made without realizing that alternatives exist. This is why he and his colleagues set out to create a process decision framework to make alternatives and their associated tradeoffs more visible. Rather than prescribing a particular pattern to be implemented, this framework provides a range of alternatives—accompanied by elaborations and decision considerations.

Assessing the challenges he has seen for scaling Agile, Ambler observed that 85 percent or more of the challenge lies in the people and culture arena, with tools and techniques coming a distant second. Furthermore, the "Agile culture" can sometimes present an obstacle itself, as orthodoxy about terminology sometimes leads to shallow interpretations. For example, the requirement to use an external verification and validation process does not automatically render an organization to be "not Agile"—as someone once suggested to Ambler. Understanding how other things like architecture, performance modeling, and documentation requirements fit together when scaling Agile is important. Ambler differentiates enterprise-level coaches, who can help with such things, from team-level coaches, who help the individual teams. He told the story of an FDA audit t turned from complete disaster to "the best team I ever audited" when the coach was able to explain how the team met the intent of each of the audit criteria—but with non-typical approaches.

Appendix C contains Ambler's written comments on a set of topics in follow-up correspondence.

[DISTRIBUTION STATEMENT A] This material has been approved for public release and unlimited distribution.

4.2 The DSDM Agile Project Framework (DSDM)

The DSDM Agile Project Framework (previously known as Atern) was established in 1994 (well before the Agile Manifesto was signed). DSDM emerged from Rapid Application Development (RAD) to create an independent RAD framework. Some of its proponents were signers of the Agile Manifesto. In fact, DSDM helped shape the Agile Manifesto.

DSDM can be used both for IT and non-IT projects as it addresses the whole product lifecycle. DSDM is organized as a model that can be used with other methods or a wrapper to ensure the whole lifecycle is addressed. It can integrate with other methods such as Scrum and XP.[8]

As an iterative approach, you can revisit any previous step in the DSDM process as needed, so the idea is to only finish enough in one step so you can move on to the next step, as shown in Figure 3. DSDM is also considered a convergent approach where basic foundations (architecture) are agreed at an early stage.

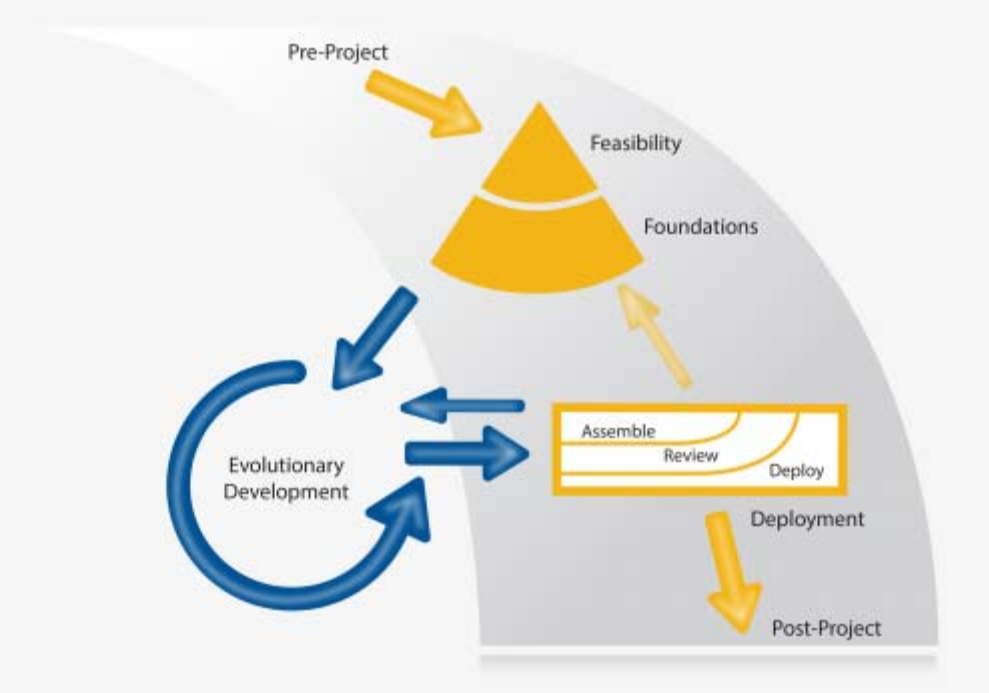

Figure 3: The DSDM Process (used with permission from DSDM.org)

Incorporating Agile and Lean principles, DSDM implements the 80/20 rule—asserting that 80 percent of the solution is done in 20 percent of the time. This framework assumes that nothing is built perfectly the first time and proposes building simpler solutions that are fit for purpose. As the lifecycle evolves, maintenance can be treated as a future increment.

DSDM is based on the following eight working principles (Section 4.1 of the DSDM handbook):

1. Focus on the business need.
2. Deliver on time.

[8] See http://www.dsdm.org/dig-deeper/book/dsdm-agile-project-framework

[DISTRIBUTION STATEMENT A] This material has been approved for public release and unlimited distribution.

3. Collaborate.
4. Never compromise quality.
5. Build incrementally from firm foundations.
6. Develop iteratively.
7. Communicate continuously and clearly.
8. Demonstrate control.

DSDM can be scaled by configuring and calibrating it for larger projects that need stronger governance. This is achieved by configuring the lifecycle for a specific project and appropriate level of formality with which the DSDM products are defined, created, and approved. If necessary, specific techniques, such as workshops for planning at scale, can be designed to help make large-scale development a reality without conflicting with DSDM philosophy and controls or compromising innate Agility. Essentially, the organization will be refined to support multiple teams and products. The solution architecture definition, development approach definition, management approach definition, and delivery plan and time box review records can be more elaborate and formal. See Figure 4 for how these fit together.[9]

[9] See http://www.dsdm.org for additional information. The majority of the DSDM framework is free to view and free to use for end users.

[DISTRIBUTION STATEMENT A] This material has been approved for public release and unlimited distribution.

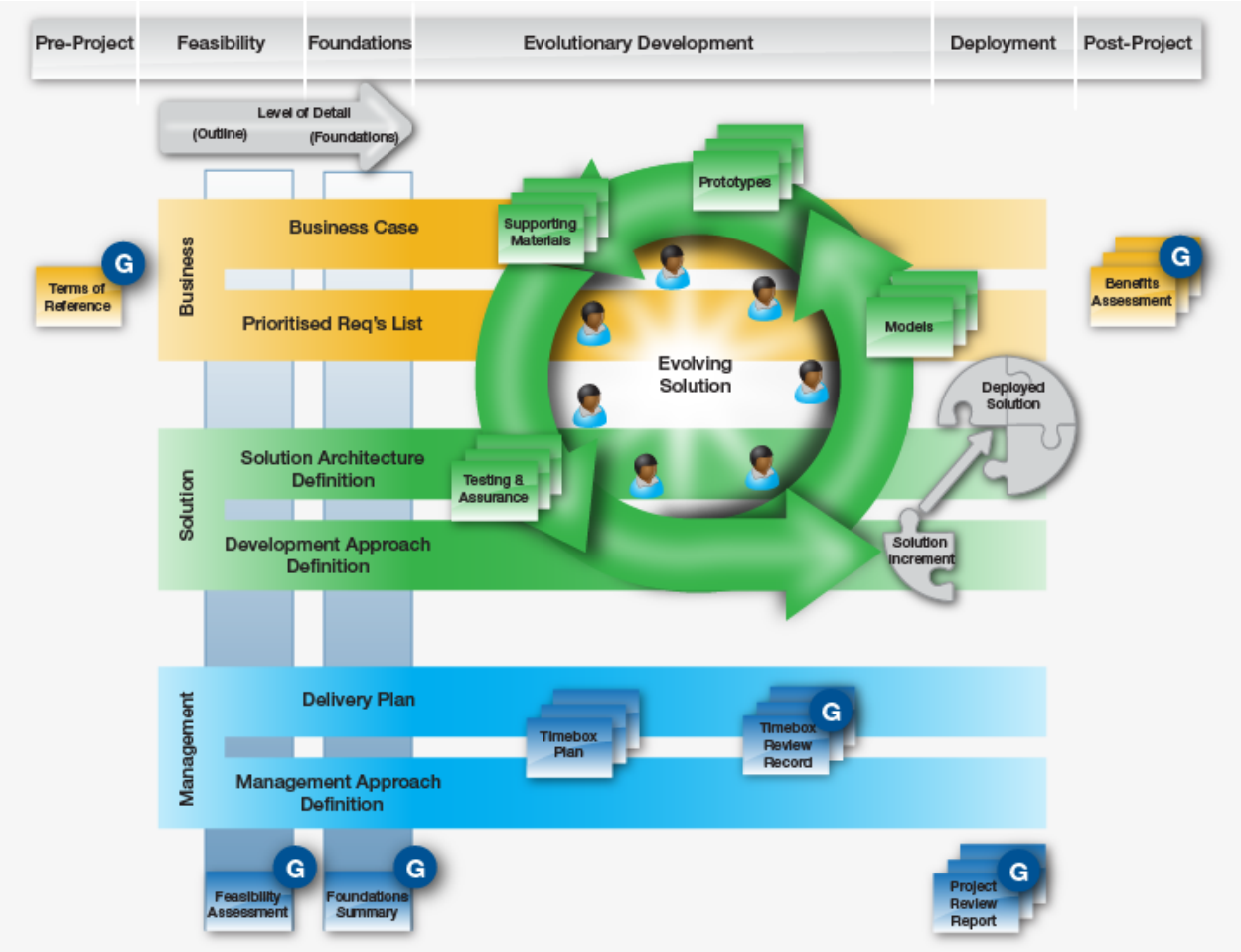

Figure 4: DSDM Products

We were able to secure an hour with Steve Messenger from the DSDM consortium for a telephone interview. Our conversation focused on the things we need to do to make it feasible for small Agile teams to do the work they need to do. Defining architecture and scope considerations well enough to scope the work of small teams is an essential ingredient to success. The challenge may involve scaling down the big governance processes as much as it involves scaling-up the development approach. It is unlikely that "everything will be Agile" when we go to the large scale. When multiple teams must collaborate, or a large complex product is involved, there is a need to establish a baseline architecture that can be changed as needed. As well, an efficient, non-bureaucratic change control process is needed—one that does not introduce a delay in the system.

Messenger recalled an experience with a major pharmaceutical company as it worked to scale Agile approaches to its work. His observation was that the regulators expressed concern at first, worried about the quality of the products. As they gained experience with the company's new approach, it became apparent that there were actually more controls on the work using this approach, and the documents needed for regulatory compliance were being built while the work was underway—rather than being done as a separate step. This meant that the regulatory obligations were being satisfied by scaling down the tasks, so they could be performed concurrently with the engineering work.

[DISTRIBUTION STATEMENT A] This material has been approved for public release and unlimited distribution.

Appendix D contains Messenger's written comments on a set of topics in follow-up correspondence.

4.3 Large Scale Scrum (LeSS)

The LeSS Framework was developed to scale Scrum upward based on the insight that "Scrum hits the sweet spot between abstract principles and concrete practices" [Larman 2008].[10] Thus, Larman and Vodde reason, to scale Scrum upward, similar balance must be achieved: "For large groups, LeSS hits the sweet spot between defined concrete elements and empirical process control." During our interview, Craig Larman emphasized the critical need to understand and evolve the organization. Focusing on differences between the espoused goals of an organization and the goals implied by the behavior of the staff leads to a deeper understanding of how structure drives culture. As Larman's comments in Appendix E reiterate, methodological details and the implementation of individual practices are secondary to the focus on the way the organization works—with a systems view.

In scaling, Larman and Vodde advocate

- simplicity—The authors advise avoiding the addition of roles, artifacts, and process. LeSS seeks to avoid providing a defined process, and rather having the product group empirically create necessary process.
- scaling Scrum—Larman and Vodde emphasize that rather than using Scrum as a building block in a framework, it is necessary to examine each element of Scrum, analyze its purpose, and determine how that *purpose* can be achieved on a larger scale.
- scaling up instead of tailoring down—The authors posit that when scaling process development for larger organizations, it is common to define a universal framework and then allow contextual tailoring. The assumptions associated with tailoring the framework lead to bloated processes, as participants often believe that all of the overarching framework is required in each context. (See Figure 5; significant structure beyond a typical depiction of Scrum process is noticeably absent. A key difference is the inclusion of the Overall Retrospective, discussed below.)

[10] At the time of this writing, Larman and Vodde had announced the pending publication of their latest book: *Large-Scale Scrum: More with LeSS* (Addison-Wesley) – publication expected August 2016.

[DISTRIBUTION STATEMENT A] This material has been approved for public release and unlimited distribution.

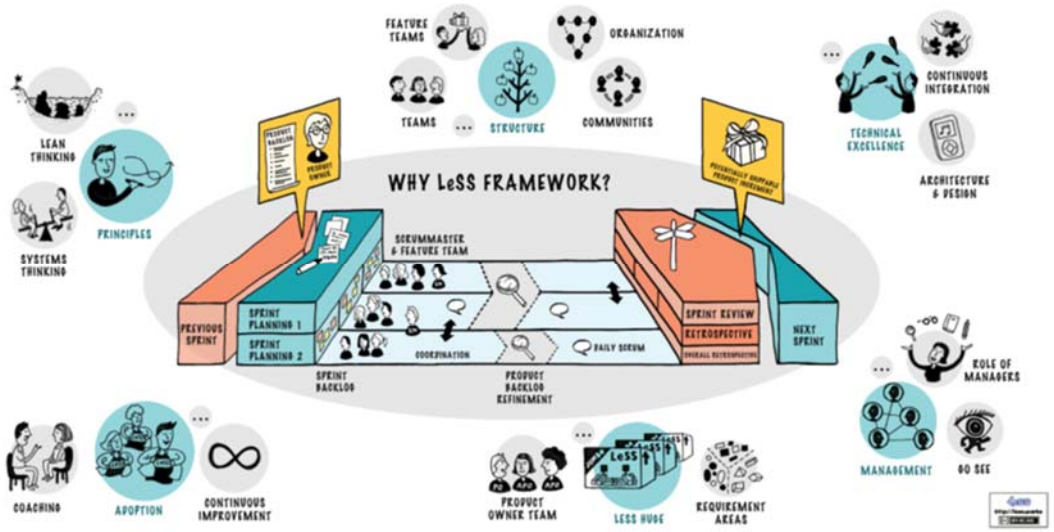

Figure 5: LeSS Framework (http://less.works/)

The authors provide a series of LeSS Rules[11] to define the LeSS Framework, strongly rooted in Lean and systems thinking. In fact, Lean thinking, systems thinking, empirical process control, and continuous improvement toward perfection are cited as four of the core "LeSS Principles." Structures and processes discussed in LeSS emphasize execution within, and constant awareness of, the entire system so that not only will a product be developed that will satisfy the customer, but the process for developing that product will be improved and adapted along the way.

Larman and Vodde emphasize establishing feature-based teams that are long lived, cross functional, and co-located, producing end-to-end-features one by one [Larman 2008]. LeSS features a single product owner and a single product backlog for the complete shippable product. Scrum masters may serve up to three teams and must focus on the overall organizational system. There is one product-level sprint (rather than different sprints for each team), and each sprint results in an integrated product. Sprint planning "Part One" occurs at the product level, in which product owners and team representatives select items to be worked during the sprint. Each feature team then engages in a "Part Two" to plan out the execution of their tasks within the team. Communication and coordination across the teams is decided by the teams, and can include observing (silently) other teams' Scrums, town hall meetings, and other approaches. LeSS includes an "Overall Retrospective" at the product level for each sprint, for discussing the system as a whole: relationships between teams, relationship with the customer, learning outcomes from the teams that may be shared. The teams are constantly seeking to improve the product and the development processes.

In *Scaling Lean and Agile Development*, Larman and Vodde provide an extensive set of experiments and "thinking tools" for consideration in scaling Agile, consistently emphasizing continuous improvement, product focus, and respect for and communication among all participants [Larman 2008]. Larman and Vodde also published the LeSS Huge framework for even larger endeavors (more than eight teams), which follows the same principles.

[11] See https://less.works/less/rules/index.html

[DISTRIBUTION STATEMENT A] This material has been approved for public release and unlimited distribution.

Our one-hour interview with Larman focused on the foundations of his work and on his emphasis on systems thinking in working with organizations. Larman cautions against a tendency toward "copying without knowledge" as organizations adopt practices without the requisite understanding of the underlying cause and effect systems that govern performance. With this profound knowledge in hand, the goal of Agile is to permit inexpensive change when it's needed: to allow you to "turn on a dime, for a dime." A search for Craig Larman on YouTube yields many recorded presentations, some of them quite comprehensive.

Appendix E contains Larman's written comments on a set of topics in follow-up correspondence.

4.4 Modular Framework for Scaling Scrum

The Modular Framework for Scaling Scrum, from Scrum Inc., provides a basis for iterative and incremental adaptation of successful Agile patterns within an enterprise. Rather than focusing on a particular solution—expressed as a comprehensive pattern to be instantiated—this framework aids in the decomposition of work needed to successfully use Scrum at scale. As Jeff Sutherland explains, the context in which a company operates drives the focus on areas that need to change.

Pre-recorded webinars available on the company website[12] as well as numerous conference presentations you can find through searching the Internet provide a great deal of free information if you want to study this framework.

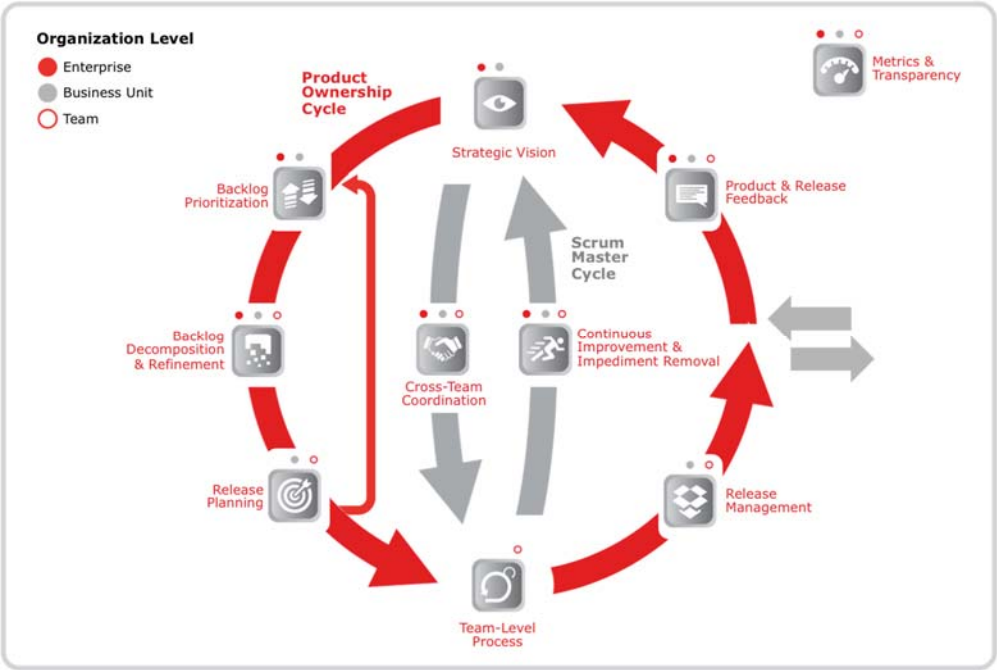

Figure 6: Modular Framework for Scaling Scrum (http://www.scruminc.com/scrum-scale-part-1/)

In applying the Modular Framework for Scaling Scrum, different elements of the framework (depicted in Figure 6) will receive emphasis based on the current approach to doing work, and the priorities for improvement. The goals, inputs, and outputs provided for elements of the framework

[12] See http://www.scruminc.com/scrum-scale-part-1/

[DISTRIBUTION STATEMENT A] This material has been approved for public release and unlimited distribution.

help guide the evolution of work practices in key parts of the enterprise—as the organization becomes more Agile. Like most scaling frameworks, there is a presumption that Scrum is a core element of the enterprise approach. In explaining his views on scaling, Sutherland told us "It's more about getting the leadership Agile than it is about Scrum." In this context, you might view scaling as a challenge of removing impediments to using Scrum—rather than devising new and more elaborate ways to "be Agile."

Our interview with Sutherland covered a lot of ground, as he told us about his most recent experiences working with large organizations. First and foremost, Sutherland wanted us to understand that scaling Agile is not about an implementation, but rather, it's about the values in the Agile Manifesto. The emphasis on speed, and identifying impediments that slow teams down dominated the conversation. Sutherland told us that a single common backlog, which can then be partitioned into the work done by contributing teams, is an essential element of success. To achieve this, Sutherland reminded us, the leadership in the organization must be able to provide clear priorities. In the successful companies, Sutherland also told us that shorter iteration length is associated with a greater completion rate. An iteration length of one week is what he is striving for, with the goal being a continuous flow of working software—integrating multiple times per iteration.

Cautioning us about the detrimental effect of evolving team membership, Sutherland advises that forming stable teams (of no more than nine individuals) is important—to a greater degree than is the focus on generalists. His experience suggests that working to reduce specialization of individual talents should come after you gain experience with maintaining stable teams delivering working code in short iterations. Sutherland explained that this is very challenging for large organizations, and that some surveys suggest a mere 31 percent success rate among those striving to be Agile (with an 11 percent success rate for waterfall projects). Sutherland's experience is that many organizations try and fail at the challenge of delivering working code at the end of each iteration. Another pattern of concern is the detrimental effect of deferring work into later sprints. Sutherland indicated that when tests slip into the next sprint, it can take as much as 24 times longer to deliver working code.

4.5 Scaled Agile Framework (SAFe)

The Scaled Agile Framework (SAFe) is widely viewed through the web interface that presents an overview image, with clickable links to articles that describe the selected topic. The so-called "big picture" (shown in Figure 7) presents an architectural view for how Lean and Agile concepts can be applied in an organization. Version 4 was nearing release when we interviewed Dean Leffingwell; version 3 is shown here.

[DISTRIBUTION STATEMENT A] This material has been approved for public release and unlimited distribution.

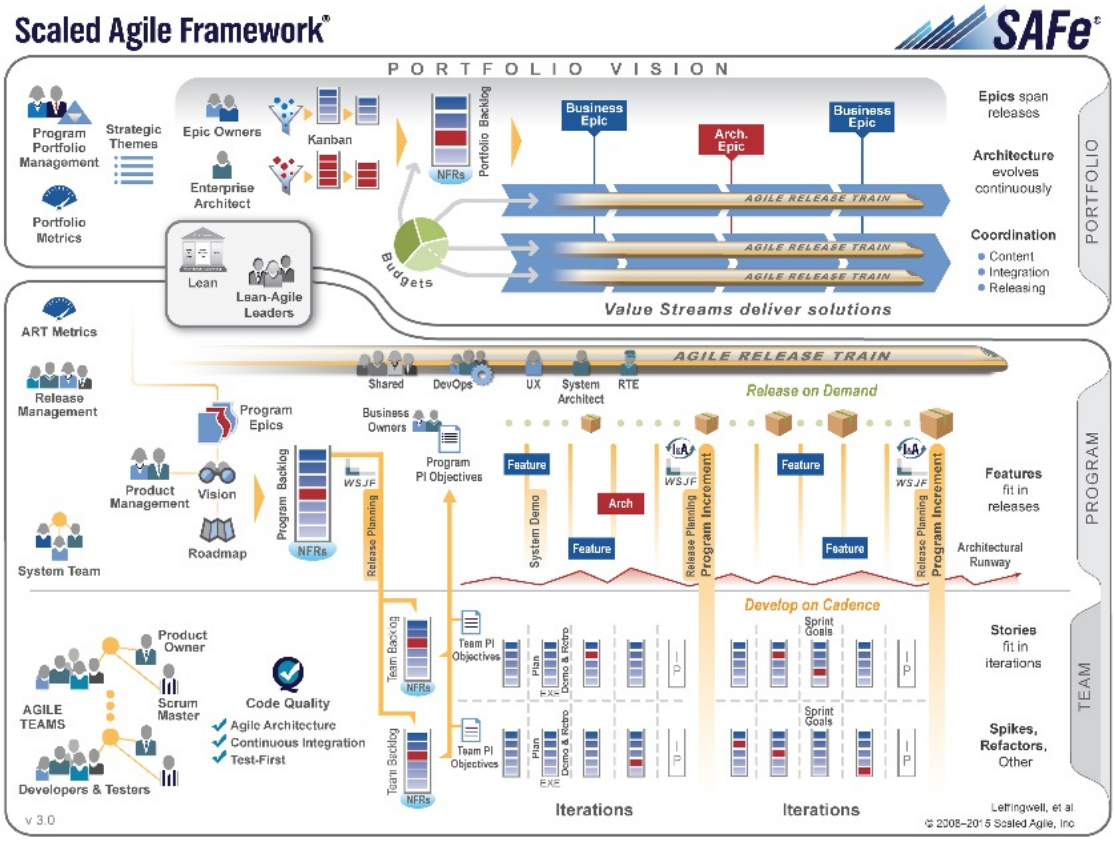

Figure 7: Scaled Agile Framework (http://scaledagileframework.com/)

A collection of Agile teams (each consisting of five to nine people) make up what is called an "Agile release train," (ART) which comprises 50-125 people. These Agile release trains build solutions that realize the intent of what are termed portfolio "Value Streams." ARTs apply a nested incremental development cadence of two-week iterations, aggregating value and time into 8-12 week "Program Increments." Lean and Agile principles are referenced throughout the material available on the website and in the training curriculum. SAFe espouses four core values:

1. quality
2. program execution
3. alignment
4. transparency

These values drive choices in how to implement concepts in your particular setting. Along with the concepts accompanying the so-called "House of Lean and Agile Manifesto,"[13] SAFe also offers nine principles that underlie the design of SAFe:

1. Take an economic view.
2. Apply systems thinking.
3. Assume variability; preserve options.

[13] See http://scaledagileframework.com/lean/ for a full explanation.

[DISTRIBUTION STATEMENT A] This material has been approved for public release and unlimited distribution.

4. Build incrementally with fast, integrated learning cycles.
5. Base milestones on an objective evaluation of working systems.
6. Visualize and limit work in process (WIP), reduce batch sizes, and manage queue lengths.
7. Apply cadence; synchronize with cross-domain planning.
8. Unlock the intrinsic motivation of knowledge workers.
9. Decentralize decision-making.

During our interview, Leffingwell explained that SAFe is designed to scale Lean and Agile concepts from the team level to the program level to the portfolio level with a "sweet spot" initially in the 300-500 person range (per SAFe portfolio)—and can grow to include more than 1,000 people who need to collaborate. Organizations of this size will likely build larger systems and have existing architectures, and legacy environments that help to define the systems they build/maintain. Roles and responsibilities that accommodate such things are described in SAFe. In addition, SAFe 4.0 features an entirely new value stream level, intended to support those building the largest software and cyber-physical systems.

Leffingwell emphasized how the so-called "requirements and design-freeze milestones" seen in many major development efforts force a too-early stake in the ground and a commitment to a specific implementation before the team knows enough about the system they are building. A number of articles on design and an elaboration of the architectural runway concept are available on the SAFe website to address this. Extending the analogy provided by Barlow's formula, Leffingwell described the goal of achieving a "laminar flow"[14] of work as the organization learns to manage its throughput by tuning batch size. The role of continuous integration as an enabler to this flow is key, in Leffingwell's experience, as well as the fundamental need to match demand and capacity. Finally, with the January 2016 release of SAFe 4.0, Leffingwell explained how the framework now accounts for the inclusion of multiple engineering disciplines and supports those building the largest software and cyber-physical systems.

Appendix F contains Leffingwell's written comments on a set of topics in follow-up correspondence.

[14] "Laminar flow" is defined by Merriam-Webster as an uninterrupted flow in a fluid near a solid boundary in which the direction of flow at every point remains constant.

[DISTRIBUTION STATEMENT A] This material has been approved for public release and unlimited distribution.

Appendix A SEI Publications on Agile Adoption

Lapham, Mary Ann; Bandor, Michael S.; & Wrubel, Eileen. *Agile Methods and Request for Change (RFC): Observations from DoD Acquisition Programs.* CMU/SEI-2013-TN-031. Software Engineering Institute, Carnegie Mellon University. 2014. http://resources.sei.cmu.edu/library/asset-view.cfm?AssetID=77732

Regan, Colleen; Lapham, Mary Ann; Wrubel, Eileen; Beck, Stephen; & Bandor, Michael S. *Agile Methods in Air Force Sustainment: Status and Outlook.* CMU/SEI-2014-TN-009. Software Engineering Institute, Carnegie Mellon University. 2014. http://resources.sei.cmu.edu/library/asset-view.cfm?AssetID=312754

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

Hayes, William; Miller, Suzanne; Lapham, Mary Ann; Wrubel, Eileen; & Chick, Timothy A. *Agile Metrics: Progress Monitoring of Agile Contractors.* CMU/SEI-2013-TN-029. Software Engineering Institute, Carnegie Mellon University. 2014. http://resources.sei.cmu.edu/library/asset-view.cfm?AssetID=77747

Wrubel, Eileen; Miller, Suzanne; Lapham, Mary Ann; & Chick, Timothy. *Agile Software Teams: How They Engage with Systems Engineering on DoD Acquisition Programs.* CMU/SEI-2014-TN-013. Software Engineering Institute, Carnegie Mellon University. 2014. http://resources.sei.cmu.edu/library/asset-view.cfm?AssetID=295943

Bellomo, Stephany. *A Closer Look at 804: A Summary of Considerations for DoD Program Managers.* CMU/SEI-2011-SR-015. Software Engineering Institute, Carnegie Mellon University. 2011. http://resources.sei.cmu.edu/library/asset-view.cfm?assetID=9751

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

Wrubel, Eileen & Gross, Jon. *Contracting for Agile Software Development in the Department of Defense: An Introduction.* CMU/SEI-2015-TN-006. Software Engineering Institute, Carnegie Mellon University. 2015. http://resources.sei.cmu.edu/library/asset-view.cfm?AssetID=442499

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

[DISTRIBUTION STATEMENT A] This material has been approved for public release and unlimited distribution.

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

Nidiffer, Kenneth; Miller, Suzanne; & Carney, David J. *Potential Use of Agile Methods in Selected DoD Acquisitions: Requirements Development and Management.* CMU/SEI-2013-TN-006. Software Engineering Institute, Carnegie Mellon University. 2014. http://resources.sei.cmu.edu/library/asset-view.cfm?AssetID=89158

[DISTRIBUTION STATEMENT A] This material has been approved for public release and unlimited distribution.

Appendix B Follow-Up Questions to Authors

All software engineering and management practices are based on cultural and social assumptions. When adopting new practices, leaders often find mismatches between those assumptions and the realities that exist within their organizations. The SEI has an analysis method called Readiness and Fit Analysis (RFA)[15] that allows the profiling of a set of practices to understand their cultural assumptions and then to use the profile to support an organization in understanding its fit with the practices' cultural assumptions. RFA has been used for multiple technologies and sets of practices, most notably for adoption of Capability Maturity Model Integration (CMMI) practices.[16] The method for using RFA and the profile that supports CMMI for Development[17] adoption is found in Chapter 12 of *CMMI Survival Guide: Just Enough Process Improvement.*[18] The SEI has extended RFA to support profiling and adoption risk identification for DoD and other highly regulated organizations that are considering or are in the middle of adopting Agile methods.

One of the fundamental principles of technology adoption is that of mutual adaptation. This principle asserts that a successful technology adoption by an organization usually requires adaptation of both the technology and the organization. The technology may adapt, for example, by being configurable—allowing switching on or off of different features—or by allowing localization to a different native language. The organization may adapt by changing some of its business workflows so they are more compatible with the technology or by changing the roles of the people involved in different processes that are affected by the technology.

When an organization adopts a new set of practices, it sees many of the same issues associated with adopting a new hardware or software technology. At the SEI, we have observed over many years that the principle of mutual adaptation applies to adopting new practices in similar ways to adopting new technologies. One of our observations has been that the closer the organization's culture is to the implied cultural assumptions of a set of practices, the easier it is for that organization to adopt those practices.

As part of our research in the adoption of Agile methods in U.S. DoD settings, we have adapted the RFA profiling technique to accommodate both the typical factors used in RFA and some factors that are more uniquely associated with the DoD acquisition environment. We found that only applying the commercial profile didn't highlight enough of the issues that we were seeing in our interviews and observations of practice.

We have characterized the following six categories to profile for readiness and fit:

- business and acquisition—adoption factors related to business strategy, acquisition strategy, and contracting mechanisms

[15] See http://www.sei.cmu.edu/sos/consulting/sos/readinessandfit.cfm for more information.

[16] See http://www.sei.cmu.edu/cmmi for more information.

[17] See http://resources.sei.cmu.edu/library/asset-view.cfm?assetID=9661 for more information.

[18] See http://resources.sei.cmu.edu/library/asset-view.cfm?assetID=30511 for more information.

[DISTRIBUTION STATEMENT A] This material has been approved for public release and unlimited distribution.

- organizational climate—adoption factors related to sponsorship, leadership, reward systems, values, and similar "soft" issues
- system attributes—adoption factors related to the actual characteristics of the system(s) being developed
- project and customer environment—adoption factors related to project management norms, team dynamics and support structures, and customer relationships and expectations
- technology environment—adoption factors related to the technologies that are in place or planned to support the selected Agile methods
- practices—a taxonomy of Agile practices that is used to understand which practices an organization plans to adopt so that other factors can be calibrated around those expectations

Although the most common use of RFA is to analyze an adoption context for the purpose of figuring out what steps an adopting organization can take to mitigate risks related to bringing in new practices, the framework used for the analysis can also be used to characterize the way a particular set of practices address the typical adoption risks associated with each category. We sometimes refer to this as the *transitionability* of the technology or practices.

For the scaling frameworks that we have characterized in this report, we offered each of the authors of a scaling framework an opportunity to characterize how their framework addresses the adoption factors associated with each RFA category.

Table 1: RFA Categories

Fit Dimension	Agile Assumptions
Business and Acquisition	Program acquisition strategy and practices enable, or at least don't disable, differences in developing using Agile approaches.
Organizational Climate	Reward systems, values, skills, sponsorship explicitly support Agile and Lean values and principles.
Project and Customer Environment	Frequent collaboration between development team and customers/end users is actively supported. Program management practices respect team boundaries.
System Attributes	System architecture is loosely-coupled (interfaces are external vs. internal among system components). System solutions benefit from fast user/operational feedback.
Technology Environment	Technology support for automated testing and continuous integration are in place. Support for information radiators (either physical or electronic) are in place.
Team Technical Practices	Technical practices that support high quality code production in small batches from a prioritized product backlog are in place.
Team Management/Coordination Practices	Decentralized decision making that allows team members to self-organize their work are in place and supported. Team management practices that support short (2-4 week) time boxes are in place.
Program Practices	Synchronization of multiple teams is occurring. Practices that reinforce respecting team management and measurement boundaries are in place.

[DISTRIBUTION STATEMENT A] This material has been approved for public release and unlimited distribution.

Each of the Agile authors who took us up on this opportunity is included in the appendices. Each of the following appendices contains the category tag (e.g., project and customer environment) and the author's comments, sometimes edited for brevity's sake, on how their framework addresses the typical adoption issues of that category.

[DISTRIBUTION STATEMENT A] This material has been approved for public release and unlimited distribution.

Appendix C Elaborations on Disciplined Agile by Scott Ambler

Business and Acquisition:

Disciplined Agile scaling approach addresses typical business and acquisition barriers to Agile through the following mechanisms:

Disciplined Agile does not include an acquisition process area.

Vendor management is addressed by the Portfolio Management process blade[19].

Organizational Climate:

Disciplined Agile scaling approach addresses typical organizational climate barriers to Agile through the following mechanisms:

The People Management process blade in Disciplined Agile 2.0 addresses HR activities following Agile principles and practices.

The Continuous Improvement process blade addresses process improvement and sharing across teams.

Project, Customer, and Team Environment:

Disciplined Agile scaling approach addresses typical project, team, and customer environment barriers to Agile through the following mechanisms:

- Philosophy of Active Stakeholder Participation fully adopted throughout the Disciplined Agile framework.
- Stakeholder is an explicit role in the framework.
- Program Management and Portfolio Management process blades address team boundaries.
- The Coordinate Activities process goal explicitly addresses strategies for inter- and intra-team collaboration.
- Explicit governance strategies for development teams built right into original DAD framework. Addition of IT Governance process blade in Disciplined Agile 2.0 release.

System Attributes:

Disciplined Agile scaling approach addresses typical system attributes barriers to Agile through the following mechanisms:

- Architecture and design practices explicitly included in the framework.

[19] The term Process Blade is used in Disciplined Agile to denote a category of topics—similar to the notion of a process area or process domain.

[DISTRIBUTION STATEMENT A] This material has been approved for public release and unlimited distribution.

- The Explore the Initial Scope process goal explicitly calls out strategies for capturing system attributes (which they call Non-Functional Requirements).
- The Identify Initial Technical Strategy process goal describes strategies for initial Agile architectural modeling.
- The Prove Architecture Early process goal describes strategies for ensuring your architecture strategy is viable early in a project.
- The Produce a Potentially Consumable Solution process goal includes practices for ongoing architecture work and detailed design during construction.
- Disciplined Agile teams are directed to produce a consumable solution frequently so that feedback may be obtained from stakeholders.

Technology Environment:

Disciplined Agile scaling approach addresses typical technology environment barriers to Agile through the following mechanisms:

- The Form Work Environment process goal explicitly addresses the need to setup both physical work spaces and viritual workspaces (tools) when the team is being initiated.

Team Technical Practices:

Disciplined Agile scaling approach addresses typical team technical practices barriers to Agile through the following mechanisms:

- The Produce a Potentially Consumable Solution process goal includes practices for ongoing design and programming work during construction.
- The Move Closer to Deployable Release process goal includes a range of practices for validation, verification, documentation, and configuration management.
- The Address Changing Stakeholder Needs process goals includes several options for requirements change control, including but not limited to a prioritized backlog.

Team Management and Coordination Practices:

Disciplined Agile scaling approach addresses typical team management and coordination practices barriers to Agile through the following mechanisms:

- The Coordinate Activities process goal explicitly addresses strategies for inter- and intra-team collaboration.
- Explicit roles and responsiblities are defined that are based on self organization.
- Lifecycles supporting time-boxed development, continuous delivery, and exploratory (Lean Startup) strategies are included in Disciplined Agile.

Program Practices:

Disciplined Agile scaling approach addresses typical program practices barriers to Agile through the following mechanisms:

[DISTRIBUTION STATEMENT A] This material has been approved for public release and unlimited distribution.

- Explict process blades addressing each of Program Management, Portfolio Management, Enterprise Architecture, and IT Governance.

[DISTRIBUTION STATEMENT A] This material has been approved for public release and unlimited distribution.

Appendix D Elaborations on DSDM by Steve Messenger

Business and Acquisition:

The DSDM Project and Program scaling approach addresses typical business and acquisition barriers to Agile through the following mechanisms:

- incorporates "just enough" governance to satisfy organisational governance requirements
- is able to incorporate both Agile and non-Agile developments within the project or program, at the same time providing a complete Agile approach
- provides guidelines for both assessing the risk of using Agile approaches and for ongoing health checks
- incorporates a complete project and program role model, covering not just development, but all aspects of large, scaled projects and programs

Organizational Climate:

The DSDM Project and Program scaling approach addresses typical organizational climate barriers to Agile through the following mechanisms:

- DSDM does not impose restrictions on reward systems, etc. Generally, the use of DSDM has improved people's motivation. The successful outcomes that result from DSDM projects and programs are normally used in rewarding the people involved—hence there is a natural move from an individual reward model towards a more team-based reward model.

Project, Customer, and Team Environment:

The DSDM Project and Program scaling approach addresses typical project, team, and customer environment barriers to Agile through the following mechanisms:

- Part of the DSDM philosphy is that teams are empowered and multi-functional, incorporating all stakeholders from those who develop the solution to those who will use it.
- Full training is available via a network of Accredited Training Organisations for all members of DSDM Agile project or program teams, so that all understand their roles and responsibilities in the initiative and how they relate to each other.
- DSDM explains in detail the optimum composition of the DSDM teams, including the roles and responsibilities required. It is also realistic about the amount of time people can commit, particlularly at senior levels.
- DSDM defines how empowerment needs to work, and how empowerment is bounded by layers (e.g., team, project, and program).
- DSDM's instrumental success factors include team-based factors such as commitment, empowerment, skills, size, etc. This provides a mechanism to assess risk and to perform ongoing health checks.

[DISTRIBUTION STATEMENT A] This material has been approved for public release and unlimited distribution.

System Attributes:

The DSDM Project and Program scaling approach addresses typical system attributes barriers to Agile through the following mechanisms:

- DSDM incorporates "Enough Design Up Front" (EDUF), which helps to define the system landscape and set the boundaries with respect to system architecture, but also still allows detail to evolve with appropriate control mechanisms.
- EDUF also provides a full picture of the problem being solved, hence identifying interfaces, etc.

Technology Environment:

The DSDM Project and Program scaling approach addresses typical technology environment barriers to Agile through the following mechanisms:

- DSDM is an Agile project and program framework. As such, it actively promotes the use of techniques and technology related to Agile, and can incorporate the majority of techniques available.

Team Technical Practices:

The DSDM Project and Program scaling approach addresses typical team technical practices barriers to Agile through the following mechanisms:

- DSDM is an Agile project and program framework. As such, whilst it does not itself include them, it actively promotes the use of techniques and technology to ensure high quality code.
- DSDM actively embraces collaborating with other Agile approaches, for example to adopt specific Agile technical practices.

Team Management and Coordination Practices:

The DSDM Project and Program scaling approach addresses typical team management and coordination practices barriers to Agile through the following mechanisms:

- The whole DSDM framework and role model is based on setting up small, multi-functional teams that are empowered to deliver against the goals that have been set for them.
- DSDM provides guidance for understanding when decisions need to be made outside the team and how interdependencies and interfaces between teams can be handled.

Program Practices:

The DSDM Project and Program scaling approach addresses typical program practices barriers to Agile through the following mechanisms:

- EDUF provides a mechanism for understanding how teams can be set up that can be self contained and can manage themselves within the limits of the goals that have been set.

[DISTRIBUTION STATEMENT A] This material has been approved for public release and unlimited distribution.

- The DSDM role model, particularly the project and program levels, defines how to set up mullti-team project and program planning, management, and tracking whilst enabling the teams to deliver against their goals with minimal interference.

[DISTRIBUTION STATEMENT A] This material has been approved for public release and unlimited distribution.

Appendix E Elaborations on Large Scale Scrum by Craig Larman

General Comments:

The most striking point in this list is the absence of a focus on major organizational design and structural elements, including the elimination of groups, roles, and positions. Most of the list relates to practices, which is a trivial, minor aspect of Scrum or Large-Scale Scrum. The first-order and significant changes are *structural*, not practices or values or principles.

For example, in Scrum there is a cross-functional Team of multi-functional team members. That's the cornerstone element of Scrum. At scale, that means the elimination of all single-function groups (analysis group, architect group, UI design group, programmer groups, test group, documentation group) and thus inevitably related manager positions. And it means the elimination of single-function job titles and traditional career paths (e.g., "business analyst path," "tester path"). That is the major first-order change that influences the behavior of the system at scale, not any practice or principle. This major and obvious category of change is missing in the listing.

It is also interesting that the tables speak of program and projects, rather than **products**. One of the major paradigm shifts towards Scrum is the elimination of the prior project/program model towards the product model (consider the Scrum terms: **Product** Backlog, **Product** Owner …). This is not just term changes; the change from project to product has major implications on policies and behavior.

Business and Acquisition:

Scaling approach addresses typical business and acquisition barriers to Agile through the following mechanisms:

In LeSS, traditional business barriers are addressed through introducing the role of Product Owner, who is from the "business" side and directly responsible for ROI. For example, head of Product Management playing the Product Owner role.

Acquisition barriers are reduced through the use of Agile Contracts in LeSS; there is an in-depth chapter on Agile Contracts in LeSS in the second LeSS book (*Practices for Scaling Lean & Agile Development: Large, Multisite, and Offshore Product Development with Large-Scale Scrum*) and there is also the http://agilecontracts.com website.

Organizational Climate:

Scaling approach addresses typical organizational climate barriers to Agile through the following mechanisms:

Broadly, most of the organizational design and climate aspects of LeSS are covered in the Organization chapter of the first LeSS book, *Scaling Lean & Agile Development: Thinking and Organizational Tools for Large-Scale Scrum*. The org climate and change implications covered in that chapter cover purpose and strategy in LeSS, organizational structure in LeSS, roles and tasks in

[DISTRIBUTION STATEMENT A] This material has been approved for public release and unlimited distribution.

LeSS, reward and remuneration system in LeSS, people and career and evaluation in LeSS, and processes in LeSS. In general we use the classic Star Model (Galbraith: http://www.jaygalbraith.com/services/star-model) to describe the organizational design for LeSS.

In LeSS, adoption is not to be forced or rushed, and requires both a top-down and bottom-up engagement and support. LeSS encourages a slow process of deep education by all the stakeholders (senior managers and front-line workers) in the change implications, with a slow and deliberate decision to do an adoption based on thorough "informed consent." This is, in fact, the Lean Thinking (Toyota Way) "nemawashi" approach to change. Also in LeSS, we recommend an adoption be done with volunteering, so that people who do not wish to be part of the change can opt out, and those interested, opt in. Further, in LeSS, we recommend limiting the change to one (and only one) "50 person" product group, not a giant group. And not allowing a second adoption until after at least six months of focusing on that first group. And this group has a massive level of coaching full time: coaches for the managers, Product Owner, and teams.

In this way, the significant structural and organizational change to Large-Scale Scrum is bounded to a small group, with lots of support, and a go-slow approach. This increases the chance for success, and this in turn creates "street cred" for supporting further change.

Of course, all Agile approaches imply the Agile Values, including "customer collaboration over contract negotiation" and "responding to change over following a plan." And so, too, in LeSS. The implications of this at scale are more significant. Why? Because traditional large-scale org policies imply a big-batch project/program initiative which is "negotiated" and then handed over to the "dev group" to deliver. And in traditional large-scale dev there are formal org elements supporting this old model, including existing program and program/project manager roles and processes, whose function is to "deliver the project contract." Thus in Scrum (and so Large-Scale Scrum) these org elements are also eliminated, since they are part of the traditional status quo system of "delivering the contract" that is inconsistent with an Agile model that emphasizes managing learning, change, variability, and collaboration.

Related to above, in Scrum (as with LeSS) there is a change in climate and paradigm to a product-centric model of development, rather than a program/project centric model. This reduces short-term-ism behaviors and a variety of other dysfunctions that arise in the project model.

Project, Customer, and Team Environment:

Scaling approach addresses typical project, team, and customer environment barriers to Agile through the following mechanisms:

In Scrum and LeSS, elimination of projects and programs, and replaced with a product-centric model, in which a business-side Product Owner who is responsible for ROI of the product (typically, head of Product Management), steers adaptively each Sprint.

In LeSS, refinement and clarification of items is done directly between team members and real hands-on customers or users, with no intermediates in between—no business analysts, etc. In this way, many of the Lean wastes are eliminated: hand-off, overproduction, inventory, delay, and more.

[DISTRIBUTION STATEMENT A] This material has been approved for public release and unlimited distribution.

In LeSS, the one Product Owner focuses on prioritization with users, and the Teams on clarification with users.

System Attributes:

Scaling approach addresses typical system attributes barriers to Agile through the following mechanisms:

In LeSS, we say that "Agile architecture" comes from "Agile architecting." That is, without a change in behavior by people, nothing will really change in the architecture. There is a detailed chapter on Agile Architecture in LeSS in the second LeSS book (*Practices for Scaling Lean & Agile Development*), and also online at https://less.works/less/technical-excellence/architecture-design.html.

In LeSS, the product/system architectural guides covers conceptual changes, behavior changes, and technical changes. There are over 30 specific guides to succeed with Agile architecture at scale. For example, have a Design Community, do design workshops with Agile modeling, create clean code, do unit Test Driven Development (TDD) and acceptance TDD, avoid PowerPoint architects, do current-architecture learning workshops, use dependency injection, learn and apply design patterns, and many more.

Technology Environment:

Scaling approach addresses typical technology environment barriers to Agile through the following mechanisms:

In LeSS, we recognize that delayed integration also delays communication and feedback, and inhibits coordination. Therefore, in LeSS a key technical environmental feature is the practice and support for continuous integration (CI). This implies the behavior changes of frequent check-in and merge, and avoiding branching, and a fast and scalable CI system. This is so important that there is a chapter on scaling Continuous Integration in the second LeSS book, *Practices for Scaling Lean & Agile Development.*

Another key technical element in LeSS, related to a CI system, is large-scale automated test support, both at the overall acceptance level, and at the unit level.

Team Technical Practices:

Scaling approach addresses typical team technical practices barriers to Agile through the following mechanisms:

LeSS includes the subject of technical excellence.[20] In brief, in LeSS, the guide is that teams do specification by example, continuous integration, automated acceptance TDD, unit TDD, clean coding, and apply the 30-plus guides in the LeSS "architecture and design" guidelines, including design workshops with Agile modeling, and current architecture learning workshops.

[20] See https://less.works/less/technical-excellence/index.html for a full description.

[DISTRIBUTION STATEMENT A] This material has been approved for public release and unlimited distribution.

Team Management and Coordination Practices:

Scaling approach addresses typical team management and coordination practices barriers to Agile through the following mechanisms:

In LeSS, coordination between the teams is handled by the self-organizing teams themselves, with many guides to support decentralized coordination. These include Communities (e.g., of Practice), Just Talk, Communicate in Code (e.g., via CI systems and social-coding tools such as GitHub), Traveller-Teacher, multi-team shared-space events, multi-team design workshops, rotating infrastructure feature teams, component mentors, Open Space meetings, Lean Coffee, and more.

In LeSS, there is one Sprint shared by all the teams working together on the same product. The goal is to have a shippable product at the end of the (e.g., two-week) Sprint. And there is a common Sprint Planning Part 1, and a common Sprint Review. All these elements also strongly relate to and support coordination.

Program Practices:

Scaling approach addresses typical program practices barriers to Agile through the following mechanisms:

Most large-scale product development companies that I've worked with to create products (e.g., Xerox) don't have the notion or org elements of "programs." "Programs" and "projects" is a paradigm associated with internal development (such as in a bank), and also with outsourcing. In product companies, it is common to have never had the notion of project or program. Rather, the paradigm is that there is a product, and *continuous product development*, and there are new requirements to add to it, month after month, "forever." In this model there is no project or program. This is the product-centric model of development, and the one promoted by Scrum, hence the language of Product Owner and Product Backlog.

The Scrum/LeSS model can be thought of as a continuous flow rather than as a series of long projects. This is continuous product development. The teams add value to the product Sprint by Sprint, with a shippable product every Sprint that is in fact normally also shipped to production every Sprint, or when enough value is added to warrant a release. From the teams' perspective, the development just continues...forever. There is no project—nor a project manager or other person responsible for just one release. There is only the stable Product Owner and stable teams responsible for each Sprint as the years pass. The management structure for the product group does not change between releases, it stays the same..."forever."

So in this way the "Program Practice" is solved...by being replaced by a Product Practice, with related organizational design elements: long-term stable Product Owner (product manager), stable teams, shippable product every Sprint, ever-changing Product Backlog that evolves based on learning, and so forth. This eliminates the need for traditional program/project management.

[DISTRIBUTION STATEMENT A] This material has been approved for public release and unlimited distribution.

Appendix F Elaborations on Scaled Agile Framework by Dean Leffingwell

Preface:

At the time of this research, Scaled Agile Inc. currently has two frameworks in the market, SAFe[21] and SAFe for Lean Systems Engineering.[22] We are in the process of combining them in the next release, Scaled Agile Framework 4.0, for Lean Software and Systems Engineering.[23]

SAFe LSE has constructs designed specifically for scalability for the largest and most complex systems contexts. Many Department of Defense suppliers, such as Raytheon, SRA International, Ball Aerospace, Lockheed Martin, and others, have contributed, directly or indirectly, to the content. As SAFe-LSE is the most relevant context for this research, I'll answer the questions primarily from that perspective. For additional depth of understanding, I've referenced the live, online content that describes the linked topic more fully. (***Bold Italics*** indicates that there is a guidance article on SAFe or SAFe LSE for that topic.) Both frameworks are freely revealed and publicly facing. The growing body of SAFe knowledge is published and advanced on a continuous basis.

Dean Leffingwell, SAFe Chief Methodologist

Business and Acquisition

SAFe addresses business and acquisition barriers to Agile adoption in a variety of ways. ***Customer*** and ***Supplier*** articles directly address the need for a collaborative environment where customers are continuously and integrally involved in defining and assessing value. Companion articles, including ***Adaptive Requirements and Design***, ***Fixed vs. Variable Solution Intent***, ***Agile Release Train***, and others provide visibility and transparency whereby the development enterprise, Customers and Suppliers can contract and collaborate in a more Agile way to better achieve the customer's real goals.

Organizational Climate

SAFe addresses the organizational climate and change management challenge in multiple ways. First, Lean-Agile Leadership is required to adopt this new way of "being" and "doing." SAFe provides leadership ***courseware***, ***video training*** and ***online guidance*** to help develop the skills of emerging Lean-Agile leaders.

SAFe provides a structured approach to Lean-Agile *values*, *principles* and practices. Lean *values* are described in the ***House of Lean***, which includes shortest sustainable lead time, innovation, flow, respect for people and culture, and relentless improvement. SAFe's ***Lean-Agile Principles***

[21] See scaledagileframework.com for more information.

[22] See safe-lse.com for more information.

[23] Scaled Agile Framework 4.0 is now available. See http://www.scaledagileframework.com/welcome-to-safe-40/.

[DISTRIBUTION STATEMENT A] This material has been approved for public release and unlimited distribution.

provides in-depth guidance on nine immutable *Principles* of Lean-Agile development at enterprise scale. The SAFe online knowledgebase contains comprehensive *practice* guidance.

Project, Customer and Team environment

SAFe comprehensively addresses the project, customer and team environment with extensive Lean-Agile organization structure design and operating practices. First, build self-organizing ***Agile Teams***. Second, organize Agile Teams into self-organizing and self-managing Agile programs (Agile Release Trains) that foster cadence-based interaction between the development teams, ***Business Owners***, ***Customers,*** and ***Suppliers***. Third, understand, address, and manage the flow of work from the ***Value Stream*** and ***Portfolio*** levels.

SAFe's mantra is "nothing beats an Agile team," and the values and principles of the Agile Manifesto—as well as team practices based on Scrum, XP and Kanban—take center stage in the framework. Agile Team practices are taught, fostered, and protected by the enterprise's own ***Lean-Agile Leaders***. All teams in the program plan together, integrate and demo assets together and, inspect and adapt their processes on a regular cadence.

System Attributes

SAFe applies a variety of scalable, Agile technical practices to foster quality large-scale solutions with architectural robustness. ***SAFe's Seven Principles of Agile Architecture*** highlight how using emergent design and intentional architecture together help the teams create the ***Architectural Runway*** necessary to successfully build and deploy upcoming features. This "continuous architecture" approach guides and counsels teams to build resilient, loosely coupled architectures that are testable, deployable and lend themselves to change without the need for Big-Design Up-Front (BDUF) or "future proofing." SAFe also provides guidance on ***Nonfunctional Requirements***—attributes such as usability, scalability, reliability, and fitness for use.

Fast feedback is an essential part of SAFe. Full system demos—the combined work of all teams for an Agile Release Train—are held every two weeks. Larger ***Program Increment*** demos bring together key stakeholders—including Business Owners, Suppliers and Customers—for a full program review every 8-10 weeks at both the ***program*** and ***value stream*** levels. Teams work from a common ***Program Backlog***, which is under the authority of ***Product Management*** and ***Product Owners***, who serve as day-to-day customer proxies.

Technology Environment

SAFe provides the principles and guidance to support Agile infrastructure with a focus on ***Continuous Integration and Testing*** and ***Test Automation***. SAFe motivates teams to continuously enhance their technical infrastructure required for continuous integration, and fast build cycles needed for ***System and Solution demos*** that serve as "pull events" for integrating various solution elements into an integrated whole. A ***System Team*** helps Agile Teams build and maintain the necessary infrastructure.

Prescribed SAFe practices include team-based Big Visible Information Radiators, Kanban boards, Cumulative Flow Diagrams, Feature Progress Reports, and Program Boards that identify and

[DISTRIBUTION STATEMENT A] This material has been approved for public release and unlimited distribution.

track milestones, deliveries, and dependencies. SAFe provides a comprehensive view of ***Metrics*** and reporting at the Team, Program, Value Stream, and Portfolio Levels.

Team Technical Practices

SAFe provides guidance on team technical and quality practices including Test-First Development, Continuous Integration, and Agile Architecture. SAFe also provides a scalable Definition of Done that addresses ***Story, Feature, Capability, Nonfunctional Requirements*** and full, system-level ***Release*** quality and completeness criteria. Lean-Agile Principles and short Iterations emphasize small batch sizes for faster throughput and lower variability. Work-in-Process (WIP) limits are used to match actual team capacity to demand, improving the team's ability to deliver quality. Agile Teams work from a common backlog and prioritize job sequencing with using Weighted Shorted Job First (WSJF).

Team Management and Coordination Practices

SAFe's Lean Agile Principle 9 is ***Decentralize Decision-Making***. This is supported by ***Principles 1, Take an Economic View***, and ***2, Apply Systems Thinking***. These principles help establish the decision-making framework that empowers fast, local decision-making. Decentralized decision-making is also embedded in the empowerment of SAFe content authorities, ***Solution Management***, ***Product Management***, and ***Product Owner***. Self-organizing and self-managing Agile teams and programs fulfill their responsibility without managers assigning tasks. All work is governed by a shared Economic Framework. Cross-functional Agile teams are based primarily on Scrum, with overlays of XP quality practices and Kanban for flow and visibility. Short iterations are required; two weeks is the standard.

Program Practices

Agile program practices are one of the richest areas of SAFe. The primary value delivery mechanism is the ***Agile Release Train***, a team-of-Agile-teams that cooperates to deliver fully integrated value incrementally, on a regular synchronized cadence. Three specific roles create a "troika" that is most effective in guiding the train to the right value. These include the ***Release Train Engineer*** (Agile Program Manager), ***Product Management*** (content authority for the program), and the ***System Architect/Engineer***, who guides the intentional architecture necessary to assure that the system is a whole system, and not just a set of parts. Other roles such as ***UX***, ***System Team***, ***DevOps*** and ***Shared Services*** are part of the train planning and delivery.

Team boundaries are enforced by the Agile Team and Agile Release Train constructs, with clear, Lean-Agile responsibilities defined for all. Extensive Lean and Agile ***Metrics*** are provided, including measures such as Agile Team, and Agile Release Train performance self-assessments.

Alignment is one of the core values of SAFe and the construct of PI Planning with all members of the Agile Release Train in a single room, planning together, helps create the synchronization needed. Other constructs such as ART Sync meetings, joint demos and ***Inspect and Adapt*** workshops provide synchronization across the whole PI.

[DISTRIBUTION STATEMENT A] This material has been approved for public release and unlimited distribution.

References

[Ambler 2012]
Ambler, Scott W. & Lines, Mark. *Disciplined Agile Delivery: A Practitioner's Guide to Agile Software Delivery in the Enterprise*. IBM Press. 2012. ISBN-10: 0132810131.
ISBN-13: 978-0132810135

[Bachman 2012]
Bachman, Felix; Nord, Robert; & Ozkaya, Ipek. Architectural Tactics to Support Rapid and Agile Stability. *CrossTalk*. May/June 2012. Pages 20-25. http://resources.sei.cmu.edu/library/asset-view.cfm?assetid=87764

[Beck 2001]
Beck, Kent; Beedle, Mike; van Bennekum, Arie; Cockburn, Alistair; Cunningham, Ward; Fowler, Martin; Grenning, James; Highsmith, Jim; Hunt, Andrew; Jeffries, Ron; Kern, Jon; Marick, Brian; Martin, Robert C.; Mellor, Steve; Schwaber, Ken; Sutherland, Jeff; & Thomas, Dave. *Agile Manifesto*. 2001. http://agilemanifesto.org

[Conway 1968]
Conway, Melvin E. *How Do Committees Invent?* 1968. http://www.melconway.com/Home/Committees_Paper.html

[Dunbar 1992]
Dunbar, R. I. M. Neocortex Size as a Constraint on Group Size in Primates. *Journal of Human Evolution.* Volume 22. Number 6. 1992. Pages 469–493. doi:10.1016/0047-2484(92)90081-J.

[Frick 2015]
Frick, Walter. The Curious Science of When Multitasking Works. *Harvard Business Review*. https://hbr.org/2015/01/the-curious-science-of-when-multitasking-works. January 6, 2015.

[Garcia-Miller 2006]
Garcia-Miller, Suzanne & Turner, Richard. *CMMI Survival Guide: Just Enough Process Improvement.* Addison-Wesley. 2006. ISBN: 0-321-42277-5

[Harrison 1996]
Harrison, N. B. & Coplien, J. O. Patterns of Productive Software Organizations. *Bell Labs Technical Journal*. Volume 1. Number 1. May/June 1996. Pages 138-145.

[Larman 2008]
Larman, Craig & Vodde, Bas. *Scaling Lean and Agile Development: Thinking and Organizational Tools for Large-Scale Scrum*. Addison-Wesley. 2008. ISBN-13: 978-0321480965
ISBN-10: 0321480961

[Leffingwell 2007]
Leffingwell, D. *Scaling Software Agility*. Addison-Wesley. 2007.

[DISTRIBUTION STATEMENT A] This material has been approved for public release and unlimited distribution.

[Little 1961]
Little, J. D. C. (1961). A Proof for the Queuing Formula: L = λW. *Operations Research* Volume 9. Number 3. Pages 383–387. doi:10.1287/opre.9.3.383. JSTOR 167570.

[Mooz 2003]
Mooz, Hal; Forsberg, Kevin; & Cotterman, Howard. *Communicating Project Management: The Integrated Vocabulary of Project Management and Systems Engineering*. John Wiley and Sons. 2003. ISBN 0-471-26924-7.

[Nord 2012]
Nord, R. L.; Ozkaya, I.; Kruchten, P.; & Gonzalez-Rojas, M. In Search of a Metric for Managing Architectural Technical Debt. Pages 91–100. In *Proceedings of the Joint Working IEEE/IFIP Conference on Software Architecture & European Conference on Software Architecture (ECSA)*. Helsinki, Finland, August 2012. IEEE Computer Society Press, 2012.

[Reinertsen 2009]
Reinertsen, Donald. *Principles of Product Development Flow*. Celeritas Publishing. 2009. ISBN-10: 1935401009

[Reynolds 2011]
Reynolds, Susan. Are You Smothering Your Brain's True Genius? Multi-Tasking and Information Overload Impinge on Creativity [blog post]. *Prime Your Gray Cells*. 2011. https://www.psychologytoday.com/blog/prime-your-gray-cells/201107/are-you-smothering-your-brain-s-true-genius

[Rising 1998]
Rising, Linda. *The Patterns Handbook: Techniques, Strategies, and Applications.* Cambridge University Press. 1998. ISBN 0-521-64818-1

[SEI 2015]
Ernst, Neil; Bellomo, Stephany; Nord, Robert; & Ozkaya, Ipek. *Enabling Incremental Iterative Development at Scale: Quality Attribute Refinement and Allocation in Practice.* CMU/SEI-2015-TR-008. Software Engineering Institute, Carnegie Mellon University. 2015. http://resources.sei.cmu.edu/library/asset-view.cfm?assetid=439055

[Sutherland 1995]
Sutherland, Jeff & Schwaber, Ken (1995). Business Object Design and Implementation: OOPSLA '95 Workshop Proceedings. The University of Michigan. Page 118. ISBN 3-540-76096-2.

[DISTRIBUTION STATEMENT A] This material has been approved for public release and unlimited distribution.

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 December 2016	3. REPORT TYPE AND DATES COVERED Final
4. TITLE AND SUBTITLE Scaling Agile Methods for Department of Defense Programs		5. FUNDING NUMBERS FA8721-05-C-0003
6. AUTHOR(S) William Hayes, Mary Ann Lapham, Suzanne Miller, Eileen Wrubel, Peter Capell		
7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES) Software Engineering Institute Carnegie Mellon University Pittsburgh, PA 15213		8. PERFORMING ORGANIZATION REPORT NUMBER CMU/SEI-2016-TN-005
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)

Most introductory discussions of Agile software development have focused on team management concepts and the implications of the Agile Manifesto for a single, small team. The focus now includes scaling these concepts for a variety of applications. The context in which Agile methods are employed drives important choices for how the work is done. Published frameworks and commercial training available in the market offer a variety of solutions for scaling Agile. This report addresses what is meant by scaling, contextual drivers for implementation choices, and the frame-works available for use today.

14. SUBJECT TERMS Agile, Scrum, development	15. NUMBER OF PAGES 62
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