A Case Study in Applying Agile Software Engineering Practices to Systems Engineering

by

Matthew Ryan Kennedy

A dissertation submitted to the Graduate Faculty of
Auburn University
in partial fulfillment of the
requirements for the Degree of
Doctor of Philosophy

Auburn, Alabama
May 04, 2013

Keywords: Agile, System Engineering, Scrum, Software Engineering, Lean, Process

Approved by
David Umphress, Chair, Associate Professor of Computer Science and Software Engineering
Drew Hamilton, Professor of Computer Science and Software Engineering
Robert Lord, Professor of Systems Engineering

Abstract

With the fast-paced nature of technology, the need to rapidly field systems has never been more important. Success does not just depend on well-defined requirements but also on the ability to respond to change during and after deployment. The inability to rapidly respond to change may cause the system to become obsolete before initial fielding. Creating a structure where processes allow for changes to occur during system development requires a restructuring of system development values and practices at all levels. This paper addresses the progress toward agility and defines the agile values and practices being used by agile organizations in both the Business and Software Aspects. It provides an Agile System Engineering Framework and guiding Practices to increase agility in the system engineering process. A case study was conducted using the described Agile Systems Engineering Framework and Practices and found that incorporation of the Framework and Practices in the Systems Aspect improved cost, schedule and functionality estimates.

Acknowledgments

Writing this dissertation is the most significant academic challenge I have had to face. Its completion would not have been possible without the knowledge, support and leadership of many people and organizations. First and foremost I would like to thank my advisor, Dr. David Umphress, for his guidance and leadership throughout this process. Without his subject matter expertise this dissertation would not have been possible. I would like to thank my committee, Dr. Drew Hamilton and Dr. Robert Lord, who offered support which was instrumental in the completion of this dissertation.

Thank you to the corporation that allowed me to apply the agile Practices and Framework on a high profile project.

Table of Contents

List of Figures

List of Tables

Chapter 1: Introduction[1]

Today's systems are increasingly threatened by unanticipated change arising from volatility in user requirements, Information Technology (IT) refresh rates and responses to security vulnerabilities. With the rapidly changing world of IT, long static development cycles of a Software Intensive System (SIS)—"a system in which software represents the largest segment in one or more of the following criteria: system development cost, system development risk, system functionality, or development time" (Hagan, 2011, pp. B-248)—may doom the system before development begins.

Delivering a SIS that is on time, within budget and on schedule with traditional systems processes has been problematic (Group, 2009, p. 1). This problem will only increase as the complexity of SISs within the Department of Defense (DoD) grows and more functionality within systems is relegated to software (Force, 2009, pp. viii, 2), (Ferguson, 2001, p. 105).

Traditional systems engineering portrays systems development as a top-down, waterfall-centric process; one that relies on explicating requirements as early as possible. Such a perspective tends to postpone modifications until the maintenance phase (Center, 2003, pp. 2-5), thus thwarting early insertion of technology or a nimble response to changes in user needs. Though the technology refresh rate varies from system to system, a report from the State of Michigan shows the following industry computer technology refresh trends:

[1] Portions of this chapter were published in the May/June 2011issue of CrossTalk Magazine (Kennedy & Umphress, 2011, pp. 16-20)

1) 40% of companies are on a 4-year cycle for refreshing personal computers (hardware) and
2) Microsoft plans a 2-year cycle to release a new operating system (software) (Government, 2007, p. 1)

A report from the U.S. Army War College estimates that commercial electronics have a typical refresh rate of twelve to eighteen months, but may be less (Daniels, 2006, p. 7).

Cyber security further complicates the picture. The rate at which vulnerabilities are identified in a system cannot be predicted. According to the National Vulnerability Database (NVD), between 2000 and 2009 there was an average of 3,825 vulnerabilities reported each year due to software flaws alone (N. I. o. S. a. Technology, pp. 1-2). The need for a responsive systems engineering process to rapidly address unforeseen vulnerabilities is imperative for the development of a secure system.

The Office of the Assistant Secretary of Defense (OASD) for Network Information Integration (NII) conducted an analysis of thirty-two major information system acquisitions and found the average time to deliver the Initial Operating Capability (IOC) was ninety-one months (Force, 2009, p. 36). With the DoD's history of long delivery cycles and the short time required for technology refresh, the systems engineering process needs to be responsive to changes introduced both by the user and technology.

This inability to respond rapidly to change is nothing new. Software engineering recognized the pitfalls of a strictly sequential development process a number of years ago. The contemporary school of thought in software engineering has evolved away from considering a waterfall approach as the primary sequence of development activities and toward approaches

which embrace change by segmenting software development into manageable change-resistant increments and allowing change to take place at increment boundaries. Ultra-modern approaches— known as "agile" processes— have emerged to match the pace in which change is encountered during software development. Agility is “the speed of operations within an organization and speed in responding to customers (reduced cycle times)” (M. I. o. Technology). The degree of agility when developing an IT system is the organization’s ability to respond to changing requirements and technology. With the quick technology refresh rate, long development cycles could place a system in a state of obsolescence prior to initial release. With the ever-changing world of technology, the need to change without notice throughout the development lifecycle is paramount to success.

Just as the software community has moved toward a more agile approach to become more responsive to changes throughout the development lifecycle, the systems engineering community needs to follow a similar approach to remain competitive in today’s rapidly changing environment.

Past Performance

Failure to deliver a successful SIS can rarely be attributed to one project deficiency; however, the inability to rapidly adapt to change appears to be an underlying theme in many SIS development failures. A successful SIS is defined as a system that is on time, within budget and contains all of the required features and functions (Group, 2009). Instead of steadily making improvements on the successful delivery of SISs, the Standish Group 2009 Chaos Report showed a “marked decrease in project success rates,” in which only 32% of projects were successfully delivered when compared to the 35% reported in their 2006 report (Group, 2009).

The US Government Accountability Office (GAO), an "independent, nonpartisan agency that works for Congress", investigates how the government spends taxpayer's dollars (Government, 2007). The Air Force is developing an F-22 Raptor aircraft that is intended to provide increased capabilities over the current aircraft. A GAO report found the program had undergone several changes since the development began in 1986, and the Air Force cannot afford to purchase the quantities of the aircraft that were initially anticipated. This was partially attributed to the Air Force adding more robust air-to-ground attack requirements in 2002. In addition to the change in requirements, the Air Force determined that revised computing architecture as well as new computer processors were needed to support planned enhancements, both of which further increased program costs (Office, 2004, p. 5). Previous experience shows that changes within a SIS are inevitable, whether there is a change in requirements or technology. Though predicting these changes may be difficult, processes can be structured to be more responsive to these unanticipated changes. Increasing agility within the systems engineering process is one mechanism that may result in increasing the successful delivery of a SIS.

Growth of SISs

The amount of software within today's systems is increasing. Examining the correlation between the Executable Software Lines of Code (ESLOC) and time in various DoD systems (Figure 1) shows a steady increase in ESLOC in related systems over time. The Aegis system introduced in the early 1980s had less than 2 million ESLOC. The Virginia SSN introduced roughly twenty years later contained over double the ESLOC and the estimated ESLOC for the DDX system is just under 10 million.

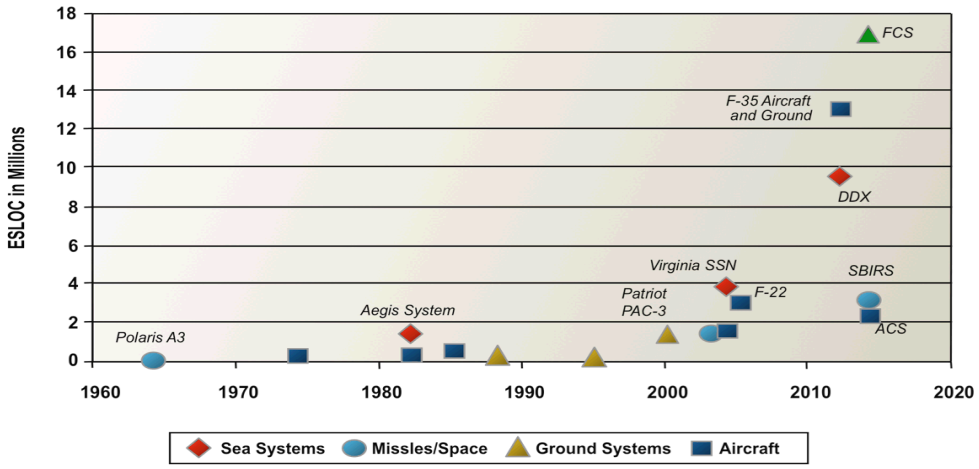


Figure 1 Increase in Software in DoD Systems (Force, 2009)

The increase in ESLOC means more of the system's functionality is being performed by software. Functions performed by software in DoD aircraft (Figure 2) has increased from 8 percent for the F-4 Phantom II in 1960 to 80 percent for the F-22 Raptor in 2000. With the proliferation of software within current systems, problems that were inherently software are evolving into system problems (Ferguson, 2001, p. 107). The issues of both system complexity and agility is not only recognized within the United States DoD but have been identified in the United Kingdom's Ministry of Defense as some of the "next great systems thinking challenges" (Oxenham, 2010, p. 1) .

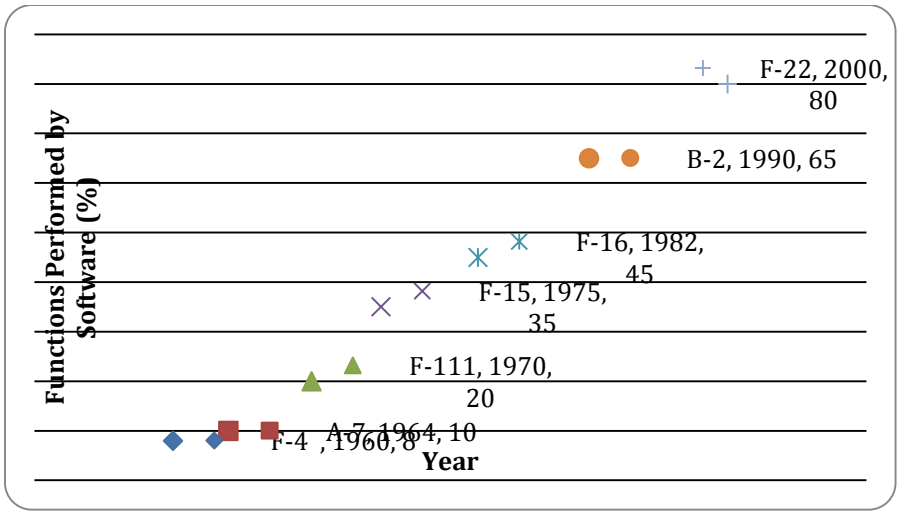


Figure 2 Functions Performed by Software (Nelson & Clark, 1999, p. 55)

Defense systems are not the only systems experiencing an increase in software, the automotive industry has also seen an increase. In 1977 General Motors' (GM) Oldsmobile Toronado contained the first productive microcomputer Electronic Control Unit (ECU) used for only electronic spark timing (Charette, 2009). Just a year later, the Cadillac Seville offered a software-driven display of speed, fuel, trip and engine information on its *Cadillac Trip Computer* (Charette, 2009). By 1981, GM was using microprocessor-based engine controls executing roughly 50,000 Software Lines of Code (SLOC). Today it is estimated that a premium automobile takes dozens of microprocessors running 100 million SLOC (Charette, 2009).

When determining the impact of software on overall system cost, Broy notes that "the cost of software and electronics can reach 35 to 40 percent of the cost of a car" (Charette, 2009). A study conducted by the Center for Automotive Research had similar findings (Trage, 2005):

> "Software made up only 16 percent of a vehicle's total value in 1990, this figure had increased to 25 percent by 2001. By 2010, their share of a car's total value is expected to climb to almost 40 percent."

The inability to deliver a successful SIS will only be exacerbated as software continues to become an increased portion of a system's composition.

Summary

The rapid technology refresh rate coupled with the need to respond to changing requirements requires a complete agile systems development process. The increase in software within today's systems only increases the need for an agile systems engineering process.

Chapter 2: Literature Review[2]

SIS Development

Development of a SIS can be envisioned as an amalgamation of three aspects: Business, System and Software. Though there is some overlap among these aspects, general responsibilities can be attributed to each one individually.

The Business Aspect is responsible for the acquisition of the system as a whole including contracting, funding, operational requirements and overall system delivery structure. The System Aspect is responsible for the general technical and technical management aspects of the system and serves as the interface between management and engineers. The Software Aspect is responsible for the software items contained in the SIS.

When developing a SIS, all three aspects need to work in harmony to produce a successful final product. Traditionally, when using a once-through development methodology, the Business Aspect would provide funding and operational requirements to the System Aspect. The System Aspect would further decompose the requirements and allocate them to software or hardware. These items would then be developed and integrated resulting in a completed system. Given that major information systems average a ninety-one month gap from operational requirements definition to system delivery, defining requirements that far in advance of technology that is changing every twelve to eighteen months suggests that the end result will not be an up-to-date system.

[2] Portions of this chapter were published in the July 2012 issue of the Defense Acquisition Research Journal (Kennedy & Ward, 2012, pp. 249-264)

Business Aspect

The Business Aspect is where operational requirements are realized and the strategy for overall system development is identified. Currently, the Department of Defense (DoD) uses DoD Instruction (DoDI) 5000.02 to manage how they will perform the acquisition of weapon systems, services and automated information systems (AISs) (Defense, 2008, pp. 1-80).

Realizing the current DoDI 5000.02 was not responsive to the changing needs of technology, Congress signed the Fiscal Year 2010 National Defense Authorization Act (NDAA), which directed the Secretary of Defense to "develop and implement a new acquisition process for information technology systems" ("NDAA 2010," 2009, p. 214). This new Defense Acquisition System (DAS) process must include ("NDAA 2010," 2009, p. 214):

1) Early and continual involvement of the user;
2) Multiple, rapidly executed increments or releases of capability;
3) Early, successive prototyping to support an evolutionary approach; and
4) A modular, open-systems approach.

Moreover, this process should be based on the March 2009 report of the Defense Science Board (DSB) Task Force on Department of Defense Policies and Procedures for the Acquisition of Information Technology ("NDAA 2010," 2009, p. 214). The DSB report concluded that:

> The conventional DOD acquisition process is too long and too cumbersome to fit the needs of the many IT systems that require continuous changes and upgrades (Force, 2009, p. xi).

The report noted that an agile acquisition approach would increase IT capability and program predictability, reduce cost and decrease cycle time.

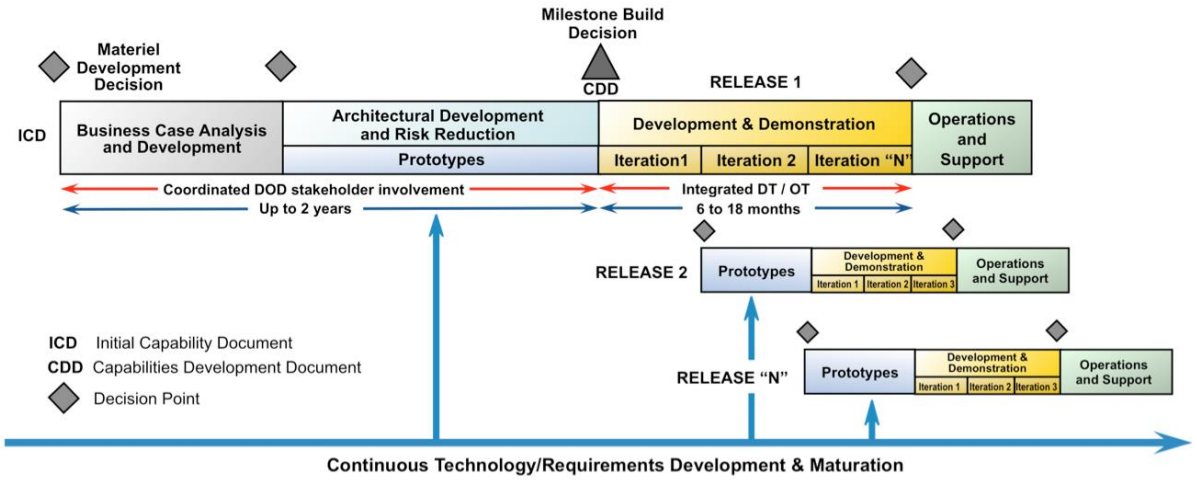


Figure 3 Agile Business Aspect Framework (Force, 2009, p. 48)

The DSB has developed an Agile Business Aspect framework (Figure 3) which is divided into four phases (Force, 2009, pp. 48-49):

1) Business Case Analysis and Development: "Establish the need for the proposed capability and develop the concept for the proposed solution and perform a cost benefit analysis to quantify the benefits of the solution."
2) Architectural Development and Risk Reduction: "The core architecture is built and architecturally significant features demonstrated. Prototyping begins during this phase and continues throughout the acquisition life cycle to assess the viability of technologies and minimize high-risk features."
3) Development and Demonstration: "The period when operational capability is built and delivered for a discrete number of releases. Capabilities are prioritized and parsed into groupings to establish release baselines for the sub-programs. Includes

development of training programs and testing in realistic environments to ensure successful fielding of new capabilities."

4) Operations and Support: "Provides materiel readiness, user training and operational support over the total program life cycle".

In addition to the emerging IT Acquisition Framework, the DoD developed an agile requirements process for IT Systems called the "IT Box" (*Manual for the Operation of the Joint Capabilities Integration and Development System*, 2012, p. B17). The Joint Requirements Oversight Council Memorandum (JROCM) 008-08 stated, "IT programs are dynamic in nature and have, on average, produced improvements in performance every 12-18 months" (JROC, 2009). Recognizing the need for performance improvements, the "IT Box" allows IT programs the flexibility to incorporate evolving technologies. This allows for greater agility in the current DoD requirements process (*Manual for the Operation of the Joint Capabilities Integration and Development System*, 2009, p. C1).

To be used in conjunction with the framework, there is a guiding value set called FIST (Fast, Inexpensive, Simple and Tiny), which may be utilized throughout the process (Ward, 2010, pp. 31-32). The FIST approach identifies a set of priorities and preferences that should be employed by project leaders during the development process (Ward, 2010, pp. 31-32). These values are declared in *The FIST Manifesto* as:

1) Talent trumps process;

2) Teamwork trumps paperwork;

3) Leadership trumps management;

4) Trust trumps oversight.

The FIST Manifesto also contains a series of principles and implementation guidelines. These principles are (Lapham, Williams, Hammons, Burton, & Schenker, 2010, pp. 56-57):

1) A project leader's influence is inversely proportional to the project's budget and schedule;
2) Constraints foster creativity;
3) Fixed funding and floating requirements are better than fixed requirements and floating funding;
4) An optimal failure costs a little and teaches a lot;
5) Complexity is cost, Complexity reduces reliability, Simplicity scales, Complexity does not;
6) Iteration drives learning, discovery and efficiency.

The implementation guidelines are (Lapham et al., 2010, pp. 56-57):

1) Minimize team size and maximize team talent;
2) Use schedules and budgets to constrain the design;
3) Insist on simplicity in organizations, processes and technology;
4) Incentivize and reward under runs;
5) Requirements must be achievable within short time horizons;
6) Designs must only include mature technologies;
7) Documents and meetings: have as many as necessary, as few as possible;

8) Delivering useful capabilities is the only measure of success.

The FIST approach has been successfully utilized on various DoD programs including the Marine Corps "Harvest Hawk," which incorporated a gunship modification onto a C-130 airframe. This modification was fielded just eighteen months after the program was announced (Axe, 2010). This is a divergence from the typical decade-long weapons system program and shows the DoD can deliver systems on short timelines.

The Business Aspect is making advancements toward becoming more agile and adaptive to changing requirements which is required to keep pace with today's rapidly changing environment.

Software Aspect

Software engineering has standards available such as ISO 12207, which "contains processes, activities and tasks that are to be applied during the acquisition of a system that contains software" ("Systems and software engineering -- Software life cycle processes - Redline," 2008, p. xiii). A limitation identified within ISO 12207 is that it does not specify details on how to implement the identified activities or tasks ("Systems and software engineering -- Software life cycle processes - Redline," 2008, p. 1). Both ISO 9001 and CMMI (discussed in more detail in the "System Aspect" Section) can be applied to the software engineering portion of the system, but they only provide *what* needs to be done and not *how* it needs to be accomplished.

Software development has been on a continuous process improvement track for decades. Initially the waterfall software development methodology was used, where software was

developed in one long release cycle (Royce, 1970, p. 330), as shown in Figure 4. The waterfall software development methodology provides the fundamental steps required to develop software. However, it has one major flaw in that it assumes that once the requirements process is complete, the requirements will remain unchanged throughout the development lifecycle. This assumption rarely holds true in practice, as change is inevitable in all large software projects (Sommerville, 2004, p. 71).

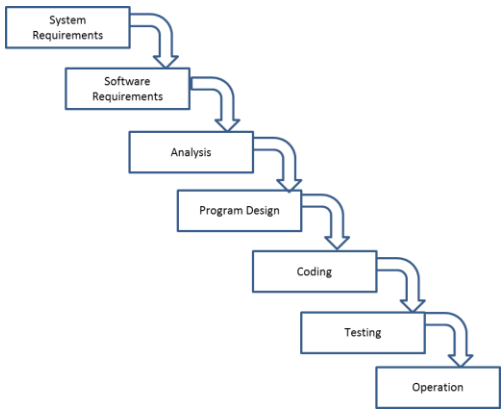


Figure 4 Waterfall Software Development Model (Royce, 1970, p. 330)

Long, waterfall-like development cycles do not allow for requirement changes. Breaking software development cycles into a series of increments allows one to better adapt to changing requirements.

In the incremental model, an increment is a potentially shippable piece of functionality. Incremental delivery allows the user to gain value from a portion of the system prior to the entire system being released (Figure 5).

There are several advantages to the incremental model (Center, 2003, pp. 2.5 - 2.6):

1) Requirements are relatively stable and may be better understood with each increment;

2) Allows some requirements modification and may allow the addition of new requirements;
3) More responsive to user needs than the waterfall model;
4) A usable product is available with the first release, and each cycle results in greater functionality;
5) The project can be stopped any time after the first cycle and leave a working product;
6) Risk is spread out over multiple cycles;
7) This method can usually be performed with fewer people than the waterfall model;
8) Return on investment is visible earlier in the project;
9) Project management may be easier for smaller, incremental projects;
10) Testing may be easier on smaller portions of the system.

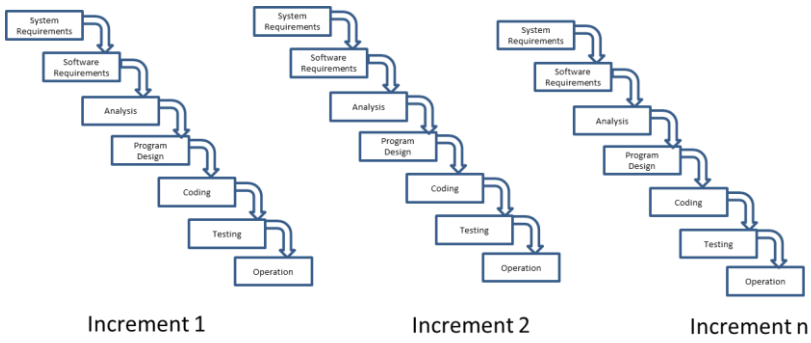


Figure 5 Incremental Development

Though seen as an improvement over the waterfall software development methodology, the incremental approach has several disadvantages, namely that the majority of requirements must be still be known upfront (Center, 2003, p. 2.6).

While incremental development provides several advantages over the traditional waterfall development model, the software engineering community determined that something else was needed in order to better respond to the rapid change in technology and user needs. Thus, to increase the responsiveness with the software process, the software community has moved toward agile development.

"Agile software development" is a broad term used to describe development methodologies that adhere to a set or subset of principles defined by *The Agile Manifesto* (Schwaber & Beedle, 2002). *The Agile Manifesto* was formed when a group of twelve people calling themselves the Agile Alliance gathered to find an alternative to the current documentation driven, lengthy and complex software development process (Schwaber & Beedle, 2002). Through this effort they framed the following set of values to improve the way software is developed ("Manifesto for Agile Software Development," 2012):

1) Individuals and interactions over processes and tools;
2) Working software over comprehensive documentation;
3) Customer collaboration over contract negotiation;
4) Responding to change over following a plan.

The Agile Manifesto states: "While there is value in the items on the right, we value the items on the left more" ("Manifesto for Agile Software Development," 2012). It is not that there isn't any value in comprehensive documentation, simply that there is more value in working software. As with most software development efforts there are tradeoffs that need to be made throughout the process. It is when analyzing these tradeoffs that the items on the left provide more value than the items on the right.

The Agile Manifesto also defines a series of principles which is used to separate agile practices from their heavyweight counterparts ("Manifesto for Agile Software Development," 2012):

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.

The application of these principles varies in practice as there is no predetermined number of principles a development methodology must utilize to be deemed "agile". There are several development methodologies in use today, though a survey conducted by Version One, which included 1681 individuals and 71 countries, found Scrum and eXtreme Programming (XP) to be the most widely followed methodologies (One, 2007, p. 5).

Scrum

Scrum is an overarching process used for project management which is designed for projects where it is difficult to look ahead (Moe, Dingsoyr, & Dyba, 2008, p. 77). It provides a framework (Figure 6) with which these activities will be executed. Scrum is comprised of self-organizing and self-managing teams that release a potentially shippable product in sprints (increments) of two-to-four weeks.

The process starts with a product backlog (requirements) that is prioritized by the user prior to the start of each sprint. The team then selects what can be accomplished within the designated sprint duration but the team must select the requirements in the order specified by the user. These selected requirements then become the sprint backlog. The items on the sprint backlog are what is to be delivered to the customer at the end of the sprint.

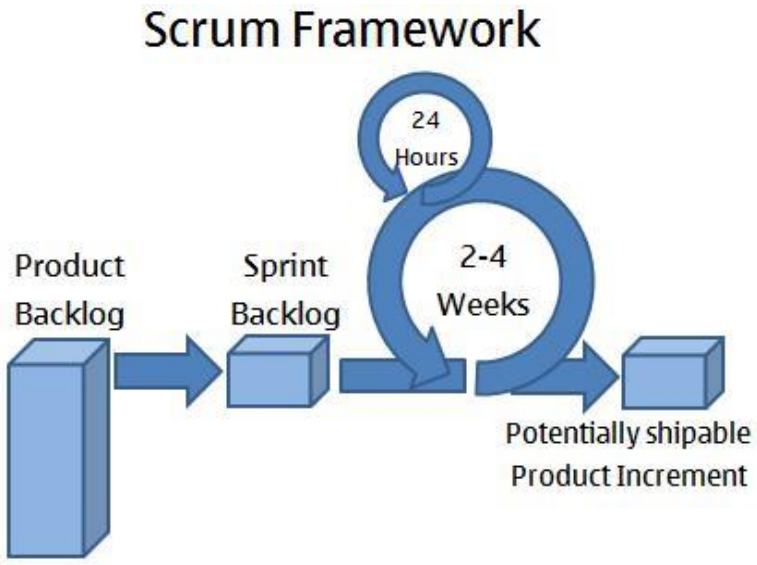


Figure 6 Scrum Framework ("The SCRUM Process | SCRUM Framework," 2012)

Extreme Programming

Whereas Scrum is a process to manage a product, eXtreme Programming (XP) is an agile development methodology focused toward software development as a whole. XP is one of the most well-documented agile methodologies consisting of twelve rules (Cohen, Lindvall, & Costa, 2003, p. 8):

The Planning Game	Refactoring	Small Releases	Metaphor
40-hour Weeks	Collective ownership	Pair Programming	Open Workspace
On-site customer	Continuous Integration	Simple Design	Tests

There is no set number of rules that need to be practiced by a team to claim they are doing XP (Wolak, 2001, pp. 6-7). However, the strength of XP is in the combination of the rules and not solitary implementation (Cohen et al., 2003, p. 13).

Dynamic Systems Development Methodology

Dynamic Systems Development Methodology (DSDM) is based on Rapid Application Development (RAD) with a heavy reliance on prototyping and provides a framework for delivering quality solutions quickly (Krebs, 2008, p. 16). The DSDM is an iterative and incremental process which can be considered the European equivalent to the agile manifesto (Leffingwell, 2007, pp. 66-67).

The DSDM defines nine core principles (Leffingwell, 2007, p. 66):

1) Active user involvement is a must;
2) Design groups are empowered to make system development decisions;
3) Frequent and regular delivery of components is a priority;
4) The primary acceptance criterion for a system or component is its fitness for business purposes—the design driver is business benefit;
5) The business solution is the goal, and iterative and incremental development is necessary to converge on that solution;
6) All changes made during development are reversible;
7) Initial requirements are defined very generally;
8) Testing is not a specific project phase; it occurs constantly;

9) It's essential to have collaboration and cooperation between all project participants.

DSDM is based on the assumption that requirements cannot be fixed and may be inaccurate (Leffingwell, 2007, p. 67). It approaches project management differently than the typical approach where requirements are fixed and then cost and schedule are estimated based on the requirements. In DSDM, cost and schedule are fixed and the requirements are variable (Leffingwell, 2007, p. 67).

Crystal

Crystal is an iterative and incremental development methodology that is heavily focused on people and interaction (Krebs, 2008, p. 18). Crystal increases face-to-face communication to reduce the amount of documentation and improve the likelihood of delivering the system (Cohen et al., 2003, p. 15). It defines a series of color schemes, each containing a set of roles, work products and techniques based on the criticality and the number of people involved (Cohen et al., 2003, p. 16). The color scheme ranges from maroon to clear, where maroon is for heavy projects and clear is for lighter projects, in terms of the number of people involved (Sliger & Broderick, 2008, p. 297).

Lean Development

Lean Development (LD) has its roots in Lean Manufacturing found in the auto industry (Cohen et al., 2003, p. 18). It is a "management approach for streamlining the process of providing value to the customer" (Sliger & Broderick, 2008, p. 298). LD defines twelve principles:

1) Satisfying the customer is the highest priority;
2) Always provide the best value for the money;
3) Success depends on active customer participation;
4) Every LD project is a team effort;
5) Everything is changeable;
6) Domain, not point, solutions;
7) Complete, do not construct;
8) An 80 percent solution today instead of 100 percent solution tomorrow;
9) Minimalism is essential;
10) Needs determine technology;
11) Product growth is feature growth, not size growth;
12) Never push LD beyond its limits.

Kanban

Kanban is Japanese which translate to “Signboard,” and was developed by Toyota as "a tool for managing the flow and production of materials” (Gross & McInnis, 2003, p. 1; Liker, 2003, p. 35). It provides operators visual indicators (Figure 7) as to how much they run and when to changeover as well as allows management to view the schedule status at a glance (Gross & McInnis, 2003, pp. 2-3). David Anderson identified 5 core principles observed in each successful implementation of Kanban (Anderson, 2010, p. 15):

1) Visualize workflow;
2) Limit Work-in-Progress (WIP);
3) Measure and manage flow;
4) Make process policies explicit;

5) Use models' to recognize improvement opportunities.

Though Kanban has its roots in manufacturing its visual indicators and lean principles are being utilized in agile software development (Ikonen, Kettunen, Oza, & Abrahamsson, 2010; Ikonen, Pirinen, Fagerholm, Kettunen, & Abrahamsson, 2011); however, it is not a software development lifecycle methodology or project management tool but can be used to change the underlying process (Anderson, 2010, p. 16).

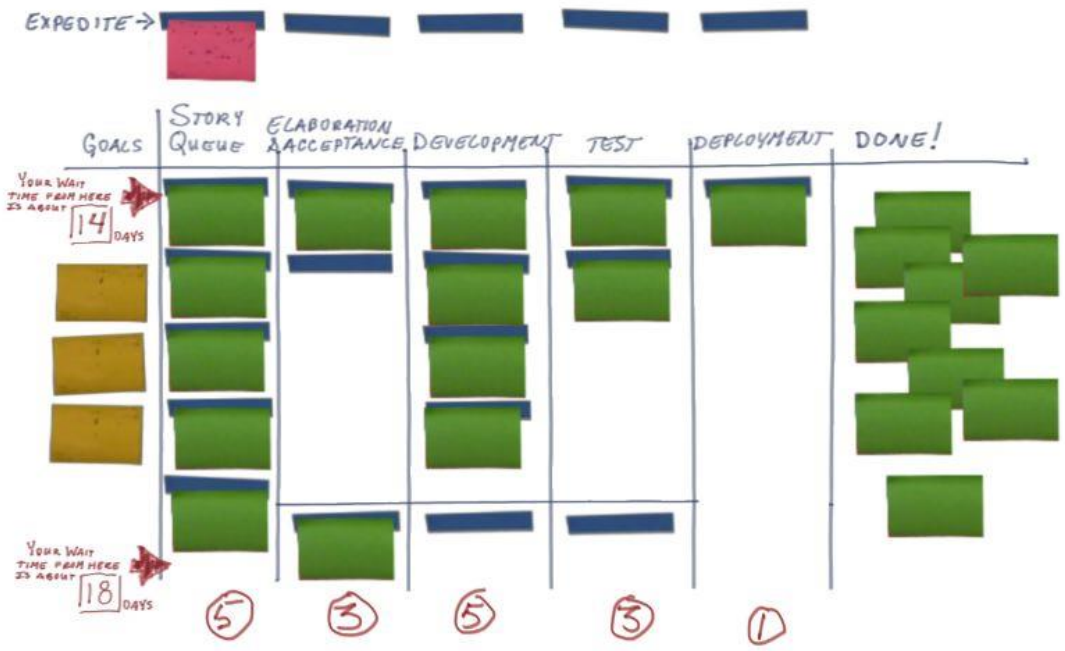


Figure 7 Kanban Board (Patton, 2009)

Agile Practices

Agile development methods – including the most popular ones mentioned here – consist of multiple practices. Although each is distinct, they all have a common thread running through each.

Increment

An increment is a time-boxed period that "represents a complete subset of functionality" (Cohn, 2004, p. 166). The deliverable at the end of an increment meets the Definition of Done (see below) for a specific increment.

Iteration

An iteration is "working, tested, value-delivered code in a short time-box" (Leffingwell, 2007, p. 123). An iteration may not be a completed product but may be an initial piece of functionality which requires user feedback. If the product requires modifications, another iteration begins to address the required changes. An iteration is the improvement of an increment.

Time-Box

A time-box is a predetermined time period comprised of a start/end date and time to accomplish a task. Agile development methodologies use a series of time-boxed events in various locations throughout the development process. Time-boxes are used for releases, iterations and meetings.

Time-boxed releases generally map out a series of iterations that will be delivered to the user. These releases are delivered to the user frequently, which increases responsiveness by incorporating newly discovered customer needs in a shorter time frame and reduces risk by allowing the customers to provide feedback as to whether the software actually addresses their needs (Leffingwell, 2007, pp. 81-85). Unlike the

traditional waterfall approach, the user is able to provide feedback periodically throughout the development lifecycle versus only at the final software delivery.

During time-boxed iterations, the development team is responsible for delivering a piece of potentially shippable functionality. The definition of "potentially shippable" varies based on the functionality being delivered, but can include designing, coding and testing the functionality. There is no predefined duration for an iteration time-box, though it is preferred to deliver working software as frequently as possible, typically from one to six weeks (Northern, Mayfield, Benito, & Casagni, 2010, pp. 3-8). These short time-boxes are another mechanism for allowing the user to provide feedback since each iteration is not complete until it has successfully passed user acceptance testing (Leffingwell, 2007, pp. 84-85).

Time-boxed meetings are used continually throughout the agile development process, which includes release planning, iteration planning, retrospectives, daily standups and demos. These time-boxed meetings allow every team member to "meet his or her commitments by knowing in advance how much time can be committed to any specific overhead activity" (Leffingwell, 2007, p. 83).

Planning

During this phase, planning activities include release and iteration planning, where a release consists of one or more iterations (Cohn, 2004, p. 10). Planning for large projects should rely on five levels (Smits, 2006, p. 4):

1) Project Vision;

2) Project Roadmap;

3) Release Plan

4) Iteration Plan

5) Daily Commitment.

User Story

User stories are a method of communication to describe the functionality from a user's point of view (Cohn, 2004, p. 4). According to Mike Cohn, user stories are comprised of three aspects (Cohn, 2004, p. 4):

1) A written description of the story used for planning and as a reminder;

2) Conversations about the story that servers to flesh out the details of the story;

3) Tests that convey and document details and that can be used to determine when a story is complete.

Unlike formal requirements, user stories are negotiable, even during development (Cohn, 2004, pp. 18-20). They are used to provide a common understanding of the product and are able to be refined or modified throughout the process.

Metaphor

Metaphors are used to provide a common understanding or shared vision between the development team and customer; they define how a program works. Ron Jefferies, one of the founders of XP, provides the following example when describing an agent-

based information retrieval system: This program works like a hive of bees, going out for pollen and bringing it back to the hive (Jefferies, 2012).

Sustainable Pace

Software development is generally a long-term commitment that needs to be approached as such. Just as running in a 100-yard dash would be approached differently than running a marathon, the development team needs to work at a pace that is sustainable during a long-term commitment. Starting a marathon with 100 percent effort will likely result in an unsuccessful completion of the marathon, as the goal is the finish line and not the first 100-yards.

Small / Co-Located Teams

One of the values defined by *The Agile Manifesto* is: "Individuals and interactions over processes and tools" ("Manifesto for Agile Software Development," 2012). Co-location means the team resides in a single room and not just a geographic area. Rather than relying on emails, teleconferences, or video conferences as the primary means of communication, co-locating teams enables continual face-to-face interactions between team members.

Since face-to-face communication is the preferred means of communication, team size plays an important role in the team's success. If the team's size is too large "the quality of interaction between its members decreases and this impairs team effectiveness which results in high costs and process losses" (Duygulu & Ciraklar, 2008, p. 4), in agile

development the optimal number of team members that can fit in a single room is generally two to ten (Cohen et al., 2003, p. 13)

Definition of "Done"

Traditionally used in Scrum, the team must define what "done" means for each iteration. For instance, a definition of "done" could be:

- Help files completed;
- 80% of the functions contain Unit Tests;
- 80% of the code contains comments;
- Integration tested;
- Training materials updated;
- Installation build complete.

This definition can vary from product to product and iteration to iteration, but that which deems a certain product "done" must be predetermined by the team and there is no credit for partially-completed items. If at the end of an iteration, a product does not meet the team's definition of "done", then it is placed back on the backlog list for possible incorporation into the next sprint.

Multi-Disciplinary Teams

Agile teams are comprised of team members with various skillsets. The actual team composition varies greatly based on the product under development; however, a team may consist of developers, documentation experts, testers, database administrators, usability experts and security experts. The idea is to staff the team with all of the

required resources needed to complete the product according to team's definition of "done".

Empowered / Self-Organizing Teams

Agile teams must be empowered to make decisions during design and coding. In essence, management tells the team what needs to be done and *not* how to do it. The team organizes itself around the tasks necessary to complete the assignment.

Product Backlog

A product backlog is a list that is prioritized by the user prior to the start of each time-boxed increment. The team then selects what can be accomplished within the designated sprint duration; however, the team must select the requirements in the order specified by the user. For instance, if the list contains ten items and the team can only complete three, they must select the first three from the list. These three items are added to something called the sprint backlog, which is a list of the product(s) produced at the end of the sprint. The user can reprioritize the product backlog at the end of each sprint allowing for greater agility within the process.

Daily Standup

Commonly used in Scrum, a daily fifteen-minute time-boxed standup meeting is conducted where three questions are answered by each team member (Westmoreland, 2009, p. 11):

1) What did you do today?

2) What will you do tomorrow?

3) Were there any impediments?

The standup provides a near real-time status of items being developed in the iteration.

Retrospectives

Retrospectives are time-boxed meetings where the team gathers to inspect and adapt their methods following a period of work (Derby, Larsen, & Schwaber, 2006). Commonly held at the end of each release and iteration, retrospectives allow the team to continually improve how work was done addressing several aspects of development such as productivity, capability, quality and / or capacity (Derby et al., 2006).

Continuous Integration

Continuous Integration is "where members of a team integrate their work frequently; usually each person integrates at least daily - leading to multiple integrations per day" (Fowler, 2006).

Removing Obstacles

Obstacles may arise during development and the faster these obstacles can be identified and removed the faster the team can continue development as planned. The daily standup is one place in which obstacles can be identified and brought to the attention of the team for resolution. Removing obstacles is an important part of agile development and as such, some agile development methodologies have a team member whose primary purpose is to remove obstacles.

Continual User Involvement

The user works closely with the team throughout development of the product and is responsible for acceptance testing. Preferably, the user is co-located with the team to answer questions and ensure that development is progressing as expected.

Burn Down Charts

Burn down charts “show work remaining over time”("Glossary of Scrum Terms," 2007). These charts show the work remaining on the Y-axis and time on the X-axis ("Glossary of Scrum Terms," 2007). They are used in several areas to track progress. Both the increment and release burn down charts are updated as frequently as possible. The increment burn down is generally updated on a daily basis to ensure progress is accurately tracked.

Demonstrations / Prototyping

The team frequently delivers demonstrations to the user to gather feedback to ensure they are progressing as expected. At a minimum, a demonstration will occur following each iteration.

Table 1 maps agile practices with their related development methodology. Note that most practices may be used with any development methodology but the table below identifies practices or similar practices that are explicitly/implicitly defined by their associated methodology. Lean development was not assessed since its primary focus is on management.

	Agile Development Methodologies			
Agile Practices	**Scrum**	**eXtreme Programming (XP)**	**Dynamic System Development Method (DSDM)**	**Crystal**
Time-Box	X	X	X	X
Planning	X	X	X	X
Increment	X	X	X	X
Iteration	X	X	X	X
User Story		X		
Metaphor		X		
Sustainable Pace	X	X		
Small / Co-Located Teams	X	X		X
Definition of "Done"	X			
Multi-Disciplinary Teams	X	X	X	X
Empowered / Self-Organizing Teams	X	X	X	
Product Backlog	X		X	X
Daily Standup	X			
Retrospectives	X			X
Continuous Integration		X		
Removing Obstacles	X			
Continual User Involvement	X	X	X	X
Burn Down Charts	X			X
Demonstrations / Prototyping	X	X	X	

Table 1 Agile Software Development Methodologies

System Aspect

A system is defined as “the organization of hardware, software, material, facilities, personnel, data and services needed to perform a designated function with specified results, such as the gathering of specified data, its processing and delivery to users” (Hagan, 2011, p. B176). Systems engineering is “the overarching process that a program team applies to transition from a stated capability to an operationally effective and suitable system” (Hagan, 2011, p. B179). Because systems engineers serve as the primary interface among management, customers, suppliers and specialty engineers, it is an interdisciplinary activity drawing its program teams from various backgrounds.

The systems engineering process takes into account both the business and the technical needs of stakeholders from inception to sustainment to retirement, also known as cradle-to-grave development. It does not provide a prescription for success, but provides a high-level structure to serve as a guide. The process needs to be adapted through each phase of development to meet the unique needs of each system.

Lack of evidence implies the System Aspect does not have similar agile initiatives. There are several systems engineering guides and standards available, such as the Defense Acquisition Guidebook (DAG) Chapter 4, IEEE, std 1220-2005, ISO/IEC 15288 and ISO/EIC 26702 ("Adoption of ISO/IEC 15288:2002 Systems Engineering-System Life Cycle Processes," 2005; *Defense Acquisition Guidebook,* 2010, pp. 162-335; "IEEE Standard for Application and Management of the Systems Engineering Process," 2005; "ISO/IEC Standard for Systems Engineering - Application and Management of the Systems Engineering Process," 2007). In practice, no single systems engineering standard is used, but a combination of standards is. For

example, the Air Force (AF) produced AF Instruction 63-1201, Life Cycle Systems Engineering, which references numerous systems engineering standards and is used in the development of all AF systems ("Life Cycle Systems Engineering," 2007). These guides and standards provide the overall structure of the systems engineering process as well as identify characteristics required during the process.

IEEE Std 1220-2005 defines a Systems Engineering Process (SEP) (Figure 8) as "a generic problem-solving process, which provides the mechanisms for identifying and evolving the product and process definitions of a system." It notes that the SEP should be applied throughout the system lifecycle for development and further identifies the lifecycle stages (System definition stage; Preliminary design stage; Detailed design stage; Fabrication, Assembly, Integration and Test (FAIT) stage; Production and customer support stages). However, it does not detail how the SEP should be applied from an agile project management perspective.

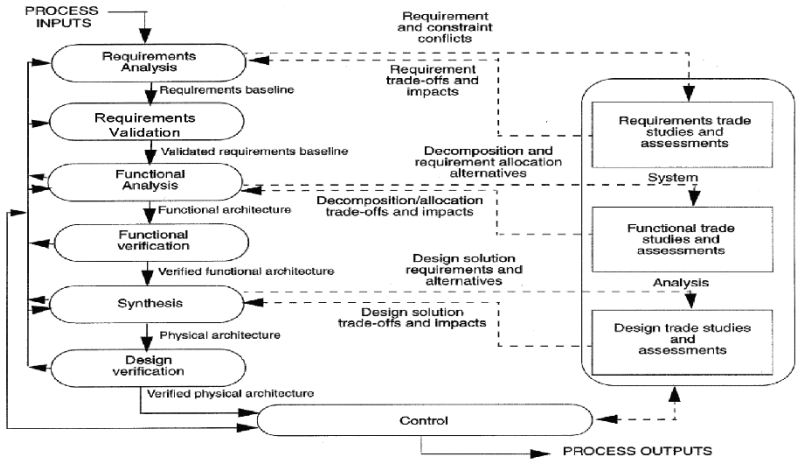


Figure 8 Systems Engineering Process ("IEEE Standard for Application and Management of the Systems Engineering Process," 2005)

Current systems engineering guides and standards provide a waterfall-like structure and key systems engineering characteristics, which are imperative for successful system development. However, they do not provide a framework for planning and managing projects that allows systems engineers to rapidly respond to the changes. The design and implementation of such a framework is left to the systems engineers who are often provided with little guidance. The structure and characteristics provided need to remain intact while their application needs to be framed such that it allows for an agile implementation.

DAG, Chapter 4

The DAG, Chapter 4, divides the systems engineering process into two categories: Technical Management Processes and Technical Processes (Figure 9) (*Defense Acquisition Guidebook*, 2010, p. 181). At a high level, the generic Technical Processes frame the steps necessary to develop a system whereas the Technical Management Processes are used to manage the technical development (*Defense Acquisition Guidebook*, 2010, pp. 181-210).

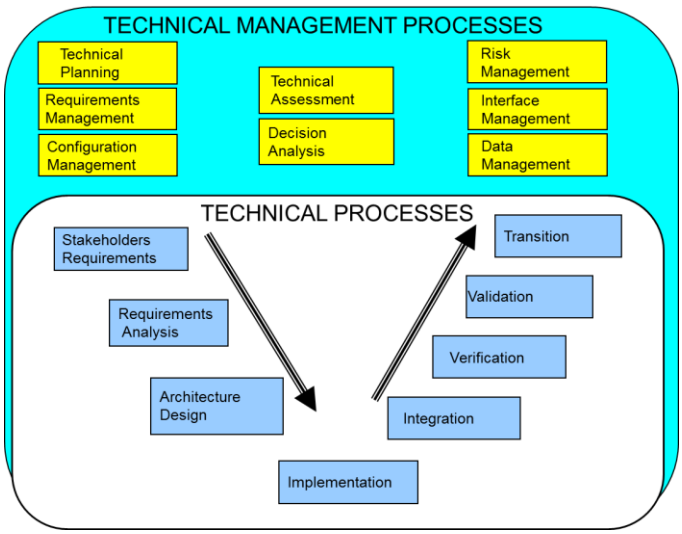


Figure 9 DAG Systems Engineering Processes (*Architecture and Systems Engineering in IT Acquisition*, 2010)

The DAG defines the Technical Processes as (*Defense Acquisition Guidebook*, 2010, pp. 203 - 210):

Stakeholder Requirements: Elicits inputs from relevant stakeholders and translates the inputs into technical requirements.

Requirements Analysis: Encompasses the definition and refinement of system, subsystem and lower-level functional and performance requirements and interfaces to facilitate the Architecture Design process.

Architecture Design: A trade and synthesis process. It translates the outputs of the Stakeholder Requirements Definition and Requirements Analysis processes into alternative design solutions and selects a final design solution. The alternative design solutions include hardware, software and human elements; their enabling processes; and related internal and external interfaces.

Implementation: The process that actually yields the lowest level system elements in the system hierarchy. The system element is made, bought, or reused.

Integration: The process of incorporating the lower-level system elements into a higher-level system element in the physical architecture.

Verification: Confirms that the system element meets the design-to or build-to specifications as defined in the functional, allocated and product baselines.

Validation: Works in conjunction with the Stakeholder Requirements, Requirements Analysis and Architecture and Design processes. It evaluates the requirements, functional and physical architectures and ultimately evaluates the implementation.

Transition: The process applied to move any system element to the next level in the physical architecture.

Technical Management Processes are defined as (*Defense Acquisition Guidebook*, 2010, pp. 181-202):

Decision Analysis: A discipline that employs many procedures, methods and tools for identifying, representing and formally assessing the important aspects of alternative decisions (options) to select an optimum (i.e., the best possible) decision.

Technical Planning: Provide the critical quantitative input to program planning and ensure that the systems engineering processes are applied properly throughout a system's life cycle.

Technical Assessment: Measure technical progress and assess both program plans and requirements. Activities within Technical Assessment include those associated with Technical Performance Measurement.

Requirements Management: Provides traceability back to user-defined capabilities as documented through either the Joint Capabilities Integration and Development System or other user-defined source and to other sources of requirements.

Risk Management: The overarching process that encompasses: identification, analysis, mitigation planning, mitigation plan implementation and tracking.

Data Management: Applies policies, procedures and information technology to plan for, acquire, access, manage, protect and use data of a technical nature to support the total life cycle of the system.

Interface Management: Ensures interface definition and compliance among the elements that compose the system, as well as with other systems with which the system or system elements will interoperate (i.e., system-of-systems (SoS)).

Configuration Management: The application of sound program practices to establish and maintain consistency of a product's or system's attributes with its requirements and evolving technical baseline over its life.

Though the systems engineering process is generally portrayed in a waterfall-like fashion, the systems engineering community has moved toward an incremental delivery approach. The DAG identifies incremental development as a capability that "is developed and fielded in increments with each successive increment building upon earlier increments to achieve an overall capability" (*Defense Acquisition Guidebook*, 2010, p. 262). This incremental approach relies heavily on prototyping and allows for technology maturation in subsequent releases (*Defense Acquisition Guidebook*, 2010, p. 262). This move toward an incremental delivery allows the systems engineering process to better adapt to change than the waterfall-like implementation.

Accompanying these systems engineering guides and standards are quality and performance improvement standards such as ISO 9001 and the Capability Maturity Model Integrated (CMMI) (Cianfrani, Tsiakals, & West, 2009; University, 2006). Though these standards are similar, the ISO standard is less detailed and pertains to the entire organization whereas the CMMI is more detailed and is more focused on engineering and program management (Spaulding).

CMMI

CMMI is a process improvement model consisting of best practices addressing activities throughout the product lifecycle (University, 2006). The development of CMMI was a collaborative effort by the US government, industry and Carnegie Mellon (Institute, 2007, pp. 1-2). It has undergone several transformations since its inception in 2000 which was intended to improve on the Software Capability Maturity Model (SW-CMM) released in 1991(Institute, 2007, pp. 1-2). Version 1.2, contains three main areas or constellations: CMMI-Services, CMMI-Acquisition and CMMI-Development. For the purpose of this summary, we will only be concerned with CMMI-Development (Institute, 2007, pp. 1-2).

CMMI-Development version 1.2 contains the twenty-two process areas shown below. A process area is "a cluster of related practices in an area that, when implemented collectively, satisfy a set of goals considered important for making improvement in that area" (University, 2006, p. 18).

The process areas are grouped into maturity levels in order to assess the overall process maturity of the organization. CMMI defines five maturity levels: Initial, Managed, Defined,

Quantitatively Managed and Optimizing (Figure 10). The stages are cumulative, meaning if one performs the process areas defined in Level 3, one has also perform the process areas in Level 2.

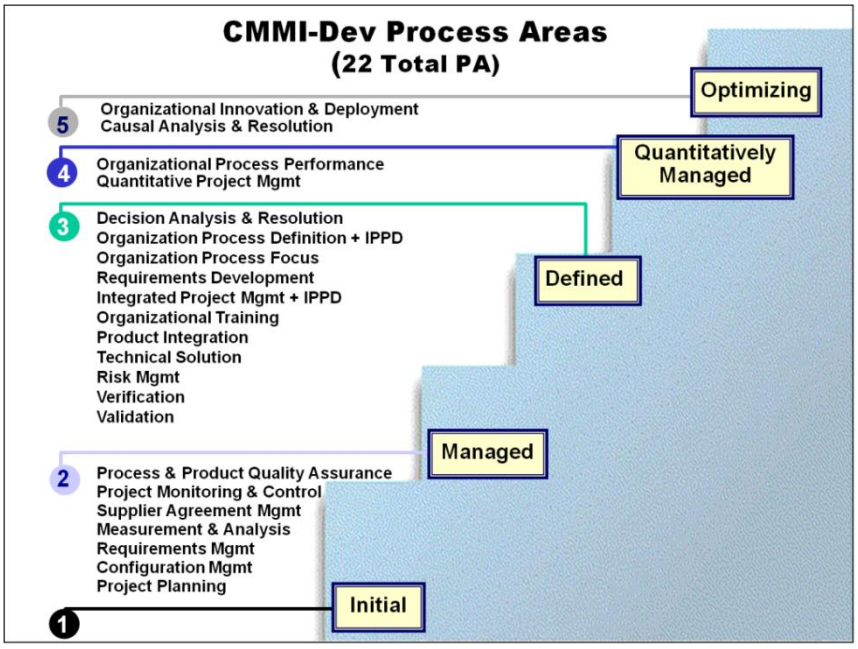


Figure 10 CMMI Levels by Process Areas (Urbon & Charles, 2009)[3]

The twenty-two CMMI-Development process areas and associated purposes are shown below (University, 2006):

> **Causal Analysis and Resolution (CAR)**: to identify causes of defects and other problems and take action to prevent them from occurring in the future.

[3] Chart structure was derived from a SPAWAR presentation depicting CMMI Process Areas and modified to represent CMMI-Dev process areas.

Configuration Management (CM): to establish and maintain the integrity of work products using configuration identification, configuration control, configuration status accounting and configuration audits.

Decision Analysis and Resolution (DAR): to analyze possible decisions using a formal evaluation process that evaluates identified alternatives against established criteria.

Integrated Project Management +IPPD (IPM+IPPD): to establish and manage the project and the involvement of the relevant stakeholders according to an integrated and defined process that is tailored from the organization's set of standard processes.

Measurement and Analysis (MA): to develop and sustain a measurement capability that is used to support management information needs.

Organizational Innovation and Deployment (OID): to select and deploy incremental and innovative improvements that measurably improves the organization's processes and technologies. The improvements support the organization's quality and process performance objectives as derived from the organization's business objectives.

Organizational Process Definition +IPPD (OPD+IPPD): to establish and maintain a usable set of organizational process assets and work environment standards.

Organizational Process Focus (OPF): to plan, implement and deploy organizational process improvements based on a thorough understanding of the current strengths and weaknesses of the organization's processes and process assets.

Organizational Process Performance (OPP): to establish and maintain a quantitative understanding of the performance of the organization's set of standard processes in

support of quality and process-performance objectives and to provide the process performance data, baselines, and models to quantitatively manage the organization's projects.

Organizational Training (OT): to develop the skills and knowledge of people so they can perform their roles effectively and efficiently.

Product Integration (PI): to assemble the product from the product components, ensure that the product, as integrated, functions properly, and deliver the product.

Project Monitoring and Control (PMC): to provide an understanding of the project's progress so that appropriate corrective actions can be taken when the project's performance deviates significantly from the plan.

Project Planning (PP): to establish and maintain plans that define project activities.

Process and Product Quality Assurance (PPQA): to provide staff and management with objective insight into processes and associated work products.

Quantitative Project Management (QPM): to quantitatively manage the project's defined process to achieve the project's established quality and process-performance objectives.

Requirements Development (RD): to produce and analyze customer, product, and product component requirements

Requirements Management (REQM): to manage the requirements of the project's products and product components and to identify inconsistencies between those requirements and the project's plans and work products.

Risk Management (RSKM): to identify potential problems before they occur so that risk-handling activities can be planned and invoked as needed across the life of the product or project to mitigate adverse impacts on achieving objectives.

Supplier Agreement Management (SAM): manage the acquisition of products from suppliers.

Technical Solution (TS): to design, develop and implement solutions to requirements. Solutions, designs and implementations encompass products, product components and product-related lifecycle processes either singly or in combination as appropriate.

Validation (VAL): to demonstrate that a product or product component fulfills its intended use when placed in its intended environment.

Verification (VER): to ensure that selected work products meet their specified requirements.

Together, systems engineering, quality and performance improvement standards provide the structure to enable the successful delivery of a SIS. The incremental delivery of a system has aided the systems engineering process to better respond to change; however, with the rapid rate of change the incorporation of an incremental model alone may not be enough.

Applying Agile Software Practices to System Engineering

Applying agile software practices within the Software Aspect has shown a positive Return on Investment (ROI) in various areas such as cost, quality and productivity when compared to traditional methods (Rico, Sayani, Sutherland, & Sone, 2009). The application of agile software practices outside of the Software Aspect to determine the ROI of these methods is starting to emerge. Opinions as to the applicability of applying the agile practices to areas such as embedded systems or the aviation industry have surfaced and have begun to evolve into preliminary research on how they may be best applied (Dahlby, 2004; Glas & Ziemer, 2009; Matthews, 2011).

Software Engineering Institute (SEI) at Carnegie Mellon "works closely with organizations globally to continually improve software-intensive systems" ("SEI Agile Research Forum: Meeting the Challenges of Large-Scale Systems," 2012). In response to the need to expand agile techniques outside of their traditional use within the Software Aspect the SEI hosted the 2012 Agile Research Forum and continues to supply a series of webinars focused on applying agile tools, methods and techniques to the acquisition and development of large scale systems ("SEI Agile Research Forum: Meeting the Challenges of Large-Scale Systems," 2012).

Using questionnaires, research has been conducted to measure stakeholder satisfaction when applying agile practices to software systems development (Ferreira & Cohen, 2008). The research focused on the overall user satisfaction of using five characteristics of agile development: iterative development, continuous integration, collective ownership, test-driven design and feedback on systems development. The study found a strong positive relationship

between the use of the five agile characteristics and user satisfaction (Ferreira & Cohen, 2008, p. 53).

Single case studies have been conducted on applying agile techniques to niche fields within systems engineering such as their application on embedded real-time medical devices and have found in medical devices "the development and acquisition platforms reduced substantially development time of the product" (Cordeiro et al., 2007, p. 11). Multiple case studies and research was found that provides data on the application of agile methods on "software systems"; however, these case studies define "system" differently than what is described in this research (Capiluppi, Fernandez-Ramil, Higman, Sharp, & Smith, 2007; Koehnemann & Coats, 2009; Lee, 2006). In these studies the term "system" refers to a system which is comprised of only software versus a "system," as described in this paper which includes other components such as firmware, manufacturing and mechanical.

A large amount of research exists which focuses on applying agile practices to software development. Ideas on how these software engineering principles could be applied to systems engineering is starting to emerge and research was found that focuses on applying agile software engineering practices to a single system engineering area such as firmware development. This dissertation expands the use of agile software engineering practices to encompass multiple areas within the systems engineering domain to include firmware, mechanical and manufacturing to measure their the effectiveness on system cost, schedule and functionality and not on quality or user satisfaction as found in other research. In addition to expanding the use of the agile software practices, this research provides a framework to manage and organize the practices within the larger systems context and measures their effectiveness on a real-world project.

Although each agile development methodology utilizes different practices, they share several common practices enhancing agility throughout the process. Regardless of the agile software development methodology selected, there is a high focus on time management / short releases, people and interactions.

The rapid technology refresh rate coupled with the need to respond to changing requirements requires a complete agile development process. This is one where the Business, System and Software Aspects contain an agile framework and work in unison to create a successful SIS. A deficit in any of the three areas will cripple the overall process. The increase in the amount of software within today's systems only increases the need for an agile systems engineering process.

The emerging DoD Agile IT acquisition lifecycle and "IT Box" provide the foundation for the Business Aspect's transformation to agility. Currently, nothing is being done to address the lack of responsiveness within the System Aspect. The System Aspect provides the critical link between the Business and Software Aspect; as such, lack of agility in the System Aspect can have a debilitating effect on the overall development process. This increases the risk of negating both the improvements being made in the Business Aspect and the existing agile processes in the Software Aspect.

The merger of agile practices and CMMI is nothing new in software development. Scrum was introduced into a CMMI Level 5 organization which resulted in improved performance while maintaining CMMI compliance (Sutherland, Jakobsen, & Johnson, 2007). Likewise the combination of agile practices with each other has also been applied in the Software Aspect (Kniberg & Farhang, 2008; Terlecka, 2012). Creating a systems engineering

framework which allows for a similar combination of systems engineering and agile practices would increase the agility of the systems engineering process.

The software and systems engineering processes have undergone similar transformations in an attempt to increase the ability with which they can rapidly respond to change. Both systems and software engineering contain a series of standards, guides and quality and performance improvement processes containing the necessary characteristics to successfully develop a SIS. They have also moved toward incremental delivery in an attempt to better to respond to change.

The software engineering community found that even with the incorporation of incremental development, they were unable to keep pace with today's changing environment. The addition of agile practices has enabled the software engineering community increased agility over the incremental approach. Software engineering has laid the foundation for utilizing agile practices within the development process and the systems engineering community needs to follow suit to remain competitive in today's changing environment.

The development of an agile system engineering framework is required to enhance the overall effectiveness of the SIS development process. Currently, both the Business and Software Aspects contain an agile framework and guiding values and practices to increase agility during the development of a SIS. The System Aspect has not shown similar frameworks or values to assist system development in keeping pace with a rapidly changing environment.

Chapter 3: Agile Systems Engineering Framework and Practices

Practices

Based on analysis from the agile values and practices utilized in the Business and Software Aspects, a core set of practices emerged. These practices are intended to be used with the Agile Systems Engineering Framework. Some of the practices are reinforced in the Framework such as Iterative Development, Incremental Development and Retrospectives. Other omitted practices such as the Definition of "Done" are enforced in the input and exit criteria of the Agile Systems Engineering Framework. Other practices were broken apart to be better adopted at the systems level such as small / collocated teams since you may have small teams that are not collocated. Table 2 lists practices identified to be applicable to the System Aspect.

Incremental Development	Small Teams
Iterative Development	Time-Boxing
Short Time-lines	Lean Initiatives
Retrospectives (Lessons learned)	Prototyping
Empowered/ Self-organizing/Managing teams	Continuous User Involvement
Prioritized Product Backlog (Requirements)	Co-located Teams

Table 2 Agile Systems Engineering Practices

The implementation of these practices varies greatly from project to project. Using co-located teams as an example, a large program retrofitting military aircraft may be structured in such a way as to have the contracting, development and testing portions located on the same installation. This is in contrast to software development teams, which implement the practice of co-located teams by having the development team work in the same room.

These practices are well documented and demonstrated in the Software and Business Aspect and offer great promise for helping deliver affordable systems that are available when needed and effective when used. One should see a similar effect by implementing these proven practices within the Systems Aspect.

Agile System Engineering Framework

To accompany the Agile Practices, a framework is required to manage the complexity of today's large SISs. The Agile System Engineering Framework seeks to foster agile practices in the Software Aspect as well as retain the ability to rapidly respond to changes from the Business Aspect.

The Agile Systems Engineering Framework is comprised of a single release divided into a series of Increments, with each Increment containing one or more Sprints and Integration blocks. The Framework defines three main phases: Release, Increment and Integration. Each phase is completed with a retrospective. A retrospective assesses each phase and provides lessons learned in order to improve the current processes the next time the phase is implemented. An example of the Framework can be seen in Figure 11 and descriptions and inputs / outputs for each phase are described in the successive section.

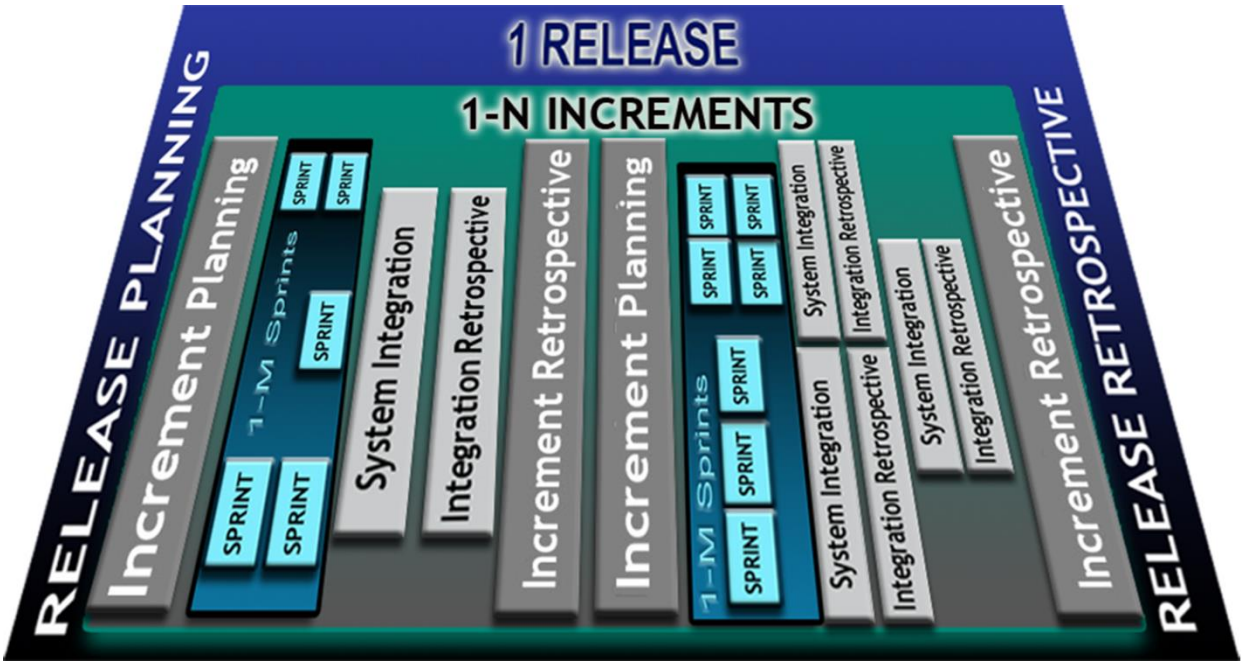


Figure 11 Agile System Engineering Framework

Retrospectives

All retrospectives (Release / Increment / Integration) will assess the execution of the specified phase. They will not be explicitly defined in the descriptions of each phase below.

Release Phase

The product of the release is the delivered system. It starts with a Release Planning meeting, consists of multiple Increments and is completed with a Release Retrospective.

Input Criteria: High Level Design

Exit Criteria: Finished product to be fielded

Release Planning

Activities in this Phase

- Requirements Engineering
    - User Story / Use Case Generation / Refinement
- Increment Time Estimation
- Identify Key System Interfaces

Exit Criteria: Prioritized Release Backlog

Increment Phase:

The Increment Phase receives the prioritized Release backlog from Release Planning Phase. The output of an Increment is an item that is placed under configuration management. Each Increment consists of one or more Sprints and Integration phases.

Input Criteria: Prioritized Release backlog

Exit Criteria: Finished "Configuration Item"

Increment Planning

Input Criteria: Prioritized Release Backlog

Activities in this Phase

- Decompose the system into functional items
- Identify high risk items
- Identify Key System Integration Points
- Identify / further define Key System Interfaces

- Specify temporal dependencies among Sprints; i.e., determine which Sprints can be conducted concurrently and which must be conducted sequentially
- For each Sprint
  - Identify ‘customer’
  - Specify the definition of ‘Done’
  - Specify / Identify expected outputs / specifications for each Sprint
- Select what can be done at each Increment based on the prioritized release backlog
- Identify the personnel/resources/skill set that should be involved in the Increment

Exit Criteria

- Prioritized Sprint backlog(s)
- Incremental program plan identifying Sprints and Integration points

Sprint

A Sprint consists of a time-boxed window for producing a potentially shippable product to be integrated into the system in the parent iteration. The Sprint block is where development of any kind occurs and is handled as a black box to the Agile System Engineering Process. Here a form of “black box trust” occurs allowing each to develop the product freely provided the product is completed using the minimum specifications and interfaces. The risk of these items is managed by a combination of the input specifications / interfaces and development time-boxed window length. This includes, but is not limited to, software and hardware. Multiple Sprints can be underway concurrently.

Input Criteria

- Sprint Backlog
- Specifications / interfaces
- Identification of Customer
- Definition of "Done"

Activities in this Phase: Item Development

Exit Criteria: Completed / user-accepted product(s)

System Integration Phase

The integration phase combines various elements of the overall SIS. These elements could be a combination of hardware and / or software produced by the Sprints and / or the incorporation of Government-Off-The-Shelf (GOTS) / Commercial-Off-The-Shelf (COTS) products required by the SIS.

Input Criteria: Completed Items and specifications/interfaces for the systems to be integrated– From previous Sprints, Increments and / or GOTS / COTS.

Activities in this Phase

- System Integration
- Specifications/interfaces validation / review / refinement

Exit Criteria: User accepted, integrated system

Chapter 4: Validation Alternatives

Various methodologies could be used to validate the proposed Agile Systems Engineering Framework and Practices; however, some lend themselves better in determining the operational suitability of the concepts under study. Regardless of the methodology used, there will be both supporters and critics (Lauck-Dunlop, 2008).

Applying the Agile Systems Engineering Framework and Practices to the entire system development effort allows for the measurement of the application of agile software engineering practices at the systems engineering level. It also allows for assessment of their effect on the complete system engineering effort from requirements to production. Considerations such as testing, laws, regulations, standards, personnel, manufacturing, design, development and integration are only a small subset of real-world issues that arise during systems development which contribute to the overall success of the system. Some complex problems, when broken apart and studied independently, may seem more manageable but the ability to assess their effect on the system as a larger whole is lost (Senge, 1990, p. 9). One approach considered was to assess only the retrospective piece of the agile software practices at the systems engineering level; however, it was felt that we would lose the ability to assess their effect on the Systems Aspect as a whole.

Another validation mechanism considered was presenting the Agile Systems Engineering Framework and Practices to members of the systems engineering community utilizing surveys or interviews as a mechanism to assess their potential use in an operational environment. Several studies have been completed using this validation mechanisms but conclude with future work needed of applying the tool, process and / or methodology to a pilot project. The primary

concern with this approach is at times "the obvious isn't always apparent" (Underhill, 2008, p. 19). When asking questions regarding the applicability of the Framework and Practices, there may be a large divide between what is perceived to be effective and what is actually effective. Though this is one mechanism to validate the project, this is typically a precursor to case study development.

Modeling was also considered as a viable validation approach. One limitation of models is that they are only as good as their known inputs; meaning, the effectiveness of the model is dependent on knowing and accurately capturing the anticipated environment. Models currently in use for systems engineering rely on a waterfall approach to engineering that react to problems rather than anticipate them. The very nature of the agile philosophy is to detect and address unpredictable events as early in the life cycle as possible. A model of an agile system would have to be validated before the model could be used to validate the effectiveness of the Framework.

A case study approach was selected as the most cost effective and feasible approach to validate the application of agile software engineering practices at the systems engineering level. The case study approach capitalized on the opportunity to apply the described Framework and Practices directly on an operational project and measure their impact on the predictability of cost, schedule and functionality of the system under developed. This is taking the research past the level of assessing its potential applicability based on models and / or surveys by actually applying the research in an operational context. Critics of the case study method note the perceived inability to generalize or replicate the results; however, these criticisms are countered

by supporters of case study research showing case study research as a valid research mechanism (Flyvbjerg, 2006; Tellis, 1997; Yin, 2002).

Each mechanism described above is an effective means of validation, but have scenarios in which they are best suited. Formal experimentation is "likely to be useful for investigating alternative methods of performing self-standing tasks"; whereas, case studies are "particularly important for industrial evaluation of software-engineering methods and tools"(Kitchenham, Pickard, & Pfleeger, 1995, p. 52). Though this case study focuses on a single project, the project is composed of 5 sub teams (Firmware, Mechanical, Software, Manufacturing and Quality Control (QC) which were affected by the insertion of the Agile System Engineering Framework and Practices. Because of the large number of interdependent pieces involved in most system engineering efforts isolating a "self-standing" task or team is not possible; lending the validation mechanism in this instance best suited for the case study approach.

Case Study Analysis

Agile software development methodologies have been in use for over a decade but metrics have just recently started to form. Scrum was formalized at the OOPSLA conference in 1995, but definitive metrics on the effectiveness took over decade to start becoming available. Case studies have been used extensively when measuring various effects of agile principles or the tools used to implement these principles on software engineering projects (Babar, 2009; Hajjdiab & Taleb, 2011; Jiangping, Weiping, & Xiaoyao, 2011; Moe & Aurum, 2008; Srinivasan & Lundqvist, 2009; Swaminathan & Jain, 2012). These case studies focus on applying principles and tools to the Software Aspect in contrast to this case study where agile principles are applied to the System Aspect. Because metrics are now available on the application of agile

software engineering practices to software development efforts this dissertation will extend these practices by taking a bottom up approach in applying agile software practices to systems engineering.

In case study research it is important to follow strict guidelines to ensure its validity. Barbra Kitchenham and Lesley Pickard explain a series of case study guidelines when applying a case study to a software development effort which include (Kitchenham et al., 1995, p. 55):

1) Define a hypothesis,
2) Select a pilot project,
3) Identify the method of comparison,
4) Minimize the effect of confounding factors,
5) Plan the case study,
6) Monitor the case study against the plan and
7) Analyze and report the results.

There are similar case study guidelines available (Palena Neale, Thapa, & Boyce, 2006, pp. 5-6; Yin, 2002); however, the guidelines above were used because they target case studies focused on software development efforts.

The goal of the research is to propose an Agile Systems Engineering Framework and Practices based on agile software practices and validate they can be used at the system level to increase the likelihood of delivering a successful system. The effectiveness of their application will be measured through tracking the cost, schedule and functionality delivered at system

completion. These are the three criteria used by the Standish Group to define a successful project.

Hypothesis (Step 1)

- Insertion of agility in the Systems Engineering Processes will increase the accuracy of delivery estimates
  - Metric—Comparing the estimated vs. actual delivery schedule against similar past performance data.
- Insertion of agility in the Systems Engineering Processes will increase the accuracy of the budget estimates
  - Metric—Estimated budget vs. actual budget against similar past performance data.
- Insertion of agility in the Systems Engineering Processes will increase the likelihood of delivering the planned functionality
  - Metric—Functionality planned vs. functionality delivered compared to similar past performance data.

Any two of these metrics taken without the results of the third could be misleading; it is the combination of all three metrics that assist in measuring the overall accuracy of the project estimates. During most system engineering projects there is something called "trade-space", this is a combination of "trade-offs" and "play-space", where the project manager can make "trade-offs" within his allotted "play-space". For instance, the project manager could choose to overrun

the budget estimate in order to meet the estimated functionality or select to remove some of the functionality in order to meet the cost estimate. This idea holds true for any two of the three metrics as the schedule could be overrun in order to produce the estimated functionality. In order to measure the true effectiveness that the Agile System Engineering Framework and Practices have on the accuracy of the estimations, all three metrics need to be measured and reported.

Project Selection Criteria (Step 2)

The International Council on Systems Engineering (INCOSE) defines Systems Engineering as:

> Systems engineering integrates all the disciplines and specialty groups into a team effort forming a structured development process that proceeds from concept to production to operation. Systems Engineering considers both the business and the technical needs of all customers with the goal of providing a quality product that meets the user needs (Engineering, 2012).

Because the definition of systems engineering is so broad it is imperative the pilot project encompass not only a software component but several system engineering components in order for the results to be relevant to other SIS. This would include other components such as mechanical, hardware and /or manufacturing.

The company from which the case study is taken must have past performance metrics on similar (cost, functionality, application types and personnel) projects completed in the past. This similarity will be assessed by the project manager. These metrics must include all three of the metrics defined in Step 1 and are required for the comparison method identified in Step 3. These metrics include:

1) Estimated and actual time to completion
2) Estimated and actual functionality delivered
3) Estimated and actual cost.

The company was also required to use a normal staff-allocation to reduce confounding variables identified in section: minimize the effect of confounding factors (Step 4).

Identify the Method of Comparison (Step 3)

Kitchenham and Pickard identify three options to organize a case study in order to "avoid bias and ensure internal validity" (Kitchenham et al., 1995, p. 56):

1) Select a system project with which to compare,
2) Compare the results of using the new method against a company baseline or
3) If the method applies to individual components, apply it at random to come product components and not others.

The third option was disregarded as the Agile Systems Engineering Framework and Practices do not apply to individual components but to the system development effort as a whole. Selecting a "system project with which to compare" was considered as a strong comparison mechanism; however, it was felt that if the data was available within the company the second option "compare the results of using the new method against a company baseline" was the strongest mechanism of validation since the project under study would be compared to numerous past projects versus a single comparison.

Minimize the Effect of Confounding Factors (Step 4)

Confounding factors occur when "one factor cannot be properly distinguished from the effect of another factor" (Kitchenham et al., 1995, p. 56). Due to the complex interdependencies of system engineering projects it is not practical to eliminate all of the confounding factors in the case study. Kitchenham and Pickard describe the most significant confounding factors on software case studies are (Kitchenham et al., 1995, pp. 56-57):

1) Learning to use the method or tool as you try to assess its benefits,
2) Using a staff who are either very enthusiastic or very skeptical about the method or tool and
3) Comparing different application types.

Learning to use the Framework and Practices while trying to assess its benefits was a major concern during the case study and one that cannot be completely eliminated. This confounding factor was controlled by using a multifaceted approach. First, the project under study was given at least two face-to-face sessions introducing the new Agile Systems Engineering Framework and Practices. Second, quarterly meetings (teleconferences and / or web conferences) were conducted to assist in the application to reduce initial learning curve. Third, ad hoc communications (teleconferences, web conferences and / or emails) were had with the project to assist with any process issues that arose during the project. It was felt that with these three control factors, this confounding factor was effectively mitigated to minimize the learning curve.

To reduce the confounding factor of staff moral it is recommended that you "staff a case-study project using your normal staff-allocation method" (Kitchenham et al., 1995, p. 57). As identified in the selection criteria (Step 2), this selected case study project used similar staff

allocation and personnel as they had in the past. In addition, the project manager was asked about the team's overall resistance to change with respect to the new Agile Systems Engineering Framework and Practices, this response was documented within the results.

The third confounding factor identified was eliminated by only using past performance metrics on projects that are of similar application types. By similar application types, it is meant that comparison will not be done on a project that is primarily a design, development and manufacturing effort against projects that are web based applications.

Though only the three confounding factors above were directly addressed, the others were reviewed and considered. A primary source to assist in the identification of potential confounding factors was the review of cost drivers identified in the Constructive Cost Model (COCOMO) II developed by Dr. Barry Boehm at the University of Southern California (Boehm, 2000, p. 15). COCOMO II is a software cost model which:

> Focuses on issues such as nonsequential and rapid-development process models; reuse-driven approaches involving commercial-off-the-shelf (COTS) packages, reengineering, applications composition, and application generation capabilities; object-oriented approaches supported by distributed middleware; software process maturity effects and process-driven quality estimation. (California, 1998, p. 2)

The COCOMO II model identifies several software development characteristics that affect the effort to complete the project in addition to assigning productivity factors based on 161 projects in the COCOMO II database (Boehm, 2000, p. 15; California, 1998, p. 13). COCOMO II is the successor to the earlier COCOMO 81 model which also identifies cost drivers and assigns productivity ranges, the higher the productivity range the greater the negative influence on the project. These cost drivers have remained relatively static over the years demonstrating

the identified cost drivers of software projects are stable over time. The cost drivers for both COCOMO II and COCOMO 81 can be seen in Figure 12.

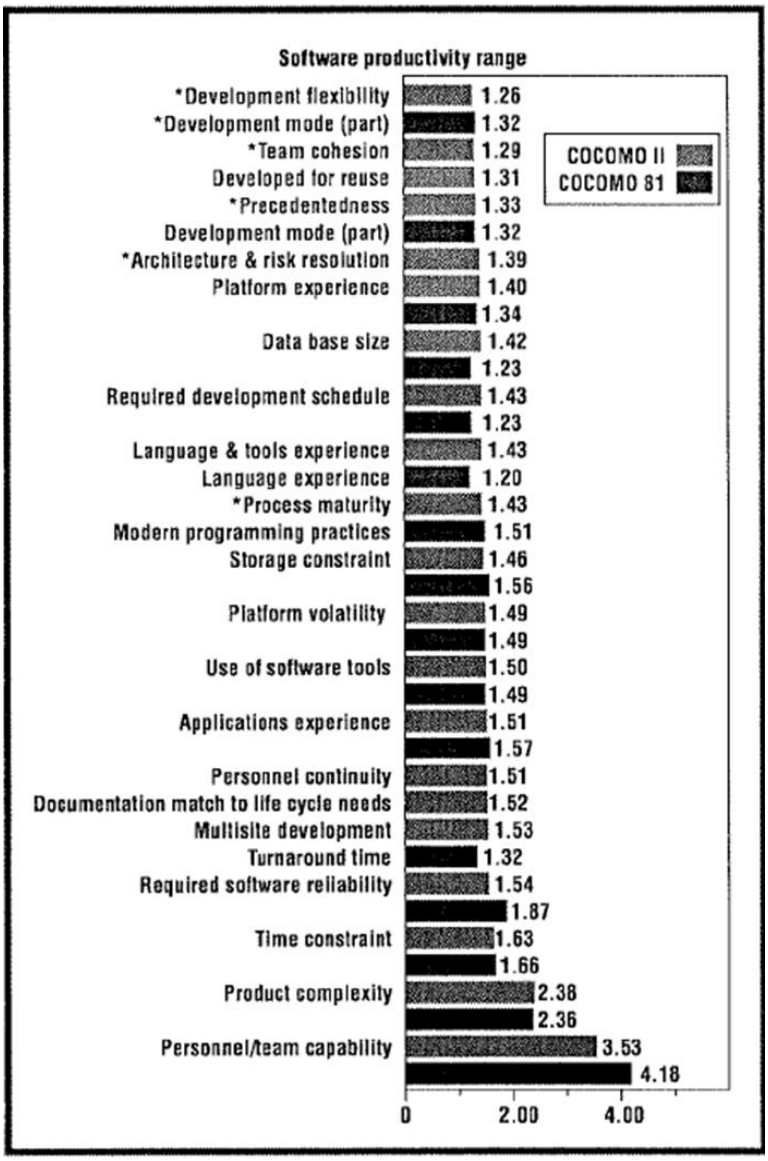


Figure 12 COCOMO Productivity Ranges (Boehm, 2000, p. 15)

Dr. Boehm states that the simplest way to assign your productivity ranges is to base them on your previous projects (Boehm, 2000, p. 14). Since the project selection identified in Step 2 requires the past performance metrics be derived from similar past projects, including size, scope and personnel these potential confounding factors would have a trivial effect on the outcome of

the case study. Taking the largest impact on a software development project, personnel/team capability, as an example, since the past performance metrics in the case study are based similar personnel, the cost drivers assigned productivity would be irrelevant in the comparison since this value would be the same for both the case study as well and the past performance metrics. Due to the careful selection of the pilot project the majority of the confounding factors identified can be control or eliminated.

Plan the Case Study (Step 5)

A case study plan should include training requirements, necessary measures, data collection procedures and the people responsible for the data collection and analysis and a budget schedule and staffing plan (Kitchenham et al., 1995, p. 57).

Training Requirements

Training was conducted through two face-to-face sessions conducted by the primary researcher. Training will be provided to the case study project manager and lead engineers. This training will be a two-way knowledge exchange. The case study project team will provide an overview of the product under development as a final check to ensure it meets the project selection specifications outlined in Step 2 and the research team will provide: an introduction to Agile Development as a whole, the Agile Systems Engineering Framework and an Introduction to the Agile Practices.

Necessary Measures

Measures will be collected as defined in Step 1: estimated and actual time to completion, estimated and actual functionality delivered and estimated and actual cost.

Past performance measure will also be required for final analysis when comparing the results of using the new method against a company baseline. These measures will be collected by the primary researcher and supplied by the case study project manager.

Data collection procedures and the people responsible for the data collection

An important consideration while developing the case study plan is being cautious that the data collection itself does not have an effect on the case study results. The plan shall take caution that this "observer effect" will have minimal influence on the project metrics. The plan needs to weigh the benefits of data collection against the overall effect on the project. For example, it may be unreasonable to plan for bi-weekly 2-hour meetings with the project team to collect data since each meeting is taking away from team's development time thus affecting the outcome of the case study. The frequency of observation and reporting will be assessed to provide a balance of metric integrity as well as minimal project interruptions.

Data will be collected on a quarterly basis through a web conferencing session to assess the current status of the necessary metrics and capture any lessons learned and / or issues encountered at that stage in development. A final meeting will occur at the completion of the project after the product has successfully passed Quality Control (QC) to collect and document the final necessary metrics and any final lessons learned and / or issues encountered. The primary researcher will be responsible for the organization, management, facilitation and documentation of the data collection meetings in addition to data analysis. The case study project program manager will be responsible for supplying lessons learned, metrics and issues encountered by his team or any sub-team.

Budget schedule and staffing plan

There is no allocated budget for this effort and the staffing levels for the case study research team are static and consist of only a primary researcher.

Monitor the Case Study Against the Plan (Step 6)

The company found to do the case study has done so under the agreement that they shall remain anonymous. To comply with this request the company will be referred to as "Juggernaut" for the purposes of this case study. The name Juggernaut has no relation to the company in the study. The product technical specifications, design documents and any detailed information that could be used to trace the case study back to the product or company will not be contained in this report.

Company Background

Juggernaut is an ISO 9001:2008 registered company with over a 100-year history and offices in multiple countries. Juggernaut produces various Information Technology (IT) solutions to customers worldwide.

Juggernaut was initially a manufacturing organization that has expanded to include manufacturing, mechanical and software departments. As their product line increased in complexity and software became a larger part of their systems, their traditional manufacturing top-down development methodology was found ineffective. Their products were becoming routinely late, over budget and did not include the planned functionality.

Small changes to the waterfall-like manufacturing process were found to be ineffective and traditional agile software processes did not provide the framework needed to incorporate

manufacturing, mechanical and software components into a single delivery. Juggernaut soon realized a new development approach was required which would allow for the rapid delivery of systems containing more than just software.

Agile software development had been successfully adopted within Juggernaut's software department but the need to expand those practices to include the entire systems engineering process was becoming evident. Although in-part portions of the currently utilized agile software practices could be employed in the larger systems engineering effort, not all practices were found to be easily adopted at the system level.

Case Study Summary

Juggernaut was provided with the Agile System Engineering Framework and Practices. The Framework and Practices were delivered in two face-to-face training sessions lasting roughly 1.5 hours each. An overview of the slides presented can be found in **Appendix A: Case Study Training Slides**[4]. During these meetings the case study primary researcher was provided an overview of the product under development (this information is unable to be published due to confidentiality agreements). In addition to the training meetings, quarterly teleconferences and virtual meetings were held to provide progress and feedback on the implementation of the agile systems engineering process. Information was also collected and shared through email correspondence which totaled over 75 email exchanges throughout the development process.

Upon acceptance of the Framework and Practices, Juggernaut reengineered its project plan. Shortly after the project was converted to the new approach it was cancelled due to a cost

[4] These slides are not designed to be a stand-alone training module but are intended to facilitate discussion while a subject matter expert is present.

savings effort by the Business Aspect to design uniform underlying hardware on multiple projects across Juggernaut. This new project was able to use some of the functionality of the previous project, which implies that the modular Framework provided allowed for greater reuse even with the short-notice cancellation. The initial project was able to complete the sprints already in process since the output was reusable in the new project. This is potentially an unanticipated benefit of the new Framework and Practices.

The new project was led by the same project manager and engineers so the training sessions were not redone. The new project structure was structured using the same Framework and Practices as the initial project in the case study. The new size and anticipated delivered functionality is more in line with previous Juggernaut development efforts so more of Juggernauts past projects cost, schedule and functionality estimates are able to be used in the past performance metrics.

Project Background

The project selected to utilize the Agile System Engineering Framework and Practices is comprised of a hardware, firmware, software and manufacturing component meeting the project selection criteria of containing several system engineering components. The software component was already using agile software practices. This familiarity allows for the agile software practices to be modified and adapted at the systems level. This effort is a major modification of an existing product and includes the incorporation of new functionality and updates to the system's hardware, firmware, software and manufacturing elements.

Internally, Juggernaut is comprised of several departments including quality control, engineering, operations and manufacturing. The engineering department consists of multiple projects of which the project involved in the case study is just one the ongoing projects. Each engineering project has a series of sub-elements depending on the product under development. The personnel assigned to each sub-element may not be 100% dedicated to the one project but may be working on several projects simultaneously. Each project within the engineering department is also competing for other department resources for manufacturing, operations and / or Quality Control (QC). A system development organizational structure can be seen in Figure 13.

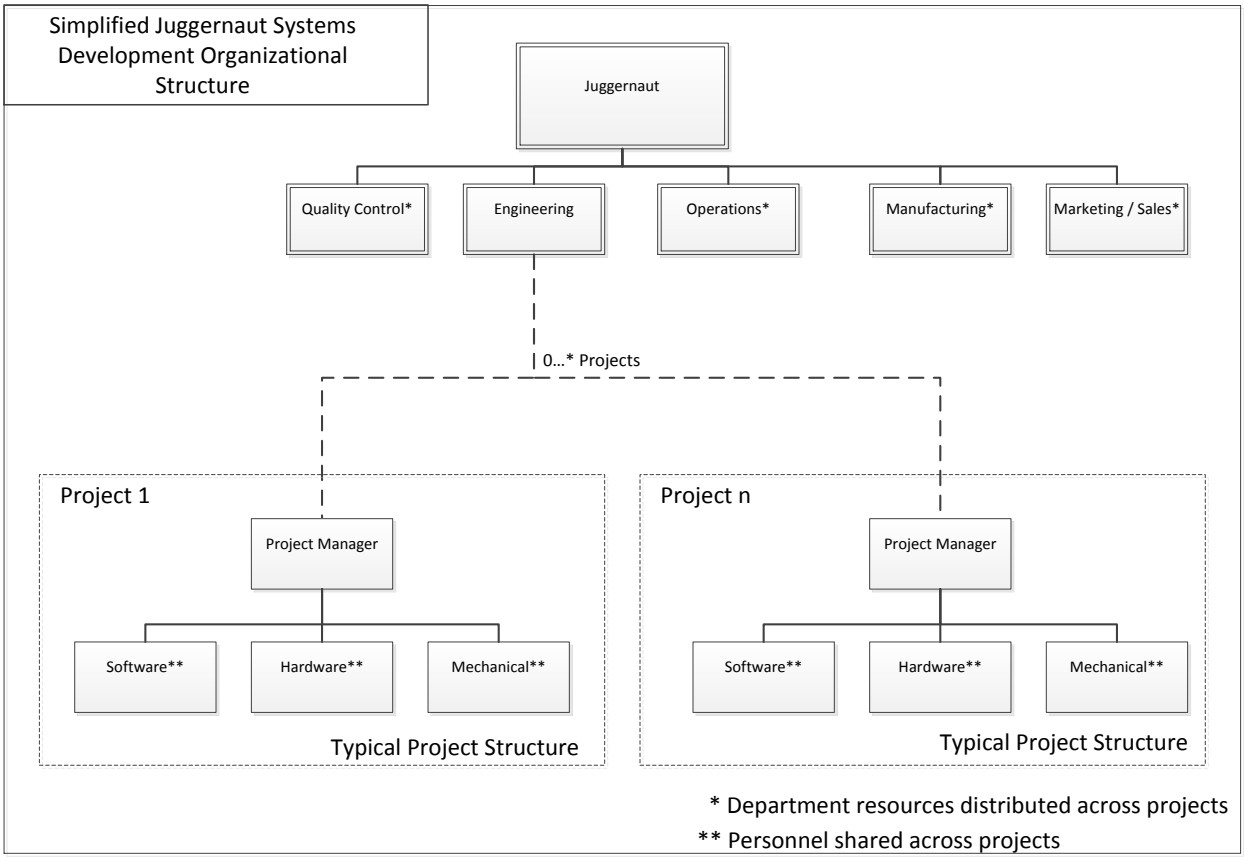


Figure 13 Juggernaut Organizational Structure

In addition to the internally developed hardware, firmware, software and manufacturing, portions of these components are outsourced to leverage external expertise in emerging technologies. Roughly 50 percent of the design and mechanical components are outsourced. These outsourced components need to be accounted for and managed within the Agile Systems Engineering Framework to allow for external components to be tested at various integration points. In addition to these outsourced components, Juggernaut uses manufacturing facilities that are located both in and outside of the United States (US) to assemble the final product, increasing the coordination effort required during development.

The delivered product must conform and / or be certified in several specifications, including American National Standards Institute (ANSI) and United States Military Standard (MIL-STD) specifications. The combination of internally and externally developed components coupled with the certification process make the identification and definition of the integration points paramount to the success of the project.

The internal team, team members working directly for Juggernaut, consisted of sixteen multidisciplinary members in the following specialties: Project Management, Firmware Development, Software Development, Firmware Testing, Systems Testing, Hardware Development and Hardware Testing. The internal team has worked together in the past on similar projects; however, they were utilizing a waterfall-type development methodology. This effort was the first implementation of an agile system engineering methodology being employed project-wide.

Past Performance

Juggernaut was able to provide complete past performance metrics including cost schedule and functionality delivered on twelve projects that were of similar size and scope. Based on the twelve projects, Juggernaut was habitually behind schedule, over budget and not delivering the planned functionality. The past performance metrics were calculated by taking the average of the estimated versus actual numbers for all three data points on the twelve projects. The results can be seen in Table 3 Past Performance

Average Cost Difference from Estimate	+2.5%
Average Schedule Difference from Estimate	+30%
Average Functionality difference from planned	-5%

Table 3 Past Performance

Applying the Framework and Practices

Since Juggernaut was already using an agile software development methodology, the incorporation of most of the Practices (such as time-boxing and sprints) was not a difficult change for the organization at the system level. However, during certain phases of system development these practices were found less effective, particularly during phases that contained difficult to determine deliverables and exit criteria, such as design. During these phases, Practices such as time-boxing and sprints were not as strictly enforced.

Based on analysis from the Release Planning Phase, three Increments were planned each containing Sprints that were designed to coincide with key integration points in which a

combination of internally and externally developed hardware, firmware, enclosures and / or software needed to be integrated and tested by the Quality Control (QC) group.

At these key integration points, the hardware and software were required to have certain functionality and the QC group had to have the necessary outside sources, equipment and personnel to run specific integration tests. As defined in the process, each integration point had a set of predetermined input and exit criteria used to measure whether a successful integration was accomplished.

The Framework structure and Practices were utilized; however, Juggernaut was careful not to deviate from their standard terminology during implementation. During implementation and when presenting the project to other department managers, Juggernaut used their terminology and not the terminology described in the Framework. For instance, Juggernaut used the standard Alpha and Beta terminology to describe various increments during development. The new approach added Alpha1, Alpha2, etc., versus using the term iteration, which is used in the agile Framework. Terminology such as time-boxing and retrospectives, where they had no previous equivalent to the terms described in the Framework, were used. This semantic difference had no effect on the Framework structure but decreased the organizational learning curve to utilize the agile Framework and Practices which may have led to the high level of adoption within the project.

One of the most difficult tasks during any change management effort was to resist the urge to go back to "the old" way. During the application of the Framework and Practices at Juggernaut project management experienced very little resistance to change once the new

processes were in place. In addition, the project manager noted that "all future projects will be run in a similar fashion".

The benefits of the Framework and Practices were seen outside of the engineering division and stretched into marketing. The increase in predictability of delivery dates allowed the marketing division to better plan for the marketing aspect of the product.

As intended, Juggernaut customized the agile practices provided to meet their unique system needs. Below identifies Juggernaut's old practices and how they incorporated the new agile practices into their system development lifecycle.

Incremental Development

Previous Approach

Development was implemented using a waterfall-type approach where the product was completed using a once-through development methodology. There was one step of each development, integration and product testing defined. Incremental development was not utilized.

Agile Approach

The development was decomposed into three Increments based on analysis from the Release Planning Phase. Each Increment produced discrete pieces of functionality that provided value to the overall system. The Increments were then decomposed into multiple Sprints that encapsulated a development and testing step for each piece of functionality. The Increments also contained integration phases allowing for system integration to occur throughout development. This is contrary to the previous approach where there was a single integration for the product.

Iterative development

Previous Approach

Since development was completed in a once-through development methodology; iterative development was used in the standard Alpha, Beta type release system. However, there was only one Alpha and one Beta release per development cycle.

Agile Approach

At the release level the standard Alpha, Beta system was used, however, as problems were identified internally during an increment or integration step, there were several iterations prior to the first Alpha release. This allowed for problems to be identified and corrected early in development. Juggernaut added Alpha 1, Alpha 2 increments prior to entering into Beta which are defined below:

Alpha 1: Internal, engineering release only

Alpha 2: Second iteration of an internal engineering release

Beta: Roughly a 90 percent completed product which may include a controlled release to customers

Time-boxing

Previous Approach

Time-boxing was not used.

Agile Approach

Time-boxing was utilized for each Sprint, Increment and tri-weekly meetings. Sprints were decomposed until they were able to be completed within five weeks, on average. If a Sprint contained too much work to be completed in five weeks, it was broken down into sub-sprints allowing them to be completed within a five-week time-box and then rolled up into a major sprint. Five weeks was the duration that Juggernaut felt they were able to effectively manage the project while allowing enough time for system level functionality to be developed. During phases that contained difficult to determine deliverables, such as design timelines were not as strictly enforced as they were in other areas such as Sprints and Increments.

Co-located teams

Previous Approach

Teams were co-located in the same facility but did not share a common workspace.

Agile Approach

Agile software development teams are typically collocated in a common workspace; however, the teams followed a similar approach to what they had done previously and were co-located in the same facility but did not share a common workspace. At the system level, project management at Juggernaut did not feel there would be value in moving everyone to the same facility and the physical layout of the facility was not conducive to this environment. Also, several team members were working on multiple projects so colocation could prove problematic for other projects.

Prototyping

Previous Approach

Prototyping was done at an Alpha and Beta stage of the development lifecycle as seen in a typical waterfall development methodology. The prototypes were also much farther along in development that the prototypes used during the agile approach. They were virtually completed products where any identified change could cause a rework of multiple subsystems.

Agile Approach

Prototypes were utilized during each increment. Overall they increased the number of prototypes by one during system development. This increase in prototypes allowed for the early identification of issues allowing for minimal rework to be scheduled to correct the problems. The initial prototypes were released early while system development was still occurring unlike the previous approach where development was nearing an end or in some cases had already ended. When problems were identified the issues could be rolled into the existing development schedule versus having to startup a new development effort in hopes the resources were still available.

Prioritized Requirements

Previous Approach

In general, the easiest features were put into the product first. In some products where multiple projects needed certain features, the features were aligned to help the other projects. This approach led to early success as the easiest pieces were done first; however, the more

difficult tasks scheduled later in development were causing the schedule to slip and increase in costs. Initially the project would appear on schedule but during the later portion of the system development the system would routinely fall behind schedule and over budget.

Agile Approach

The requirements were assessed based on their overall development risk as well as their value-added to the overall systems. This risk assessment was based on overall development risk, other project dependencies, new technology insertion, manufacturing material limitations and other project specific attributes. The risk was assigned based on project managements' past experience. Once identified, these high-risk items were developed first in order to detect possible complications early in development. By delivering a combination of high-risk and value-added components first, Juggernaut was able to ensure at a minimum the high priority items would be included in the product.

Small Teams

Previous Approach

Previous projects ranged anywhere from five to twelve internal team members.

Agile Approach

This project contained sixteen internal team members broken up into four sub-teams based on domain (firmware, software, QC, etc). There were five engineers, four electrical and one mechanical. Each team had a designated team leader who was also responsible for communicating with the external firms in their domain. The increase in team members was due

to bringing in other departments such as QC early in development versus simply handing the product off for testing after it left the development team.

Short timelines

Previous Approach

Not applicable. The timeline for the project was based on the overall development and not decomposed into smaller timelines. This structure created decreased visibility into the overall progress of the project.

Agile Approach

The timelines were based on the prioritized requirements list, Juggernaut time-boxed the Increments and Sprints to the minimum time feasible to deliver the decomposed functionality. For example, each Sprint was decomposed into five-week time-boxed periods with each Sprint having predetermined input and output criteria. Project management identified that this timeframe would allow for sufficient visibility into the project during development.

Retrospectives

Previous Approach

No official retrospective. There were unofficial lessons learned that were poorly documented and generally contained no root cause analysis. This led to the same problems happening in future projects.

Agile Approach

Retrospective meetings were scheduled after each Sprint. In addition, each month an informal retrospective meeting was held specifically to discuss, and possibly make changes to, the Sprint process or to the meeting process (times, frequency and duration) that was implemented allowing for project process optimizations to occur throughout the project. These frequent retrospectives allowed for quick adoption of lessons learned and ensured similar mistakes would not be repeated on the next Sprint or Increment.

Continuous user involvement

Previous Approach

User testing was always done, but in an ad hoc fashion and not well documented. This caused issues in which one user would test the product using certain steps while another user would test it using different steps. This inconsistency led to test data that was incomplete, confusing and often misleading.

Agile Approach

A Quality Control (QC) group was added to the project early in development. The QC group was responsible for testing the devices like a customer would use them and documenting the results. The QC group was external to the project but internal to Juggernaut.

Lean Initiatives

Previous Approach

Lean initiatives were typically done during late in system development which did not allow for these initiatives to be utilized by the system under development.

Agile Approach

This was done during the planning stages instead of development / design stages. Moving lean initiatives earlier in development helped eliminate duplication and utilize the initiatives on the system under development.

Applying the Agile Systems Engineering Framework

As the name implies the Framework only provides the basic structure to organize and manage a project. Implementation of the Framework will vary greatly from project to project. Juggernaut's implementation of the Framework is described below.

As indicated in the Framework, each phase contained a series of input, activities and exit criteria. The input criteria are artifacts that are required prior to the phase beginning. In the case of release planning, this project could not start until the use cases and Marketing Request Specification (MRS) are complete. The input, activities and exit criteria will vary for each project. Likewise, the implementation could vary. A project could create requirements using use cases, user stories or other mechanisms. The importance is that the requirements *are* specified and not *how* they are specified.

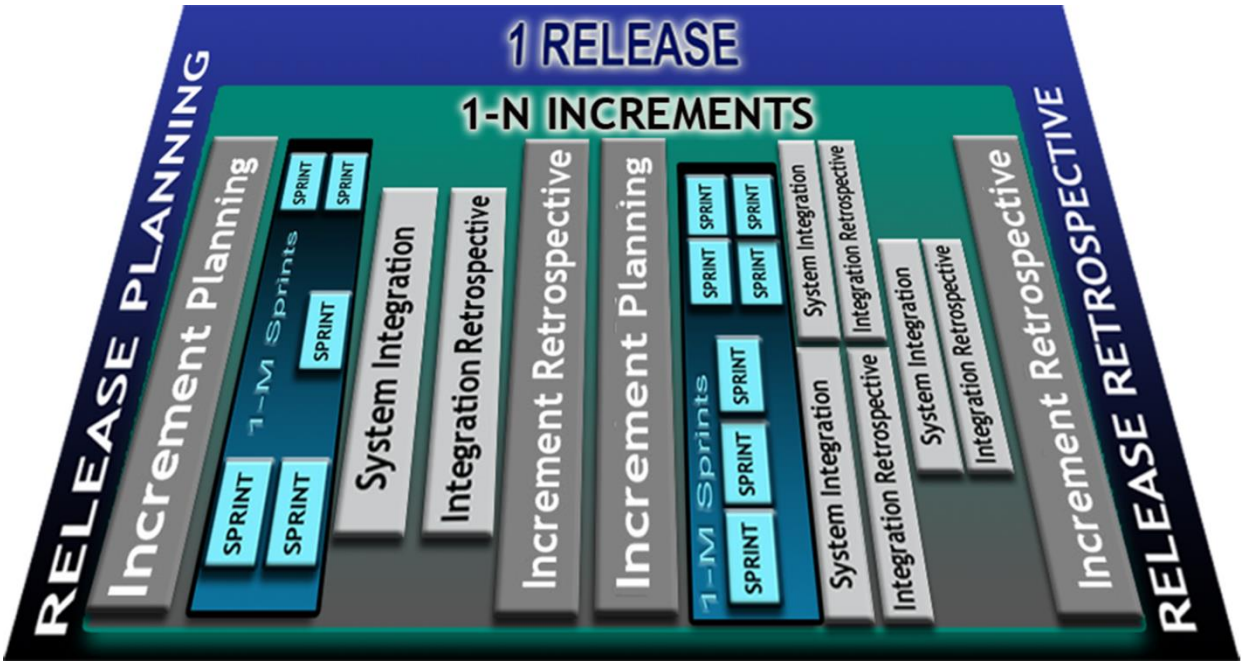


Figure 14 Agile System Engineering Framework

Release Planning

Input Criteria

As input to the release planning phase, Juggernaut received high-level use cases and a Marketing Request Specification (MRS) which laid the foundation for the high-level design required to complete the activities during this phase.

Activities in this Phase

Using the use cases and MRS, Juggernaut crafted more detailed designs for the various components by creating a Software Operational Document, a High Level Design Goal Specification and a Hardware Design Specification. These specifications contained required interface specifications and any high level data exchange requirements need by the system. The refinement of the MRS and use cases allowed for more detailed requirements to be formed and

prioritized. From this refinement a preliminary Sprint / Increment structure was formed which also identified critical system integration points.

Exit Criteria

The resulting product of the release planning phase was a list of prioritized requirements and a high-level schedule defining Increments, Integration points and Sprints that would occur during the release.

Increment Planning

In the previous approach, everything was combined. There was little design prior to development and testing. The incremental planning allowed testing to be handled incrementally and was recognized by Juggernaut's project management as a critical piece that contributed to project success since defects were found and corrected early. This brought all of the components together to validate the product at various stages of development. Since the requirements are prioritized on a risk-based approach, the high-risk items were tested earlier in development.

Input Criteria

Prior to starting this phase juggernaut required the high level schedule and prioritized requirements list.

Activities in this Phase

Using the high level schedule as a guide, Juggernaut continued to identify and further define Sprints. At the end of each series of Sprints was an integration point for hardware,

enclosures and / or software. These integration points required explicit interfaces which were defined through specification documents used during the integration phase.

Further analysis was conducted to identify dependencies between Sprints to ensure the Sprints could be conducted in the order specified based on both internal and external dependencies such as manufacturing availability and the estimated completion of externally developed functionality.

Once the Sprints were defined, Juggernaut identified a customer, time estimate and testing procedures for each sprint. For most Sprints, the customer was the QC group and the test group. The developmental tests were documented with specific pass / fail criteria. If the product did not pass the developmental testing or QC testing, it was deemed inefficient and would have to be fixed in a later Sprint. Depending on the Sprint, a hardware and / or software document was created that was similar to a datasheet. These documents were used to create the required tests as well as direct the development both internally and externally. A final check was then conducted to ensure the increment could be completed in the timeline specified.

Exit Criteria

The result of increment planning was a defined increment structure based on identified sprints, dependencies and integration points.

Sprints

Input Criteria

Each Sprint received the specifications based on the component that was under development, typically a hardware or software document, along with the customer identified for the sprint and the associated testing procedures.

Activities in this Phase

The development of the component occurred based on the specifications provided. The process with which the item was developed was not mandated or defined. The Sprint members could use any development methodology while developing the component. For instance, they could use Scrum, XP or an internally defined process allowing for infinite flexibility at the Sprint level.

Exit Criteria

The product of the Sprint is a completed component that passes the specified testing procedures. If the product did not pass it was deemed unsuccessful and would have to be fixed in a later Sprint.

Integration

Input Criteria

The input criteria varied based on the components being integrated. The input criteria could contain any combination of hardware, software and / or manufacturing packages that were developed internal or external to Juggernaut. Regardless of the components there was a

specification document outlining the interfaces undergoing integration and a series of pass / fail criteria which answered the questions: do the parts fit as designed, does the metrology of the unit work as designed once integrated and / or does the mechanical design meet the dimensional requirements?

Activities in this Phase

Overall integration was conducted in accordance with the specification document and the predefined pass / fail criteria of the subsystems.

Exit Criteria

The final product of the integration phase was an integrated system which passed the predefined pass / fail criteria.

Other Changes

The outlined process change was only a piece of the change that Juggernaut undertook to implement the Agile Systems Engineering Framework. Increased coordination between departments such as QC, engineering and manufacturing were realized in order to plan for multiple touch points during development. For example, the system used in this study was not the only system under development at Juggernaut, therefore QC and manufacturing resources had to be appropriately managed to ensure the proposed schedule was feasible for all departments.

Meetings were restructured to include both the background of the attendees as well as the frequency of the meetings. The meetings were focused on certain aspects of development resulting in less people attending each meeting, but more meetings overall. It was noticed that

having smaller (both in number of attendees and time) meetings increased the amount of feedback that was provided at each meeting.

Juggernaut routinely held team meetings (various teams met depending on the stage in development) every other day, which were time-boxed to fifteen minutes. During these meetings the status of each team was provided in which they identified any risks / issues their teams are facing as well as any schedule variance.

Project-level status meetings were held every other week and were time-boxed to forty-five minutes. These project meetings allowed for overall impact analysis based on the feedback from the team meetings.

Both the team and project meetings were time-boxed and kept as an informational, and not a problem-solving, session. Ad hoc meetings were also scheduled if issues arose during development which required external participation or additional time for analysis or resolution.

Previously, the meetings were held weekly, had more attendees, and were longer in duration. Though the overall number of meetings increased, the time spent at the meetings decreased and the feedback from the team increased, allowing problems to be identified as soon as possible versus waiting for the weekly meetings.

The projects were structured by domain area: Project Management, Electrical, Mechanical and Firmware. Each domain area was assigned a designated lead that would attend the team meetings to provide the status of their project. The structure allowed the team members to continue working while the lead reported on their status. Previously, all team members were

involved in the status meetings effectively taking time away from development. In addition, each team had internal coordination meetings as needed for each subproject.

Framework Implementation Example

During the Increment planning phase Juggernaut designated their Sprint Phases as hardware, mechanical, software and / or firmware sprints. Each Integration Phase required input from a predetermined number of sprints and / or external dependencies (designs, hardware components, Commercial-Off-the-Shelf products, etc.). The case study example below requires input from three internally developed Sprints, though other Integration Phases may require both internal and external input to achieve the specified objectives. In this example details have been removed such as the actual specification requirements and additional details required to adequately test the exit criteria such as required response times and detailed accuracy ranges. Names of the customer POC are also not included. The example is intended to provide the overall organization Juggernaut used to structure their systems development using the Agile Systems Engineering Framework. A graphical depiction of the example can be seen in Figure 15.

Integration 1: Wired Functionality

Objectives

- Decision on the Solar Panel / Internal Battery concept.
- Metrology meets specified tolerance ranges
- Hardware Interface meets current profile requirements.
- Initial wire testing is complete with list of defects.
- Validation of Test Point access (Test Engineering)

Input Criteria:

Sprint 1 – Hardware

Input Criteria:

- Hardware Specification Document

Exit Criteria:

- Support multiple energy sources
- Discrete solution for the register interface

Sprint 2 – Mechanical

Input Criteria:

- Design Specification Document

Exit Criteria:

- Rolled Sealed Assemblies
- Housings According to Specification

Sprint 3 – Firmware

Input Criteria:

- Firmware Specification Document

Exit Criteria:

- Wire Communication Standard Conformance
- Liquid Crystal Display (LCD) functionality version, volume and rate
- LCD Activation Level
- Metrology Functionality

Exit Criteria

- Current Consumption/Profile
- Wire Interface HW Testing
- Primary Power Supply Capabilities
- LCD Testing
- E&M Field Testing
- Electrostatic Discharge Testing
- Forward and Reverse Flow Accuracy
- LCD Activation Level (multi-source)Wire response
- Clock Detection Accuracy
- ASIC Read.

- Flag validation
- Submersion Test
- Torque Testing
- Sensor Pulse level testing
- Metrology accuracy
- Pulse counting

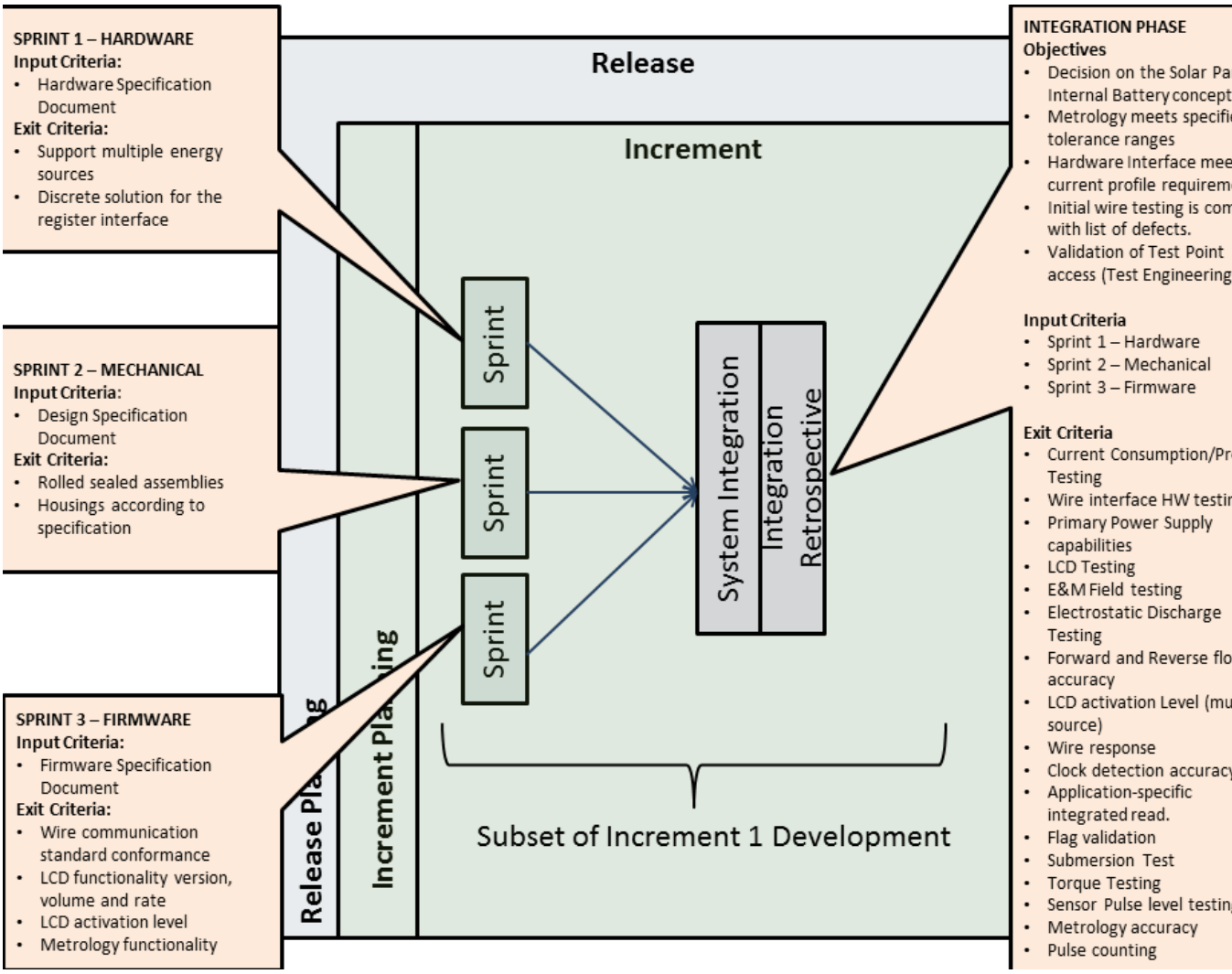


Figure 15 Agile Systems Engineering Framework Example

Project Summary

For case study tracking purposes, the project was divided into three critical milestones: the first milestone marked the successful completion of internal QC; the second milestone was the released to a limited user base and external testing; and the third milestone was mass production and customer sales. The metrics for this case study focus on the first milestone.

Completion of Milestone 1 is the focal point of this case study as it included design, development and internal QC of the product. Juggernaut produced seventy units during this phase to put through internal QC. In the design phase, fifty QC checks were identified and of the fifty QC checks, twenty were deemed high-risk and were required to be successfully run based on predetermined criteria before going to Milestone 2. These QC checks included tests such as vibration, over voltage, thermal shock and current consumption. The remaining QC tests were either low risk or required outside testing such as International Organization for Standardization (ISO) or United States Military Standard (MIL-STD) testing and were scheduled for the completion of Milestone 2.

After the successful completion of Milestone 1, the product specifications were sent to several production plants in various countries for manufacturing resulting in 500 production units. The 500 units that are produced are either distributed to a limited user base or used to complete the remaining QC checks. After receiving the results of the user feedback and QC tests, Juggernaut performs an assessment to determine if the units will be allowed to be mass produced. If approval is granted the specifications are sent for mass production, marking the successful completion of Milestone 2. Milestone 3 includes the mass production and customer sales of the final product.

Analyze and Report the Results (Step 7)

The completion of Milestone 1 was scheduled for 27.5 weeks and the actual completion took 29 weeks. This was a 5.5 percent increase in duration from the initial estimate. When compared to the past performance of Juggernaut, there was a 24.5 percent improvement in predicting their schedule using the Agile Systems Engineering Framework and Practices. The primary cause of the schedule delay on this project was due to the assembly of components performed by outside contractors causing the overall schedule slippage.

Since Juggernaut has been developing similar systems for years, their cost estimation is typically accurate prior to using the Agile Systems Engineering Framework and Practices. Throughout the project, the project teams were assigned a "cost goal" based on the overall cost estimate. These cost goals are then used by the teams to make tradeoffs throughout the project. The teams are able to meet their cost goal by balancing cost factors such as scope, material costs, or labor. On this project, the largest contributor to the under run in budget was that one of Juggernaut's vendors agreed to decrease their profit margin resulting in a decrease in overall project cost. Based on Juggernaut's past performance metrics, Juggernaut typically ran 2.5 percent over budget. At the completion of Milestone 1, Juggernaut was 5 percent under budget marking a 7.5 percent difference in cost estimation. Because the cost was due to a vendor renegotiation, the cost fluctuation was not attributed to the Framework or Practices utilized during product development.

Typically Juggernaut has experienced a decrease in 5 percent of the planned functionality in order to better meet their cost and schedule goals. At the completion of milestone one they delivered 100 percent of the planned functionality showing an overall improvement of 5 percent.

	Past Performance	*New Model*	*Result*
Cost Difference from Estimate[5]	N/A	N/A	N/A
Schedule Difference from Estimate	+30%	+5.5%	24.5% Improvement
Functionality difference from planned	-5%	0%	5% Improvement

Table 4 Case Results Data

Lessons Learned

Release Planning

While performing systems design in the Release Planning Phase it was felt that additional time should have been spent identifying and using open standards, such as ANSI, for their system interfaces. The increase of open standards would have allowed Juggernaut to cost effectively outsource portions of the testing, enabling more "trade-space" during development.

Though the systems design was well documented the operational deployment requirements for the system was discussed during the Release Planning Phase but not formally specified. The need for an increase in the formal documentation of the operational deployment requirements was identified and thought that this formal documentation would facilitate a smoother transition of the systems to operations.

[5] The cost variance was due to a vendor renegotiation and was not attributed to the Framework or Practices.

Hardware Failures on Concurrently Developed Sprints

Juggernaut experienced a higher than usual failure rate on ordered hardware for the prototypes which forced the dynamic reallocation of some hardware components to development testing. Due to this reallocation they were unable to perform some of the tasks on concurrently planned Sprints and the tasks had to be consolidated due to this hardware limitation. In the future an increase in the number of hardware will be ordered to account for this failure rate and allow for the Sprints to run concurrently as planned.

Release Structure

It was felt that breaking apart the release into two separate releases would have enabled a quicker release to the field in addition to adding the ability to capture lessons learned from the first release and incorporate those in subsequent releases. This strategy does not mean that Juggernaut would have to wait to start the second release until the first release was successfully fielded, they would be able to overlap the development of the releases, so when the first release was fielded they would be into the development of the second release. This strategy would allow for the second release to be delivered faster while still retaining the ability to capture and incorporate lessons learned from the first release.

Chapter 5: Conclusions

The incorporation of the Agile Systems Engineering Framework and Practices showed an improvement in estimating the systems cost, schedule and functionality. Delivery of a successful system relies on more than just processes and practices. In addition to process, there is a technological and personnel aspect of system development which could be catastrophic if ignored. This research addresses the process aspect of system development, but recognizes the importance that people and technology play in overall system development.

The Agile Systems Engineering Framework and Practices do not remove typical project management issues encountered during systems development, but they enable early identification and resolution of issues. Juggernaut encountered many of the same issues faced by projects regardless of the Framework and / or Practices used during systems development. Issues include:

1) Scheduling priorities – Other project took priority in manufacturing, testing, or development;
2) Staffing issues;
3) Fluctuation in material costs;
4) Manufacturing lines shut down;
5) Delays in receiving ordered parts;
6) Retesting of subsystems during development due to unsuccessful QC.

Though Juggernaut faced these issues during development, by structuring their project into the specified Framework and using the agile Practices, they were able to identify, restructure and adapt to these issues with minimal impact to the overall project. Though some unanticipated

events caused minor schedule slips to occur, other scheduled milestones were completed ahead of schedule. One of the Juggernaut's initial "phase reviews" was completed ahead of schedule, which was noted by management as the "first time in company history".

Taking status meetings as an example, Juggernaut found three days a week was the optimal meeting time whereas another system may require daily meetings. The optimal meeting time was discovered through the periodic retrospectives held throughout development. At the system level one can easily overburden the team with daily meetings as there will probably be subsystem daily meetings as well. This needs to be accounted for when customizing the agile practices for a project. These meeting times also need not be static. The frequency of the meetings may need to be readdressed throughout the project. One may need to meet more or less at different times throughout the project. Avoid being caught in the rut of "what worked last time will work now." It may be a good starting point but it is important to readdress needs periodically.

Future Work

The case study described here was limited in scope. It measured the cost, schedule and functionality of system development from design to internal QC (Milestone 1). Overall usability was not measured. Additional research can be conducted to measure overall user satisfaction based on User Acceptance Testing (UAT) from Milestone 2 and overall system usability based on post Milestone 3 data.

As Juggernaut incorporates the Framework and Practices into more projects, these projects should be measured in a similar fashion to evaluate more accurately the total value of the agile systems engineering process. In addition, studies can be conducted to compare and

contrast the customization of the Framework and Practices on different projects within Juggernaut.

System engineering is a large domain with virtually infinite combinations of disciplines. This case study accounts for the incorporation of mechanical, firmware, manufacturing and software, but there are numerous other system engineering disciplines that could utilize the Framework and Practices. A natural extension of this research would be to apply the Framework and Practices to different systems engineering disciplines to measure if similar results are found.

In this study the measure of a "successful" system was based on three variables: cost, schedule and functionality. One could also research the effects on variables such as:

1) Team Satisfaction
2) User Satisfaction
3) Usability

Though the application was shown effective on a project with sixteen team members, the effectiveness on projects of varying team sizes could also be explored. Keeping similar system engineering disciplines, the Framework and Practices could be applied to projects of both smaller and larger scope to measure the scalability. As other potential system engineering frameworks and practices emerge, a model could be constructed to using data from this case study to assist in measuring their likelihood of success.

Bibliography

Adoption of ISO/IEC 15288:2002 Systems Engineering-System Life Cycle Processes. (2005). *IEEE Std 15288-2004 (Adoption of ISO/IEC Std 15288:2002)*, 0_1-67. doi: 10.1109/IEEESTD.2005.96287

Anderson, D.J. (2010). *Kanban*: Blue Hole Press.

Architecture and Systems Engineering in IT Acquisition. (2010). Paper presented at the Information Resource Management (IRM) 202 Course.

Axe, David. (2010). Marines' Instant Gunship Blasts Taliban, Pentagon Bureaucracy. Retrieved from Wired.com website: http://www.wired.com/dangerroom/2010/11/marines-instant-gunship-blasts-taliban-pentagon-bureaucracy/

Babar, M. A. (2009, 14-17 Sept. 2009). *An exploratory study of architectural practices and challenges in using agile software development approaches.* Paper presented at the Software Architecture, 2009 & European Conference on Software Architecture. WICSA/ECSA 2009. Joint Working IEEE/IFIP Conference on.

Boehm, Barry. (2000). Safe and Simple Software Cost Analysis. *Software, IEEE, 17*(5), 14-17. doi: 10.1109/52.877854

California, University of Southern. (1998). COCOMO II Model Definition Manual (pp. 37).

Capiluppi, A., Fernandez-Ramil, J., Higman, J., Sharp, H. C., & Smith, N. (2007). *An Empirical Study of the Evolution of an Agile-Developed Software System*. Paper presented at the Proceedings of the 29th international conference on Software Engineering.

Center, Department of the Air Force: Software Technology Support. (2003). Guidelines for Successful Acquisition and Management of Software-Intensive Systems: Weapon Systems Command and Control Systems Management Information Systems (D. o. t. A. Force, Trans.) *Condensed GSAM Handbook* (pp. 209). Hill AFB, Utah 84056.

Charette, Robert N. (2009). This Car Runs on Code. *IEEE Spectrum*, 12. http://spectrum.ieee.org/green-tech/advanced-cars/this-car-runs-on-code

Cianfrani, C.A., Tsiakals, J.J., & West, J. (2009). *ISO 9001: 2008 Explained*: McGraw-Hill.

Cohen, David, Lindvall, Mikael, & Costa, Patricia. (2003). A State of the Art Report: Agile Software Development (pp. 71): The University of Maryland.

Cohn, M. (2004). *User Stories Applied: For Agile Software Development*: Addison-Wesley.

Cordeiro, Lucas, Barreto, Raimundo, Barcelos, Rafael, Oliveira, Meuse, Lucena, Vicente, & Maciel, Paulo. (2007). TXM: an agile HW/SW development methodology for building medical devices. *SIGSOFT Softw. Eng. Notes, 32*(6), 4. doi: 10.1145/1317471.1317476

Dahlby, Doug. (2004). Applying Agile Methods to Embedded Systems Development. In I. ArrayComm (Ed.).

Daniels, Jody J. (2006). *Review of Acquisition for Transformation, Modernization, and Recapitalization.* (Master of Strategic Studies), U.S. Army War College.

. *Defense Acquisition Guidebook*. (2010). Ft. Belvoir, VA.

Defense, Under Secretary of. (2008). *Operation of the Defense Acquisition System*. Washington D.C.

Derby, E., Larsen, D., & Schwaber, K. (2006). *Agile Retrospectives: Making Good Teams Great*: Pragmatic Bookshelf.

Duygulu, Ethem, & Ciraklar, Nurcan. (2008). Team Eectiveness and Leadership Roles. 10.

Engineering, International Council on Systems. (2012). What is Systems Engineering? Retrieved 10/2012, 2012, from http://www.incose.org/practice/whatissystemseng.aspx

Ferguson, J. (2001). Crouching dragon, hidden software: software in DoD weapon systems. *Software, IEEE, 18*(4), 105-107. doi: 10.1109/MS.2001.936227

Ferreira, Carlos, & Cohen, Jason. (2008). *Agile systems development and stakeholder satisfaction: a South African empirical study*. Paper presented at the Proceedings of the 2008 annual research conference of the South African Institute of Computer Scientists and Information Technologists on IT research in developing countries: riding the wave of technology, Wilderness, South Africa.

Flyvbjerg, Bent. (2006). Five Misunderstandings About Case-Study Research. *Qualitative Inquiry, 12*(2).

Force, Defense Science Board Task. (2009). Department of Defense Policies and Procedures for the Acquisition of Information Technology (T. Office of the Under Secretary of Defense For Acquisition, and Logistics, Trans.) (pp. 109). Washington, D.C. 20301-3140.

Fowler, Martin. (2006, 05/01/2006). Continuous Integration. Retrieved 10/21/2012, 2012, from http://martinfowler.com/articles/continuousIntegration.html

Glas, Martin, & Ziemer, Sven. (2009). *Challenges for agile development of large systems in the aviation industry*. Paper presented at the Proceedings of the 24th ACM SIGPLAN conference companion on Object oriented programming systems languages and applications, Orlando, Florida, USA.

Glossary of Scrum Terms. (2007, 03/07/2012). Retrieved 10/21/2012, 2012, from http://www.scrumalliance.org/articles/39#1127

Government, Michigan State. (2007). Information Technology Equipment Life Cycle (pp. 4). Michigan.

Gross, J.M., & McInnis, K.R. (2003). *Kanban Made Simple: Demystifying and Applying Toyota's Legendary Manufacturing Process*: Amacom.

Group, Standish. (2009, 04/23/2009). Standish Newsroom - CHAOS 2009. *New Standish Group report shows more project failing and less successful projects.* Retrieved 01/01/2011, 2011, from http://www1.standishgroup.com/newsroom/chaos_2009.php

Hagan, Gary. (Ed.) (2011) (14th ed.). Fort Belvoir, Virginia 22060-5565: Defense Acquisition University Press.

Hajjdiab, H., & Taleb, A. S. (2011, 15-17 July 2011). *Agile adoption experience: A case study in the U.A.E.* Paper presented at the Software Engineering and Service Science (ICSESS), 2011 IEEE 2nd International Conference on.

IEEE Standard for Application and Management of the Systems Engineering Process. (2005). *IEEE Std 1220-2005 (Revision of IEEE Std 1220-1998)*, 0_1-87. doi: 10.1109/IEEESTD.2005.96469

Ikonen, M., Kettunen, P., Oza, N., & Abrahamsson, P. (2010, 1-3 Sept. 2010). *Exploring the Sources of Waste in Kanban Software Development Projects.* Paper presented at the Software Engineering and Advanced Applications (SEAA), 2010 36th EUROMICRO Conference on.

Ikonen, M., Pirinen, E., Fagerholm, F., Kettunen, P., & Abrahamsson, P. (2011, 27-29 April 2011). *On the Impact of Kanban on Software Project Work: An Empirical Case Study Investigation.* Paper presented at the Engineering of Complex Computer Systems (ICECCS), 2011 16th IEEE International Conference on.

Institute, Carnegie Mellon Software Engineering. (2007). Brief History of CMMI. 2.

ISO/IEC Standard for Systems Engineering - Application and Management of the Systems Engineering Process. (2007). *ISO/IEC 26702 IEEE Std 1220-2005 First edition 2007-07-15*, c1-88. doi: 10.1109/IEEESTD.2007.386502

Jefferies, Ron. (2012). What is Extreme Programming. Retrieved 10/21/2012, 2012, from http://xprogramming.com/book/whatisxp/

Jiangping, Wan, Weiping, Luo, & Xiaoyao, Wan. (2011, 13-15 May 2011). *Case study on Critical Success Factors of agile software process improvement.* Paper presented at the Business Management and Electronic Information (BMEI), 2011 International Conference on.

Kennedy, Matthew R., & Umphress, David A. (2011). An Agile Systems Engineering Process The Missing Link? *CrossTalk Magazine, 24*(3), 36.

Kennedy, Matthew R., & Ward, Dan. (2012). Inserting Agility in System Development. *Defense Acquisition Research Journal, 19*(3).

Kitchenham, B., Pickard, L., & Pfleeger, S. L. (1995). Case studies for method and tool evaluation. *Software, IEEE, 12*(4), 52-62. doi: 10.1109/52.391832

Kniberg, H., & Farhang, R. (2008, 4-8 Aug. 2008). *Bootstrapping Scrum and XP under Crisis A Story from the Trenches.* Paper presented at the Agile, 2008. AGILE '08. Conference.

Koehnemann, H., & Coats, M. (2009, 24-28 Aug. 2009). *Experiences Applying Agile Practices to Large Systems.* Paper presented at the Agile Conference, 2009. AGILE '09.

Krebs, J. (2008). *Agile Portfolio Management*: Microsoft Press.

Lapham, Mary Ann, Williams, Ray, Hammons, Charles (Bud), Burton, Daniel, & Schenker, Alfred. (2010). Considerations for Using Agile in DoD Acquisition (D. o. t. A. Force, Trans.) (pp. 83): Carnegie Mellon University.

Lauck-Dunlop, Penny. (2008). *Marketing War: A Case Study Comparison of Wars between the United States and Iraq.* (Doctor of Philosophy), Auburn University.

Lee, Jason Chong. (2006). *Embracing agile development of usable software systems*. Paper presented at the CHI '06 Extended Abstracts on Human Factors in Computing Systems, Montréal, Québec, Canada.

Leffingwell, D. (2007). *Scaling Software Agility: Best Practices for Large Enterprises*: Addison-Wesley.

. Life Cycle Systems Engineering. (2007) (Vol. AFI-63-1201, pp. 43): Secretary of the Air Force.

Liker, J. (2003). *The Toyota Way*: McGraw-Hill.

Manifesto for Agile Software Development. (2012). Retrieved 10/2012, 2012, from http://agilemanifesto.org/

. *Manual for the Operation of the Joint Capabilities Integration and Development System*. (2009). Washington D.C.

. *Manual for the Operation of the Joint Capabilities Integration and Development System*. (2012). Washington D.C.

Matthews, Charles E. (2011). *Agile practices in embedded systems*. Paper presented at the Proceedings of the compilation of the co-located workshops on DSM'11, TMC'11, AGERE!'11, AOOPES'11, NEAT'11, & VMIL'11, Portland, Oregon, USA.

Moe, N. B., & Aurum, A. (2008, 3-5 Sept. 2008). *Understanding Decision-Making in Agile Software Development: A Case-study.* Paper presented at the Software Engineering and Advanced Applications, 2008. SEAA '08. 34th Euromicro Conference.

Moe, N. B., Dingsoyr, T., & Dyba, T. (2008, 26-28 March 2008). *Understanding Self-Organizing Teams in Agile Software Development.* Paper presented at the Software Engineering, 2008. ASWEC 2008. 19th Australian Conference on.

National Defense Authorization Act for Fiscal Year 2010, Pub. L. No. 111–84 656 (2009 2010).

Nelson, Mike, & Clark, James. (1999). Curing the Software Requirements And Cost Estimating Blues. *Program Manager, 28*(6).

Northern, Carlton, Mayfield, Kathleen, Benito, Robert, & Casagni, Michelle. (2010). Handbook for Implementing Agile in Department of Defense Information Technology Acquisition (pp. 107): MITRE.

Office, United States General Accounting. (2004). Changing Conditions Drive Need for New F/A-22 Business Case (Vol. GAO-04-391, pp. 34). Washington, DC 20548.

One, Version. (2007). 2nd Annual Survey - "The State of Agile Development" (pp. 9).

Oxenham, D. (2010, 22-24 June 2010). *Agile approaches to meet complex system of system engineering challenges: A defence perspective.* Paper presented at the System of Systems Engineering (SoSE), 2010 5th International Conference on.

Palena Neale, Thapa, Shyam, & Boyce, Carolyn. (2006). PREPARING A CASE STUDY: A Guide for Designing and Conducting a Case Study for Evaluation Input. 16.

Patton, Jeff. (2009). Kanban Development Oversimplified. *How Kanban-style development gives us another way to deliver on Agile values.* Retrieved 10/25/2012, 2012, from http://www.agileproductdesign.com/blog/2009/kanban_over_simplified.html

Rico, D.F., Sayani, H.H., Sutherland, J.V., & Sone, S. (2009). *The Business Value of Agile Software Methods: Maximizing ROI with Just-In-Time Processes and Documentation*: J. Ross Pub.

Royce, Winston. (1970). *Managing the Development of Large Software Systems.* Paper presented at the IEEE WESCON.

Schwaber, K., & Beedle, M. (2002). *Agile software development with scrum*: Prentice Hall.

The SCRUM Process | SCRUM Framework. (2012). Retrieved 10/21/2012, 2012, from http://www.expertprogrammanagement.com/2010/08/the-scrum-process/

SEI Agile Research Forum: Meeting the Challenges of Large-Scale Systems. (2012). Retrieved 11/10/2012, 2012, from http://www.sei.cmu.edu/go/agile-research-forum/

Senge, P e t e r. (1990). *The Fifth Discipline - The Art and Practice of the Learning Organization*. New York, New York 10036: Doubleday.

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

Smits, Hubert. (2006). 5 Levels of Agile Planning: From Enterprise Product Vision to Team Stand-up (pp. 11): Rally Software Development Corporation.

Sommerville, Ian. (2004). *Software Engineering* (7th ed.): Addison Wesley.

Spaulding, Dale R. *CMMI - ISO "Can't we all just get along?"*.

Srinivasan, J., & Lundqvist, K. (2009, 27-29 April 2009). *Using Agile Methods in Software Product Development: A Case Study.* Paper presented at the Information Technology: New Generations, 2009. ITNG '09. Sixth International Conference on.

Sutherland, Jeff, Jakobsen, Carsten, & Johnson, Kent. (2007). *Scrum and CMMI Level 5: The Magic Potion for Code Warriors.* Paper presented at the Agile 2007, Washington D.C.

Swaminathan, B., & Jain, K. (2012, 17-19 Feb. 2012). *Implementing the Lean Concepts of Continuous Improvement and Flow on an Agile Software Development Project: An Industrial Case Study.* Paper presented at the AGILE India (AGILE INDIA), 2012.

Systems and software engineering -- Software life cycle processes - Redline. (2008). *ISO/IEC 12207:2008(E) IEEE Std 12207-2008 - Redline*, 1-195.

Technology, Massachusetts Institute of.). Keywords & Definitions for Interesting Organizations. *Keywords & Definitions for Interesting Organizations.* Retrieved 06/15/2010, 2010

Technology, National Institute of Standards and. National Vulnerability Database Retrieved 09/18/2010

Tellis, Winston. (1997). Introduction to Case Study. *The Qualitative Report, 3*(2).

Terlecka, K. (2012, 13-17 Aug. 2012). *Combining Kanban and Scrum -- Lessons from a Team of Sysadmins.* Paper presented at the Agile Conference (AGILE), 2012.

Trage, Sylvia. (2005). Electronics: Driving Automotive Innovation. *Pictures of the Future*, 1.

Underhill, P. (2008). *Why We Buy: The Science of Shopping--Updated and Revised for the Internet, the Global Consumer, and Beyond*: Simon & Schuster.

University, Carnegie Mellon. (2006). CMMI® for Development Version 1.2 (pp. 573). 2006: Software Engineering Institute.

Capability Maturity Model Integration (CMMI) Orientation, (2009).

Ward, Dan. (2010, 12/2010). The FIST Manifesto. *Defense AT&L*, 64.

Westmoreland, Joshua. (2009). *Scrum as Project Management in a Software Engineering Classroom Setting.* (Master of Science), University of West Georgia.

Wolak, Chaelynne M. (2001). *Extreme Programming (XP) Uncovered*. Information Systems. Nova Southeastern University. Retrieved from http://www.scisstudyguides.addr.com/papers/cwdiss725paper4.pdf

Yin, R.K. (2002). *Case Study Research: Design and Methods*: SAGE Publications.

Appendix A: Case Study Training Slides

Applying Agile Principles to Firmware/Hardware Development

Matthew R. Kennedy

Narration

- The next slides show the core values/principles of agile software development and the framework provided (Scrum).

Agile SW Development The Agile Manifesto

Values

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

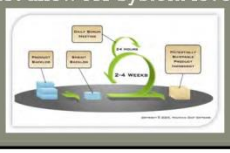
A few principles

- Our highest priority is to satisfy the customer through early and continuous delivery of valuable software.
- Welcome changing requirements, even late in development. Agile processes harness change for the customer's competitive advantage.
- At regular intervals, the team reflects on how to become more effective, then tunes and adjusts its behavior accordingly.

Standard Sprint Framework

Major Hurdles in Applying Scrum Framework to Firmware/Hardware

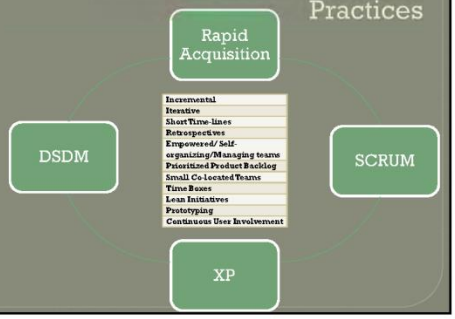
- Manufacturing process
- Requires integration testing prior to declaring a "Potentially Shippable" product
- Does not allow for system level planning



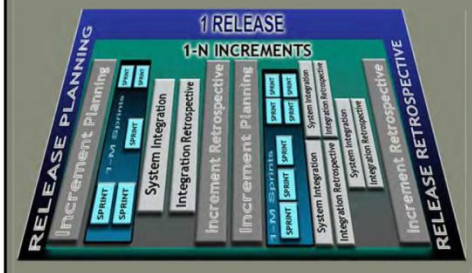
Narration

- The next slides show the common practices seen when implementing agile system / software development (hardware and / or software) and a system engineering framework.

Identifying Common Agile Practices



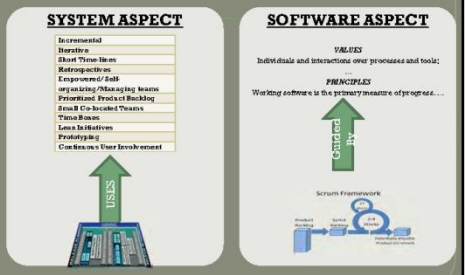
Agile System Engineering Framework



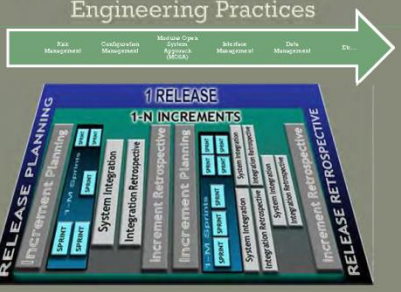
Addressing Hurdles in Applying Framework to Firmware/Hardware

- Manufacturing (Hardware) process
- Requires integration testing
- Does not allow for system level planning

Inserting Agility Into the System Aspect



Does Not Eliminate Core System Engineering Practices



Questions