

MTR100489

MITRE TECHNICAL REPORT

MITRE

Handbook for Implementing Agile in Department of Defense Information Technology Acquisition

**Carlton Northern
Dr. Kathleen Mayfield
Robert Benito
Michelle Casagni**

15 December 2010

Approved for public release: 11-0401.
Distribution Unlimited.
Copyright © 2010 - The MITRE Corporation.
All rights reserved.

The author's affiliation with The MITRE Corporation is provided for identification purposes only, and is not intended to convey or imply MITRE's concurrence with, or support for, the positions, opinions or viewpoints expressed by the author.

Report Documentation Page

Form Approved
OMB No. 0704-0188

Public reporting burden for the 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. Respondents should be aware that notwithstanding any other provision of law, no person shall be subject to a penalty for failing to comply with a collection of information if it does not display a currently valid OMB control number.

1. REPORT DATE 15 DEC 2010	2. REPORT TYPE	3. DATES COVERED 00-00-2010 to 00-00-2010	
4. TITLE AND SUBTITLE Handbook for Implementing Agile in Department of Defense Information Technology Acquisition		5a. CONTRACT NUMBER	
		5b. GRANT NUMBER	
		5c. PROGRAM ELEMENT NUMBER	
6. AUTHOR(S)		5d. PROJECT NUMBER	
		5e. TASK NUMBER	
		5f. WORK UNIT NUMBER	
7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES) MITRE Corporation, 7525 Colshire Drive, McLean, VA, 22102		8. PERFORMING ORGANIZATION REPORT NUMBER	
9. SPONSORING/MONITORING AGENCY NAME(S) AND ADDRESS(ES)		10. SPONSOR/MONITOR'S ACRONYM(S)	
		11. SPONSOR/MONITOR'S REPORT NUMBER(S)	
12. DISTRIBUTION/AVAILABILITY STATEMENT Approved for public release; distribution unlimited			
13. SUPPLEMENTARY NOTES			
14. ABSTRACT The ability to rapidly produce and deploy information technology (IT) based capabilities in the United States Department of Defense (DOD) that meet the ever-evolving needs of the Warfighter is a challenging endeavor. DOD acquisition projects typically follow a highly structured, top-down, step-by-step process, based on the assumption that an end state is known. Unfortunately, this is rarely the case in modern IT projects. Long development cycles and rapidly changing requirements make it difficult to properly identify the end state of an IT system at the onset of the project. Agile development is an industry accepted software development practice that is now beginning to emerge in Government programs. This report includes background information on Agile principles and methodologies from peer-reviewed industry and academic materials, MITRE technical reports, as well as interviews with members of DOD Programs that have implemented Agile development methodologies. This report describes how Agile development principles can be applied to an IT systems engineering effort, and explains how an Agile methodology could be used to benefit DOD Government acquisition and development programs. This report is intended to be used as a guidebook, to provide specific recommendations regarding the implementation of an Agile methodology. This handbook will be a living document and will be updated as lessons are learned.			
15. SUBJECT TERMS			
16. SECURITY CLASSIFICATION OF:			17. LIMITATION OF ABSTRACT
a. REPORT unclassified	b. ABSTRACT unclassified	c. THIS PAGE unclassified	Same as Report (SAR)
			18. NUMBER OF PAGES 107
			19a. NAME OF RESPONSIBLE PERSON

This page intentionally left blank.

Approved By:

Name and Title of Approval Signature

Date

MITRE

Abstract

The ability to rapidly produce and deploy information technology (IT) based capabilities in the United States Department of Defense (DOD) that meet the ever-evolving needs of the Warfighter is a challenging endeavor. DOD acquisition projects typically follow a highly structured, top-down, step-by-step process, based on the assumption that an end state is known. Unfortunately, this is rarely the case in modern IT projects. Long development cycles and rapidly changing requirements make it difficult to properly identify the end state of an IT system at the onset of the project.

Agile development is an industry accepted software development practice that is now beginning to emerge in Government programs. This report includes background information on Agile principles and methodologies from peer-reviewed industry and academic materials, MITRE technical reports, as well as interviews with members of DOD Programs that have implemented Agile development methodologies. This report describes how Agile development principles can be applied to an IT systems engineering effort, and explains how an Agile methodology could be used to benefit DOD Government acquisition and development programs. This report is intended to be used as a guidebook, to provide specific recommendations regarding the implementation of an Agile methodology. This handbook will be a living document and will be updated as lessons are learned.

Acknowledgements

The authors would like to thank the government and MITRE staff members who took the time help in the creation of this handbook. This includes all those who shared their expertise through interviews, as well as those who took the time to read the document and provide feedback. These contributors include, but are not limited to: Kevin Buck, Patricia Carbone, Dr. Robert Cherinka, Dave Edwards, Kevin Gunn, Dr. John Keller, and Dr. Jody Mandeville.

Table of Contents

1	Introduction.....	1-1
1.1	Purpose	1-1
1.2	Intended Use of Document.....	1-1
1.3	Naming Conventions.....	1-1
1.4	Document Organization	1-2
2	The Software Development Process	2-3
2.1	The Pitfalls of Traditional Software Development	2-3
2.1.1	A High Rate of Failure	2-3
2.1.2	Assumes Well-Defined Requirements and Overall Stability	2-3
2.1.3	Linear, Formal Process (“Heavy Process”).....	2-4
2.2	IT Development as it Relates to the Department of Defense	2-5
3	Agile Software Development.....	3-6
3.1	What is Agile Software Development?.....	3-6
3.1.1	Agile History	3-6
3.1.2	Agile Principles	3-6
3.1.3	Highly Disciplined Development Process.....	3-8
3.2	What Agile Software Development is Not.....	3-9
3.3	Agile Methods	3-10
3.3.1	Extreme Programming (XP).....	3-10
3.3.2	Scrum.....	3-12
3.3.2.1	Scrum Roles	3-12
3.3.2.2	Scrum Process	3-12
3.3.3	Lean Software Development	3-13
4	Guidelines for Embracing Agile	4-15
4.1	Deciding to ‘Go Agile’	4-15
4.2	Selecting the Methodology or Methodologies	4-17
4.3	New Development and Integration.....	4-18
4.4	Defining “Done”.....	4-19
4.5	Agile Training and Using an Agile Coach	4-19
4.6	Distributed Teams	4-20
4.7	Scaling Agile for Large Projects	4-21
4.8	Agile In the Enterprise	4-21
4.9	Agile Risk Management.....	4-22

4.10	Agile Contracts	4-23
4.11	CMMI and Agile	4-24
5	Agile/Rapid Acquisition	5-27
6	Development Best Practices	6-30
6.1.1	Expressive Code	6-30
6.1.2	Regression Testing	6-30
6.1.3	Continuous Integration	6-31
6.1.4	Pair Programming.....	6-31
7	Tools 7-33	
7.1	Development Tools	7-33
7.2	Agile Project Management Tools.....	7-33
7.2.1	Trade Study Methodology.....	7-34
7.2.2	Summary of Tool Results.....	7-36
8	Case Studies.....	8-38
8.1	Program A	8-38
8.1.1	Project Summary	8-38
8.1.1.1	Summary of the Agile Methodology and Related Processes	8-38
8.1.2	Challenges	8-39
8.1.3	Benefits.....	8-39
8.1.4	Lessons Learned	8-39
8.2	Joint Operational Planning and Execution System (JOPES)	8-39
8.2.1	Project Summary	8-39
8.2.1.1	Summary of the Agile Methodology and Related Processes	8-40
8.2.2	Challenges	8-40
8.2.3	Benefits.....	8-40
8.2.4	Lessons Learned	8-40
9	Systems Engineering Plan Template.....	9-41
9.1	Agile Systems Engineering Plan Introduction	9-41
9.2	Capability Requirements Analysis	9-42
9.2.1	User Stories	9-42
9.2.2	Storyboarding and Mockups	9-44
9.2.3	Prioritization.....	9-45
9.2.4	Capability Estimation	9-45
9.3	Planning.....	9-46
9.4	Sprint Execution.....	9-47

9.4.1	Sprint Development.....	9-47
9.4.2	Sprint Integration.....	9-48
9.4.3	Sprint Testing	9-48
9.4.4	Sprint Retrospective	9-49
9.4.5	Sprint Review	9-49
9.5	Tools	9-49
9.6	Roles.....	9-50
9.6.1	Product Owner.....	9-50
9.6.2	ScrumMaster	9-50
9.6.3	Integration Team	9-51
9.6.4	Stakeholders	9-51
9.6.5	Users	9-51
9.7	Trade Studies	9-51
10	Conclusion	10-53
11	References	A-1
Appendix A	Results	A-6
A.1	SoftwareForge	A-6
A.2	Atlassian JIRA with GreenHopper.....	A-8
A.3	Agilo Free.....	A-10
A.4	ScrumWorks Pro	A-11
A.5	tinyPM.....	A-13
A.6	ScrumNinja.....	A-15
A.7	ScrumDesk	A-16
A.8	ScrumWorks Basic	A-17
A.9	XPlanner+.....	A-19
A.10	Rally	A-20
Appendix B	CMMI and Agile Paradigm Comparison	B-21
Appendix C	Recommended Reading	C-24
C.1	Websites	C-24
C.2	Books.....	C-24
Appendix D	Glossary	D-26
Appendix E	Use Case Template.....	E-28
Appendix F	Program of Record Technology or Initiative Transition Barriers/Enablers.....	F-29
F.1	Funding.....	F-29
F.2	Design.....	F-30

F.3	Documentation	F-33
F.4	Test	F-35
F.5	Program of Record	F-36
F.6	Sustainment	F-38
F.7	Advocacy.....	F-40

List of Figures

Figure 2-1: Project Uncertainty and Cost Comparison over Time (Project Management Institute, 2008, p. 17)	2-4
Figure 2-2: Recommended Agile Acquisition Process	2-5
Figure 3-1: People Focused Aspects of Agile.....	3-8
Figure 3-1 eXtreme Programming Practices (source: xprogramming.com).....	3-11
Figure 3-2 Scrum Process (source: Scrum Primer V1.2).....	3-13
Figure 4-1 Dimensions Affecting Method Selection (source: Boehm & Turner, 2004, p. 56. Note: Personnel description was modified slightly for ease of understanding).....	4-16
Figure 4-2: Scrum and XP (source: http://www.controlchaos.com/about/xp.php)	4-18
Figure 5-1 New Acquisition Process	5-28
Figure 5-2 Program of Record Technology or Initiative Transition Barriers/Enablers.....	5-29
Figure 6-1 Java For Loop Without Iteration Design Pattern	6-30
Figure 6-2 Java For Loop With Iteration Design Pattern	6-30
Figure 9-1: Agile Process.....	9-42
Figure 9-2 Wire-frame Mockup of iTunes Cover Flow Feature (source: http://www.balsamiq.com/products/mockups/examples#mytunez)	9-44
Figure 9-3: Iteration Planning Meeting Flow	9-46
Figure 9-4: Sample Burn Down Chart	9-48
Figure G-10-1 Program of Record Technology or Initiative Transition Barriers/Enablers.....	F-29

List of Tables

Table 4-1 Rank Ordered Agile Success Factors (source: Chow & Cao, 2008, p. 970).....	4-16
Table 7-1 Tool Evaluation Criteria.....	7-34
Table 7-2 Tool Study Results.....	7-36
Table 9-1 Cost Effectiveness Parameters.....	9-52
Table 10-1 CMMI and Agile Paradigm Comparison Table.....	B-21
Table 10-2 Recommended Websites Table.....	C-24
Table 10-3 Recommended Books Table.....	C-24
Table 10-4 Use Case Template.....	E-28

This page intentionally left blank.

1 Introduction

Information Technology (IT) plays a large part in the Department of Defense (DOD), it offers immense capability to weapons systems, infrastructure and business systems. In 1970, software accounted for only 20 percent of weapon system functionality, whereas, in 2000 it accounted for as much as 80 percent (Defense Science Board, 2000). Unfortunately, IT systems development in the DOD is fraught with budget-overruns and time delays in achieving initial operational capability (IOC). According to a July 2008 Government Accountability Office (GAO, 2008a) report, 48 percent of the federal Government's major IT projects have been rebaselined at least twice. Another GAO (2008b) report cites that the overall portfolio of DOD IT programs has experienced a 21-month delay in delivering initial operational capability to the war fighter, and 12 percent are more than four years late. Over the last decade, commercial industry has responded to similar dismal statistics regarding IT projects by implementing Agile software development methods (Johnson, 2009).

Agile software development is an industry accepted software development practice that is now beginning to emerge in Government programs. This report describes how Agile development principles can be applied to a IT systems engineering effort, and explains how an Agile methodology could be used to benefit DOD Government acquisition and development programs.

1.1 Purpose

The purpose of this document is to provide DOD programs with guidance on implementing an Agile process. This document presents the findings and recommendations of multiple sources to include MITRE technical reports, peer reviewed journals from industry and academia, open source materials, interviews with members of DOD Agile teams, and a market survey of Agile project management tools.

1.2 Intended Use of Document

This handbook is geared towards DOD projects; however it can be adapted and used by any program within the government. For DOD program managers, this handbook will provide details on how to apply Agile techniques to a DOD IT intensive program. For program managers and developers it can be used to gain an understanding of Agile principles, processes, methods and best practices. It will provide guidance and insight into tools and techniques that can be introduced to help a program become more agile in how they develop, integrate, and deliver capabilities. This handbook can also provide program managers with the necessary information to help make decisions regarding whether or not to use Agile in a project.

The document can be read end-to-end, or the reader can selectively read only what is of interest to them using the available hyperlinks or Table of Contents.

1.3 Naming Conventions

The following naming conventions are used in this document:

- The term "Agile" will be capitalized when it refers to the Agile software development process, and will be lower case when being used as a verb or to describe an action or activity.
- When referring to the role, the titles for Product Owner and ScrumMaster will be capitalized.
- When referring to a specific instance, the term "sprint" will be capitalized, such as Sprint 1, all other times the term will be lower case.

1.4 Document Organization

Section 2

The Software Development Process

In this section we describe the traditional software development process, its shortcomings, and how it relates to the Department of Defense.

Section 3

Agile Software Development

This section introduces Agile Software Development, describing its history and principles, and summarizes Extreme Programming, Scrum, and Lean Development.

Section 4

Guidelines for Embracing Agile

This section discusses critical areas of that should be explored for a Program of Record when using an Agile Development methodology. These subjects include how to make the decision to go 'Agile'; deciding on an Agile methodology; defining "done"; team skills and training; how to write Agile contracts; Agile and CMMI; how to manage risk with Agile; Agile in the Enterprise and how to scale it; and lastly, using Agile with distributed teams.

Section 5

Agile/Rapid Acquisition

In this section we define and summarize the processes of Agile/Rapid Acquisition and how they relate to a Program of Record.

Section 6

Development Best Practices

In this section we discuss best practices from Agile development methodologies for developing code.

Section 7

Tools

In this section, we take a look at Agile tools which aid in development of software, and we then discuss the results of a trade study completed on Agile project management tools to determine which tools are well suited for use in the DOD environment.

Section 8

Case Studies

This section includes two case studies of DOD programs that are using Agile methodologies and processes.

Section 9

Systems Engineer Plan Template

This section aims to ease the burden of creating a Systems Engineering Plan by providing an Agile methodology to use as a template for the creation of a program's Systems Engineering Plan.

Section 10

Conclusion

This section contains the summary and recommendations of this handbook.

Section 11

References

This section contains the references that are cited in this handbook.

2 The Software Development Process

2.1 The Pitfalls of Traditional Software Development

Software development methodologies have evolved since the early 1960's mainframe computing era. They run the gamut of no formal process at all (e.g. early 'code and fix' development), to rigid development processes based on extensive up-front planning, to more recent processes that embrace change throughout the development cycle (Surendra, 2008). DOD acquisition projects typically follow a highly structured, top-down, step-by-step process, based on the assumption that an end state is known. Unfortunately, this is rarely the case in information technology (IT) projects. IT, as defined in this report, is any system or subsystem of hardware and/or software whose purpose is acquiring, processing, storing, or communicating information or data (Defense Science Board, 2009). This section describes the pitfalls of traditional software development and why these processes can lead to underestimated costs, schedule overruns and ultimately, dissatisfied customers and users.

2.1.1 A High Rate of Failure

Traditional software development methodologies, defined as a rational, highly documented, plan-driven processes, such as waterfall and spiral, use highly controlled and predictable frameworks to guide projects to completion. The intent of these approaches is to manage uncertainty by clearly defining a desired end state, and developing up-front, detailed plans to achieve this end state. This rigidity prevents projects from readily responding to emerging technical and functional requirements (Lindstrom & Jeffries, 2004).

The Standish Group (2009) reports that over 68% of IT projects are delivered late, over budget, or do not fully address the required system functionality (Johnson, 2009). In 2008 alone, the US Government placed over 400 software projects, at a cost of \$25.2 billion, on a management watch list because they were failing or performing poorly (United States General Accounting Office, 2008a). Unfortunately, success or failure is typically related to the 'iron triangle' of cost, scope, and schedule rather than the customer's satisfaction with the end product (Fernandez & Fernandez, 2008). With respect to DOD acquisition, where the focus is on the Warfighter, software project failures are not only measured by statistics or dollars, but ultimately, by their inability to perform mission critical capabilities, putting lives at risk.

2.1.2 Assumes Well-Defined Requirements and Overall Stability

IT systems development can be fraught with uncertainty and ambiguity making it difficult to accurately define the end state up front. This may be the main reason for the high rate of failure on IT development projects (Atkinson, Crawford, & Ward, 2006). There are several contributing factors to the high level of uncertainty. The product is intangible or abstract, which makes it difficult for users to define what they want up front. This can be complicated even more by stakeholders with differing objectives and perceptions of the problem space and what they think the system should do. These differences can increase uncertainty and misunderstanding. In addition, software development has a high level of technical complexity due to numerous internal and external interrelationships in the code and interfaces, which can result in unforeseen issues. It is hard to predict the impact of these factors at the beginning of the project.

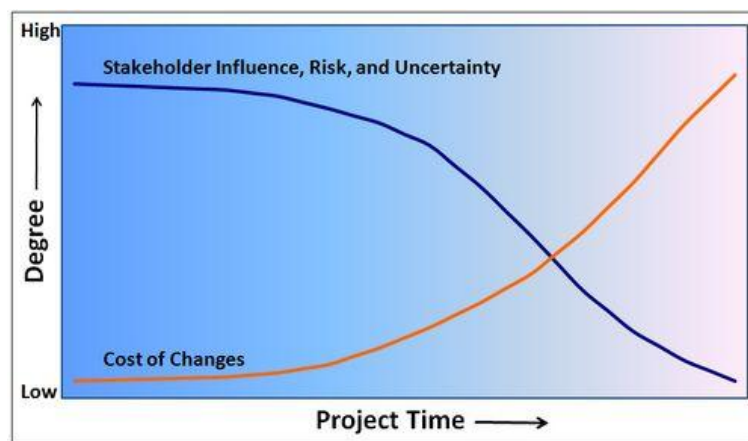


Figure 2-1: Project Uncertainty and Cost Comparison over Time (Project Management Institute, 2008, p. 17)

In a generic project life cycle as depicted in Figure 2-1, risk, uncertainty, and stakeholder influence are typically highest at the start of the project (Project Management Institute, 2008, p. 17). Yet in many IT development projects, this is when detailed project planning and design documentation is completed and baselined. In other words, planning and design is completed when the least information is known. Based on this model, as understanding increases, the cost of making corrective changes increases – creating the need to ‘stay on track’ with a sub-optimal plan rather than developing the optimal product for the customer. This creates a situation in which the team is constantly fighting with reality to keep the plan on track.

2.1.3 Linear, Formal Process (“Heavy Process”)

Some form of methodology is needed to manage the multitude of tasks associated with an IT development project; however the approach chosen should be matched to the needs of the project. Traditional software development approaches have been considered ‘heavyweight’ processes defined by up-front planning and formal documentation, and functional teams that develop the product in a linear phase approach (requirements, design, etc.). This method was referred to as a relay race by Takeuchi and Nonaka (1986) due to the way the product is handed off from one functional team to another at the end of each phase. This may be acceptable for incremental product improvement or enhancement to an existing, well-understood product, but can actually hinder creativity on a new product effort (Harmancioglu, McNally, Calatone, & Durmusoglu, 2007).

Additionally, decisions are not made during development phases; rather, they are delayed until the end of each phase, reducing the project’s ability to adapt to change (Steffens, Martinsuo, & Arto, 2007). Because of these considerations, a plan-driven, heavyweight process may not be the best choice on a highly innovative software development effort. This caution can be further justified by comparing the level of control with team performance. Davis, Eisenhardt, and Bingham (2009) explain that high control on well-defined projects is acceptable in a stable environment; but not for a highly complex project because the environment is challenging and chaotic. The team needs to be able to learn and adjust to this changing environment. Many software development projects fail because they are more complex than what the project team initially anticipated (Xia & Lee, 2004). Following a plan-based approach can hide this complexity, and cause a project to spiral out of control.

2.2 IT Development as it Relates to the Department of Defense

IT systems development of a Program of Record (POR) in the DOD adheres to the policies of DOD Directive 5000.1 and Instruction 5000.2, and the processes detailed in the Joint Capabilities Integration and Development System (JCIDS). These heavyweight processes were designed for large weapon systems development, and are not necessarily appropriate for the production of IT systems. Typically, the acquisition time frame is quite lengthy, as much as 5-10 years (Duquette, Bloom, & Crawford, 2008). This development cycle is too long for IT programs, and is made worse by the practice of locking down detailed requirements at the beginning of the project. By the time the product is finally fielded, the technology is dated and the functionality needed 5 to 10 years before may no longer address the Warfighter's current needs.

Excessive cost and schedule overruns on software projects have earned the attention of government senior leadership and oversight authorities. In December 2008, modifications were made to the DOD 5000 Directives and Instructions, and the JCIDS process which seek to reduce uncertainty early in the development process. Specifically, the creation of the "IT Box" in the JCIDS offers an alternate model to IT systems. The IT Box institutionalizes a process where the Combatant Command has requirements validation authority to reprioritize, add, or delete non-key performance parameter requirements. It also seeks to reduce the number of trips to the Joint Requirements Oversight Council in hopes of reducing time to deliver Initial Operating Capability (IOC). It is the opinion of the authors that these changes to the JCIDS process are a step in the right direction, but they do not go far enough to significantly reduce the time to IOC. Furthermore, the changes to the overall defense acquisition process include increased documentation and review processes which may slow development and increase costs for IT programs.

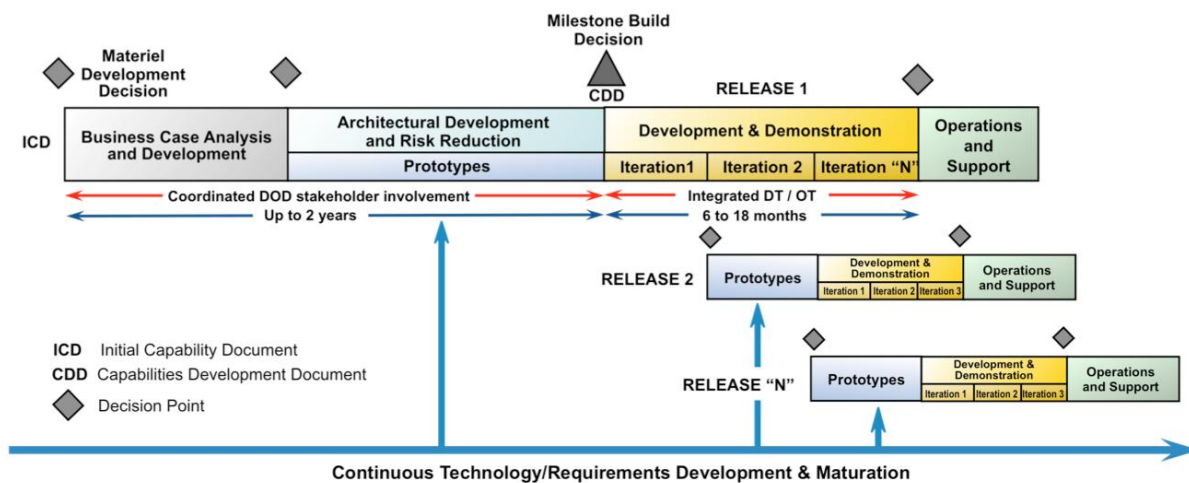


Figure 2-2: Recommended Agile Acquisition Process

Organizations such as the Defense Science Board (2009) and the National Academy of Sciences (2009) recommend a fundamentally new acquisition process for information technology (IT) programs that uses commercial Agile software development practices. Figure 2-2, from the Defense Science Board report (2009, p. 48) suggests a process that includes continuous user

involvement, smaller iterations leading to releases within 18 months, and continuous testing. This topic is discussed further in Section 5 Agile/Rapid Acquisition.

3 Agile Software Development

3.1 What is Agile Software Development?

3.1.1 Agile History

Agile methods, for which principles were agreed upon in the Agile Manifesto (Beck et al., 2001) in 2001, actually took root decades before with the introduction of iterative and incremental design and development and quality and productivity principles. These included Dr. Deming's 14 transformational points for management, which includes building in quality and breaking down departmental barriers (Deming, 1982) and his Plan-Do-Study-Act (PDSA) empirical model which included time-boxed, iterative, and incremental product development plans. These timeless principles and practices, which began in manufacturing, are beginning to emerge in software development practices.

As described by Barry Boehm in his 2006 paper titled "A View of 20th and 21st Century Software Engineering" software processes have evolved since the 1950's. Software development began by using engineering practices similar to those of hardware development. During the 1960's, this changed to a 'code and fix' model, often resulting in code that was difficult to maintain. In the 1970's, the Waterfall methodology was introduced to add structure and order to the development process. This process was an iterative approach focused on finding and fixing bugs as early in the process as possible. However, it was misconstrued as a strictly sequential approach which was further reinforced by government contracting standards (Waterfall model, 2010). The 1980's introduced software process maturity models, development tools, and software reuse. The 1990's brought about new methods to support reduced time to market constraints. Many of the early Agile methods emerged during this decade, which started a shift from plan-driven to value-driven software development.

This emergence began during the 1990's and the individual Agile approaches came together in 2001 into what is known as the Manifesto for Agile Software Development (see <http://agilemanifesto.org/>). This document, often referred to as the Agile Manifesto, describes a set of beliefs and guiding principles of Agile development and creates an alliance among the various methodologies.

3.1.2 Agile Principles

The Agile Manifesto describes the overarching beliefs of Agile software development (Beck et al, 2001):

We are uncovering better ways of developing software by doing it and helping others do it. Through this work we have come to value:

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

That is, while there is value in the items on the right, we value the items on the left more.

The authors augmented the manifesto with the following 12 principles for Agile software (<http://agilemanifesto.org/principles.html>):

1. Our highest priority is to satisfy the customer through early and continuous delivery of valuable software.
2. Welcome changing requirements, even late in development. Agile processes harness change for the customer's competitive advantage.
3. Deliver working software frequently, from a couple of weeks to a couple of months, with a preference to the shorter timescale.
4. Business people and developers must work together daily throughout the project.
5. Build projects around motivated individuals. Give them the environment and support they need, and trust them to get the job done.
6. The most efficient and effective method of conveying information to and within a development team is face-to-face conversation.
7. Working software is the primary measure of progress.
8. Agile processes promote sustainable development. The sponsors, developers, and users should be able to maintain a constant pace indefinitely.
9. Continuous attention to technical excellence and good design enhances agility.
10. Simplicity – the art of maximizing the amount of work not done – is essential.
11. The best architectures, requirements, and designs emerge from self-organizing teams.
12. At regular intervals, the team reflects on how to become more effective, then tunes and adjusts its behavior accordingly.

Of significance is that two of the four manifesto items and five of the 12 principles are people-oriented as demonstrated in Figure 3-1. This emphasizes that applying Agile methods is more than simply having a different way for developing software; the human element is just as important, as is delivering value to the customer rather than merely following a prescriptive development or management process.

People-Focused Aspects of the Agile Manifesto

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

People-Focused Aspects of the Agile Principles

1. Our highest priority is to satisfy the customer through early and continuous delivery of valuable software.
2. Welcome changing requirements, even late in development. Agile processes harness change for the customer's competitive advantage.
3. Deliver working software frequently, from a couple of weeks to a couple of months, with a preference to the shorter timescale.
4. **Business people and developers must work together daily throughout the project.**
5. **Build projects around motivated individuals. Give them the environment and support they need, and trust them to get the job done.**
6. **The most efficient and effective method of conveying information to and within a development team is face-to-face conversation.**
7. Working software is the primary measure of progress.
8. Agile processes promote sustainable development. The sponsors, developers, and users should be able to maintain a constant pace indefinitely.
9. Continuous attention to technical excellence and good design enhances agility.
10. Simplicity – the art of maximizing the amount of work not done – is essential.
11. **The best architectures, requirements, and designs emerge from self-organizing teams.**
12. **At regular intervals, the team reflects on how to become more effective, then tunes and adjusts its behavior accordingly.**

Figure 3-1: People Focused Aspects of Agile

3.1.3 Highly Disciplined Development Process

Agile methodologies require team members to be highly skilled and disciplined. Unlike traditional software development methodologies where each phase of the life-cycle sequentially informs the next, Agile methods perform these life-cycle phases essentially in parallel. A typical Agile incremental development period will last anywhere from one to six weeks and could include requirements analysis, design, development, testing, documentation and deployment.¹ This requires team members to be intimately familiar with all phases of the software development life-cycle and allows little time for “learning-on-the-job”.

Furthermore, the ability to write usable, fully-tested software in short periods of time implies strong knowledge of the computing environment, including pertinent technologies, coding standards, testing practices and design patterns. The decision to use web technologies, for instance, implies that the team member have extensive knowledge of the HTTP protocol, HTML, Javascript, server side scripting languages, and the associated coding, testing and deployment practices of each of these constructs.

¹ This does not imply that upfront planning, analysis or design is not encouraged. They are encouraged, but should be performed less rigorously than traditional development methods. The Product Backlog from the Scrum methodology is an example of such up-front planning and analysis,

This is not to say that only team members who have previous experience are eligible to work in Agile teams. Less experienced team members are appropriate, but should not make up the majority of the team. Pair programming is a good technique that will be discussed in more detail in Section 6.1.4 that helps less experienced team members get up to speed on a project. Furthermore, skill building is often encouraged of developers during time between iterations of development. This helps to ensure that the technology to be used during an iteration is better understood by those who are responsible for implementing it.

Lastly, it is up to the team to determine the necessary and sufficient level of documentation and testing to be performed. This is determined by the definition of "done", discussed more thoroughly in Section 4.6. It takes experience and discipline to determine not just what needs to be done, but more importantly, what doesn't need to be done. One all too common pitfall of IT development is not knowing when to stop. Test Driven Development (TDD) is one Agile practice in which no code is written unless it attempts to satisfy the requirements of a failing test. In other words, tests are created first followed by code. This provides traceability from requirements, to tests, to code, and ensures that a proper level of testing and pertinent code is written. To perform properly, TDD requires an extreme amount of discipline from the developer.

3.2 What Agile Software Development is Not

While it is important to describe what Agile software development is, it is also a worthwhile exercise to discuss what Agile software development is not. Agile methods are not a single approach to software development; they are a family of development processes that embody the principles of the Agile Manifesto. It is impractical to assume that any one development methodology can fit every software engineering problem. For instance, the safety critical software domain where malfunctions of the system can result in serious physical harm or death requires large amounts of initial analysis and design to ensure that safety requirements will be met. It is much cheaper to perform this work before any code has been written due to the extreme cost and time of writing safety critical code². In this case, an Agile methodology such as Lean software development, where emphasis is placed on minimizing work that has less value, may not be a wise choice. However, Scrum, an iterative and incremental project management framework, may work very well, if the practitioner includes initial sprints dedicated to upfront analysis and design.

This brings up the point that Agile methods are not "all or nothing" approaches. It is appropriate to pick and choose practices from various methodologies as needed. Scrum, for instance, relies on Agile practices from Extreme Programming (XP), TDD, and other development methodologies to carry out the actual writing of the code. Scrum focuses specifically on the management aspects of the project such as the roles, meetings and artifacts of the project (Kniberg, 2007).

While there is flexibility associated to Agile methods, it is important to understand the benefits and drawbacks of each practice. Too many Agile practitioners falsely apply a zero value to the items on the right of the Agile Manifesto core values. For example, it is not uncommon for the inexperienced Agile practitioner to blaze a trail of code without keeping up with the appropriate level of documentation, falsely assuming that Agile methods value working software to the exclusion of documentation.

² According to a report from LynuxWorks, 134,736 lines of code cost 269 person years (Matharu, (2005).

This is where experience and discipline come into play. The experienced Agile practitioner will set out to create enough documentation for himself or someone else to be able to develop new functionality in the future. This threshold should be defined in the definition of "done". The experienced Agile practitioner will also understand that documentation done while writing code is more pertinent and relevant than documentation produced at the end of the iteration or release, because it is at the moment of creation that code is most clearly understood.

In summary, it is important to understand not just what Agile development is, but also what it is not. Agile development is not a silver bullet. It is not an "all or nothing" approach. And it is not appropriate to skip proper engineering practices, claiming that is it "Agile".

3.3 Agile Methods

There are multiple Agile methodologies, but to scope the document appropriately, three methods will be summarized in this section: XP, Scrum, and Lean Development. According to Version One's "State of Agile Development Survey 2009", out of all Agile development methodologies in use, 83% of Agile teams use the following methodologies: Scrum (50%), Scrum / XP hybrid (24%), XP (6%) and Lean Development (3%).³ Furthermore, these methodologies were selected because they have slightly different objectives. XP is a software engineering Agile process, Scrum is an Agile project management framework that can be used alone or in coordination with any Agile process or processes, and Lean Development is focused on reducing waste, increasing quality, and establishing a culture of continuous improvement. As mentioned previously, adopting Agile practices is not an 'all or nothing' approach, nor is it a purist-only approach. It is important to match the best approach, mix or hybrid based on the needs of the individual project.

It is not the intent of this section to describe these three methodologies in great detail. The intent is to explain at a high-level these methodologies and to expose the reader to their core values/practices. The reader is encouraged to follow the links provided in each subsection to gain a greater understanding of the methodologies.

3.3.1 Extreme Programming (XP)

The following information is provided from "What is Extreme Programming?" on Ron Jeffries' site (<http://xprogramming.com/xpmag/whatisxp>).

"Extreme Programming is a discipline of software development based on values of simplicity, communication, feedback, and courage. It works by bringing the whole team together in the presence of simple practices, with enough feedback to enable the team to see where they are and to tune the practices to their unique situation." Of the three agile methodologies described in this section, XP is most focused on daily engineering processes for the software team. These include concepts that overlap with other agile practices, such as Scrum, by using user stories to define requirements. It also includes several 'extreme' concepts such as shared code ownership, writing the tests before the code, and developing code in pairs. One important aspect of XP is the use of automated tests that are run each time new code is developed. If new code causes a test to fail, no new code is written until the test passes. This continuous testing approach, to include integration testing, allows errors to be corrected as close as possible to when they are made.

³ It should be noted that Scrum dominates the Agile space with a large 74% of the market share when including the Scrum / XP Hybrid approach. For this reason, many people often use the terms "Agile" and "Scrum" somewhat synonymously even though their definitions are different.

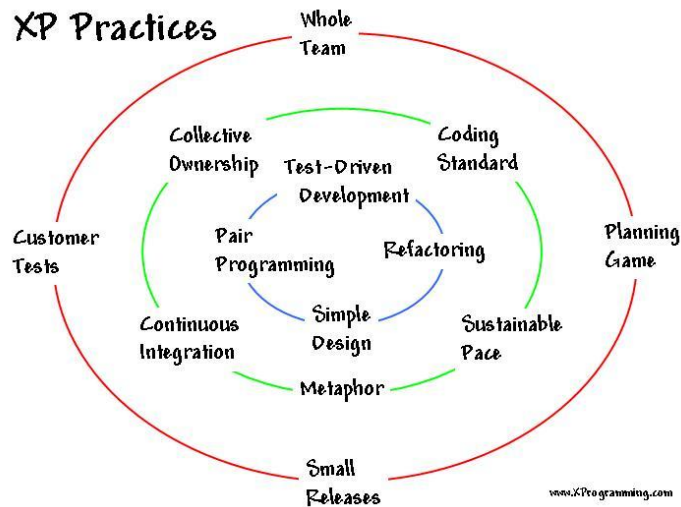


Figure 3-2 eXtreme Programming Practices (source: xprogramming.com)

XP is an Agile software engineering practice that consists of the following 13 core practices shown in Figure 3-1.

1. **Whole Team:** The team is made up of many functional roles that all work together to develop the product. It is not “us” and “them”, but “we”.
2. **Customer Tests:** The customer witnesses an automated acceptance test of the desired feature.
3. **Small Releases:** Development is broken into multiple small releases to decrease risk and to provide working software early and often.
4. **Planning Game:** The team continually monitors what is currently being developed to determine what needs to be developed next.
5. **Collective Ownership:** The code is owned by the team, not by individual team members, so everyone can modify the code.
6. **Coding Standard:** A common standard is used for code development so that it is easier to understand and modify.
7. **Continuous Integration:** Integration is not pushed off until the end, but is part of the daily workflow.
8. **Metaphor:** The team uses a simple way to describe the program vision so that it is easy to understand and communicate.
9. **Sustainable Pace:** The team works 40-hour workweeks, and surges for a short time, if needed.
10. **Simple Design:** The team builds software to a simple design to minimize waste associated with redundancy and rework.
11. **Pair Programming:** Junior developers are paired with more experienced developers for mentoring as well as providing peer review of the code.
12. **Refactoring:** The code is continually refined and simplified.
13. **Test-Driven Development:** XP uses a ‘test first’ approach that requires the test to be written prior to code development, and then the test is run frequently as the code is being written. New code is not written until the test passes.

3.3.2 Scrum

Scrum is an Agile project management practice that is based on iterative sprints, or development cycles. The following information is derived from the Scrum Primer In contrast to XP, Scrum is more focused on the planning aspects of agile development rather than on the technical processes. This includes requirements management as well as iteration and release planning. As a result, Scrum can be considered as a management framework that can be used in addition to agile technical methodologies. The following information on Scrum roles and processes is derived from the Scrum Primer V1.2, by Deemer, Benefield, Larman, and Vodde (2009) and [is](#) depicted in Figure 3-3:

3.3.2.1 Scrum Roles

1. **The Product Owner:** Identifies and prioritizes required features, and is responsible for the overall project success.
2. **The Team:** Builds the product based on direction from the product owner. The team is cross-functional (functional analyst, developer, tester, database designer, etc.) and is typically no more than 10-15 people.
3. **The ScrumMaster:** Helps the team and product owner follow the Scrum process. Removes barriers to the team's progress, resolves issues, and enables the team to be successful.

3.3.2.2 Scrum Process

1. The process begins with the product owner collecting and prioritizing requirements in a product backlog.
2. The team and product owner work together to select the highest priority requirements that can be accomplished within a given time period called a "sprint".
3. The ScrumMaster facilitates the team through daily collaborative meetings, buffers them from external interference, and removes barriers that slow their progress.
4. At the end of the sprint, the team demonstrates the completed features to the product owner. These features are a potential shippable product. The feedback gathered from the demonstration is fed back into the product backlog.
5. The team then has a retrospective to determine what worked well, and what improvements need to be made to the process for the next sprint.

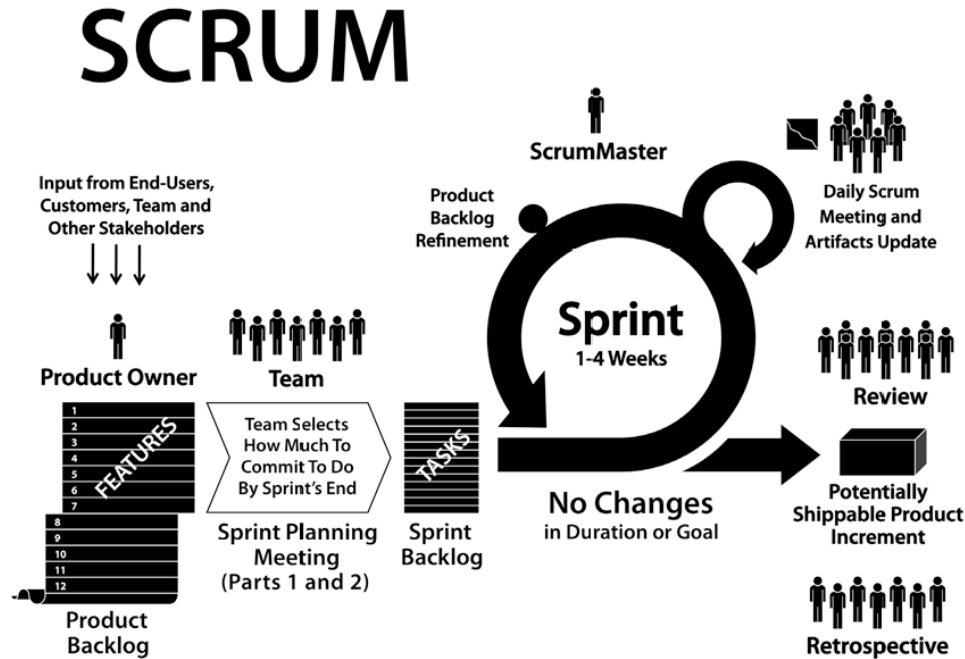


Figure 3-3 Scrum Process (source: Scrum Primer V1.2)

3.3.3 Lean Software Development

Lean software development practices and principles are derived from those used in lean manufacturing. Lean manufacturing has been around for over 50 years, and includes processes such as just-in-time production. Lean software development, a more recent concept, provides a set of overarching principles that can be used to help guide decision making during the development process. These are related to goals such as making quality an integral part of the process, and removing waste by minimizing work in progress and reducing excessive documentation. As a result, it can be used with any combination of agile methods to help optimize the development process. The 10 basic practices or rules of Lean Production from the Poppendeick LLC site at <http://www.poppendeick.com/lean.htm> are:

1. **Eliminate waste.** Reduce things that do not produce value such as excess documentation, functionality that is not used, and waiting time between processes (such as review or approval).
2. **Minimize inventory.** For software development, examples are excess documentation that will not be released with the code, and code that is not finished.
3. **Maximize flow.** Reduce the time it takes to deliver value, such as smaller release cycles.
4. **Pull from demand.** Respond to change instead of trying to predict it. Keep requirements flexible and make decisions at the last possible moment.
5. **Empower Workers.** Push decisions down to the lowest level possible. To do this, the team must understand the project vision and have the data and authority to make decisions.
6. **Meet customer requirements.** Team with the customer throughout development rather than at the beginning, when requirements are defined, and the end when they are verified.
7. **Do it right the first time.** Make quality and feedback intrinsic aspects of the development workflow rather than as separate processes.

8. **Ban local optimization.** Do not sub-optimize requirements by locking them down at the beginning of the project. Things will change. The customer's needs will change.
9. **Partner with suppliers.** Build relationships on trust and on doing what is best for the customer, rather than on highly specified contracts. These contracts can result in adversarial relationships.
10. **Create a culture of continuous improvement.** Encourage learning and feedback to continually improve the processes.

4 Guidelines for Embracing Agile

The following section addresses additional guidelines that should be taken into consideration as a program is deciding, or begins, to use Agile. Through research, case studies and experience, these practices have been observed as illustrating the best practices of Agile development. Each subsection is broken up into Observation, Discussion and resulting Guideline. This format serves to help the reader identify a situation, understand the logic behind the best practice, and to ultimately implement the best practice or attain more information from an additional resource.

These guidelines are not necessarily related nor are they meant to be taken together as a whole. They are independent observations and recommendations that are applicable during different points of the engineering process. To help users decide which guidelines may be applicable to their situation, consider the following categorizations:

- Project inception or planning
 - 4.1 Deciding to 'Go Agile'
 - 4.2 Selecting the Methodology or Methodologies
 - 4.3 New Development, Integration
 - 4.4 Defining "Done"
- Building the team
 - 4.5 Agile Training and Using an Agile Coach
 - 4.6 Distributed Teams
- Program interactions
 - 4.7 Scaling Agile for the Large Projects
 - 4.8 Agile in the Enterprise
- Managing the project
 - 4.9 Agile Risk Management
 - 4.10 Agile Contracts
 - 4.11 CMMI and Agile

4.1 Deciding to 'Go Agile'

Observation: Before deciding to adopt an Agile practice, it is wise to first determine the best approach for the given project as a whole, and/or for sub-projects within it. This includes an assessment of the project volatility (to include requirements and technology) and project criticality, the available resources, the organizational culture, and the availability and commitment of the customer and stakeholders.

Discussion: Boehm and Turner, in "Balancing Agility and Discipline: A Guide for the Perplexed" (2004, p. 56), offer a tool to help determine whether an Agile or plan-driven approach is more appropriate for a given project depicted in Figure 4-1. For example, a safety-critical project with a stable requirements baseline, in a culture that thrives on order may not be best suited for an Agile approach, even though the team is small and the personnel are experienced in Agile methods.

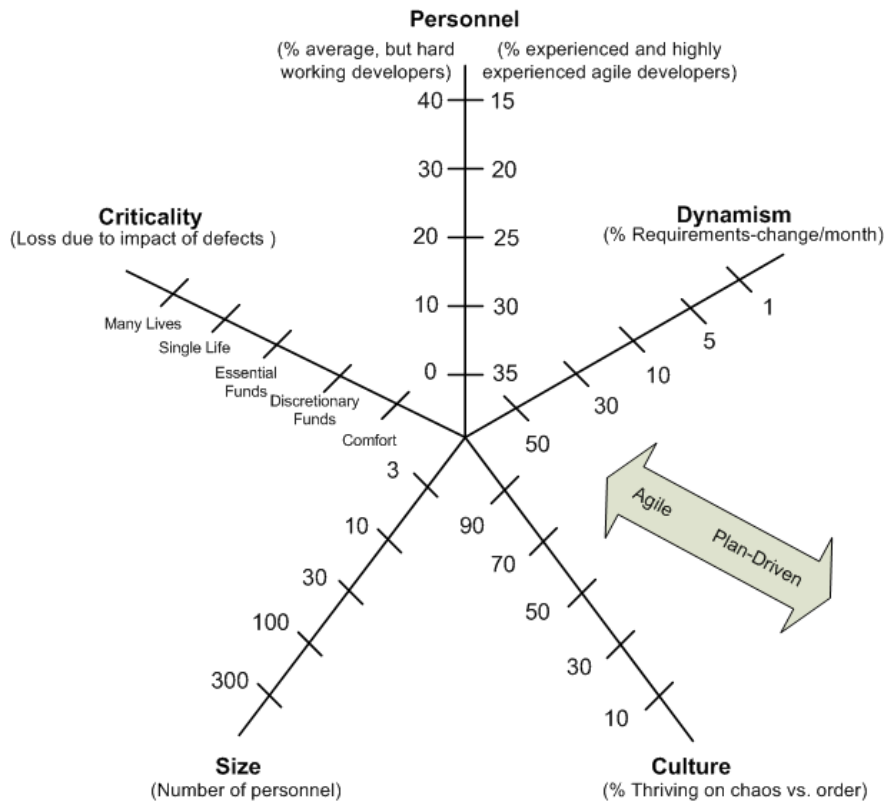


Figure 4-1 Dimensions Affecting Method Selection (source: Boehm & Turner, 2004, p. 56. Note: Personnel description was modified slightly for ease of understanding)

Even if all of the dimensions are appropriate for an Agile approach, there are additional key success factors that need to be taken into consideration. Table 4-1 offers a list of rank-ordered success factors and their attributes (Chow & Cao, 2008, p. 970). Based on the results of the Chow and Cao study, Agile project success is a factor of selecting and adhering to the Agile value-based delivery strategy, selecting a high quality team that is trained on the selected Agile method(s), and following sound Agile-software engineering practices.

Table 4-1 Rank Ordered Agile Success Factors (source: Chow & Cao, 2008, p. 970)

Rank	Factor	Attributes
1	Delivery strategy	Regular delivery of software Delivering most important features first
2	Agile software engineering techniques	Well-defined coding standards up front Pursuing simple design Rigorous refactoring activities Right amount of documentation Correct integration testing
3	Team capability	Team members with high competence and expertise Team members with great motivation Managers knowledgeable in Agile Managers who have adaptive management style Appropriate technical training to team

4	Project management process	Following Agile-oriented requirement management process Following Agile-oriented project management process Following Agile-oriented configuration management process Good progress tracking mechanism Strong communication focus with daily face-to-face meetings Honoring regular working schedule
5	Team environment	Collocation of the whole team Coherent, self-organizing teamwork Projects with small team Projects with no multiple independent teams
6	Customer involvement	Good customer relationship Strong customer commitment and presence Customer having full authority

Guideline: Following these recommendations on what to consider when deciding to adopt Agile practices, can improve the likelihood of a successful project.

4.2 Selecting the Methodology or Methodologies

Observation: Agile methods are not “all or nothing” approaches. It is appropriate to pick and choose practices from various methodologies as needed.

Discussion: Choosing an appropriate methodology or combination of methodologies may seem like a daunting task due to the numerous options and the many possible combinations of methodologies. A quick scan of Agile websites and blogs may reveal that some practitioners are either “traditionalists” or “agilists” and believe that the two approaches are mutually exclusive (and are often adamant about it), while others are more open minded and recommend process selection, to include multiple processes, be based on the needs of the project. For example, referring back to the five factors in Figure 4-1, some aspects of a project may require a traditional plan-driven approach while other sub-elements may be able to use an Agile method. In this case, organize Agile and traditional sub-units under a higher-level program governance framework (Vinekar, Slinkman, & Nerur, 2006). This will help the Program Management Office keep the Agile and traditional development projects separate to minimize confusion (because they require a different team structure, culture, and set of processes) and to provide a mechanism to synchronize the activities of both. This cross-project coordination may also include encouraging members of the traditional and Agile development teams to attend each other’s project meetings, such as the Agile teams’ daily Scrums. This communication and coordination mechanism can increase awareness on individual team development progress and potential integration dependencies across teams.

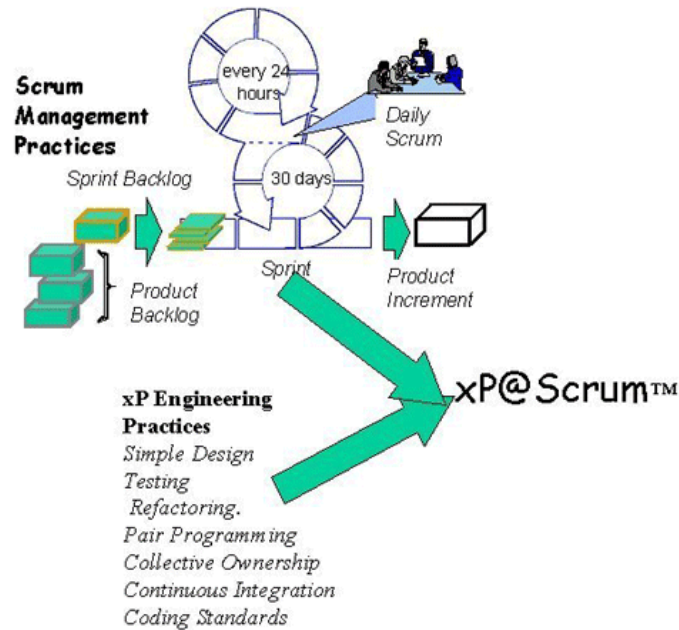


Figure 4-2: Scrum and XP (source: <http://www.controlchaos.com/about/xp.php>)

Whether using traditional and Agile methods together, multiple Agile methods can also be combined. For example, as depicted in Figure 4-2 (Advanced Development Methods, Inc., 2010), projects can adopt Scrum along with XP to gain the benefits of an Agile project management process combined with an Agile software engineering process.

Guideline: It is common practice to use a combination of methodologies, whether traditional or Agile, in a program. The main thing to remember when selecting a process, or set of processes is that they align with the unique needs of the project.

4.3 New Development and Integration

Observation: Agile should not be viewed as a solution solely for new development projects. It should be considered for all projects that involve software intensive systems, including but not limited to: IT systems, embedded systems, and equipment under control.

Discussion: The Government has different terms to categorize different types of engineering efforts. Usually, the terms "software/system development" and "software/system integration" are used to categorize and define the scope of a project.

The term "software/systems development" is used as an umbrella term and is defined as the research, new development, modification, reuse, maintenance, integration or any other activities that result in a software product or information system. Specifically, the word "development" could be perceived as having too broad a scope to be used to describe the work performed on most Government programs.

Similarly, "software/systems integration" is defined as the process of linking together different computing systems and software applications physically or functionally. The process of systems integration involves the integration of existing (often disparate) subsystems through their interfaces. The method by which these interfaces "speak" to one another varies. Translation code or mappings are often needed to act as "glue" between these systems. The systems

integrator is required to have broad knowledge varying from such fields as computer networking, business process management, computer programming, etc. Often, integration tasks require the integrator to perform research, development of new software (i.e. translation code), modification of existing software/systems, reuse of existing software/systems, and possibly even integration of even smaller sub-systems. It is for these reasons that software/systems integration can be used almost synonymously with software/systems development, and furthermore, why software/systems integration should adhere to practices of software/systems development methodologies.

Many projects aim to decrease the amount of "custom" code developed, and leverage commercial off the shelf (COTS) components as much as possible in order to decrease project cost and risk. These types of projects tend to classify themselves as "Integration" projects, and focus more on developing "glue" code that integrates disparate systems together. Similarly, some projects are responsible for updating or maintaining existing legacy software, and do not classify themselves as pure software development.

Guideline: Agile is about improving the quality of a product through early and frequent deliveries of small pieces of the product, and continual stakeholder involvement. Regardless of the type of effort (e.g. new development, integration or maintenance), Agile principles can be applied to add engineering discipline and improve product quality and user satisfaction.

4.4 Defining “Done”

Observation: In traditional projects, “done” may mean that the code or feature is completely finished (coded, tested, documented, etc.), but in Agile, this term may have different meanings based on the context of the given situation.

Discussion: How the team decides what is “done” for an iteration may be very different from what is considered “done” for a release. Iteration features may need to be minimally documented (maybe only in the code comments), bug free, and accepted by the product owner or customer. Release feature code may need to be fully integrated, bug free, documented in accordance with program of record requirements, and tested in a manner that is ready to be fielded. The accepted “done” criteria need to be defined explicitly during iteration planning (M. Loomis, personal communication, February 16, 2010), and agreed to by the entire team. Otherwise, features that are not adequately completed in an iteration will incur technical debt, resulting in additional backlog and rework during future iterations. Technical debt can result from decisions such as waiting until the next iteration to refactor code or to fix a bug. Debt is incurred because the code or function was not "done" in the earlier iteration, and will require cleanup or fixes in the in the next iteration. This will take away time that the team could have devoted to new functionality. Therefore, it is recommended that the definition and criteria for "done" be agreed upon prior to starting the first Sprint.

Guideline: Explicitly define and agree to the accepted criteria for "done" during iteration planning.

4.5 Agile Training and Using an Agile Coach

Observation: One of the keys to a successful Agile development project is that the entire team have Agile experience. This was reiterated in VersionOne’s 2009 State of Agile Survey, which specifically stated that the top reason for failure on an Agile project was lack of Agile development experience among team members. This applies not only to the development team, but to the program management team as well.

Discussion: A United States General Accounting Office Report (2002) states: “Having the right people with the right skills to successfully manage acquisitions is critical for DOD. The Department spends about \$100 billion annually to research, develop, and acquire weapon systems and tens of billions of dollars more for services and information technology. Moreover, this investment is expected to grow substantially. At the same time, DOD, like other agencies, is facing growing public demands for better and more economical delivery of products and services. In addition, the ongoing technological revolution requires a workforce with new knowledge, skills, and abilities.” While this report does not specifically call out Agile methods, it does cite the need to have highly skilled individuals in the workforce.

Agile training needs should be identified and included in the overall cost and schedule estimates during the project planning phase. Even with training, additional hands-on assistance from an Agile subject matter expert may be needed during the early stages of implementation. One way to compress the up-front learning curve is to hire an Agile coach to guide the team through the initial stages. The value of a coach has been documented in both industry and government adoption of Agile methods (Cohan, 2007; Schatz & Abdelshafi, 2005), and can help ease the challenges associated with adopting a new process, thus increasing the overall probability of success.

Guideline: Identify and include Agile training in the project plan. For inexperienced Agile teams, hire an Agile coach to enhance the training and facilitate the teams' adoption of the process.

4.6 Distributed Teams

Observation: Working a project where team members are not physically co-located can be a challenging endeavor. While distributed projects may not be optimal, they are often times the only option available, especially in the DOD where resources and personnel are scattered all over the world.

Discussion: Many Agile methodologies including Extreme Programming urge their practitioners not to work with distributed teams; citing problems with verbal communication, digital communication and even team bonding. To mitigate these drawbacks we mainly look to technology to cross the divide. A good Agile project management tool can provide the most value in this respect. It provides a single point of entry into the situational awareness of a project. In addition to an Agile project management tool, a project wiki is a great way to catalogue the details and documentation of a project. A typical project wiki would include information for all of the resources of a project such as the people working on the project and their contact information, the project's code management system, any repositories or databases that may be in use, and the details of the project's computing systems.

Collaboration tools are another way to improve the situation. Sharing a desktop screen or using a virtual white board is a good way to augment a distributed meeting; this becomes particularly useful in a distributed design session. A persistent chat room where team members can "hang-out" is another good way to keep presence between the team and to facilitate communication in an unintentional sort of way, i.e. pose a question to the group and whoever is around and knows the answer can field it. Instant messaging can also be leveraged to provide quick communication to an individual team member.

Video chat is another tool that provides very good communication and a feeling of presence, but is often seldom used outside of video-teleconferencing (VTC) in meetings. Setting up a persistent video chat room with tools such as Apple iChat or Google Voice and Video Chat

between two labs, for instance, can provide an audio-visual presence between team members. This isn't necessarily something that should be done all the time, but during an iteration of development where team members are working together on a specific goal, it can be very beneficial.

Guideline: Communication, collaboration and the proper tools are key for distributed teams to be effective and successful with an Agile approach. More information on Agile project tools and their benefits can be found in Section 7.2.

4.7 Scaling Agile for Large Projects

Observation: For large projects that consist of multiple Agile sub-projects, an inter-team communication strategy is essential.

Discussion: A recommended inter-team communication strategy is to have an additional meeting called a “Scrum of Scrums” after the daily Scrum meeting. Each Scrum team should send at least one member to the larger meeting, such as the ScrumMaster, and/or member from the development team. Consideration should be given as to who can best represent the team. This may change over time depending on factors such as results or issues from the daily Scrum meeting. During the Scrum of Scrums meeting, each Scrum team should report the following four items (Cohen, 2007):

1. What has your team done since we last met?
2. What will your team do before we meet again?
3. Is anything slowing your team down or getting in their way?
4. Are you about to put something in another team's way?

In addition to the daily Scrum of Scrums, Peter Vaihansky, Jeff Sutherland, and Anton Victorov (2008) suggest a list of recommended practices for large, distributed projects. These can be useful even if the team is not geographically distributed:

- Daily meetings of the Product Owner team.
- Hourly automated builds from the central repository.
- Seamless integration of XP practices like pair programming and code refactoring.

Guideline: The main point to remember when selecting a process, set of processes, or communication strategy is that they must align with the unique needs of the project.

4.8 Agile In the Enterprise

Observation: IT projects are no longer organized or executed in a stove-piped fashion. The success of projects requires interactions with other program offices and projects that may follow a more traditional development model.

Discussion: Interactions between projects using different development methodologies presents unique challenges with the need for identification and management of external dependencies (programmatic and technical), submitting and receiving requirements from external projects, reporting progress against those requirements, and deploying major releases to a production environment.

Developing and delivering solutions for a large business organization, commonly referred to as an Enterprise, usually requires programmatic and technical resources that are external to a

project. It is recommended that project plans (or other guiding project documents) outline that these resourcing needs must be identified and addressed as soon as possible. Often, the required resources may not be known at the onset of a project, so it is good practice to provide a mechanism for new external resources to be identified and addressed within a project plan (or other guiding project documents).

When a project requires an external resource, such as an external Program Management Office (PMO) service or technical integration point, a best practice is to establish a documented agreement with the external PMO that outlines the following:

- Inter-program requirements management process that governs how PMOs will develop, exchange and manage requirements
- Roles and responsibilities for each organization
- Mechanisms for measuring and reporting performance between PMOs
- Deployment considerations

Often it is necessary to coordinate with another program or project that doesn't follow the same style of requirements development and management. It is helpful to hold requirements development sessions with the external PMO to build proper "Agile" requirements, and store these in a collaboration tool that is accessible to both PMOs.

Likewise, estimation of effort doesn't translate well across PMOs or development teams, as different estimation methods may be employed and different teams have different production capabilities. It is recommended that a common mechanism for exchanging estimates is developed between development teams (e.g. days, hours). It is also helpful to include external PMOs in planning and review meetings, so they can monitor progress and improve the fidelity of their plans.

Deploying a capability to the Enterprise also requires some coordinating activities that are best placed in a project plan (or other guiding project documents) and release cycle. Ideally, releases are planned that adhere to Enterprise support and training requirements. Often, an Enterprise has mandatory deployment activities (e.g. training, interoperability testing etc.) that must be completed prior to deployment. It is good practice to account for these activities in the release plan so the team can properly prepare for these necessary activities.

Guideline: Project plans (or other guiding project documents) should outline programmatic and technical resources that are external to your project and establish a documented agreement with these external resources. On-going communication and coordination with other program management offices and projects is a critical component

4.9 Agile Risk Management

Observation: Risk management is an organic component of Agile development. Risks are implicitly identified and managed through frequent communication, frequent plan refinement, requirements prioritization, and the increased transparency, these activities bring.

Discussion: Despite the fact that risks are implicitly tracked and managed in an Agile process, there should be a mechanism to codify this approach and have a semi-formal to formal risk management approach. Common sense should be applied when developing a risk mitigation approach. The formality of the risk management process depends on the size and criticality of the project. An Agile development process needs to make risk management organic to the process, and ensure that all team members actively participate in the risk management process.

Whatever risk management approach is used, it should meet the following objectives as identified by Sliger & Broderick (2008, p 189):

- Risk Identification
 - Develop a risk register that is available to the entire team
 - Risk register is updated regularly and anyone can submit a risk to be considered by the team for inclusion
 - Revisit the risk register at every planning meeting
 - Create risk exposure charts to measure risk as a whole and provide quick and easy mechanism to see the project's risk exposure at a glance
- Risk Analysis
 - Quantitatively and qualitatively assess risk and make results visible in the risk register
 - Prioritize risks and develop response plan for the highest priority risks
- Risk Response Planning
 - When the team decides to respond to a risk, the following options are available and should be documented in the risk register:
 - Avoid: Don't do the project or part of the project that entails the risk
 - Mitigate: Take steps to reduce or eliminate the risk before the risk materializes (e.g. move high risk items to earlier Sprints)
 - Contain: Set aside resources (time, money, materials) to address the risk if it becomes an issue
 - Evade: When none of the above is performed and the risk doesn't become an issue (usually the least optimal response)

Guideline: Develop a semi-formal to formal risk mitigation approach that aligns with project size, criticality, and availability of resource management resources and ensure that all team members active participants.

4.10 Agile Contracts

Observation: Typical DOD IT acquisition projects assume that the product end state is well known and that requirements are stable, however, this is usually not the case. Different contracting approaches are needed that can accommodate high levels of uncertainty for IT systems.

Discussion: For Agile projects, since the end-state is not well defined up front, a different approach might include alignment of funding to software delivery releases (or iterations, if desired), rather than to delivery of specific requirements or functionality (Koch, 2005). Smaller periods of performance, correlated with short capability delivery cycles, can increase the probability of success.

According to the Standish Group (2009, p. 30), software projects with labor costs less than \$750K have a 71% probability of success, those costing between \$750K to \$3M only have a 19% chance of success, and for projects over \$10M, success rapidly falls to an abysmal 2%. It is possible that the reason smaller contracts have a higher success rate is that they are easier to manage, encourage frequent feedback and rapid delivery of value, and are easier to terminate early, if necessary, than large contracts. Additionally, performance incentives based on user feedback at capability demonstrations can be used to reward desired behavior (Stevens, King, & Halley, 2009). In contrast, rather than terminating the contract, many large, traditional contracts reward vendors who are over budget or behind schedule with additional time and money, thus encouraging bad performance.

Koch (2005) provides additional recommendations on the contract “shalls” that protect both the contractor and the government, as well as encourage a collaborative relationship:

- Identify the Agile development process or processes that will be used on the project (such as Scrum, XP, or a hybrid of several methods)
- Specify the contractor/developer role
- Specify the government’s role (such as product owner) and the level of involvement they will have in the project
- Identify how requirements will be managed
- Clarify the contractor’s level of interaction with the customer
- Identify release time-boxes (such as, sprints, iterations, releases) and associated funding strategy
- Specify that the contract can be terminated early and the decision criteria for this action

Duquette, Bloom, Crawford, and Osgood (2007) have identified contracting guidance in the Federal Acquisition Regulation (FAR) supporting Agile contracting. The following is a summary of these findings:

- Subpart 16.5 - specifies the use of *indefinite-delivery* and *indefinite quantity* contracts, which could prove particularly useful in letting contracts without fixed deliverables and requirements.
- Subpart 6.3 - specifies the policies and procedures for contracting without providing full and open competition. While full and open competition is generally regarded as a good thing, frequent contracts of a smaller size may be held up in the competition process.
- Subpart 16.207 - specifies the use for *firm-fixed-price, level-of-effort term* contracts suitable for Agile projects where the work can be stated in general terms and the Government pays the contractor in a fixed dollar amount.
- Subpart 16.603 - specifies the use of *letter contracts* that are "used as a preliminary contractual instrument that authorizes the contractor to begin immediately performing services", "in cases where negotiating a definitive contract is not possible in sufficient time to meet the requirement."
- Subpart 43.2 - specifies the use of *change orders*, which allow for the changes in the general scope of a contract after it has been let. This of course does not require the contractor to incur any costs beyond the limits established in the initial contract. This subpart could prove useful in projects where emerging requirements can cause a fundamental change in the scope of the project not initially accounted for in the original contract.

Guideline: Smaller contract award sizes have shown to improve project success rates. Using specific verbiage that encourages a collaborative relationship in a contract is a good idea. Existing sections of the FAR can support Agile contracting without modification.

4.11 CMMI and Agile

Observation: Capability Maturity Model Integration (CMMI) compliance has become a standard compliance yardstick for bidders on government projects. Unfortunately, it is a common misconception that the CMMI and Agile methodologies are at odds with each other.

Discussion: The reasons for this discord are best summarized in the Carnegie Mellon and Software Engineering Institute’s (SEI) paper "CMMI or Agile: Why Not Embrace Both!" (Glazer, Dalton, Anderson, Konrad, & Shrum, 2008, p. 1):

1. “Early adopters of both CMMI and Agile methods represent extreme examples of their software development paradigms. Early CMMI adopters were developers of large-scale, risk averse, mission-critical systems, often with high levels of management oversight and hierarchical governance; whereas the early adopters of Agile methods generally focused on smaller, single-team development projects with volatile requirements in a software-only environment. These two extremes set the tone for all that followed.
2. The inaccurate information about CMMI and Agile and the misuse of both resulted in misperceptions in both camps about the other. These negative perceptions that position CMMI and Agile at odds with each other arose largely from the following factors:
 - Misuse—two decades of experience, first with the Capability Maturity Model (CMM®) and then with CMMI models, in which practices were sometimes misused or applied to (i.e., overlaid on) development activities that may have already been perceived by software development teams as productive without them
 - Lack of Accurate Information—a dearth of accurate information about CMMI in the Agile community and the corresponding dearth of accurate information about Agile methods in the CMMI community
 - Terminology Difficulties—the use of terminology in CMMI (e.g., discipline, quality assurance, and predictability) and Agile methods (e.g., continuous integration, test-driven development, and collective code ownership) that carries context-specific connotations and is thus easily misunderstood and abused
3. Top-Down Versus Bottom-Up Improvement Approach—the introduction of an approach that sometimes favors one “voice” (i.e., management versus practitioner) over the other, which neglects the other important voice in how to effectively run the business”

However, CMMI and Agile are compatible. CMMI focuses on process objectives, while Agile methods focus on how projects develop products. These two objectives are orthogonal, and actually complement each other very well (Sutherland, Jakobsen & Johnson, 2007). For instance, CMMI does not impose a particular development methodology, nor do Agile methods impose a particular model for development in which activities and process outputs are to be measured. A case study performed by Sutherland, Jakobsen and Johnson (2007) on Systemic, a CMMI level 5 organization that uses Lean Software Development, showed that productivity doubled, reducing costs by 50%, rework by 42%, and defects by 40%.

A comparison of CMMI and Agile paradigms in the Carnegie Mellon and Software Engineering Institute (SEI) paper (Glazer, Dalton, Anderson, Konrad, & Shrum, 2008, p. 35) is listed in 11Appendix B. A point of interest to note in this comparison is the dimension of “trust” in which Glazer et al argue that CMMI is well suited for low-trust environments and Agile for high-trust environments. CMM (and subsequently CMMI) were created in response to the DOD software dilemma of the 1980’s. A culture of distrust persisted due to a wealth of software projects that were over budget and under producing. As a result, protecting one’s own interests was the standard modus operandi verses finding the most efficient win-win solution (Glazer, Dalton, Anderson, Konrad, & Shrum, 2008). CMM was created with the aim to combat this mentality by ensuring that all interested parties could do their part in an open and transparent operating environment.

Agile methods take a different approach to the problem of trust. At a cursory glance, it would appear that trust is not important. Contracts are let with open ended deliverables and requirements. Teams are encouraged to self organize (i.e. specific role responsibility is not assigned). Inspection from stakeholders and management is allowed only during pre-scheduled

demonstrations. From an outsiders perspective it would appear as if Agile development could only work in a high-trust, low-risk environment. However, this is not the case, as will be explained.

Contracts are let without overly specific deliverables and requirements for numerous reasons. The first of which is described in [Section 2.1.2](#), which states that uncertainty is at the highest point during the beginning of a project and it is therefore unreasonable to assume that detailed specifications and release plans can be adequately documented at this stage of the project. The risk of not specifying requirements and deliverables is mitigated with early termination clauses, frequent demonstrations of functionality, and code iterations that are potentially shippable. If at any point in the project the contractor is not producing quality code or a user/stakeholder approved capability in a timely manner the contract holder reserves the right to terminate the relationship. Since code is written in such a way that it is potentially shippable, a new contractor can be brought on to finish the work.

Agile processes employ self-organizing, cross-functional teams to achieve the goals of the project. Responsibility is placed upon the team, rather than the individual team member or the project manager. This allows individuals to take on tasks as they become ready to be executed (Moe, Dingsoyr, Dyba, 2008). In this way, the team manages their own workload internally, rather than having it dispersed by a manager. In addition, the team interacts directly with the customer and users, rather than through the project manager. These processes effectively transfers customer and user trust directly to the team, the group responsible for doing the work.

Lastly, a team that is constantly under supervision and expected to drop what they are doing to perform demonstrations or give briefs will not be able to perform at their optimum level. Writing quality code is a time-intensive and focus-consuming task. It is for this reason that scheduled demonstrations at the end of iterative development periods are not only recommended, but are intended to be the only demonstrations given by the team. During this demonstration information flows bi-directionally; the current status of the project is demonstrated to stakeholders, users and management, and feedback is collected by the team for future iterations.

Guideline: Agile and CMMI are not at odds with one another and Agile can be employed in an organization or a project that has a CMMI requirement.

5 Agile/Rapid Acquisition

Duquette, et al (2007) define *Agile Acquisition* as "the actions taken by a capable, experienced government management team to adjust program parameters throughout the life-cycle to respond to changes in the program's environment and its customers' expectations." They further define *Rapid Acquisition* as "the actions taken by a government management team to acquire a capability in the shortest time possible within or outside of the government acquisition process (definition, funding, procurement, development, testing, and fielding)." Based on these definitions, we see that Agile Acquisition has an emphasis on reacting to change in the program's environment and customers' expectations, while Rapid Acquisition focuses on the fielding of capabilities in the shortest amount of time possible. One can clearly see the relationship between these two definitions and the tenets of Agile Software Development. Mapping the focus of these concepts to the 12 Agile principles described in Section 3.1.2, we see that Agile Acquisition corresponds to principles 1, 2, 4, 8 and 12, and Rapid Acquisition corresponds to principles 1, 3, 7 and 8.

Most policy and authority in the DOD relating to Agile and Rapid Acquisition deals with Rapid Acquisition. Rapid Acquisition Authorities exist at the Joint, Service, and Combatant Command levels. These authorities exist to satisfy immediate Warfighter needs and are intended to field capability as soon as possible, in some cases a matter of days, but can span up to two years. They operate outside of the normal acquisition process to achieve these results. Typically, these authorities are meant to assist with the development of new technologies to be transitioned to a POR. The following is a list of programs that can be used to transition rapid capability to a POR or in some cases even an operational unit (Duquette et al., 2008):

- Advanced Technology Demonstrations
- Air Force Applied Technology Councils
- Joint Concept Technology Demonstration Program
- Defense Production Act Title III Program
- Joint Experimentation Program
- Manufacturing Technology Program
- Small Business Innovation Research Program
- Defense Acquisition Challenge Program
- Small Business Technology Transfer Program
- Technology Transition Initiative
- Value Engineering
- Warfighter Rapid Acquisition Programs
- Navy Rapid Technology Transition Program
- Army Aviation Technology Assessment and Transition Management (TATM) Process
- Foreign Comparative Testing
- National Technology Alliance
- Independent Research and Development
- Department of the Navy Future Naval Capabilities
- USSOCOM Advanced Technology Directorate

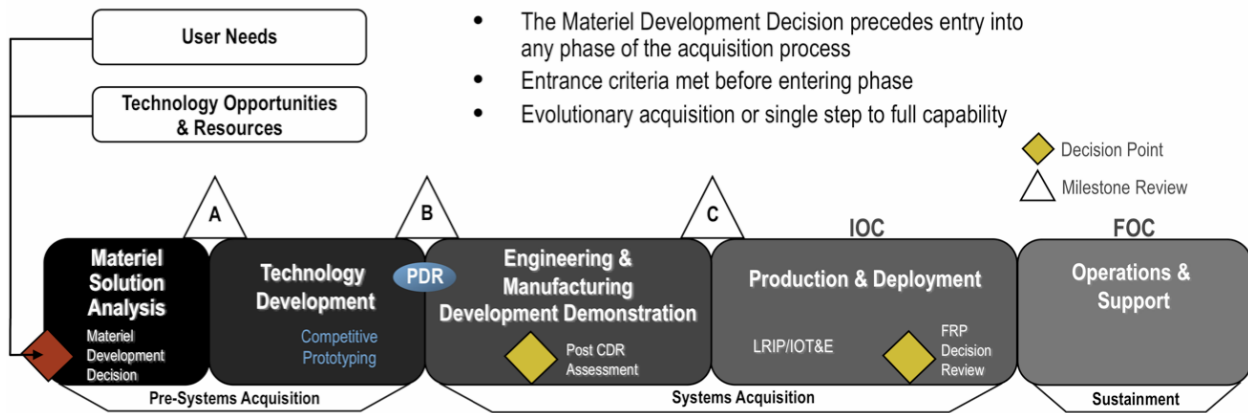


Figure 5-1 New Acquisition Process

There are Agile Acquisition efforts in progress, as stated in [Section 2.2](#). The modifications made in December 2008 to the DOD 5000 Directives and Instructions sought to reduce uncertainty early in the development process, through earlier development and increased prototyping. Unfortunately, many of these changes have also included increased documentation and review processes that could slow the process down for IT systems. Figure 5-1 depicts the changes to the process.

Policy changes embedded in the new process include the following:

- "All programs will proceed through a formal acquisition process entry point, the Materiel Development Decision (MDD). Programs will no longer immediately proceed to Milestone B. Consequently, the vast majority of programs will benefit from the improved conception and technical maturity resulting from the early phases of development.
- Programs requiring technology development will conduct competitive prototyping at the system or sub-system level, when appropriate, to ensure that technologies have been demonstrated in a relevant environment and, consequently, key risks have been retired before programs are initiated.
- Where consistent with the strategy for the Technology Development Phase, preliminary designs will be prepared to ensure that requirements are well understood and cost estimates well informed.
- The Engineering and Manufacturing Phase has been redesigned to place additional emphasis on systems engineering and manufacturing readiness.
- Configuration Steering Boards have been established to ensure that requirements changes/creep, a traditional contributor to increased cost and extended schedules, are not casually approved." (Defense Science Board 2009)

While these changes are a step in the right direction, they do not go far enough to address the unique characteristics of information technology programs and the rapid timeframes in which they evolve. The Defense Science Board's new acquisition process for information technology systems, outlined in their March 2009 report, recognizes the need for a compressed development cycle for IT systems and recommends a completely new process, drawing from Agile principles. In fact, the National Defense Authorization Act for fiscal year 2010, becoming Public Law 111-84, in section 804 directs changes to the DOD acquisition process for IT systems, using the recommendations provided from the Defense Science Board. Specifically, the act states the changes made to the acquisition system will be designed to include:

- Early and continual involvement of the user
- Multiple, rapidly executed increments or releases of capability
- Early, successive prototyping to support an evolutionary approach
- A modular, open-systems approach

However, until this new acquisition system is created, the existing acquisition process must be followed. In light of this, Duquette, et al (2008) focus on how to successfully transition Agile/Rapid Acquisition based capability initiatives to the Warfighter in the current DOD acquisition atmosphere. They postulate that an Agile Acquisition approach can work for the creation or transfer of capabilities to a POR, but that it must comply with the funding, design, documentation, testing, sustainment and advocacy required of the acquisition process. This is depicted in Figure 5-2 in greater detail. For convenience, the complete description of each barrier/enabler is listed in [Appendix G](#).

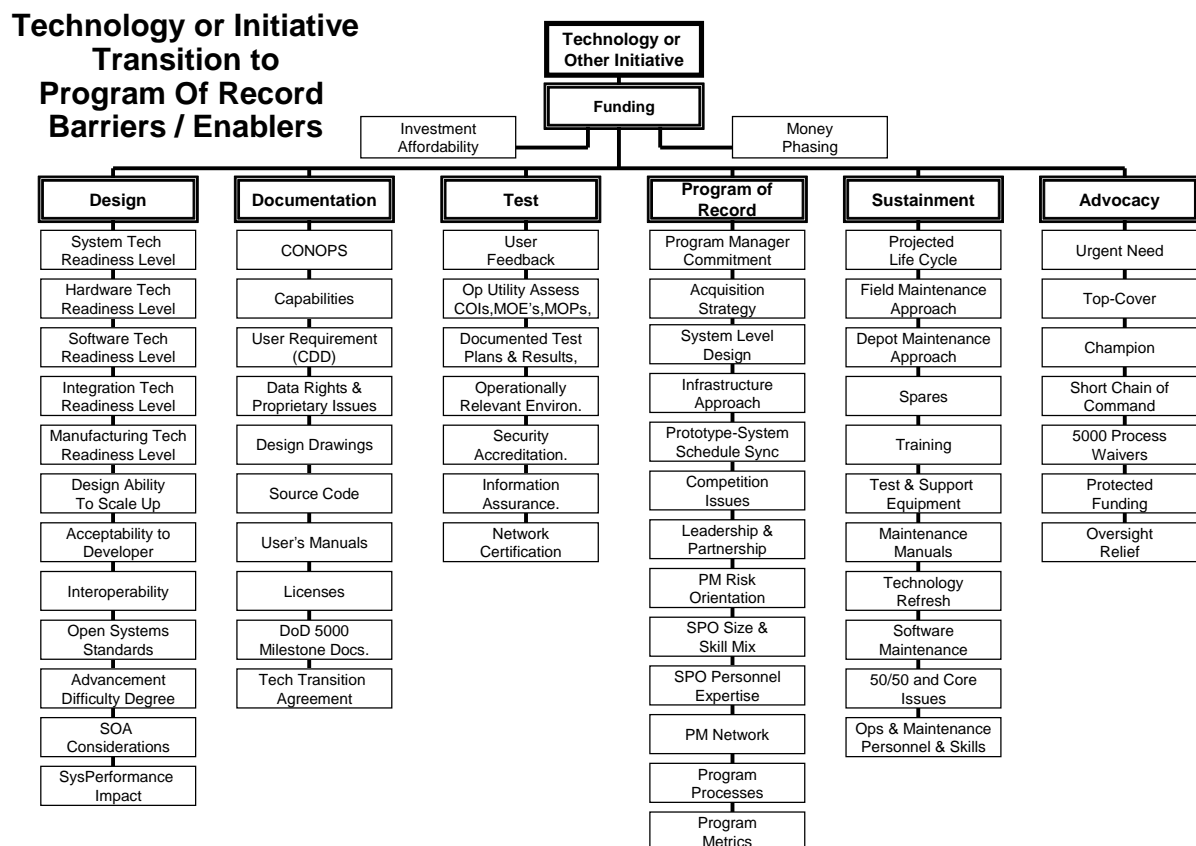


Figure 5-2 Program of Record Technology or Initiative Transition Barriers/Enablers

Furthermore, the Agile development roadmap of the program must be aligned with the acquisition roadmap. Rather than waiting to deliver capability to the Warfighter at Initial Operational Capability (IOC), which can take 5 - 10 years, capability can be delivered at milestone decision reviews. This capability should be created for two reasons, either because it justifies the existence of the POR to solve capability gaps identified previously, or it responds to emergent user needs to counter a threat. Either way, it can be used as an Agile approach to developing and delivering capability to the user in a rapid fashion.

6 Development Best Practices

The following section describes best practices for developing code. While best practices for developing code are not specific to Agile development, the concepts in this section are considered to be a core component of Agile development. It is assumed that the reader of this section is familiar with common programming concepts and paradigms.

6.1.1 Expressive Code

Code that is clear and easy to understand is considered to be 'expressive'. Expressive code should convey its intent to the reader without excessive inline documentation. Simple practices, such as using meaningful class and variable names, providing consistent formatting, avoiding complex Boolean logic, and using whitespace appropriately leads to more expressive code. A more complex practice of expressive code can be found in the use of design patterns.

Design patterns are general reusable solutions to commonly occurring software design problems (Design Patterns, 2010). Examples of commonly used design patterns are builder classes, factory methods, lazy initialization and iterator patterns. More information on design patterns can be found in the Wikipedia article *Design Patterns* (2010).

Design patterns make code expressive because they are easily recognizable thus aiding in comprehension. Often, programming languages embed design patterns directly into the language itself. For example, Java makes use of the iterator pattern in container classes that implement the `Iterable` interface in version 1.5 of the Java Development Kit (JDK). This includes almost all of their container classes including lists, vectors, stacks, hashes, etc. Figures Figure 6-1 and Figure 6-2 demonstrate the printing of objects in a list to standard output, both with and without the iterator design pattern. Notice the succinctness of the code in Figure 6-2 and how the intent is easily understood.

```
List list = oldList;
for (int i = 0; i < list.size(); i++) {
    System.out.println(list.get(i).toString());
}
```

Figure 6-1 Java For Loop Without Iteration Design Pattern

```
List list = oldList;
for (Object o : list) {
    System.out.println(o.toString());
}
```

Figure 6-2 Java For Loop With Iteration Design Pattern

6.1.2 Regression Testing

Software is in a constant state of evolution when it is being developed. Whenever new code is created, or existing code is modified, there is the potential to break existing functionality. Regression testing is an attempt to discover when new code has broken previous code, and allows the developer to correct the problem at the moment it is discovered. Regression testing is

achieved through extensive and repeatable automated testing, usually involving unit and functional testing.

Unit tests verify the functionality of the smallest subset of the system, such as an object or method. The goal of unit testing is to isolate each part of the program and show that the individual parts are correct (Kolawa & Huizinga, 2007). Unit tests should be automated and run at regular intervals, such as at build time and especially before checking code into version control. It is considered good practice to create a unit test to expose a bug that has been located and fixed, so that it can be retested after subsequent changes to the program, as bugs sometimes have a way of resurfacing. Unit testing packages exist for most programming languages.

Functional testing can be thought of as black box testing, i.e. the internal mechanics of the system are inconsequential, only the inputs and outputs of the system are of concern. In Agile software development, functional testing takes place at the user story level and is used to determine whether or not a user story has been completed. This is also known as 'acceptance testing'.

By utilizing regression testing, the development team can ensure that code is operating the way it was intended to operate. Further, it is an integral part of what is known as *Continuous Integration*, another best practice of Agile software development.

6.1.3 Continuous Integration

Continuous integration, like regression testing, uses automation to ensure consistency in the project, but focuses on the software build rather than software testing. Continuous integration is defined as the practice of frequently integrating one's new or changed code with the existing code repository; it should occur frequently enough so that no intervening window remains between commit and build, and such that no errors can arise without developers noticing them and correcting them immediately (Fowler, 2006).

For instance, take a project where developers working on an independent problem branch their code, work on it for a couple weeks, and then check in their code. The delta between a branch and the baseline could vary wildly as commits are performed. In some cases, modifications to check the branch back into the baseline may take longer than the actual branch modifications performed in the first place. Continuous integration aims to solve this problem by committing early and committing often. Furthermore, automation of the build, complete with regression testing, ensures that new modifications to code don't break the build or previous functionality.

6.1.4 Pair Programming

Pair programming is the physical act of two programmers developing together at one station. Typically, one person types while the other observes, switching periodically. The thought behind this concept is that two programmers working together on the same problem will produce overall better code with less defects.

Pair programming is a controversial topic in software engineering. Proponents of pair programming cite benefits of greater design quality, reduced cost, better training, and greater focus and time management (Cockburn & Williams, 2000). Opponents of pair programming cite drawbacks of slower development time, conflicts of personality, and personal freedom issues (Pair Programming, 2010). This controversy is understandable considering the extreme deviance from the typical single programmer approach. However, the benefits that pair programming provides can in many cases outweigh the drawbacks. At the very least, pair programming can be used on an as-needed basis.

Take, for example, the scenario of a new employee joining the team of a well established project. It is likely that the project has in place a set of practices that are used for configuration management, that knowledge has been accumulated and shared among team members which outsiders do not possess, and that relationships and roles have been formed in the project that are critical to the project's success. A new team member must learn this/these information/practices/roles in order to be a functioning contributor to the project. By pairing a novice or new team member with a senior team member, information can be exchanged at a much faster rate.

Pair programming facilitates this kind of knowledge transfer and it can be a useful tool that every team should have in their Agile software development toolbox.

7 Tools

Tools are an important part of software engineering. Agile development downplays the importance of tools in favor of individuals and their interactions, which is a valid point. Without the individuals and the interactions between those individuals, the tools they use are meaningless. However, in a distributed environment like the DOD, where it's almost always the case that users and stakeholders are not co-located with the development team, or even the development team itself isn't co-located, tools become more important, especially tools that foster collaboration.

In this section, we take a look at Agile tools which aid in development of software, and we then discuss the results of a trade study completed on Agile project management tools to determine which tools are well suited for use in the DOD environment.

7.1 Development Tools

As discussed in the [Development Practices Section](#), Agile advocates the use of automation in every area in which automation makes sense: the build, testing, and in some cases, even the creation of the code itself. The following is a list of the types of tools that should be used in an Agile development environment. These are not specific recommendations, as performing an exhaustive study on all tools used in Agile development is outside the scope of this effort.

- **Continuous Integration and Build** – Necessary features include easy integration with popular source code management tools, support for commonly used test frameworks, support for various build tools, notification and alerting, and reporting. Examples include Hudson (MIT-licensed) and Bamboo (Atlassian Software Systems).
- **Automated Testing** – Necessary features include testing of web applications with AJAX and JavaScript elements, automated capture and replay of user actions, support for functional, compatibility (cross-browser) and regression testing, and support for code-driven testing. Examples include Selenium (open source) and TestComplete (AutomatedQA).
- **Source Code Management or Revision Control** – Necessary features are support for baseline or trunk, revisions at the source file level, version merging, conflict management, commits, release tagging, and branching. This may be included or integrated with a project management tool. Examples include SubVersion (open-source), Mercurial (open-source) and Perforce (Perforce Software Inc.).
- **Integrated Development Environment** – Necessary features include a source code editor, a compiler or interpreter, a debugger, integration with a source code management tool. Examples include Eclipse (open source), NetBeans (Sun) and Komodo (ActiveState).

7.2 Agile Project Management Tools

Agile project management refers to the planning, organizing, and managing of resources associated with an Agile development project. Agile project management encompasses:

- Requirements management (product backlog)
- Planning (releases, iterations, tasks)
- Tracking (progress tracking/reporting)
- Quality assurance (testing, defect management)
- Feedback gathering (user/stakeholder priorities, ideas, and issues)

Each of these areas are important and all are required for a project to be successful.

Typically, with traditional project management, tools are used individually to satisfy these areas. For instance, a spreadsheet or a requirements tool such as Rational RequisitePro is used for requirements management. For planning and tracking Gantt charts are often used, such as Microsoft Project. Bug trackers such as BugZilla and JIRA are often used to track defects in software. Lastly, feedback is often garnered through surveys, email or word of mouth. It may or may not be aggregated into a document or recorded in a requirements tool. These tools do the job, but they are acting independently, without coordinating with each other. Often, there is no integrated source information that ties everything together.

Furthermore, Gantt charts have some inherent flaws that keep them from being a useful day-to-day planning tool for Agile projects. The first flaw is that every task must be time estimated and assigned a date to be performed. When planning a year out, or even further, this is impractical. Typically, tasks are chronically underestimated and the schedule keeps sliding to the right. This leads us to the second flaw: there isn't a way to track changes that have been made to the schedule. In other words, as the schedule slides to the right, there is no way to go back to where you were 6 months ago to gauge your progress. Lastly, Gantt charts are focused on dependencies. The fault in being overly concerned with dependencies is that there really isn't much of a need to track dependencies on a project level. Most dependencies are well known on an IT project and can be handled ad hoc (Dubakov & Stevens, 2008). This isn't to say that tracking dependencies on a programmatic level isn't effective. Gantt charts can be very effective at the high level of a program where the goal is to deconflict project timelines, milestones and dependencies. However, trying to overload a Gantt chart at the low level of an individual project has limited usefulness.

The answer to the above problems is an integration of the entire planning and development process into an overarching, dedicated tool that can be accessible by everyone who is associated with the project. Finding a tool that can achieve this for your project can be hard. There are roughly 80 of these tools available for free or for purchase, each possessing their own set of advantages and drawbacks. Furthermore, one must determine the desired characteristics of an Agile project management tool tailored to the specific environment in which it will operate.

The DOD environment differs from many environments in industry due to its strict security constraints and its distributed nature. Furthermore, Agile development in the DOD isn't as widespread as it is in industry; as such, there isn't a clear front-runner among Agile project management tools. For this reason, the authors have conducted a trade study of Agile project management tools to determine the "best-of-breed" for use in the DOD.

Table 7-1 lists the criteria used to evaluate Agile project management tools for this trade study.

7.2.1 Trade Study Methodology

Table 7-1 Tool Evaluation Criteria

Weight	Evaluation Criteria	Description
5	Collaboration	DOD programs perform development with distributed teams, stakeholders and users. These individuals are located at such sites as government installations, contractor sites and home residences. Collaboration between these individuals is a constant need and for this reason the tool must provide easy access to members on-site and off-site to allow information to flow freely. This is mainly achieved through web-accessible tools that exchange data on port 80; however, other implementations are theoretically feasible.

5	Security	Information systems that operate on government owned or controlled hardware must comply with the 8500 series of DOD Directives. This characteristic will grade how well the tool complies with these directives and how likely it will be to obtain a certification or waiver to operate on DOD owned or controlled servers.
3	User stories/defect management	Takes into account the ability to adequately capture a user story and its associated supplemental information. Typical associated information includes notes, tasks, the ability to prioritize against other user stories, and assignment of responsibility to a team member. The tool should also be able to manage defects in the software as user stories are considered to be new feature requests.
3	Configuration management	This characteristic grades the tool on how well it allows a person or group of people to approve, deny, or modify functional changes to user stories.
3	Management of multiple projects/releases/sprints	A project management tool should have the ability to handle multiple projects with multiple releases and multiple iterative development cycles.
3	Time reporting/estimation	Time estimation is extremely important in software development. It is also extremely difficult to predict. A project management tool should be able to accurately record and display time as it relates to a project so that estimation can be improved upon on a continual basis.
3	Ease of use/access	The tool should be easy to use and easy to understand. Tools that are not easy to use are frequently not used. Also, an Agile project management tool should be used on a daily basis. Any barriers that keep a team member from using a tool should be looked upon negatively. For instance, if a user must first setup a virtual private network to obtain access to the tool the user will not use the tool as frequently.
3	Portability	Members of the development team often use varied operating systems. A tool that can run on more than one OS is preferred.
3	Support	Support by the creator or vendor of the tool is important, but not required. A tool that is no longer supported could lead to problems if a vulnerability is found and there is no way to remedy it. This is especially problematic of applications in which no source code has been provided and there is no support. For applications where the source is provided, at least a fix could be developed.
2	Cost	Cost of the tool should not be prohibitive. Maintenance updates, upgrades and support is also a part of the total cost of ownership and needs to be evaluated.
2	Automation of metrics/charts/reports	Automatic statistic and report generation is an important part of an Agile project management tool. Providing these statistics and reports automatically allows for on-the-fly status briefs to any interested party including senior level management and stakeholders.
2	Integrates with development tools/repositories	The ability to integrate with a wiki for the purposes of documentation or the ability to integrate with a code versioning system for the purposes of documentation or bug tracking would be looked upon favorably. This is not a crucial feature of the tool but enhancements of this nature tend to foster better documentation and project awareness.
1	Ease of installation	Installation of the tool shouldn't be overly complex or require large amounts of hardware. A somewhat complex install can be tolerated because it is usually a one-time process, but overly complicated install processes tend to lead to errors, which could lead to vulnerabilities.
1	Ability to make modifications	Access to the underlying source code is the most straightforward way to make modifications to software. Open source software, for instance, has the benefit of being tailorable to a specific environment. For instance, if vulnerability is found in the tool, and a patch from the vendor is not forthcoming, an in-house fix could be made, whereas with a closed source tool this is not possible.

The approach taken on this tool study is fairly straightforward. An exhaustive list of tools was created by searching the web and from <http://www.userstories.com/products>, which contains a list of Agile PM tools (about 60). Next, tools were selected for evaluation. The selection process removed tools that were not able to operate in the DOD environment, including tools that were not installable (i.e., hosted by a 3rd party), have poorly implemented security, or offer no ability to collaborate between distributed users. Ten tools were selected, and each was installed and evaluated based on the evaluation criteria in Table 7-1.

The evaluation criteria were developed based on the authors' experiences working with Agile PM tools, and interviews conducted with members of the DOD and MITRE. The criteria attempt to strike a balance between security, collaboration and Agile features. Each category has an associated weight between 1 and 5. This weighting was multiplied against the grade received for each category which is also a value between 1 and 5. The weighted category values were then added together to achieve the total grade for the tool. The maximum value a tool could receive is 195 and the minimum is 39. Each tool was also reviewed to discuss features and drawbacks. All evaluations are in [Appendix A](#). The reviews were performed from a time period of January - March, 2010. A summary of the tool results is provided in the following section.

7.2.2 Summary of Tool Results

Table 7-2 Tool Study Results

Score	Tool
179	SoftwareForge
178	JIRA with GreenHopper
168	Agilo for Trac
166	ScrumWorks Pro
153	TinyPM
136	ScrumNinja
134	ScrumDesk
131	ScrumWorks Basic
127	Xplanner+
N/A	Rally

The results in Table 7-2 turned out in favor of DISA's open source collaboration service, SoftwareForge, with a score of 179. SoftwareForge enables the collaborative development and distribution of open source software and DOD community source software. It provides software development tools such as version control, bug tracking, requirements management, and release packaging along with collaboration tools such as wikis, discussion forums, and document repositories to enable collaborative development amongst distributed developers. SoftwareForge is currently built on the open source Subversion version control system and CollabNet TeamForge application life cycle management tool. As of February 2010, SoftwareForge was updated to include the TeamForge version 5.3 which includes Agile planning capabilities, making SoftwareForge an Agile project management tool.

SoftwareForge stands out from the rest of the tools in that it is the only tool that is hosted as a cloud-based, software-as-a-service application provided by DISA. As such, DISA has done the legwork of certification and accreditation to operate on DOD networks. Furthermore, they have

tailored the tool to the DOD environment, making it a very attractive offering to any program that needs a project management and code hosting solution. For these reasons, and the fact that it is offered for free, it is selected as the "best-of-breed".

The second place tool is JIRA with GreenHopper which received a score of 178. JIRA is an issue tracking and project management tool for software development that aims to improve code quality and increase the speed of development. GreenHopper is an Agile project management add-on to JIRA that allows for customizable work flows. JIRA without GreenHopper does not utilize Agile concepts; however, JIRA with GreenHopper is a very mature tool and has a well-thought out design. It is offered commercially, but can be used for free for teams smaller than 10 people.

JIRA with GreenHopper comes as either a hosted solution or an installable application. It fosters collaboration with users outside of the firewall by operating over the Web. It complies with DOD regulations regarding security with the exception of authentication through PKI certificate. It handles multiple projects, releases, sprints, and user stories very well, and it also has very good time reporting and estimation capabilities. It is generally easy to use, though there is a somewhat steep learning curve due to its inherent flexibility and customization. The level of support provided by JIRA is excellent. They have community forums and seem to engage well with their customers. It supports multiple operating systems including Windows, Mac and Linux. The report generation is very robust and includes capabilities to help forecast future costs. The install is fairly straightforward, but not turn-key. It integrates well with various code repositories and has built in bug tracking support through JIRA. It is a proprietary product and not an open source tool, so there is no access to the underlying code.

The third place tool which is worth an honorable mention due to its open source and free nature is Agilo Free. Agilo Free is a plug-in for the popular development tool Trac. Trac is an open source, web-based, project management and bug-tracking tool. Along with Agilo it delivers a robust platform of streamlined functionality for managing Scrum. Agilo targets the challenges of a distributed development environment, it is highly configurable, allowing you to adapt the tool to a specific workflow and provides a single tool to support a team's daily work.

Agilo scores very well in the areas of collaboration, portability, support and cost. The project was created with the intent to allow distributed teams to collaborate easily on open source projects. It installs on any operating system that runs Python (as this is a Python based tool). The open source version is free, but there is also a professional version of this tool available for purchase. The professional version only offers a few extra features and most likely isn't worth the cost unless professional support is a must have.

Overall, this tool study found many suitable Agile PM tools that could be used in the DOD environment. Some were better than others. This trade study should aid in the decision of selecting an Agile PM tool.

8 Case Studies

The use of Agile development methodologies in DOD is on the rise. Many programs of record are using some form of Agile development including: National Senior Leadership Decision Support Service, Joint Space Operations Center Mission System, Joint Communications Support Element, Air and Space Operations Center - Weapons System, Global Command and Control System – Joint, Distributed Common Ground Station - Marine Core (DCGS-MC), DCGS - Intelligence Backbone (DIB) and DCGS - Intelligence Community (DCGS-IC). Two case studies are provided in this section detailing the use of Agile processes in government programs of record. The first case study is on a program (referred to as "Program A") that is creating a collaborative, web-based tool for sharing information and services and the second is on the Joint Operational Planning and Execution System (JOPES). Data for the first case study was collected through interviews conducted in February and August of 2010 with program representatives. The second case study was derived from a published article. Each of the cases includes a description of the project, information on the Agile methodology and processes used; challenges and benefits of adopting Agile, and recommendations.

8.1 Program A

8.1.1 Project Summary

In 2007, Program A was created to be a collaborative, web-based tool to share information and services. The effort is fairly large and serves multiple government programs of record. Development is ongoing.

8.1.1.1 Summary of the Agile Methodology and Related Processes

- *Team:* The team structure includes a breakdown of the team into a government team and a vendor team. The government team includes a Contracting Officer's Technical Representative (COTR), Contracts Officer, Federally Funded Research and Development Center (FFRDC) support, and three System Engineering and Technical Assistance (SETA) contractor personnel. The vendor teams size is about 4-5 persons and includes testers, systems engineers, developers, and security representatives. The Scrum team roles include the Product Owner (a government or SETA contractor assigned to the project), the team (vendor development team plus test and security reps), and the ScrumMaster (vendor). An additional role called the Product Manager was also added. This role is assigned to a vendor team member who serves as the vendor interface to the government Product Owner.
- *Process:* High level requirements are tracked in a spreadsheet or a System /Sub-system Specification (SSS). They are decomposed into features (such as the system must be Protection Level 3 (PL3) compliant), which are then decomposed further into user stories and associated story points. Iterations are tracked at the feature level, using feature points. A Verification Cross-Reference Matrix is used to manage traceability from the SSS to features, to the lower level requirements (such as tasks). Iterations to build features that are not yet integrated are about 4-6 weeks in length. At the end of the iteration, features are demonstrated and signed off by the government, and planning for the next Iteration begins. User feedback is added to the backlog. If features were not implemented in the previous iteration, they become candidates for the next one, along with other high priority features from the product burn down log. Releases occur about every 6 months (there are 4-6

iterations in a release), and include integration and documentation tasks. The team system engineers build Department of Defense Architecture Framework (DODAF) 2.0 document artifacts during the integration iteration. At the end of a release, a Requirements Review Board is convened to look at outstanding requirements and reprioritize open ones, if needed. The team engages users through numerous means: 1) new releases are deployed with team members who engage users and show them the new features; 2) training sessions are offered for users; and 3) feedback can be posted on the program's portal.

- *Tools:* The program is using [Rally](#) as their project management tool. It was considered the best tool for managing multiple teams, but needs to be modified to pull data from the vendor's time accounting system. The Program Office is working to automate the connection of the SSS requirements to Rally to complete the end to end monitoring of requirements to features satisfaction. This data will to be used to calculate earned value. In addition, the team uses two automated test tools: 1) [Load Runner](#) and 2) [Hudson](#).

8.1.2 Challenges

The team experienced challenges with testing and security to meet their organizations requirements. The team initially had problems deciding how to test features, because there were misunderstandings on what 'done' meant. This was resolved by defining acceptance criteria up-front during Iteration planning. The team also experienced challenges with waiting until the end of a release to initiate security planning. This results in unnecessary rework. This problem was resolved by including security representatives as team members during Iteration Planning.

8.1.3 Benefits

Using flexible contracting mechanisms saved money and allowed the program to develop/integrate new technologies quickly. The Agile development process gave the Program Office the opportunity to build out and field capabilities in a more rapid manner.

8.1.4 Lessons Learned

Because Agile software development requires continual improvement and learning, mistakes are to be expected. However, as long as problems are identified and resolved quickly, the team can keep moving forward. One recommendation offered was to leave a couple of blank iterations at the end of a release to handle any unforeseen problems, or testing/security oversights. Another recommendation was to keep the teams small. If a team gets too large (more than 5 or so), break the team into smaller sub-teams.

8.2 Joint Operational Planning and Execution System (JOPES)

8.2.1 Project Summary

The information provided in this section is from an article written by Sean Cohan (2007) of Pragmatics, Inc. Mr. Cohan was the lead engineer on a Defense Information Systems Agency (DISA) project to Service Oriented Architecture (SOA) enable the JOPES program. The paper highlighted that the typical DOD acquisition release duration is about 18 to 36 months, and that DISA wanted to compress that down to 30-60-90 day releases, similar to Google release cycles. The article describes how the team was able to meet these shorter duration cycles.

8.2.1.1 Summary of the Agile Methodology and Related Processes

- *Team:* The initial team consisted of 5 members: the lead engineer, a mid-level developer, 2 junior developers, and an Agile coach. The team later grew to over 20 developers.
- *Process:* The team used software engineering Agile practices such as Test Driven Development, refactoring, and pair programming. The JOPES customer was included early in the development process in order to understand requirements management (story cards), observe automated tests and builds, and review release plans. The team included the test community early in the planning process and provided the DISA testers automated tests with the code delivery.
- *Tools:* The team used an open source tool called [Dokuwiki](#) for knowledge management, [Subversion](#) as a code repository, [Vulcan](#) for continuous integration, and [Cobertura](#) as a test code coverage reporting tool. Mr. Cohan mentioned that the test reports helped Pragmatics, Inc. become CMMI level 4 compliant.

8.2.2 Challenges

The team experienced challenges with the following: having a geographically separated customer, integration of non-Agile team members, and a long testing process that delayed fielding. The team did not have an on-site customer/end user, so they created web applications that allowed the end user to provide feedback and still be involved with development. There were some difficulties integrating non-Agile team members into an Agile development environment. At first they did not like the noise in the team area, having their code open for review and refactoring, and fear of losing their status as senior engineers and team leaders. The Agile coach helped ease transition by teaching the new members Agile processes, such as story card development, refactoring, and pair programming. Additionally, the team worked to keep the noise levels in check. Initially, the team had some difficulty getting the testers to be part of the development team. As a result, it would take months for the software to be approved for fielding. The team resolved this by including the test team earlier in the development cycle, and worked with them to synchronize their test activities to the feature development schedule. Also, at the time the article was being written, DISA was developing a Federation Development Certification Environment (FDCE) that included automated testing and certification capabilities.

8.2.3 Benefits

Cohan cited that management support and having an experienced Agile coach were major contributing factors to the project's success in reaching DISA's overall desire for shorter release cycles.

8.2.4 Lessons Learned

The main lesson learned in the article is to use an Agile coach during early implementation. This helped introduce new Agile concepts to the team, integrate new non-Agile team members, and successfully deal with difficulties as they emerged.

9 Systems Engineering Plan Template

A Systems Engineering Plan (SEP) is a "living" document that captures a program's current and evolving systems engineering strategy and its relationship with the overall program management effort. The SEP's purpose is to guide all technical aspects of the program and is typically established early in the program's life and continually updated. It is used in the Milestone Decision Authority (MDA) approval at each milestone (ACQuipedia, 2008). Creating a SEP can be a hard task, especially if the program doesn't currently have much systems engineering rigor. Furthermore, it can be even harder if the program is making a transition to Agile.

This section aims to ease the burden of creating a SEP by providing an Agile methodology to use as a template for the creation of a program's SEP. The remainder of section 9, is written as sections that could be used in a SEP. To use the template, paste the relevant sub-sections into your program's SEP and modify it to suit the needs of the program.

9.1 Agile Systems Engineering Plan Introduction

This SEP template uses an Agile methodology based on existing models and practices as frameworks to build a hybrid approach that allows a program to capitalize on the benefits and minimize the drawbacks of various approaches. When selecting an Agile methodology, it is important to understand that each practice comes with benefits and drawbacks, and there is no "one size fits all" approach to adopting a single method. As with all engineering processes, tailoring is required to satisfy the idiosyncrasies of a given project.

This hybrid approach employs an Agile development methodology based primarily on Scrum and supplemented with Extreme Programming (XP) practices. Scrum itself is not a software development methodology (Mountain Goat Software, 2010), rather an iterative, incremental process for development that focuses on delivering capabilities to the end user; it is thought of more as a framework or an Agile project management methodology (Paulik, Davis, & Maccherone, 2009). When implemented correctly, Scrum focuses on quick insertion of incremental capability that is driven by user feedback. It also creates a program management structure that is conducive to dealing with constant change.

While Scrum's strong points focus around project management, it does not specify low-level engineering practices that help ensure product quality. To address this lack of specificity, and to ensure product quality, low-level engineering practices from XP are recommended as a supplement to the Scrum approach. As Kniberg describes in *Scrum and XP from the Trenches* (2007), this is a common approach and these two methodologies complement each other well.

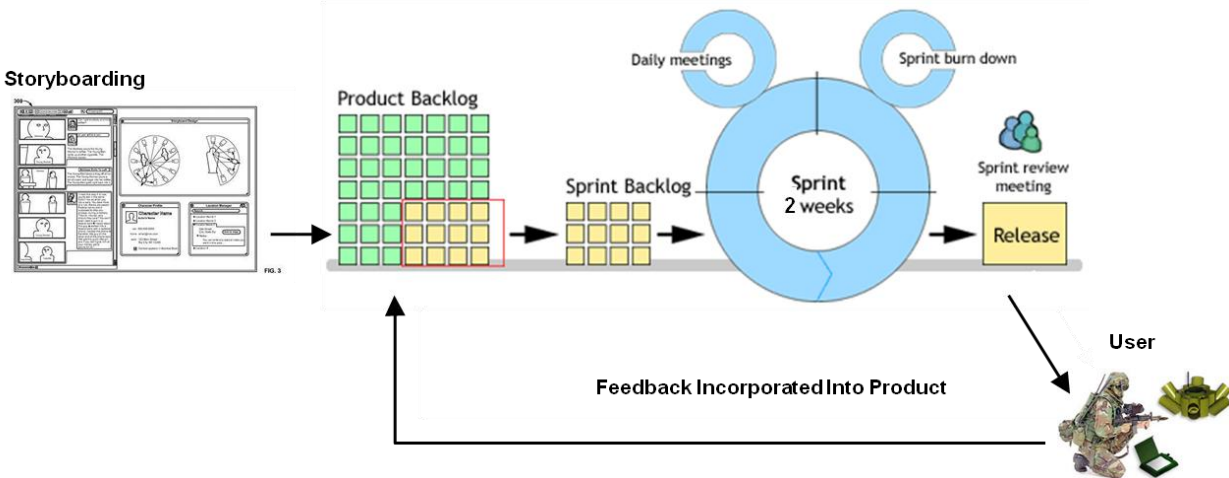


Figure 9-1: Agile Process

Scrum provides a project management framework that incorporates, developers, stakeholders, and end users together to produce a potentially fieldable capability at the end of every time boxed iteration, called a “sprint”. Figure 9-1 illustrates this process and the following sections detail how Scrum can be tailored for a DOD IT program.

9.2 Capability Requirements Analysis

Capability requirements are organized in a living document called the “product backlog”. The product backlog is an evolving, prioritized queue of operational and technical requirements, as depicted in Figure 4-1. It can contain requested features, change enhancements, and defect corrections; in other words, anything that constitutes a change to the product that should be made available for future releases. All stakeholders, end users, and developers can add capability requirements to the product backlog. In effect, the product backlog is a prioritized “wish list” of features to be incorporated into the system, and can change as end user's needs change.

Initially, the product backlog should be populated from the program's documentation such as the Capability Description Document (CDD) and Capability Production Document (CPD). Classified information should be handled in a separate classified product backlog.

After information is initially collected and added to the product backlog, a storyboarding workshop should be held. Storyboarding is described in Section 9.2.2. This storyboarding workshop is an opportunity to present the capability requirements to the user community and to refine them further with storyboards and user stories. During a storyboarding workshop the development team, product owner (described further in the Section 9.6.1) and users meet to walk through operational threads to refine capabilities already on the product backlog and to identify new capabilities. The product backlog will become a living document and should be refined based on user evaluation and feedback at the end of every sprint, or at predefined evaluation events.

9.2.1 User Stories

Capability requirements should contain a user story. A user story is a concise, written description of a capability that will be valuable to a user (or owner). It is written in a narrative fashion from the user's perspective. It contains just enough information, "the who, what and why" necessary to describe a requirement. User stories should be:

- **Independent** – User Stories should be as independent as possible, and minimize dependency on other user stories to deliver full value.
- **Negotiable** – User Stories are not detailed specifications. They are high level descriptions of features for the team to discuss with the story owner and collaborate to clarify the details near the time of development.
- **Valuable** – User Stories should be valuable to the user (or owner) of the solution. They should be written in user language, not technical jargon. They should be at the feature level of description, not the task level.
- **Estimatable** – User Stories need to provide enough information to estimate level of effort, without being too detailed.
- **Small** – User Stories should be small and succinct.
- **Testable** – User Stories should be worded in a way that provides for a testable result, i.e. not subjective and clearly defined, stating true or false outcomes.
- **Traceable** – User Stories are traceable to overarching mission threads and CONOPS.

A well written user story takes this form:

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

For example:

As a registered user I want to log in so I can access subscriber-only content.

A well written user story provides the key components of a capability:

- **Who** - The user role.
- **What** - The user's goal.
- **Why** - The reason for the user wanting to achieve the goal. This:
 - Gives clarity as to why a feature is useful
 - Can influence how a feature should function
 - Provides a basis for the functional test (that is written before the code)
 - Can give you ideas for other useful features that support the user's goals

9.2.2 Storyboarding and Mockups

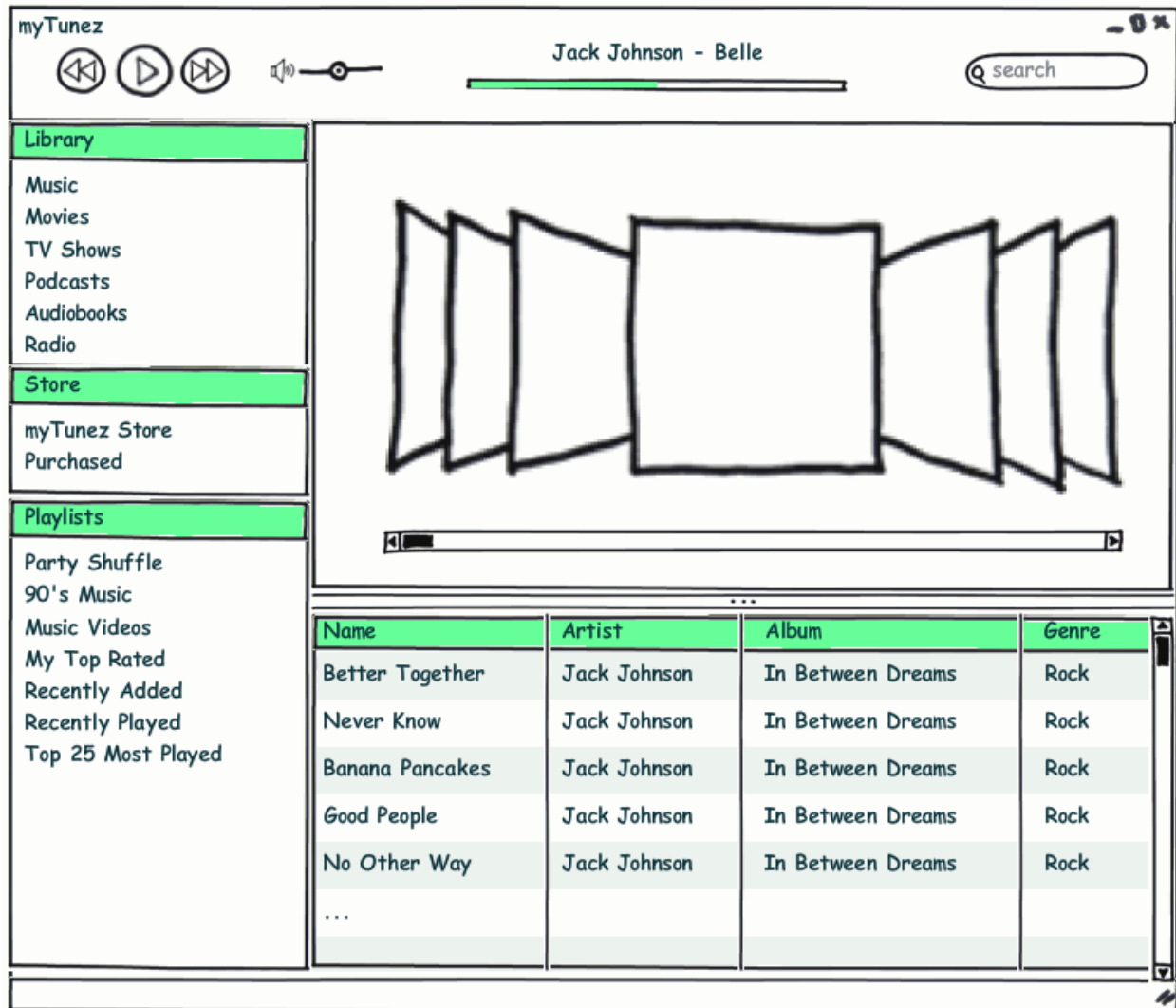


Figure 9-2 Wire-frame Mockup of iTunes Cover Flow Feature (source: <http://www.balsamiq.com/products/mockups/examples#mytunez>)

The program should use storyboarding and mockup demonstrations as a way to help visualize the uses of the system and its features. A storyboard is a narrative visual depiction set in time that describes how the system will be used. Google uses storyboards for almost every product they launch. The implementation of their storyboards range from comic strips to animated videos, but all of them describe how and why someone should use the product. A mockup is a visual depiction of a feature of the system. For example, when Apple created Cover Flow in iTunes so that users could visually browse their music collection using album cover artwork, it was obviously very useful for them to show executives and developers what cover flow would look like before it was built. Using a mockup, they could easily show exactly where in iTunes Cover Flow would be displayed and roughly how it would look and function. Figure 9-2 demonstrates a wire-frame mockup of iTunes Cover Flow.

9.2.3 Prioritization

The product owner is responsible for establishing the product backlog priorities so that higher priority items will be implemented before lower priority items. This ensures scope optimization throughout the life of the project. The product owner is responsible for socializing with the user community to understand their needs in order to make decisions regarding the backlog item priorities.

9.2.4 Capability Estimation

Agile estimation is also an iterative and incremental process. Estimations are performed by the development team and are used to gauge complexity. Estimations can become more accurate over time (if used correctly), as past performance information emerges throughout the project. Estimates are starting points, from which sprints can be empirically constructed and managed.

User stories should be estimated by the development team using a measure called 'story points'. A story point is a unit of measure that expresses the development complexity of a user story relative to the other user stories in the product backlog. Each user story is assigned a story point value. The raw value assigned is unimportant unless it is relative to every other story point value. For example, a user story that is assigned a value of '2' should be twice as complex as a user story that is estimated to have a value of '1'. Engineering judgment and analogy will be the primary method to develop the basis of estimate for each estimated user story. This basis of estimate will become refined prior to each sprint when the user stories are decomposed into tasks and time required to completing them is associated.

The scale of story point values should be Fibonacci numbers (i.e. 1, 2, 3, 5, 8, 13, 21, 34, 55, 89, ...). Fibonacci numbers are derived recursively by taking the sum of the previous two numbers in the sequence, (i.e. $1 + 2 = 3$, $2 + 3 = 5$). A value of '1' would be trivial, such as performing a find and replace refactor on a term. A value of '89' would be very complex, usually reserved for user stories where the design is unknown or a way forward is unclear. Fibonacci numbers make good story point values because the increasing gap between the available values nicely represents the increase in uncertainty when estimating larger user stories. By using story points, the team can determine the amount of work they can perform in a given time period, known as 'velocity'.

Velocity is a measure of a team's rate of progress. It is calculated by summing the number of story points assigned to each user story that the team completed during a sprint. If a team averages X story points in the past sprints, they are likely to complete X story points in future sprints. Because story points are estimates of relative size, this will be true whether they work on two five-point stories or five two-point stories.

As the development sprints begin, the team's true velocity will become apparent over the first few sprints. This points-based approach to estimation is self-correcting. As the true velocity becomes known, the schedule can be re-adjusted to reflect the actual number of story points that can be completed during a sprint.

So far we've discussed 'why' you estimate, but not 'how'. How you estimate can be performed through a fun process called planning poker. Planning poker takes place by gathering the team and giving each team member cards with estimate (Fibonacci) numbers on them. The team discusses a user story and then, at the same time, each person throws down a card with the number of the estimation for that user story. Then the team discusses the numbers that were thrown and normalizes on a number. For instance, if a team consisting of five members throws a 2, 5, 5, 5, and a 13, the person who threw the 2 would be asked "Why so low?" and the person

who threw a 13 would be asked "Why so high?". Through this process the team each gets to voice their opinion about the complexity of the requirements and a group consensus forms on the estimation of this complexity. More info on planning poker can be found on its Wikipedia page http://en.wikipedia.org/wiki/Planning_poker.

Time required to achieve a completed user story is purposely left out of this section. This takes place during a sprint planning session which will be described in the following section. It is important to treat time estimation and complexity estimation separately, as complexity does not always directly equate to time. Furthermore, time estimations require a breakdown of work at the task level which is not performed until a user story has been committed to a sprint.

9.3 Planning

Just like other activities within Agile methodologies, planning also occurs iteratively and incrementally in order to reflect reality and to embrace change. Each planning activity is driven by the cost, schedule and performance constraints. It is recommended that a program have three levels of planning: *release*, *iteration*, and *daily*.

Release planning is strategic in nature, and considers the user stories that will be developed for major releases of the program. The goal of release planning is to determine the appropriate scope, schedule, and resources for the entire project. This planning occurs at the beginning of the project after the initial product backlog has been developed. The release plan is updated throughout the project (usually at the start of a sprint). It is important to note that release planning is a form of intent, not commitment. Commitment is agreed upon at the individual sprint level. This ensures scope is properly optimized and remains current throughout the life of the project.

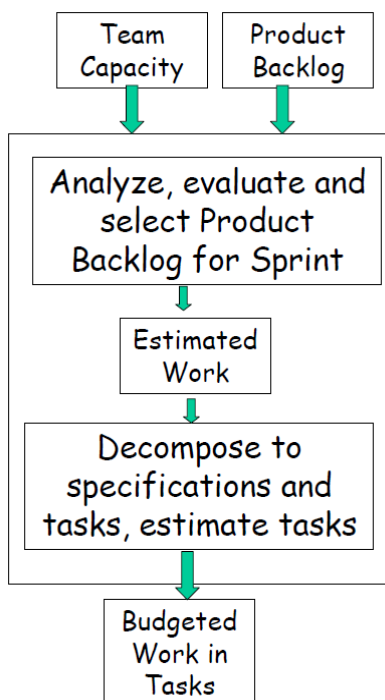


Figure 9-3: Iteration Planning Meeting Flow

Iteration or sprint planning, shown in Figure 9-3, is where the development team commits to completing a set of user stories in a given time period. Iteration planning is conducted by the development team and the product owner. This planning is more tactical in nature, and is conducted before the start of each sprint. Based on the work accomplished in the previous sprints, the product owner identifies high priority items from the product backlog to be completed in the sprint. During iteration planning, the team establishes the engineering tasks needed to transform a backlog item into working and tested software. It includes all tasks necessary to start with a user story and to finish with a fully functioning, potentially shippable product. For the high risk user stories, use cases should be developed that outline the details of the user story (a use case template can be found in 11Appendix E). During this planning meeting the team estimates (in hours) how long each task will take. All analysis, design, user interface design, integration, testing, certification, accreditation, and documentation tasks need to be identified and estimated in terms of hours. After a user story's tasks have been estimated, hours are summed and stored for the user story along with its story point value. This comparison will also help the estimates improve over time.

Daily planning (referred to as the 'daily scrum' or stand-up) occurs daily and is the forum where the scrummaster, product owner, and development team assess and revise their sprint plan. Its goal is make adaptations that optimize the value of the next workday. During the daily stand-up sessions, the product owner, scrummaster and development team constrain the planning horizon to be no further away than the next day, when they will meet again. The focus is on the planning and coordination of individuals' activities and the assignment of the tasks produced in iteration planning to individuals. These daily stand-up meetings should last no more than fifteen minutes. This is not the forum to discuss any topic in great depth. These discussions should be directly coordinated with only the required individuals and discussed outside of the daily stand-ups. Each development team member discusses the following:

- 1) What he or she has done since the last daily stand-up.
- 2) What he or she is going to do before the next daily stand-up.
- 3) What obstacles are in his or her way.

9.4 Sprint Execution

9.4.1 Sprint Development

Program work should be confined to a regular, repeatable cycle, known as a sprint. Sprints are usually 2 – 6 weeks in length. The length of a sprint is usually determined by the speed in which important requirements emerge. In a very fast paced environment where competing high-priority requirements change from week to week, or even day to day, a smaller sprint length should be chosen. Sprint length is often tweaked based on past performance, but it should not be changed during a sprint, as sprints are time-boxed once underway.

Once the development team commits to the work in a sprint, the sprint's scope cannot be altered. This is important because it ensures the sprint will have an achievable finish line. The work is finite and fixed, i.e. new user stories are not being added to the sprint and existing user stories are not being augmented to include new functionality. This also ensures that work, once started, is carried through to completion. In environments where high priority requirements change weekly, or even daily, it is common to start a development task without finishing it to move on to

the newer, higher priority task. In environments like this, it may be beneficial to undertake short sprint durations such as 2 weeks.



Figure 9-4: Sample Burn Down Chart

At the end of each day, development team members should enter the hours remaining for his/her assigned tasks in an Agile project management tool. Team members' data will be aggregated and displayed in a burn down Chart, as shown in Figure 9-4. The X-axis represents days in the sprint, while the Y-axis is the estimated effort remaining (usually in ideal engineering hours). The planned activities, with associated completion times, can be compared to the actual activities, with associated actual completion times, to help monitor project performance.

Source code repository, internal configuration management, automated testing, Agile management and defect management tools should be used by the development team. These tools are discussed more in Section 7.

9.4.2 Sprint Integration

The development team should strive for continuous integration, described in Section 6.1.3. Developers should integrate their work at least once daily, using continuous integration software to automate the verification of a build and detect integration errors as quickly as possible. Integration testing will ensure that newly developed capabilities integrate successfully into the evolving enterprise software baseline.

9.4.3 Sprint Testing

Testers should be fully integrated into the development team. This ensures the testers are always up-to-date and can readily communicate with the developers and the product owner.

During planning meetings, testing should be discussed amongst the development team to identify the best way to fulfill testing requirements. Automated testing and continuous integration should be implemented to the fullest extent possible. Having continual builds running unit tests and functional tests will allow testers to perform more exploratory testing.

No user story can be marked as complete until its testing has been successfully completed. There should be four major types of testing performed during the sprint:

1. **Unit/Component Testing** – Unit tests verify functionality of the smallest subset of the system, such as an object or method. Component tests verify the behavior of a larger part of

the system, such as a grouping of classes that provide a service or a logical grouping of functionality. Both types of tests are usually automated with a member of the xUnit family of test automation tools.

2. **User Story Testing** – User story tests are derived from examples provided by the customer team and documented in the capability requirements. They describe the details of each capability. Operational-facing tests run at a functional level, each one verifying an operational satisfaction condition. They are written in a way end users can easily understand using the operational domain language. The domain experts will use these tests to define the external quality of the product. It's possible this testing could duplicate some of the tests that were done at the unit level; however, these tests are oriented toward illustrating and confirming desired system behavior at a higher level.
3. **User Assessment Testing (UAT)** – UAT gives end users a chance to evaluate the system in an operational context to assess its effectiveness and to suggest new capability requirements.
4. **Ility Testing** – Ility tests are technology-facing tests that are discussed in technical, rather than end user terms. Technology-facing are intended to critique product characteristics such as human factors, performance, robustness, and security (Barbaci, Klein, Longstaff, Weinstock, 1995).

The testing process is continuously improved by applying feedback from the team. Improvements to the testing process (as well as any other process in the sprint) are also discussed in the sprint retrospective.

9.4.4 Sprint Retrospective

The program should hold a sprint retrospective at the end of each sprint. A sprint retrospective is a meeting in which only the development team, scrummaster, and product owner discuss how the sprint went and how future sprints can be improved. When providing feedback, it is important not to dwell on specific failings or shortcomings of specific team members, rather discuss how things could be improved the next time around. Scrum is an adaptive framework and retrospectives are a chance to ensure that Scrum is being properly applied to the specific environment in which you operate.

9.4.5 Sprint Review

The sprint review is a meeting that takes place at the end of each sprint, in which the development team demonstrates the functionality created in that sprint. This meeting is led by the development team and is attended by anyone who is interested, but specifically the product owner and other interested stakeholders. Users are encouraged to participate as well.

Typically, the developer who implemented the user story will demonstrate the functioning of that user story. Questions are fielded and feedback is garnered. Once the demonstrations have completed, and the product backlog has been updated to reflect the feedback, the product backlog is then re-prioritized by the group, but with ultimate approval from the product owner.

9.5 Tools

Based upon the Agile project management tool study, it is recommended that SoftwareForge offered by Forge.mil be the Agile project management tool of choice. SoftwareForge provides development tools such as software version control, bug tracking, requirements management, and release packaging along with collaboration tools such as wikis, discussion forums, and document repositories to enable collaborative development amongst distributed developers. As

discussed in Agile Project Management Tools (Section 7.2), SoftwareForge stands out from the rest of the tools in that it is the only tool that is hosted as a cloud-based, software-as-a-service application provided by DISA.

In addition, a product from each of the following categories of tools would be beneficial:

- **Continuous Integration and Build Tool** – Necessary features include easy integration with popular source code management tools, support for commonly used test frameworks, support for various build tools, notification and alerting, and reporting. Examples include Hudson (MIT-licensed) and Bamboo (Atlassian Software Systems).
- **Automated Testing Tool** – Necessary features include testing of web applications with AJAX and JavaScript elements, automated capture and replay of user actions, support for functional, compatibility (cross-browser) and regression testing, and support for code-driven testing. Examples include Selenium (open source) and TestComplete (AutomatedQA).
- **Source Code Management or Revision Control Tool** – Necessary features are support for baseline or trunk, revisions at the source file level, version merging, conflict management, commits, release tagging, and branching. This may be included or integrated with a project management tool. Examples include SubVersion (open-source), Mercurial (open-source) and Perforce (Perforce Software Inc.).
- **Integrated Development Environment Tool** – Necessary features include a source code editor, a compiler or interpreter, a debugger, integration with a source code management tool. Examples include Eclipse (open source), NetBeans (Sun) and Komodo (ActiveState).

9.6 Roles

This section discusses five specific roles that should be present on a DOD IT program: 1) Product Owner, 2) ScrumMaster, 3) Integration Team, 4) Stakeholders, and 5) Users.

9.6.1 Product Owner

The Product Owner, a critical role in Scrum, is responsible for understanding and communicating the users' and stakeholders' concept of operations to the engineering team, including the product vision and value proposition. The Product Owner is responsible for managing the product backlog and refining its contents on a daily basis, including prioritization of the backlog to ensure the most important operational concepts are being addressed by both the defined and derived requirements. The Product Owner also must collaborate with the team on a daily basis to answer questions when they arise, provide feedback, and sign off on work results. The Product Owner is often thought of as “a single wringable neck” because this one person is ultimately responsible for the success or failure of the project.

9.6.2 ScrumMaster

It is recommended that the program designate a team ScrumMaster whose job is to perform ScrumMaster duties. The ScrumMaster facilitates the scrum process, protects the team from interference, and removes obstacles so that the team can deliver the sprint goal. The ScrumMaster role is loosely mapped to the role of project manager in traditional software engineering methodologies; however, he does not control the team as a traditional project manager would, since the team is self-organizing.

The ScrumMaster has many duties, but most importantly, the ScrumMaster should:

- Act as a buffer between the team and any distracting influences.

- Ensure that the Scrum process is used as intended.
- Be the enforcer of team rules.
- Remove obstacles and protect the team to keep them focused on the task at hand.

9.6.3 Integration Team

The integration team is responsible for developing and integrating the software required to meet program capability requirements. The term "integration team" is used synonymously with "development team". The integration team includes software engineers, data specialists, security engineers, testers, and other specialists as needed. This team should be self-organizing. The concept of self-organizing teams is discussed in Section 4.11

9.6.4 Stakeholders

The term "stakeholder" is applied to anyone who has a vested stake in the project. This includes those people who enable the project or who receive an agreed upon benefit.

9.6.5 Users

The term "user" refers to anyone who will be a direct or indirect user of the program system. A direct user is anyone who uses a program service or component. An indirect user includes those who use a service or component, not directly, but through another piece of software. In other words, another software application is consuming a service and the use of this software makes that user an indirect user of the program.

9.7 Trade Studies

System analysts use trade studies to support decisions about capability requirements and design alternatives, especially when analyzing commercial off the shelf (COTS) or Government off the shelf (GOTS) products for integration. These trade studies target performance drivers and constraints from the program.

Agile development uses a similar concept referred to as 'spikes', which are time boxed periods of research and development to investigate a concept or COTS product to assess its suitability of fulfilling a requirement. The duration and objective(s) of a spike should be agreed upon between the product owner, scrummaster and development team before it starts. Spikes will normally be shorter than a standard sprint; however, they can range in duration from a few hours to a few weeks.

Unlike sprints, spikes may or may not deliver tangible, shippable, valuable functionality. For example, the objective of a spike might be to research COTS products to determine if they are the best way to implement a capability. The recommendation may very well be to create an in-house product because no suitable COTS, GOTS, or open-source alternative could be found. This is consistent with DISA's strategy of adopt-buy-create that has been successfully used by NCS and other DISA programs (Browne, 2006).

Spikes will usually take place in between sprints and are often introduced before the delivery of capability in order to:

- Secure budget
- Expand knowledge
- Proof of concept prior to committing resources to a particular vendor

Table 9-1 Cost Effectiveness Parameters

Cost Containment Parameters	System Effectiveness Parameters
Research, design, and development cost Production cost System operation cost Maintenance and support cost	System performance Availability, reliability, supportability and usability System quality Security Other technical factors

Part of the trade study process should include cost-benefit and cost-containment analyses. This will be used to provide economic balance to the systems engineering decision-making process. Cost-benefit analyses weigh the total cost of design alternatives against their effectiveness in order to determine the relative value of solutions. These analyses attempt to capture all short-term and long-term costs associated with an item. The potential costs and effectiveness parameters to be considered in the analyses are listed in Table 9-1.

10 Conclusion

This handbook provides guidelines focused on Agile development methodologies and tools to enable a DOD IT program to rapidly field new capabilities to the Warfighter. Agile development is a set of industry accepted software development practices that are now beginning to emerge in DOD acquisition programs. This handbook includes background information on Agile principles and methodologies from multiple open source and peer reviewed materials, as well as interviews with members of Government Agile software programs. It also includes the results of a market survey on Agile project management tools.

The information provided in this handbook demonstrates that the emerging Agile practices within the Government Acquisition community combined with the proven Agile development methodologies of the commercial sector can decrease time-to-field of valuable new capabilities. Agile software development is not a "silver bullet" and these practices and methodologies must be implemented with care.

Each commercial practice comes with benefits and drawbacks and there is no "one size fits all" approach to adopting a single method. As with all engineering processes, tailoring is required to satisfy the uniqueness of a given project. Using existing models and practices as frameworks, a hybrid approach is one option that allows a DOD program to capitalize on the benefits and minimize the drawbacks of various approaches.

An exemplary hybrid approach is an Agile development methodology based primarily on Scrum supplemented with Extreme Programming practices. Scrum itself is not a software development methodology, rather an iterative, incremental process for development that focuses on delivering capabilities to the end user; it is thought of more as a framework or an Agile project management methodology. When implemented correctly, Scrum focuses on quick insertion of incremental capability that is driven by user feedback. It also creates a program management structure that is conducive to dealing with constant change.

While Scrum's strong points focus around project management, it does not specify low-level engineering practices that help ensure product quality. To address this lack of specificity, and to ensure product quality, low-level engineering practices from XP are recommended as a supplement to the Scrum approach. As Kniberg describes in *Scrum and XP from the Trenches* (2007), this is a common approach and these two methodologies complement each other well.

Section 9 contains a systems engineering plan (SEP) modeled around Scrum and XP. This SEP, which incorporates the findings of this report, is intended to serve as a tailorable foundation for DOD programs.

Forge.mil's SoftwareForge received the highest evaluation score in the Agile Project Management tool trade study. SoftwareForge provides development tools such as software version control, bug tracking, requirements management, and release packaging along with collaboration tools such as wikis, discussion forums, and document repositories to enable collaborative development amongst distributed developers. As discussed in Agile Project Management Tools Section 7.2, SoftwareForge stands out from the pack in that it is the only tool that is hosted as a cloud-based, software-as-a-service application provided by DISA.

Agile training is crucial to the success of the project for teams that are transitioning to an Agile methodology. This is based on a finding of Section 4.4 and a lesson learned from the JOPES program in Section 8.2. A course on Agile development or hands-on training from an Agile coach are both effective means to train teams. Specialized Agile training in the form of Scrum training is recommended for teams adopting an Agile methodology using Scrum.

A change to the Defense Acquisition System resulting from the Defense Science Board's (2009) report is likely in 2011. Therefore, as a final recommendation, programs are urged to take advantage of the Agile Acquisition and Contracting guidance discussed in Section 5 and Section 4.10, respectively.

11 References

- ACQuipedia (2008). Systems Engineering Plan (SEP). Retrieved from <https://acc.dau.mil/CommunityBrowser.aspx?id=29041>
- Advanced Development Methods, Inc. (2010). XP @ Scrum diagram. Retrieved from <http://www.controlchaos.com/about/xp.php>
- Atkinson, R., Crawford, L., & Ward, S. (2006). Fundamental uncertainties in projects and the scope of project management. *International Journal of Project Management*, 24(8), 687-698. doi: 10.1016/j.ijproman.2006.09.011
- Barbaci, M., Klein, M., Longstaff, T., & Weinstock, C., (1995). Quality attributes. Carnegie Mellon University Software Engineering Institute. Retrieved from <http://www.sei.cmu.edu/reports/95tr021.pdf>
- Beck, K., Beedle, M., Bennekum, A., Cockburn, A., Cunningham, W., Fowler, M., et al. (2001). Manifesto for Agile software development. Retrieved from <http://agilemanifesto.org/>
- Boehm, B., & Turner, R. (2004). *Balancing agility and discipline: A guide for the perplexed*. Boston, MA: Pearson Education, Inc.
- Barry, B. (2006). A view of 20th and 21st century software engineering. Paper presented at the Proceedings of the 28th international conference on Software engineering, Shanghai, China.
- Browne H. A., (2006). DISA enters new era of opportunity. Signal Online, Retrieved from http://www.afcea.org/signal/articles/templates/SIGNAL_Article_Template.asp?articleid=1095&zoneid=9
- Chow, T., & Cao, D.-B. (2008). A survey study of critical success factors in Agile software projects. *Journal of Systems and Software*, 81(6), 961-971. doi: 10.1016/j.jss.2007.08.020
- Cockburn, A., & Williams, L., (2000). [The costs and benefits of pair programming](#). Proceedings of the First International Conference on Extreme Programming and Flexible Processes in Software Engineering (XP2000). Retrieved from <http://collaboration.csc.ncsu.edu/laurie/Papers/XPSardinia.PDF>
- Cohan, S. (2007). Successful integration of Agile development techniques within DISA. Paper presented at the Agile 2007 Conference.
- Cohen, M. (2007). Advice on conducting the Scrum of Scrums meeting. Retrieved from <http://www.scrumalliance.org/articles/46>
- Davis, J. P., Eisenhardt, K. M., & Bingham, C. B. (2009). Optimal structure, market dynamism, and the strategy of simple rules. *Administrative Science Quarterly*, 54(3), 413-452. doi: 10.2189/asqu.2009.54.3.413

Defense Science Board. (2000). *Report of the Defense Science Board Task Force on: Defense Software*. Retrieved from <http://www.acq.osd.mil/dsb/reports/ADA385923.pdf>

Defense Science Board. (2009). *Report of the Defense Science Board Task Force on: Department of Defense policies and procedures for the acquisition of information technology*. Retrieved from <http://www.acq.osd.mil/dsb/reports/ADA498375.pdf>

Deming, W. E. (1982). *Out of the crisis*. Cambridge, MA: MIT Press.

Demmer, P., Benefield, G., Larman, C., & Vodde, B. (2009). *The Scrum primer, V1.2*. Retrieved from http://scrumtraininginstitute.com/home/stream_download/scrumprimer

Design Patterns. (2010, March 26). In *Wikipedia, the free encyclopedia*. Retrieved April 1, 2010, from [http://en.wikipedia.org/wiki/Design_pattern_\(computer_science\)](http://en.wikipedia.org/wiki/Design_pattern_(computer_science))

Dubakov, M., & Stevens, P. (2008). *Agile tools. The good, the bad and the ugly*. Retrieved from <http://targetprocess.com/download/whitepaper/agiletools.pdf>.

Duquette, J., Bloom, M., & Crawford, L. (2008). *Transitioning Agile/Rapid Acquisition initiatives to the Warfighter. The MITRE Corporation, MITRE Technical Report WN080041*.

Duquette, J., Bloom, M., Crawford, L., & Osgood, D. (2007). *Final report: E520 technical initiative project Agile/Rapid Acquisition. The MITRE Corporation, MITRE Working Note WN070047*.

Fernandez, D., & Fernandez, J. (2008). *Agile project management - Agilism versus traditional approaches. The Journal of Computer Information Systems, 49(2), 10-17*.

Fowler, M., (2006). *Continuous integration*. Retrieved April 1, 2010, from <http://martinfowler.com/articles/continuousIntegration.html>.

Glazer, H., Dalton, J., Anderson, D., Konrad, M., & Shrum, S. (2008). *CMMI or Agile: Why not embrace both!* Retrieved on January 5, 2010 from <http://www.sei.cmu.edu/reports/08tn003.pdf>

United States General Accounting Office. (2002). *Acquisition workforce Department of Defense's plans to address workforce size and structure challenges*. Retrieved from <http://www.gao.gov/new.items/d02630.pdf>

Government Accountability Office. (2008a). *OMB and agencies need to improve planning, management, and oversight of projects totaling billions of dollars*. Retrieved from <http://www.gao.gov/new.items/d081051t.pdf>

Government Accountability Office. (2008b). *Better weapon program outcomes require discipline, accountability, and fundamental changes in the acquisition environment*. Retrieved from <http://www.gao.gov/new.items/d08782t.pdf>

Harmancioglu, N., McNally, R. C., Calantone, R. J., & Durmusoglu, S. S. (2007). Your new product development (NPD) is only as good as your process: An exploratory analysis of new NPD process design and implementation. *R&D Management*, 37(5), 399-424. doi: 10.1111/j.1467-9310.2007.00486.x

Jeffries, R. (2001). What is Extreme Programming? Retrieved from <http://xprogramming.com/xpmag/whatisxp>

Johnson, J. (2009). CHAOS 2009. Chaos Activity News, 4(3). Retrieved from http://www.standishgroup.com/chaos_news/2009/march/index.php

Kniberg, H., (2007). Scrum and XP from the trenches – How we do Scrum. Retrieved from <http://www.infoq.com/minibooks/scrum-xp-from-the-trenches>

Koch, A. S. (2005). *Agile software development: Evaluating the methods for your organization*. Norwood, MA: Artech House.

Kolawa, A., & Huizinga, D., (2007). [Automated defect prevention: Best practices in software management](#). Wiley-IEEE Computer Society Press. p. 75. ISBN 0470042125.

Lindstrom, L., & Jeffries, R. (2004). Extreme Programming and Agile software development methodologies. *Information Systems Management*, 21(3), 41-60. doi: 10.1201/1078/44432.21.3.20040601/82476.7

Matharu, J., (2005). Reusing safety-critical software components. COTS Journal. Retrieved from <http://www.cotsjournalonline.com/articles/view/100380>

Moe, N., Dingsoyr, T., & Dyba, T. (2008). Understanding self-organizing teams in Agile software development. Retrieved from <http://ieeexplore.ieee.org/stamp/stamp.jsp?arnumber=04483195>

Mountain Goat Software (2010). Introduction to Scrum – An Agile process. Retrieved from <http://www.mountangoatsoftware.com/topics/scrum>

National Academy of Sciences. (2009). Achieving effective acquisition of information technology in the Department of Defense. Retrieved from http://www.nap.edu/catalog.php?record_id=12823

Pair Programming. (n.d.). In *Wikipedia, the free encyclopedia*. Retrieved March 5, 2010 from http://en.wikipedia.org/wiki/Pair_programming

Paulik, M., Davis, N., & Maccherone, L. (2009). On empirical research into Scrum. Retrieved from http://www.scrumalliance.org/resource_download/1156

Poppendeick, LLC (2009). Lean programming part 1 and 2. Retrieved from <http://www.poppendeick.com/lean.htm>

Project Management Institute. (2008). *A guide to the project management body of knowledge (PMBOK guide) (4th ed.)*. Newtown Square, PA: Project Management Institute.

Schatz, B., & Abdelshafi, I. (2005). Primavera gets agile: a successful transition to Agile development. *IEEE Software*, 22(3), 36-42.

Sliger, M., & Broderick, L. (2008). *The Software Project Manager's Bridge to Agility*. New York: Addison-Wesley

Steffens, W., Martinsuo, M., & Arto, K. (2007). Change decisions in product development projects. *International Journal of Project Management*, 25(7), 702-713. doi: 10.1016/j.ijproman.2007.01.008

Stevens, R. G., King, M. K., & Halley, M. R. (2009). Acquisition strategies for dealing with uncertainty. The MITRE Corporation. MITRE Public Release 09-1310.

Surendra, N. C. (2008). Using an ethnographic process to conduct requirements analysis for Agile systems development. *Information Technology and Management*, 9(1), 55-69. doi: 10.1007/s10799-007-0026-6

Sutherland, J., Jakobsen, C., & Johnson, K. (2007). Scrum and CMMI Level 5: A magic potion for code warriors. Agile Conference. Denver, CO, July 2005. Retrieved from <http://jeffsutherland.com/2007/09/scrum-and-cmmi-level-5-magic-potion-for.html>

Takeuchi, H., & Nonaka, I. (1986). The new product development game. *Harvard Business Review*, 64(1), 137-146.

The Standish Group, Inc. (2009). Chaos manifesto: The laws of chaos and the CHAOS 100 best PM practices. Retrieved from http://www.standishgroup.com/custom_chronicles/showfile.php?File=CHAOS-Manifesto.pdf

Vaihansky, P. Sutherland, J. & Victorov, A. (2008). Hyperproductivity in large projects though distributed Scrum. Agile Journal, Dec 2008. Retrieved from <http://www.agilejournal.com/articles/columns/column-articles/190-hyperproductivity-in-large-projects-though-distributed-scrum>

VersionOne, Inc. (2009). 4th annual state of Agile survey: State of Agile development. Retrieved March 15, 2010 from <http://www.versionone.com/Resources/Whitepapers.asp>

Vinekar, V., Slinkman, C. W., & Nerur, S. (2006). Can Agile and traditional systems development coexist? An ambidextrous view. *Information Systems Management*, 23(3), 31-42. doi: 10.1201/1078.10580530/46108.23.3.20060601/93705.4

Waterfall model. (n.d.). In *Wikipedia*. Retrieved December 15, 2010 from http://en.wikipedia.org/wiki/Waterfall_model

Xia, W., & Lee, G. (2004). Grasping the complexity of IS development projects. *Communications of the ACM*, 47(5), 68-74. doi: 10.1145/986213.986215

Appendix A Results

A.1 SoftwareForge

Grade	Weight	Total	Actual	Characteristic
4	5	25	20	Collaboration
5	5	25	25	Security
4	3	15	12	Tracking of user stories/defects
5	3	15	15	Configuration Management
5	3	15	15	Handles multiple projects, releases, sprints
5	3	15	15	Time estimation/reporting
3	3	15	9	Ease of use/access
5	3	15	15	Portability
5	3	15	15	Support
5	2	10	10	Cost
5	2	10	10	Automation of Metrics, Charts and Reports
5	2	10	10	Integrates into repositories (wiki, svn, etc.)
5	1	5	5	Ease of install
3	1	5	3	Ability to make modifications
		195	179	

SoftwareForge enables the collaborative development and distribution of open source software and DOD community source software. For open source and community source development projects within the DOD, SoftwareForge provides software development tools such as software version control, bug tracking, requirements management, and release packaging along with collaboration tools such as wikis, discussion forums, and document repositories to enable collaborative development amongst distributed developers. SoftwareForge is currently built on the open source Subversion version control system and CollabNet TeamForge application life cycle management tool. As of February 2010, SoftwareForge was updated to include the TeamForge version 5.3 which includes Agile planning capabilities, making SoftwareForge an Agile project management tool.

SoftwareForge stands out from all the rest of the tools in that it is the only tool that is hosted as a cloud-based, software-as-a-service application provided by DISA. As such, DISA has done the legwork of certification and accreditation to operate on DOD networks. Furthermore, they have tailored the tool to the DOD environment, making it a very attractive offering to any DOD program that needs a project management and code hosting solution. For these reasons, and the fact that it is offered for free, it receives a very good score of 179.

SoftwareForge is part of a larger program called Forge.mil. Forge.mil (which can be accessed at <http://www.forge.mil>) is a family of services aimed at improving the ability of the DOD to rapidly deliver dependable software, services and systems in support of net-centric operations and warfare. Currently, it has two offerings, SoftwareForge and ProjectForge. It is working on three other services, TestForge, CertificationForge, and StandardsForge. TestForge will provide hosting space for the development of testing tools and environments, CertificationForge will provide a collaborative space for managing the certification of net-centric enterprise services.

StandardsForge will provide a collaborative space for developing and reviewing standards programs.

ProjectForge is like SoftwareForge in that they use the exact same software platform. The only difference is that ProjectForge is for projects that need to operate privately. The cost for ProjectForge is hefty at \$60k per year for 100 users. SoftwareForge is completely free, but has the caveat that all content can be read by anyone who has access to SoftwareForge. Access to SoftwareForge is limited to only those individuals who possess a valid DOD Common Access Card (CAC) or a PKI certificate issued by a DOD approved External Certificate Authority (ECA). By performing authentication in this fashion, SoftwareForge effectively limits access to only the DOD community. In fact, they created a new type of software license called DOD Community Source which, like an Open Source license is free and open for anyone to use, however it is limited to only those individuals associated with the DOD. It should be noted that this license only applies to software hosted on their site. A DOD organization could use SoftwareForge for its Agile project management aspects alone and not be restricted to releasing their software with an Open Source or a DOD Community Source license.

The ability to collaborate on SoftwareForge is very good for people who have access to the service. However, for those who do not have a CAC or ECA, there is no way for them to access the content contained inside. While most of the DOD individuals who work on base have a CAC card, those contractors who do not work on base may not. This is problematic considering the widespread use of contractors throughout the DOD.

SoftwareForge receives very high marks in every category except "tracking of user stories/defects", "ease of use/access" and "open source". Currently there is no way to associate a task to a user story or to a defect. This poses a problem when trying to breakdown user stories into individual tasks. Hopefully, this will be remedied in a future version. Ease of use and access is also a problem in that the site can be kind of slow and sometimes very verbose. Lastly, the TeamForge code in which SoftwareForge is based on is not open source. This doesn't particularly pose a problem as DISA is the maintainer of this code base and they do listen to user feedback in cases of bugs or feature requests.

Overall, SoftwareForge is a very impressive Agile project management tool. It has a few caveats, but no showstoppers. Furthermore, it is actively being developed and should get even better over time.

A.2 Atlassian JIRA with GreenHopper

Grade	Weight	Total	Actual	Characteristic
5	5	25	25	Collaboration
4	5	25	20	Security
4	3	15	12	Tracking of user stories/defects
5	3	15	15	Configuration Management
5	3	15	15	Handles multiple projects, releases, sprints
5	3	15	15	Time estimation/reporting
4	3	15	12	Ease of use/access
5	3	15	15	Portability
5	3	15	15	Support
3	2	10	6	Cost
5	2	10	10	Automation of Metrics, Charts and Reports
5	2	10	10	Integrates into repositories (wiki, svn, etc.)
5	1	5	5	Ease of install
3	1	5	3	Ability to make modifications
		195	178	

JIRA is an issue tracking and project management tool for software development that aims to improve code quality and increase the speed of development. GreenHopper is an Agile project management add-on to JIRA that allows for customizable work flows. JIRA without GreenHopper does not utilize Agile concepts. JIRA with GreenHopper scored very well with a 178 due to its very mature and well-thought out design. This evaluation was performed on version 4.1 of JIRA and version GreenHopper version 4.2.

JIRA with GreenHopper comes as either a hosted solution or an installable application. It fosters collaboration with users outside of the firewall by operating over the Web. It complies with DOD regulations regarding security with the exception of authentication through PKI certificate. It handles multiple projects, releases, sprints, and user stories very well and it also has very good time reporting and estimation capabilities. It is generally easy to use, though there is a somewhat steep learning curve due to its flexibility and customization, which is to be expected. The level of support provided by JIRA is excellent. They have community forums and seem to engage well with their customers. It supports multiple operating systems including Windows, Mac and Linux. The report generation is very robust with capabilities to help forecast future cost. The installation is fairly straightforward, but not turn-key. It integrates well with various code repositories and has built in bug tracking support through JIRA. It is a proprietary product and not an open source tool, so there is no access to the underlying code.

The pricing for JIRA with GreenHopper is very competitive, though somewhat complicated. Atlassian provides JIRA for free to open source projects, and organizations that are non-profit, non-government, non-academic, non-commercial, non-political, and secular. For commercial customers, the full source code is available under a developer source license.

Starting with JIRA 4, a 10-user starter license costs \$10 with all proceeds benefiting charity. The price then jumps to \$1,800 for 25 users and \$3,300 for 50 users. There are also fees for maintenance and upgrades. Overall, JIRA with GreenHopper is a very good tool. The only

downsides are cost and the fact that there isn't PKI authentication for use with Common Access Cards (CAC).

A.3 Agilo Free

Grade	Weight	Total	Actual	Characteristic
5	5	25	25	Collaboration
3	5	25	15	Security
5	3	15	15	User stories/defect management
4	3	15	12	Configuration management
4	3	15	12	Management of multiple projects/releases/sprints
4	3	15	12	Time reporting/estimation
3	3	15	9	Ease of use/access
5	3	15	15	Portability
5	3	15	15	Support
5	2	10	10	Cost
4	2	10	8	Automation of Metrics, Charts and Reports
5	2	10	10	Integrates with development tools/repositories
5	1	5	5	Ease of install
5	1	5	5	Ability to make modifications
		195	168	

Agilo Free is a plug-in for the popular development tool Trac. Trac is an open source, web-based project management and bug-tracking tool. Along with Agilo it delivers a robust platform of streamlined functionality for managing Scrum. Agilo targets the challenges of a distributed development environment, is highly configurable, allowing you to adapt the tool to a specific workflow, and provides a single tool to support a team's daily work. This evaluation was performed on version 1.3.0.7 of Agilo Free and version 0.11.1 of Trac.

Agilo scores very well in the areas of collaboration, portability, support and cost. This is mainly due to its open-source nature. The project was created with the intent to allow distributed teams to collaborate easily on open source projects. It installs on any operating system that runs Python (as this is a Python based tool). The open source version is free, but there is also a professional version of this tool available for purchase. The professional version only offers a few extra features and most likely isn't worth the cost unless professional support is a must have.

The tool scores less highly for security and ease of use. Security is an issue in that it currently does not run over SSL and offers no authentication for PKI. Ease of use is also an issue with Agilo for Trac. There isn't much of use AJAX, thus requiring a new page load every time a request is sent to the server, which is tedious. Also, the layout of form fields is clunky.

Overall, Agilo Free is a very good tool. With some modifications to the code base to make it a little more secure it could be a very good choice for use in a DOD environment.

A.4 ScrumWorks Pro

Grade	Weight	Total	Actual	Characteristic
5	5	25	25	Collaboration
4	5	25	20	Security
4	3	15	12	Tracking of user stories/defects
2	3	15	6	Configuration Management
5	3	15	15	Handles multiple projects, releases, sprints
5	3	15	15	Time estimation/reporting
3	3	15	9	Ease of use/access
5	3	15	15	Portability
5	3	15	15	Support
3	2	10	6	Cost
5	2	10	10	Automation of Metrics, Charts and Reports
5	2	10	10	Integrates into repositories (wiki, svn, etc.)
5	1	5	5	Ease of install
3	1	5	3	Ability to make modifications
		195	166	

ScrumWorks Pro is an Agile project management tool from CollabNet that focuses on building the Scrum framework directly into the product management workflow. There is also a free version of ScrumWorks called ScrumWorks Basic that is meant to give you a taste of the tool line in hopes that you will buy the professional version. The basic version is also reviewed in this study because it is almost a completely different tool. This evaluation was performed on version 4.5 of ScrumWorks Pro.

ScrumWorks Pro is a very nice tool with many well-thought out features. It comes complete with drag and drop interface, return on investment analyst tools, coordination between multiple projects and integration with other tools such as JIRA and Eclipse. While the features they've implemented are generally good, there are a few questionable "features" of the GUI. In particular, there is a component docking system that allows you to place components where you want them. Secondly, they've gone overboard with the notion of tabs. Seeing their interface, it becomes obvious now why there are quite a bit of Agile development tools that stress simplicity.

ScrumWorks Pro allows for distributed collaboration by using Java Web Start as a mobile application platform for deploying the application. The data is stored in a central database that is accessed through web services from the Web Start application. This works, but may cause the organization to need to open a port in the firewall to allow outside access.

The security of ScrumWorks looks to be fairly sound. They have support for SSL, but no support for PKI. There is a lack of configuration management for user stories, however the user could add a configuration management field into the notes section for each user story if needed. The pricing seems reasonable at \$500 per seat for life, or \$289 per seat per year. There is also enterprise and concurrent licensing options available upon request.

The return on investment feature adds the notion of business weight that takes into account the benefit of implementing a user story verses the penalty of implementing it (time it takes to do

so). From these two pieces of information they can deduce a return on investment for each user story.

There is also integration with Bugzilla and JIRA for bug and issue tracking. Being able to track your user stories, bugs and issues all from the same place is nice. In this case, they let someone else who has worked with this area for many years do the heavy lifting, a commendable approach.

Another useful feature is their web "Team Task Board", that lets users quickly update their status from the web, rather than having to start the somewhat lengthier process of launching the Java app.

Overall, the Pro version is a good tool, but not the best available. By adding a wealth of features and GUI options the tool is powerful but can be overwhelming.

A.5 tinyPM

Grade	Weight	Total	Actual	Characteristic
5	5	25	25	Collaboration
2	5	25	10	Security
4	3	15	12	Tracking of user stories/defects
5	3	15	15	Configuration Management
5	3	15	15	Handles multiple projects, releases, sprints
2	3	15	6	Time estimation/reporting
5	3	15	15	Ease of use/access
5	3	15	15	Portability
5	3	15	15	Support
4	2	10	8	Cost
2	2	10	4	Automation of Metrics, Charts and Reports
4	2	10	8	Integrates into repositories (wiki, svn, etc.)
4	1	5	4	Ease of install
1	1	5	1	Ability to make modifications
		195	153	

tinyPM is a lightweight Agile project management tool written in Java. It emphasizes its simple and straightforward approach to project management while still packing in the features. While it scored very well in most of the grading categories, it scored poorly in security, time estimation/reporting, and automation of metrics, charts and reports, resulting in a score of 153. This evaluation was performed on version 2.0 of tinyPM.

tinyPM does a very good job at fostering an environment of collaboration around the project. It comes with email and RSS notifications to ensure team members stay in the loop. It also operates over the web as a web application ensuring that anyone with a web browser can get access to the tool.

There is no mention of it being able to run over SSL or using PKI authentication, thus without modifications it is lacking adequate security. It does a fairly good job of tracking user stories and bugs, however there is no way to differentiate between and user story and a bug so in order to track bugs you would need to somehow designate a user story to be a bug, possibly in the title or notes section. It handles multiple projects, releases and sprints very well. The time estimation and reporting is a little lacking. There doesn't seem to be a way to keep track of fine grained hours worked on the project, which is important for determining burn rates and velocity, which most likely explains why it lacks in the generation of metrics and meaningful charts.

tinyPM does a very good job of making the tool easy to use and easy to access. It can be installed on the three major operating systems which makes it very portable. tinyPM is not open source, and offers support with the purchase of the license. The install is fairly straightforward and easy.

tinyPM packs a few unexpected features and does them quite nicely. There is a wiki embedded in the tool for documentation. It also does a very good job at tracking changes and compiling changes into an activity stream. It also integrates with other tools such as JIRA and supports

web services so that it can be extended if needed. tinyPM is a nice, lightweight tool that takes minimal time to set up and can get your team up and working quickly. If it can fix a few of the problems listed previously it could be a very good choice for use in the DOD environment.

A.6 ScrumNinja

Grade	Weight	Total	Actual	Characteristic
5	5	25	25	Collaboration
4	5	25	20	Security
3	3	15	9	Tracking of user stories/defects
2	3	15	6	Configuration Management
4	3	15	12	Handles multiple projects, releases, sprints
3	3	15	9	Time estimation/reporting
3	3	15	9	Ease of use/access
5	3	15	15	Portability
5	3	15	15	Support
3	2	10	6	Cost
2	2	10	4	Automation of Metrics, Charts and Reports
0	2	10	0	Integrates into repositories (wiki, svn, etc.)
3	1	5	3	Ease of install
3	1	5	3	Ability to make modifications
		195	136	

ScrumNinja, as its name implies, is a Scrum based project management tool that aims to keep it simple. Overall, it is a fairly straightforward and simple tool. It's web based, though you can host it yourself.

There is a backlog for managing your user stories and there is a "card wall" for managing the tasks associated with the current sprint. Both seem easy to use and understand. One thing that was disappointing with the backlog, however, is that there isn't a backlog. All user stories are committed to sprints as soon as they are created. In order for a user story to be in the backlog, it must not have been committed to a sprint. That's why it's called the backlog, because it's on the backburner and hasn't yet been agreed upon to be implemented.

The pricing is staggered depending on how many users are buying a seat. 1-9 members is a one-time payment of \$289 per user and the price increases from there. This includes support and maintenance for the first year, after which support and maintenance is 20% per user of the original price.

A.7 ScrumDesk

Grade	Weight	Total	Actual	Characteristic
3	5	25	15	Collaboration
2	5	25	10	Security
3	3	15	9	Tracking of user stories/defects
2	3	15	6	Configuration Management
5	3	15	15	Handles multiple projects, releases, sprints
4	3	15	12	Time estimation/reporting
3	3	15	9	Ease of use/access
5	3	15	15	Portability
4	3	15	12	Support
3	2	10	6	Cost
5	2	10	10	Automation of Metrics, Charts and Reports
5	2	10	10	Integrates into repositories (wiki, svn, etc.)
2	1	5	2	Ease of install
3	1	5	3	Ability to make modifications
		195	134	

ScrumDesk is an intuitive Agile project management tool geared for performing Scrum with distributed teams. ScrumDesk comes with many features including planning poker, a story board, integration with external repositories like bug trackers, a sprint retrospective tool, and informative graphs. It's not a web application, rather it's a .Net application that runs on the desktop and open a database connection directly to a SQL server. This can pose security problems. This evaluation was performed on version 4.0.5.6 of ScrumDesk.

ScrumDesk has pretty much every feature one could ask of a Scrum tool, though it tends to be a bit buggy. It's difficult to tell how much of it is user error and how much is system error. ScrumDesk uses the Microsoft .NET Windows Presentation Foundation which strays from typical GUI conventions like using "Ok" and "Cancel" buttons. It takes a little getting used to and can lead to some frustrating moments.

ScrumDesk has a good grasp of Scrum practices. Everything inside the tool looks dead on and very useful. There are alternate views of the content, and they have a very nice way to visualize your user stories called TreeMap. It takes each user story and based on the amount of task hours projected to complete the User Story, it prints a square on the screen representing the size of that user story until the screen is completely full. This allows you to easily see what your biggest and smallest user stories are.

The pricing structure for this tool is fair. The complete tool, not a watered down version, is free for up to 5 users (up to 20 for a non-profit company), and then it's \$15 per user per month after that.

ScrumDesk only installs on Windows. The install is fairly complex as well.

Overall, ScrumDesk is a decent tool but it probably isn't a good choice to use in a DOD environment due to its security problems.

A.8 ScrumWorks Basic

Grade	Weight	Total	Actual	Characteristic
4	5	25	20	Collaboration
3	5	25	15	Security
3	3	15	9	Tracking of user stories/defects
2	3	15	6	Configuration Management
5	3	15	15	Handles multiple projects, releases, sprints
3	3	15	9	Time estimation/reporting
3	3	15	9	Ease of use/access
5	3	15	15	Portability
3	3	15	9	Support
5	2	10	10	Cost
3	2	10	6	Automation of Metrics, Charts and Reports
0	2	10	0	Integrates into repositories (wiki, svn, etc.)
5	1	5	5	Ease of install
3	1	5	3	Ability to make modifications
		195	131	

ScrumWorks Basic is a free Agile project management tool from CollabNet that focuses on building the Scrum framework directly into the product management workflow. This is the free version of ScrumWorks that is like the Pro version, but hasn't been developed in a few years. ScrumWorks Basic is a bare-bones tool that gets the job done. There are a few quirks with the tool and few missing features. This evaluation was performed on version 1.8.4 of ScrumWorks Basic.

ScrumWorks Basic allows for distributed collaboration by using Java Web Start as a mobile application platform for deploying the application. The data is stored in a central database that is accessed through web services from the Web Start application. This works, but may cause the organization to need to open a port in the firewall to allow outside access. There is no support for SSL or for PKI which is a drawback.

The tool tracks user stories and defects in an acceptable fashion, though there is some room for improvement. Defect tracking isn't specifically called out, but can still be performed. There is no configuration management built-in, though it could be performed in the notes section of user stories. The time estimation and reporting could also be improved upon. There is initial estimation of work to be performed for the user story and then for each task there is a work left to be done. There is no way to see the history of how much work was left to be done for a task which would be nice to have.

Since the tool is a Web Start application it has to load itself each time before it can run, which can be cumbersome. A web application tends to provide faster access. Also, it isn't the most straightforward of tools, though after some practice it can be understood and used quite well.

CollabNet offers support for this tool but not as substantial as with the Pro version. There is no integration with other repositories unless the integration is performed manually through the web services that are offered by ScrumWorks. The install was very easy. Lastly, modifications

couldn't be made to the tool, but they could be made to the data with access of the tools web services.

ScrumWorks Basic, while lacking in features and sophistication is a good tool that can get the job done. The security is lacking which can be a deal breaker for use in the DOD environment unless remedied.

A.9 XPlanner+

Grade	Weight	Total	Actual	Characteristic
5	5	25	25	Collaboration
2	5	25	10	Security
2	3	15	6	Tracking of user stories/defects
3	3	15	9	Configuration Management
4	3	15	12	Handles multiple projects, releases, sprints
2	3	15	6	Time estimation/reporting
3	3	15	9	Ease of use/access
5	3	15	15	Portability
3	3	15	9	Support
5	2	10	10	Cost
3	2	10	6	Automation of Metrics, Charts and Reports
1	2	10	2	Integrates into repositories (wiki, svn, etc.)
3	1	5	3	Ease of install
5	1	5	5	Ability to make modifications
		195	127	

XPlanner+ is an open source project planning and bug tracking tool. It is written in Java and is based on the open source tool XPlanner (XPlanner is not reviewed because it only runs on Java 1.4.2 and has a disclaimer that states "not for production use"). XPlanner+ gets a modest score of 127 mainly due to its free, portable and open source nature. This evaluation was performed on version 1.1a2 of XPlanner+.

XPlanner+ can be installed on any operating system that can run Java. It allows for collaboration between users by running over the web as a web application. There is no mention of it being able to run over SSL or using PKI authentication, thus, without modifications it is lacking adequate security. The handling of user stories and bugs isn't as mature as some other tools. It does however, handle multiple projects and iterations very well, though it doesn't allow for release management. Configuration Management of user stories is somewhat lacking in that a user story can be in various stages like "planned", "defined" or "estimated", but there isn't a concrete process for gaining approval from a person or a group of persons. The time reporting and estimation is a little lacking. There is only the ability to record estimated remaining hours. The cost of the tool is free because it is open-source; however, there is no option for paid support by the main developer. The report/chart generation is also somewhat lacking in that there are only a handful of charts and metrics to view. The install looks to be fairly straightforward, but not turnkey. Lastly, there is no integration with a documentation system or code versioning system.

XPlanner+ is vast upgrade from XPlanner, but it is still not quite a strong enough or mature tool to be used in the DOD environment for Program of Record use.

A.10 Rally

Rally was slated for inclusion in this report, but the Rally Software Development Corporation failed to respond to multiple requests for a trial version, therefore an evaluation could not be performed.

Appendix B CMMI and Agile Paradigm Comparison

This comparison table is from *CMMI or Agile: Why Not Embrace Both!* (Glazer, Dalton, Anderson, Konrad, & Shrum, 2008, pp. 35-37).

Table 11-1 CMMI and Agile Paradigm Comparison Table

Dimension	CMMI Paradigm	Agile Paradigm
Organizational focus of attention	The focus is on the organization or enterprise. Most benefit occurs when CMMI is implemented at organizational level so that all functions and capabilities contributing to the development of products and services are addressed by the process improvement effort.	The focus is on the project and team. Agile methods can isolate (i.e., insulate) the project/team from the organization and still be effective.
Management	Management plays an important role in ensuring project success. There is much attention to project management, including ensuring that plans affecting the project are integrated with the project plan, dependencies are managed, coordination issues are resolved, there is a shared vision for the work, and risk management is performed.	Management is a coaching function (as opposed to traditional command-and control) that helps to eliminate barriers to progress. This view of management may be expanding as Agile approaches are extended to address larger project contexts.
Trust	Some CMMI practices assume the need to compensate for a low-trust environment (a key concern of the Agilistas). A low trust environment is often characterized by (1) safety and mission critical objectives, (2) high risk of failure, and most of all (3) multiple stakeholders who cannot be totally transparent relative to their intentions and commitments.	Agile methods originated from the recognition that teams work best when they are composed of task-mature individuals operating in high trust groups. An Agile environment fosters high trust.
Planning	CMMI promotes macro (project-level) planning with an emphasis on establishing a suitable defined process enabling the project to achieve its objectives. Traditional planning approaches assume a long time horizon. Detailed plans are not required by CMMI but many of the examples encourage such an interpretation. Nevertheless, many who use CMMI also use “rolling plans” (detailed only to the next iteration or quarter). There is emphasis on re-planning and conditional change.	In Agile methods, there are multiple levels of planning, including high-level product planning (including release planning) and at the beginning of each iteration, more detailed planning around the features to be addressed in that iteration. There is a strong emphasis on flexibility and re-planning as conditions change. Use of Gantt charts (that map tasks to calendar time) and graphs of task networks is discouraged because the requirements on which the tasks are based change frequently.
Market/User Assumption	CMMI is broadly beneficial but particularly so when the target market becomes more mature and process innovation becomes a more important differentiator to organization success.	Agile methods have the most benefit in an emergent and not well-understood target/market.
Design Presumptions	CMMI presumes the product architecture is selected or created in the early stages of a project and is revisited when it becomes clear the selected architecture is no longer valid or when using an iterative lifecycle.	Projects are most successful when corporate standard architectures are adopted with flexibility applied as the project progresses. Some Agile methods downplay the importance of architectures on the general principle that such are often prematurely specified.

Learning	Learning happens in many ways, including: (1) organizational training based on analyses of process and skill needs and priorities and the design of appropriate training vehicles; (2) through the development activities of the project itself (e.g., the evaluation of proposed product requirements and solutions for feasibility and cost, schedule, quality, and risk impact); (3) through the planning and use of the processes on projects, measurements, and lessons learned are gathered and shared across the organization; (4) as part of a causal analysis process; and (5) through an analysis of how well the organization's and project's quality and process performance objectives are being met by the processes in use.	Learning happens at project/iteration levels, typically bottom-up and just-in-time (i.e., comparable to Kaizen).
Perspective	CMMI takes, has, and assumes a longer term view.	Agile takes, has, and assumes a short to medium-term view.
Appraisals	The SCAMPI method compares the organization's processes against the practices of CMMI to evaluate whether the organization has implemented processes that achieve CMMI goals.	The desire of Agile practitioners is that appraisals are made only by looking at results (i.e., customer satisfaction and other project outcomes).
Human Development	CMMI has a limited people focus at the project level. There is an expanded people focus at the organizational level. CMMI does not advocate heroics, but instead creating an environment in which people can excel without heroics (i.e., an effective process appropriately leverages people and technology).	Agile has a team and individual focus (i.e., people over process). There has been a tendency among many in the Agile community to advocate "just hire good people" with an implicit undercurrent of developing "hero developers" within a work-life balanced culture.
Life-Cycle Emphasis	<p>CMMI has a strong "review-as-you develop" emphasis. There is also a tendency to read in CMMI the approach "verify often and validate at the end," which is NOT the correct way to understand what CMMI is saying.</p> <p>CMMI is consistent with an environment in which there is a high cost of failure (see Trust above). The objective is to proactively avoid the high costs associated with product failure (or time-to-market failure). The conclusion is that we cannot rely on testing alone. Therefore, CMMI encourages documentation, analyses, and reviews before product components are integrated into a functional product. CMMI also encourages frequent (early and mid-course) validations to ensure the right product is being built. The project determines the appropriate application and timing of these review and testing steps.</p>	<p>Agile methods employ concurrent development, test iterations, and informal peer reviews of work products as necessary.</p> <p>There is a tendency to see in Agile a "validate often and verify second approach" which may be a more correct reading.</p> <p>"Fail early, fail fast, and learn" are central to Agile methods. The trade-off is seen as favoring the development of a useable and testable product vs. developing and analyzing requirements and product components.</p> <p>A low cost of delay or low cost of failure is assumed. There is an assumption of incremental delivery being viable.</p> <p>Burn out can arise from repeated time-boxed iterations, so special provisions should be made for this eventuality (e.g., by revising the time-box duration based on experience feedback and by providing some slack around iterations).</p>

Predictability	<p>Predictability is achieved for critical sub-processes (those process steps that are leading contributors to or indicators of quality and process performance) through statistical management (which involves, among other things, use of control charts).</p> <p>Predictability is achieved at the project, iteration, or level by monitoring the performance of critical sub-processes and periodically using the organization's established process performance models, appropriately calibrated, to evaluate whether the project is on track to achieving its objectives. Finally, at the organizational level, analysis of sub-process performance across projects provides an increased understanding of the capability of the organization's processes and thus of which areas to target for innovation.</p>	<p>Using time-boxed scope-limited iterations, evolving designs/solutions, driving out defects as early as possible, and failing quickly leads toward a predictable development velocity.</p> <p>There is also an expectation of predictability of iteration delivery and scope of delivery. The anecdotal evidence suggests that Agile teams often struggle with predictability of iteration scope, often de-scoping iterations (or sprints) in the final few days. This de-scoping is rooted in a lack of statistical convergence of the velocity data combined with the analysis technique underlying the scope breakdown.</p>
Cost of Failure	<p>Historically, CMMI was developed in a domain of high cost of failure. If a plane crashes, the cost of failure is extremely high. Examples within this domain include the development of aircraft, weaponry, spacecraft and safety-critical medical devices.</p>	<p>Agile methods have flourished in a domain of low cost of failure or linear incremental cost of failure. Examples within this domain include Internet commerce, social networking, and games development.</p>

Appendix C Recommended Reading

C.1 Websites

Table 11-2 Recommended Websites Table

Category	Name	Source	Notes
General	Agile Manifesto	http://agilemanifesto.org/	
General	Agile Alliance	http://www.agilealliance.org	Good articles to download
General	Agile Community	http://www.agile-community.com/	Community of Interest
General	InfoQ	http://www.infoq.com/	Videos, articles, and downloads available.
General	Mountain Goat Software	http://www.mountaingoatsoftware.com/	Lots of downloadable resources
General	Version One	http://www.versionone.com/	Version One develops Agile tools but also offers lots of great information
Lean	Lean Software Development	http://www.poppendieck.com/	The Poppendiecks offer lean references, papers, and book reviews
Scrum	Manager 2.0: Role of the Manager in Scrum	http://scrumtraininginstitute.com/home/stream_download/%23%3cUUID:0xb747aba0%3e-71972	This paper gives an excellent description of how the role of a manager changes on a Scrum team.
Scrum	Scrum Alliance	http://www.scrumalliance.org/	
Scrum	Scrum Log Jeff Sutherland	http://www.jeffsutherland.com/	Jeff Sutherland offers Scrum-related posts and articles
Scrum	Control Chaos	http://www.controlchaos.com/	Ken Schwaber offers several downloads and a very clear explanation of Scrum
XP	eXtreme Programming: A Gentle Introduction	http://www.extremeprogramming.org	Provides a good primer on XP
XP	eXtreme Programming Yahoo Group	http://tech.groups.yahoo.com/group/extremeprogramming/	Community of Interest
XP	XP	http://www.xprogramming.com	Ron Jeffries offers XP-related posts, articles, and book reviews

C.2 Books

Table 11-3 Recommended Books Table

Category	Source	Name	ISBN
General	The Mythical Man-Month: Essays on Software Engineering, Anniversary Edition (2nd Edition)	Frederick P. Brooks	9780201835953
Lean	Lean Software Development: An Agile Toolkit	Mary and Tom Poppendieck	0321150783
Scrum	Agile Project Management with Scrum	Ken Schwaber	073561993X
XP	Extreme Programming Explained: Embrace Change (2nd Edition)	Kent Beck and Cynthia Andres	0321278658

Appendix D Glossary

C&A	Certification and Accreditation
CDD	Capabilities Development Document
CMM	Capability Maturity Model
CMMI	Capability Maturity Model Integration
CM	Configuration Management
COBOL	Common Business-Oriented Language
COTR	Contracting Officer's Technical Representative
COTS	Commercial Off the Shelf
CPD	Capability Production Document
DCGS	Distributed Common Ground Station
DCGS-DIB	Distributed Common Ground Station -Intelligence Backbone
DCGS-IC	Distributed Common Ground Station - Intelligence Community
DCGS-MC	Distributed Common Ground Station - Marine Corps
DISA	Defense Information Systems Agency
DLOE	Distributed Limited Objective Event
DOD	Department of Defense
DODAF	Department of Defense Architecture Framework
FDCE	Federation Development Certification Environment
FDD	Feature Driven Development
FFRDC	Federally Funded Research and Development Center
GOTS	Government Off-the-Shelf
HTML	Hyper Text Markup Language
HTTP	Hyper Text Transfer Protocol
ICD	Initial Capability Document
IIDD	Iterative and Incremental Design and Development
IT	Information Technology
IV&V	Independent Verification and Validation
JCIDS	Joint Capabilities Integration Development System
JOPES	Joint Operational Planning and Execution System
JROC	Joint Requirements Oversight Council
NASA	National Aeronautics and Space Administration
NCES	Net-Centric Enterprise Services
NIPRNET	Non-Classified Internet Protocol Router Network

NRO	National Reconnaissance Office
PL3	Protection Level 3
POR	Program of Record
RAD	Rapid Application Development
RUP	Rational Unified Process
SEI	Software Engineering Institute
SETA	System Engineering and Technical Assistance
SEP	Systems Engineering Plan
SIPRNET	Secret Internet Protocol Router Network
SSS	System /Sub-system Specification
TDD	Test Driven Development
TRD	Technical Requirement Document
US	United States
USASOC	United States Army Special Operations Command
XP	Extreme Programming

Appendix E Use Case Template

Table 11-4 Use Case Template

System:	The system section calls out the system that is being used in the use case.
Name:	A use case name provides a unique identifier for the use case. It should be written in verb-noun format (e.g., <i>Borrow Books</i> , <i>Withdraw Cash</i>), should describe an achievable goal (e.g., <i>Register User</i> is better than <i>Registering User</i>) and should be sufficient for the end user to understand what the use case is about.
Actors:	An actor is someone or something outside the system that uses system. An actor may be a person, another system or sub-system, or time.
Summary:	A summary section is used to capture the essence of a use case. It provides a quick overview, which is intended to save the reader from having to read the full contents of a use case to understand what the use case is about. Ideally, a summary is just a few sentences or a paragraph in length and includes the goal and principal actor.
Triggers:	A trigger section describes the event that causes the use case to be initiated.
Assumptions:	Conditions that must be true for the use case to begin and end. For example, if your use case deals with editing a file in Microsoft Office, an assumption would be: The file the actor is opening is in Microsoft Office format.
Success Path:	The success path of a use case represents the most important course of events or what happens most of the time, sometimes referred to as the 'Happy Day Scenario' because it is what occurs when everything goes well -- no errors or exceptions. Another reason why the success path is so critical is because it is much easier to fully comprehend the exceptions once the norm is understood and if the basic flow represents 70% of the system.
Alternate Paths:	The alternate flows providing the following: <ul style="list-style-type: none"> • An exception or error flow to any line item in your success path • An additional flow, not necessarily error based, but a flow that COULD happen A few examples of alternate flows are: <ul style="list-style-type: none"> • While a customer places an order, their credit card failed • While a customer uses an ATM machine, the machine runs out of receipts and needs to warn the customer
Post-Conditions:	The post-conditions section describes what the outcome is when the use case successfully finishes.
Business Rules:	Business rules are requirements that do not fit into the success path. An example would be: Video shall be encoded in MPEG-4 format.
Open Issues	Document any issues that need to be resolved prior to this use case being finalized.
Notes:	The Notes section has important information that does not fit under any specific heading. Traceability to higher level documents such as a CONOPS can go here.
Author:	The author of the use case.
Date:	The date the use case was last edited.

Appendix F Program of Record Technology or Initiative Transition Barriers/Enablers

This section is taken from Transitioning Agile/Rapid Acquisition Initiatives to the Warfighter (Duquette, Bloom, & Crawford, 2008, pp. 87 - 103).

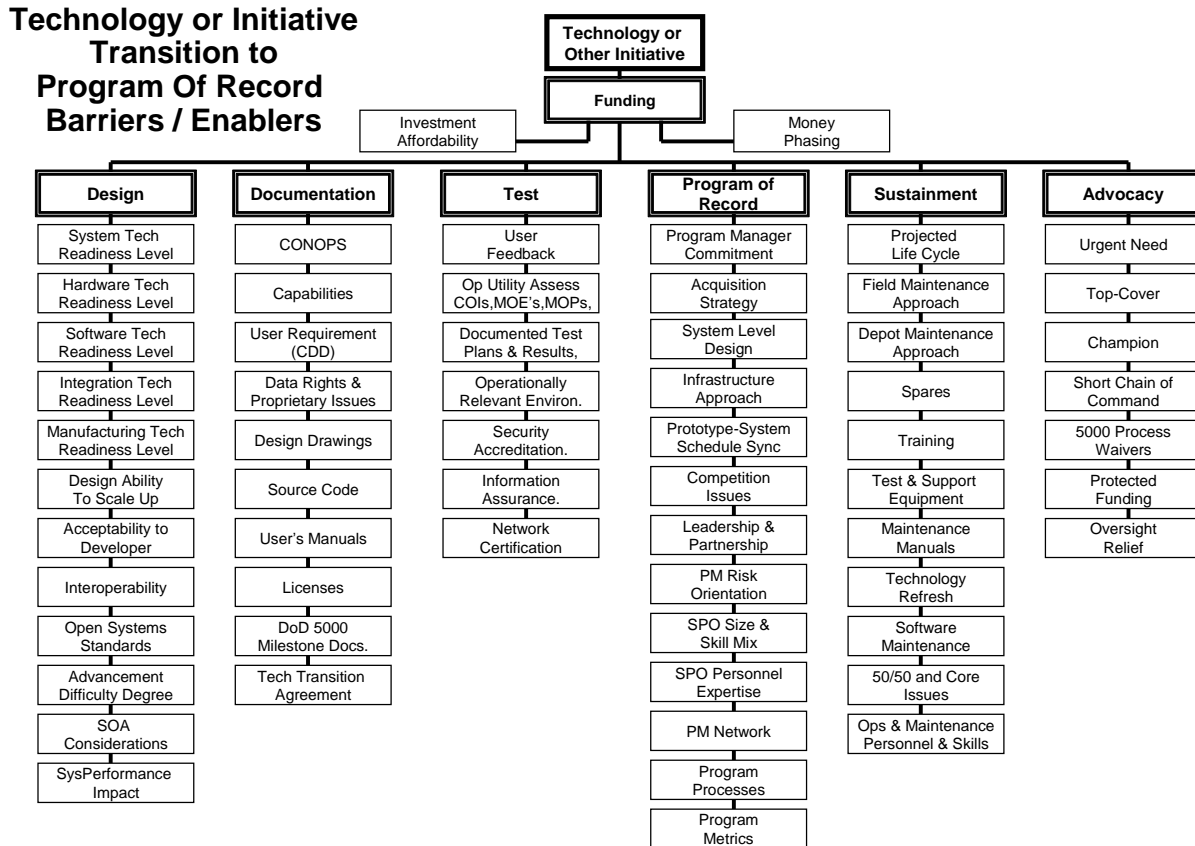


Figure G-11-1 Program of Record Technology or Initiative Transition Barriers/Enablers

F.1 Funding

In transitioning a technology or initiative to a program of record, consideration should be given to Investment Affordability and Money Phasing.

Investment Affordability is a significant concern in terms of competition for RDT&E (6.3 and 6.4), Procurement, and Operations and Maintenance funds. If you developed the technology with 6.3 RDT&E funds and need to continue development or improvement during or after transition to a program of record, you are competing with others wanting to do ATDs or JCTDs to prove operation in a relevant environment. When the program of record begins to use your initiative and maintain it, they are balancing the need to fund continued procurement and maintenance associated with the initiative with the fiscal needs of its baseline system. A successful business case for transition into a program of record will include a strong case for operational utility or the demonstration of some form of funding offset as a result of leveraging the initiative.

- **Enabler:** Clearly developed demonstration of strong operational utility, or alleviation of need for appropriations in same or related development, procurement, or operation.
- **Barrier:** Lack or inability to state distinguishing operational utility, or inability to balance expenditure for transition with savings / avoidance of cost elsewhere.

Money Phasing is also a significant transition concern. There may need to be an overlap in RDT&E 6.2 or 6.3 S&T funds and 6.4 R&D in order to manage a smooth transition from the S&T initiative manager to the R&D program manager. If it has not already been factored into the 6.4 R&D budget by the Program of Record sponsor (MAJCOM or Agency) then there may be “bridge funds” available through arrangements like the AF WRAP to carry the transition for a couple of years until it is funded in the Sponsor’s POM.

- **Enabler:** Clearly identified funded overlap, or access to bridge funds or other interim funding.
- **Barrier:** Lack of overlapping funding (if a program need), or failure to assess such a need in transition management.

F.2 Design

Design enablers and barriers in transitioning an initiative or technology into a program of record are covered in this section. They include the following:

A **System or Subsystem Technology Readiness Level 6** is required to transition into System Design and Development for a DOD 5000 program. A Technology Readiness Level 6 requires that the system or subsystem model or prototype has been demonstrated in a relevant environment.

- **Enabler:** Assessment showing transition subject matter is at or above system/subsystem TRL 6.
- **Barrier:** Assessment showing transition subject matter is below system/subsystem TRL 6. Also, failure to perform a system/subsystem technology readiness assessment.

A **Hardware Technology Readiness Level (TRL) 6⁴** is required to transition into System Design and Development (SDD) for a DOD 5000 program. A Hardware TRL 6 requires a representative model or prototype system, which is well beyond that of TRL 5, is tested in a relevant environment. This represents a major step up in a technology’s demonstrated readiness. Examples include testing a prototype in a high-fidelity laboratory environment or in a simulated operational environment.

- **Enabler:** Assessment showing transition subject matter is at or above hardware TRL 6.
- **Barrier:** Assessment showing transition subject matter is below hardware TRL 6. Also, failure to perform a hardware technology readiness assessment.

A **Software Technology Readiness Level (TRL) 6⁵** is required to transition into System Design and Development for a DOD 5000 program. A Software TRL 6 is the level at which the engineering feasibility of a software technology is demonstrated. This level extends to laboratory prototype implementations on full-scale realistic problems in which the software technology is partially integrated with existing hardware/software systems.

- **Enabler:** Assessment showing transition subject matter is at or above software TRL 6.

⁴ See DOD TRA Deskbook – May 05 <https://acc.dau.mil/CommunityBrowser.aspx?id=18545>

⁵ Ibid.

- **Barrier:** Assessment showing transition subject matter is below software TRL 6. Also, failure to perform a software technology readiness assessment.

An **Integration Readiness Level (IRL) 3**⁶ would show that there is compatibility between the technologies required for the system to orderly and efficiently integrate and interact. The IRL is a systematic measurement of the interfacing of compatible interactions for various technologies and the consistent comparison of the maturity between integration points.

- **Enabler:** Assessment showing transition subject matter is at or above IRL 3.
- **Barrier:** Assessment showing transition subject matter is below IRL 3. Also, failure to perform an integration readiness assessment.

A **Manufacturing Readiness Level (MRL) 6**⁷ would be required to transition into System Design and Development for a DOD 5000 program. A Manufacturing Readiness Level of 6 requires that the new technology has been demonstrated in a pre-production environment on design parts of the same level of complexity and using the same types of materials that would be used in the intended application. Appropriate quality levels have been achieved.

- **Enabler:** Assessment showing transition subject matter is at or above MRL 6.
- **Barrier:** Assessment showing transition subject matter is below MRL 6. Also, failure to perform a manufacturing readiness assessment.

Design Ability to Scale Up describes (in part) a system whose performance, after adding hardware, improves at least proportionally to the capacity added. This is said to be a scalable system. An [algorithm](#), design, [networking protocol](#), [program](#), or other system is said to **scale** if it is suitably efficient and practical when applied to broad situations (e.g. a large input data set or large number of participating nodes in the case of a distributed system). If the design fails when the quantity or scope increases then it does not [scale](#). If the initiative or prototype has never been tested with an operational load (or peak operational load), it is possible that the design will not scale, and instead, will result in adverse operational consequences.

- **Enabler:** Demonstration of ability to scale
- **Barrier:** Demonstrated weakness in scaling, or lack of a scaling evaluation.

Acceptability to Developer/Acquirer is the willingness of the prime contractor to the program of record to adopt a design solution from an external source and integrate it with the other hardware and software that the prime is responsible for developing and/or integrating.

- **Enabler:** Operations and maintenance information about the transition capability has been clearly communicated to developer, who with the leadership of the program of record have committed to supporting the updated capability.
- **Barrier:** The developer has adopted a negative view of the transition. Depending on context and complexity of the transition technology or its integration, failure to achieve an understanding with the developer could be a significant barrier.

Interoperability⁸ in the broadest sense is the ability of systems, units, or forces to provide services to and accept services from other systems, units, or forces, and to use the services so exchanged to enable them to operate effectively together. In this sense, the concern is how the

⁶ See Determining System Interoperability Using an Integration Readiness Level – Stevens Institute of Technology NDIA Presentation – 2006

⁷ See <https://acc.dau.mil/CommunityBrowser.aspx?id=18231>

⁸ See http://www.rand.org/pubs/monograph_reports/MR1235/MR1235.chap2.pdf

prototype or technology will improve or hinder the program of record in achieving its stated interoperability requirements.

- **Enabler:** Assessment (or demonstration) of interoperability impacts performed with favorable results.
- **Barrier:** Assessment (or demonstration) of interoperability impacts performed with negative results. Failure to consider interoperability impacts is likely a barrier.

Open Systems Standards⁹ are publicly-available documents defining specifications for interfaces, services, protocols, or data formats, established by consensus. If the target system is built using open systems standards and your initiative uses open systems interfaces, it is potentially easier to integrate your prototype or initiative with the target system.

- **Enabler:** Analysis or demonstration of similarity of open system infrastructures (both initiative and program of record)
- **Barrier:** Initiative not based on open system targeted to a program of record that leverages an open system, although analysis can prove that closed nature of initiative does not hinder integration or interoperability.

Advancement Degree of Difficulty (AD2) of about 6¹⁰ would indicate that a transition to system design and development is appropriate. AD2 is a NASA concept that measures how difficult it will be to advance the development of the technology in question to the point where it is “safe to proceed.” An AD2 degree of 6 indicates a 30% development risk where new development is required but similarity to existing experience is sufficient to warrant comparison across the board.

- **Enabler:** Assessment showing transition subject matter is at or above AD2 level 6.
- **Barrier:** Assessment showing transition subject matter is below AD2 level 6. Also, failure to perform an assessment of advancement degree of difficulty.

Service Oriented Architecture (SOA) Considerations According to SOA for Dummies, “SOA is about reuse. SOA is about taking what you have and structuring it in a way that allows you to not only continue to use it, but to use it secure in the knowledge that future change will be simple, straightforward, safe and fast.” If your target operational system is using SOA, the consideration is how your prototype will help or hinder that system, the network, and other systems that it supports with its SOA services.

- **Enablers and barriers** may be evaluated subjectively based on the facts and circumstances of your specific initiative and the program of record.

System Performance Impact If the program manager of the target system is depending on your prototype or initiative to meet his/her system key or critical operational performance objectives, the question is how well your present prototype of initiative meets those objectives. Even if the operational unit commander is not depending on your prototype or initiative to meet his/her key or critical performance objectives, it is still important to evaluate the extent to which the present prototype or initiative will help or hinder the unit’s operational performance.

- **Enabler:** Prototype or initiative demonstrated to meet expectations relative to impact on operational performance objectives. In the alternative, prototype / initiative is not relied

⁹ See <http://www.sei.cmu.edu/opensystems/faq.html>

¹⁰ See Mitigating the Adverse Impact of Technology Maturity, James Bilbro, [NASA] Feb 2007 http://pmchallenge.gsfc.nasa.gov/docs/2007Presentations/Presentations/Bilbro_James.pdf

upon to improve operational performance and analysis or demonstration has established that use of the initiative or prototype will not hinder operational performance.

- **Barrier:** Prototype / initiative is relied upon to improve operational performance, but assessment indicates a significant risk that use of the initiative or prototype will hinder operational performance. Also, failure to perform an assessment of the impact on operational objectives regardless of reliance factor.

F.3 Documentation

Documentation requirements for a program of record can be a show-stopper for the transition of technology or an initiative. The following documentation attributes and associated enablers and barriers should be considered.

CONOPS - What is the Concept of Operations (CONOPS) that the initiative or prototype supports? Is it referenced in the Initial Capability Document (ICD) or Capability Description Document (CDD) for the target program of record? If it is a new CONOPS has it been written and reviewed by the Joint Requirements Oversight Council (JROC)?¹¹

- **Enabler:** CONOPS supported by the initiative is consistent with / a subset of the CONOPS of the program of record.
- **Barrier:** There is significant potential that the CONOPS supported by the initiative is inconsistent with the CONOPS of the program of record. Also, failure to assess the potential conflict of differing CONOPS between initiative and program of record.

Capabilities – The issue is what Capabilities (e.g. JCS Joint Capability Areas (JCA) or Air Force Master Capabilities Library (MCL) capabilities) are supported by the prototype or initiative.

- **Enabler:** State JCA or MCL Item
- **Barrier:** Does not support either JCA or MCL item.

User Requirement – The question is what requirements in the current or draft capabilities development document¹², technical requirements document, or other user requirements document for the target program of record does the prototype or initiative satisfy.¹³

- **Enabler:** Initiative or prototype satisfies explicitly stated requirements.
- **Barrier:** Initiative or prototype does not correspond to a requirement, even a derived requirement.

Data Rights and Proprietary Issues¹⁴ - The focus is on restrictions on the data for your prototype or initiative that would prevent the target program of record from furnishing the data or design to their development contractor or integrator.

- **Enabler:** There are no such restrictions.
- **Barrier:** There are restrictions that may limit the ability of responsible contractors to the program of record to obtain data on the initiative relevant to operations, maintenance, or system evolution. Also, failure to establish certainty that there are no such restrictions.

¹¹ See CJCSM 3170.01C May 07 http://www.dtic.mil/cjcs_directives/cdata/unlimit/m317001.pdf

¹² See CJCSM 3170.01C May 07, http://www.dtic.mil/cjcs_directives/cdata/unlimit/m317001.pdf

¹³ See SEPO Requirements Toolkit, <http://sepo1.mitre.org/kits/requirements/index.html>

¹⁴ See FAR Part 27, <http://www.acquisition.gov/far/current/html/FARTOCP27.html>

Design Drawings¹⁵ - It may be important to have a design drawing package that would permit follow-on design or fabrication of the prototype or initiative by someone other than the original developer.

- **Enabler:** There is such a package.
- **Barrier:** There is no such package, and the timely procurement of such a package is not possible or economically feasible.

Source Code¹⁶ - This topic is concerned with the availability of source code to support software modification and maintenance of the prototype or initiative.

- **Enabler:** Source code is available.
- **Barrier:** Source code is not available, and the timely procurement of source code is not possible or economically feasible

User's Manuals¹⁷ - Key questions follow: Are user's or operator's manuals available for the operators of the prototype or initiative? Are the manuals consistent with the skill level of the operators of the target program of record system?

- **Enabler:** Properly documented manuals are available.
- **Barrier:** Properly documented manuals are not available, and the timely procurement of them is not possible or economically feasible.

Licenses¹⁸ - The issue is if the prototype or initiatives involve licenses that must be transferred to or renegotiated by the target program of record for development, use, or deployment to users.

- **Enabler:** The prototype or initiative does not rely on licenses. Or, the conveyance of any such licenses does not represent an administrative or economic burden to the program of record.
- **Barrier:** The prototype or initiative relies on licenses whose conveyance to program of record represents an administrative or economic burden.

DOD 5000 Milestone Documentation¹⁹ - The key question for milestone documentation is if you have either developed [for a new start program] or evaluated the impact [transition to an existing program of record] the regulatory and statutory information requirements for milestone B or C [depending on the technology insertion point].

- **Enabler:** The acquisition documentation is developed, or is partially developed, and it is likely the documentation will be ready within a prescribed schedule.
- **Barrier:** The acquisition documentation is not developed, nor is there a plan in place, properly resourced, to ensure the documentation will be prepared in a timely manner.

Technology Transition Agreement²⁰ - The question is if there is a negotiated Technology Transition Agreement (TTA) with the target program of record.

¹⁵ See ASME Y14.24-1999 – Table A-1,

http://catalog.asme.org/Codes/PrintBook/Y1424_1999_Types_Applications.cfm

¹⁶ See IEEE/EIA 12207.1-1997, http://standards.ieee.org/reading/ieee/std_public/description/se/12207.1-1997_desc.html

¹⁷ Ibid.

¹⁸ See FAR Part 12, <http://www.arnet.gov/far/current/html/FARTOCP12.html> and FAR Part 27 <http://www.acquisition.gov/far/current/html/FARTOCP27.html>

¹⁹ See DODI 5000.2, Enclosure 3, May 12, 2003, <https://akss.dau.mil/dag/DOD5002/Subject.asp>

²⁰ See *DOD Technology Readiness Assessment [TRA] Deskbook, Appendix G, May 05.* <https://acc.dau.mil/CommunityBrowser.aspx?id=18545>

- **Enabler:** A TTA has been successfully coordinated.
- **Barrier:** A TTA has not been successfully coordinated, nor is one likely to be completed in a timeframe consistent with the successful transition of the initiative.

F.4 Test

Test requirements for a program of record are significant. The following are test attribute enablers and barriers that should be considered in planning for the transition of a technology or initiative.

User Feedback – Key questions are as follows: Have you demonstrated the initiative or prototype for the intended users in a relevant or operational environment and recorded their feedback? Have you altered the design of the initiative or prototype based on the user feedback? Has the user requested changes that you have not made? Are the users providing the feedback involved in requirements and/or budgeting for their organization?

- **Enabler:** User feedback (including key decision makers or designee, and those who determine budget recommendations) based on demonstration or use in a relevant or operational environment has been obtained and as appropriate applied to refine the design and/or implementation of the initiative.
- **Barrier:** Such feedback has not been obtained, or the feedback indicates a lack of interest in furthering the initiative, or actionable feedback was obtained but not applied to refine the design or implementation.

Operational Utility Assessment COIs, MOEs, MOPs - It is desirable to conduct an Operational Utility Assessment with a structured set of Critical Operational Issues (COIs), Measures of Effectiveness (MOEs), and Measures of Performance (MOPs). Positive, documented results should be contrasted with any deficiencies or recommendations for improvement.

- **Enabler:** An operational utility assessment was conducted, with minimal deficiencies and actionable recommendations.
- **Barrier:** An operational utility assessment revealed significant deficiencies, or such an assessment was not performed.

Documented Test Plans and Results - The plans used for testing or demonstrating the prototype or initiative were well documented and include specific objectives, measurements, and/or expectations. The results of the test or demonstration were well documented and reviewed/validated with the operational users following test or demonstration completion.

- **Enabler:** Test plans and results were well documented and properly reviewed.
- **Barrier:** Test plans or results were not completely documented, and their review was limited.

Security Accreditation²¹ - If the initiative or prototype processes classified information, has it been accredited for use in an operational environment? A DOD Information Assurance Certification and Accreditation Process (DIACAP) Authority To Operate (ATO) is an authorization granted by a Designated Approving Authority (DAA) for a DOD intelligence system to process, store, or transmit information. An ATO indicates a DOD intelligence system has adequately implemented all assigned information assurance controls to the point where residual risk is acceptable to the DAA. ATOs may be issued for up to 3 years.

- **Enabler:** All ATOs have been granted.

²¹ See DODI 8510.01, November 28, 2007, <https://acc.dau.mil/CommunityBrowser.aspx?id=186871>

- **Barrier:** At least one ATO has not been issued, and there is an indefinite or unacceptable delay associated with its issuance.

Information Assurance²² - Mission Assurance Category (MAC) I, II, or III and Confidentiality Level (Classified, Sensitive, or Public) have been designated in the User Requirements document and tested.

- **Enabler:** MAC and confidentiality levels have been designated, and testing is both complete and satisfactory.
- **Barrier:** A MAC or confidentiality level has been designated, or testing is either incomplete or unsatisfactory.

Network Certification²³ - The system been certified as Net-Ready (if applicable) by Joint Interoperability Test Command or been given an Interim Certificate to Operate (ICTO).

- **Enabler:** The prototype or initiative has been certified net-ready or given an ICTO.
- **Barrier:** The operational system has a net-ready requirement, and the prototype or initiative has not passed certification testing or has not undergone such testing.

F.5 Program of Record

The following are attributes of the program of record that can be enablers or barriers to the transition of a technology or initiative.

Program Manager Commitment - The Program Manager (intended receiver of the technology or initiative) has signed a Technology Transition Agreement²⁴ (TTA) (or other Memorandum of Agreement(MOA)) with the Science and Technology developer to develop, deliver and integrate the technology or initiative into an acquisition program.

- **Enabler:** A TTA or MOA has been signed.
- **Barrier:** There is no compelling evidence indicating a commitment to the transition on the part of the program manager.

Acquisition Strategy - The program receiving the initiative has an acquisition strategy that will permit the transition of the technology or initiative to the program of record.

- **Enabler:** The acquisition strategy is in place and the transition is incorporated in the strategy or is well positioned to be incorporated.
- **Barrier:** The transition appears poorly positioned relative to the acquisition strategy of the operational unit, or no strategy has been developed.

System Level Design - The system receiving the initiative has a system design that is capable of accepting the technology or initiative without major rework.

- **Enabler:** Rework to ensure design consistency is minimal at most.
- **Barrier:** Rework to ensure design consistency is or may be significant, and that level of rework is unsatisfactory to the operational unit, or such information is unknown but may be significant to those deciding on or overseeing the transition.

²² See DODI 8500.2, 6 February 2003, http://fas.org/irp/doddir/dod/d8500_2.pdf

²³ See CJCSI 6212.01D, 8 March 2006, Current as of 14 March 07, http://www.dtic.mil/cjcs_directives/cdata/unlimit/6212_01.pdf

²⁴ See *DOD Technology Readiness Assessment [TRA] Deskbook*, <https://acc.dau.mil/CommunityBrowser.aspx?id=18545>

Infrastructure Approach - The infrastructure approach (e.g. power, network) for the system receiving the initiative is consistent or compatible with the technology or initiative being transferred.

- **Enabler:** The infrastructure approaches are consistent.
- **Barrier:** The infrastructure approaches are not consistent, or their relative consistency is undetermined.

Prototype-System Schedule Synchronization - The technology or initiative is ready for transition at a point in time where it can be synchronized with the requirements or early design stage of the receiving program or other technology insertion point (pre-production or sustainment modification).

- **Enabler:** Schedule synchronization is established and planning for the transition is complete or considered non-problematic.
- **Barrier:** Schedules do not synchronize, and the inherent delay in transition is unacceptable or significantly reduces the value of the initiative. Or, synchronization has not been evaluated.

Competition Issues - There are no competition issues that would prevent the technology or initiative design from being given to a development contractor for incorporation in the system without redesign, or a Justification and Approval (Federal Acquisition Regulation (FAR) Part 6²⁵) has been approved. One issue is if the contractor involved in development/ implementation of the initiative or prototype has a defined contractual right pertaining to the continued production or the maintenance of the initiative that is in conflict with the acquisition approach in use by the program of record.

- **Enabler/Barrier:** There are no / there are competition issues.

Leadership and Partnership - Leadership is generally charismatic, dynamic, and hard-charging. This is not to imply ad hoc leadership. Leaders in these circumstances have a clear understanding of the goal and a success orientation that infects the entire team. Motivation, innovation, and inspiration are the methods to obtain the maximum from the program team. Partnership is the willingness of the Program Manager to partner with the S&T Manager to achieve the transition.

- **Enabler:** These attributes are in evidence in both the initiative and the program of record.
- **Barrier:** The opposite of these attributes have been demonstrated by either the S&T management or the program of record.

Program Manager Risk Orientation - Liberal Risk Orientation of management indicated a willingness to take risk or seize upon opportunities or "game changing" technologies or innovative approaches. While the team does not "gamble" the team does not have "risk aversion."

- **Enabler:** Program of record leadership has demonstrated well-reasoned risk taking.
- **Barrier:** Program of record leadership has demonstrated that it is strongly risk averse.

Program Office Size and Skill Mix - The program team is of the Right Size and the Right Mix of qualified individuals to optimize communication and independent action. Each member of the team is "self starting" and to a large extent "self directed." Communications are often optimized because of the "like mindedness" of the program team members. Respect, trust, and teamwork are characteristic of these teams.

²⁵ See <http://www.acquisition.gov/far/current/html/FARTOCP06.html>

- **Enabler:** The program of record can be characterized as having right-sizing and right-mix of staff and it appears this will continue to be the case indefinitely.
- **Barrier:** The program of record cannot be characterized as having right-sizing and right-mix of staff.

Program Office Personnel Expertise - Expertise is valued and given the freedom to make critical and authoritative program decisions. A "value added" versus an organizational hierarchy is in place for decision making. Individual experts have established credibility through a proven "track record" of sound decision making. They are sought out to participate in the most critical of program decisions.

- **Enabler:** The program of record has sufficient expertise in the initiative or technology to integrate it and operate with it.
- **Barrier:** The program of record has insufficient expertise.

Program Manager's Network - The program leadership and team's interpersonal networks are leveraged to obtain advantageous results in transition.

- **Enabler:** There is evidence that such networks have been and will continue to be leveraged in ways salient to the transition.
- **Barrier:** Limited to no networking has been demonstrated by the leadership of the program of record.

Program Processes - The Internal Program Office Business Processes are integrated, predictable, responsive, and well understood by all team members. The program makes an ally of process and uses it to bind the team together and raise efficiency.

- **Enabler:** The use of the new initiative or technology can be easily integrated into existing operational business processes.
- **Barrier:** There may be issues with the integration of the initiative into existing business processes, or such processes may not be in place.

Program Metrics - Program Office Agility/Rapidity is sustained and monitored through the collection of a small but critical set of Program Metrics.

- **Enabler:** The introduction of the initiative or prototype has a positive influence on the program of record's performance metrics.
- **Barrier:** The initiative has a negative influence on the program of record's performance metrics.

F.6 Sustainment

Generally in a program of record sustainment planning is a deliberate activity. In transitioning a technology or initiative the following sustainment attributes and associated enablers and barriers should be considered.

Projected Life Cycle - The projected life cycle of the technology or initiative is long enough (more than 5 to 10 years) to warrant inclusion in a program of record rather than: (1) throwing the item away and replacing it with a newer version when it breaks; (2) handling sustainment with a "command support" contract; or (3) relying on the S&T developer to sustain it.

- **Enabler:** The projected life cycle of the initiative is long enough that it makes sense for a program of record to assume the responsibility of its sustainment.

- **Barrier:** The projected life cycle of the initiative is of such potential brevity that it does not make sense for a program of record to assume the responsibility of its sustainment.

Field Maintenance Approach - A field (operator, organizational, and intermediate) maintenance approach exists or is not required (e.g. throw-away or warranty or return to depot) for the technology or initiative.

- **Enabler:** A maintenance approach consistent with the initiative exists, or field maintenance is not required.
- **Barrier:** Field maintenance is required and the maintenance approach of the program of record is inconsistent with the initiative.

Depot Maintenance Approach - A depot maintenance approach exists that is compliant with Title 10 50/50 and Core provisions or is not required (e.g. commercial item warranty or throw-away).

- **Enabler:** As described above.
- **Barrier:** There is no depot maintenance system available, but the initiative is not supported by warranty, nor is it managed as a throw-away item.

Spares - If the technology or initiative is or includes hardware, spares have been identified and are available to support maintenance at the specified maintenance levels (operator, organizational, intermediate, and/or depot) or they are not required (e.g. commercial item warranty or throw-away).

- **Enabler:** As described above.
- **Barrier:** Spares are required, but they are not available, or they are available but are not composed in a manner consistent with the maintenance level applied by the program of record.

Training - Operator training and training for organizational, intermediate and depot maintenance has been conducted and continues to be available for both operator and maintenance at skill levels appropriate for the designated military or civil service specialty codes.

- **Enabler:** As described above.
- **Barrier:** Training is required, but it has not been conducted. Furthermore, there is some significant obstacle to the development of training materials and the performance of initial training consistent with the needs of the program of record.

Test and Support Equipment - Any unique test and support equipment (and associated software) that was developed to support the technology or initiative is available and its design, operation, and maintenance are documented.

- **Enabler:** As described above.
- **Barrier:** Such support equipment is likely to be required, but it is not developed, and there is some significant obstacle to its timely procurement. Or, available support equipment is unsupported by documentation to such an extent that its continued use in sustainment is impaired.

Maintenance Manuals – Hardware, software, and other maintenance manuals have been developed, verified, and validated. They are up to date and available for maintenance at skill levels appropriate for the designated military or civil service specialty codes.

- **Enabler:** As described above.

- **Barrier:** Manuals have not been developed, and there is some obstacle to their timely development. Or, available manuals are not consistent with the skill levels appropriate for the designated military or civil service specialty codes.

Technology Refresh - A technology roadmap has been developed for the technology or initiative. Recommended technology refresh cycles have been established and are consistent with the maintenance and spares approaches.

- **Enabler:** As described above.
- **Barrier:** Recommended technology refresh cycles are not consistent with the maintenance and spares approaches of the program of record. Or, assessment of technology refresh of the initiative/prototype has not been performed, and it is of a complexity that indicates failure to perform this assessment represents an unacceptable risk to the transition.

Software Maintenance - Provisions have been made for software maintenance including documentation, assignment of responsibility, source code, programmer's manuals, development/maintenance hardware, software and tools. Software maintenance provisions are consistent with the skill level of the designated software maintenance organization.

- **Enabler:** As described above.
- **Barrier:** Appropriate provisions for software maintenance have not been made, and sustainment of the initiative or prototype includes software maintenance.

50/50 and Core Issues - The introduction of the technology or initiative does not impact Depot Source of Repair decisions for the program receiving the technology or initiative.

- **Enabler:** As described above.
- **Barrier:** The transition of the initiative into the program of record will or might plausibly represent a conflict with pre-established Depot Source of Repair decisions.

Operations and Maintenance Personnel and Skills - The skill levels and specialty codes for operating and maintaining the technology or initiative have been specified and are assigned in sufficient numbers to the operational units which will be operating and maintaining the receiving system.

- **Enabler:** As described above.
- **Barrier:** The skill levels and specialty codes have not been identified, or they are inconsistent with the approach used by the program of record.

F.7 Advocacy

The following advocacy attributes and the associated enablers and barrier should be considered in planning the transition of a technology or initiative.

Urgent Need - An Urgent Need exists. This does not necessarily indicate that a rapid acquisition is needed; it could be a vital or necessary element missing from the current military set of capabilities.

- **Enabler:** There is a documented and coordinated level of urgency.
- **Barrier:** Urgency is not a motivating factor in the transition of the initiative.

Top Cover - Top Cover is a condition of protection provide to the program management and program team. This top cover is usually provided by senior or political leaders that have an interest in the program.

- **Enabler:** The initiative or the program of record has a demonstrated, effective level of top cover.
- **Barrier:** Neither the initiative nor the program of record in the context of this transition can be characterized as having top cover.

Champion - A Champion for the effort exists. It may be the same person(s) providing top cover; however it could be another person in the chain-of-command or someone that has strong political, social, technological, economic, or operational needs and leverage. Unlike top cover whose support may be passive in nature, the champion is active in supporting the effort and keeps the initiative on the front burner.

- **Enabler:** As described above.
- **Barrier:** There is no champion for the initiative.

Short Chain of Command - Short Chain of Command allows quick decisions and eliminated dilution of decisions. Often the program leader is a direct report to the final decision maker. In these cases senior decisions make themselves available to the program team.

- **Enabler:** With respect to the transition of the initiative, there is a short chain of command.
- **Barrier:** There are multiple levels of decision making, influence, and approval in the chain of command in the context of this transition.

DOD 5000 Process Waivers or By-pass - By-passing process/rules/regulations etc. is usually a part of the Agile/rapid environment. This could consist of either finding a path through existing process, rules, regulations, or law that allows for exception. It could also be accomplished by requesting and receiving a waiver or deviation (e.g. skunk-works, classified program, etc.).

- **Enabler:** Significant acquisition process and documentation overhead does not exist, or, there is a robust set of by-pass and waiver opportunities available to the initiative and its transition into the program of record.
- **Barrier:** Significant acquisition process and documentation overhead exist, and there are no foreseeable by-pass and waiver opportunities.

Protected Funding - Predictable funding flow is obtained (developed or designed by the team) and used to ensure appropriate planning and execution. The program champion(s) often play a big role in this area.

- **Enabler:** As described above.
- **Barrier:** Funding is not protected, and this vulnerability affects the planning and implementation of the transition (e.g., inability to secure contractual support of a desired duration).

Oversight Relief - Often, those responsible for Agile/rapid technology or initiatives are not comfortable with higher levels of management oversight and management and find ways to avoid it. Programs of this nature seek out methods to shelter the program team. This is accomplished through the insulation provided by the program leader, classification of the program, proactive anticipation of the needs of the HQ, or assuming a posture of "it is easier to apologize than ask permission".

- **Enabler:** Oversight relief is unnecessary, or the means to achieve relief are identified and dependable.
- **Barrier:** Oversight relief is necessary, and the means to achieve relief are not identified.