Systems Engineering for Software Intensive Projects Using Agile Methods

Larri Rosser
Raytheon
Intelligence, Information and Services
Garland, TX
larri_rosser@raytheon.com

Phyllis Marbach
Boeing
Huntington Beach, CA
phyllis.r.marbach@boeing.com

Gundars Osvalds, CSEP
Praxis Engineering
Annapolis Junction, MD
gosvalds@praxiseng.com

David Lempia
Rockwell Collins
Cedar Rapids, IA
dllempia@rockwellcollins.com

Abstract. "Systems engineering is an interdisciplinary approach and means to enable the realization of successful systems" as defined in the INCOSE Systems Engineering handbook. When software development teams apply agile software methodologies such as Scrum, test driven development and continuous integration (collectively referred to as "Agile software development" hereafter); there are challenges in coordination with traditional systems engineering efforts. This paper, developed by the INCOSE Agile Systems Engineering Working Group, proposes methods for cross-functional teams that include Systems and Software Engineers working on mid-size (~80 contributors), customer "pull" projects to produce software products. This paper defines a proposed Agile SE Framework that aligns with agile software development methodology, and describes the role of the Systems Engineer in this context. It presents an iterative approach to the aspects of development (requirements, design, etc.) that are relevant to systems engineering practice. This approach delivers frequent releasable products that result in better customer alignment and the ability to absorb changes in mission requirements through collaboration between systems engineers and software engineers.

Introduction

Over a span of forty plus years, systems engineering has proven to be a value-added activity on complex software intensive projects.[1] Over the last decade and a half, agile software development

[1] A software-intensive project is defined as one in which software contributes essential influences to the design, construction and deployment of a project.

methodologies have offered a faster, leaner, and more flexible approach to developing software.[2] Systems Engineers (SE) and Software Engineers (SWE) have been challenged to integrate value added systems engineering activities into an agile software development approach. This challenge has been met most successfully on small projects, in which the necessary systems engineering activities can be owned by members of the development team and definition of capabilities and determination of system readiness can be handled by the customer and stakeholders with minimal formality.

Success in these small commercial environments encourages application of Agile software development to larger and more complex projects, and to those with different business models, such as Department of Defense (DoD) [U.S.] projects. Given the current economic climate and the U.S. government's fiscal challenges, the DoD and other federal customers are focused on lower costs and greater value for money. One of the banners on the *Better Buying Power* web site (DoD 2010) states, "Ensuring Our Nation Can Afford The Systems and Services It Acquires." The first Focus Area of Better Buying Power is to "Achieve Affordable Programs." Additional focus areas emphasize "Control Costs Throughout the Product Lifecycle" and "Eliminate Unproductive Processes and Bureaucracy." The Engineered Resilient Systems (ERS) initiative, one of the DoD Science and Technology Office's top seven priorities, focuses on creation of affordable, effective and adaptable solutions through faster implementation, reduced rework, better informed decision making and the support of a broader range of mission contexts. Both the acquisition and technical communities in the DoD are sending the same request: systems that meet their mission needs, quickly and affordably. The Agile Defense Adoption Proponents Team (ADAPT) is composed of industry and government representatives who are interested in advancing the adoption of Agile software development in DoD acquisition. ADAPT has published a White Paper *"Achieving Better Buying Power 2.0 For Software Acquisition: Agile Methods"* submitted for consideration to USD (AT&L), DoD CIO and DCMO (ADAPT 2013). This paper was written in response to the "*Better Buying Power*" challenge (DoD 2010).

While agile software development shows promise for providing more value at lower cost on DoD and federal programs, its application has not been without its challenges. DoD programs have a mature operational framework with long-standing practices and methods that are not well aligned with agile software development concepts. These projects often have expectations of formal milestones and an approach to delivery that are inconsistent with the agile software development approach.

With respect to the definition of stakeholder needs, two general models are identified. The "push" project, typical of commercial product development, is defined as one in which the enterprise plans, proposes and implements a product that is then released to the market. The "pull" project has a stakeholder or end customer who specifies the capabilities required and presents them to a contractor for implementation. For purposes of this paper, the challenges and best practices are examined as they apply to integrating systems engineering with the use of agile software development on "pull" projects for DoD or federal programs in the Engineering and

[2] This paper addresses agile software development **not** the development of systems that are designed to have agile capabilities.

Manufacturing Development (EMD) phase, with an assumed team size of approximately eighty people.

This paper summarizes a traditional systems engineering approach, and proposes how systems engineering can work on projects using agile software development. It describes some of the unintended consequences and undesirable effects that have been experienced when combining agile software development with formal systems engineering practices, and offers suggestions for overcoming them. An Agile SE Framework is introduced which consists of architecture, process, and roles describing the changes necessary to align SE and SWE in an agile methodology context. Finally, a list of "Challenges" is described along with "Enablers" identified from the Agile SE Framework that help resolve the identified challenges. Now software intensive projects using agile software development methods can pick and choose the enablers most important to their teams' success in working with SE.

Traditional Systems Engineering and DoD Acquisition

Historically, systems engineering provides value to projects in areas of cost, schedule and technical quality (Honour 2004). Systems engineering delivers value through a variety of activities, including technical management, mission and needs analysis, requirements articulation and management, system architecture and design, and technical analysis and trades (Frank 2000). Given this range of activities, systems engineering provides value to several stakeholders: the customer, the user, the program manager and the implementation team. SE works with stakeholders (customers and users) to articulate and prioritize needs, to coordinate prioritization and progress reporting between the implementation teams and the program office, to remove barriers, and to provide architectural focus and technical analysis to the implementation team. While acknowledging that the role of SE includes working with the customer and the program office, this paper will focus analysis and recommendations on the role of SE in supporting implementation in the context of an agile software development paradigm. This paper will address some of the technical processes described in the in *Systems and software engineering -- System life cycle processes* (ISO/IEC 2008) standard as presented in Figure 1 from the INCOSE *Systems Engineering Handbook* (INCOSE 2011). The technical processes addressed are: stakeholder requirements definition, requirements analysis, architectural design, implementation, integration, and verification.

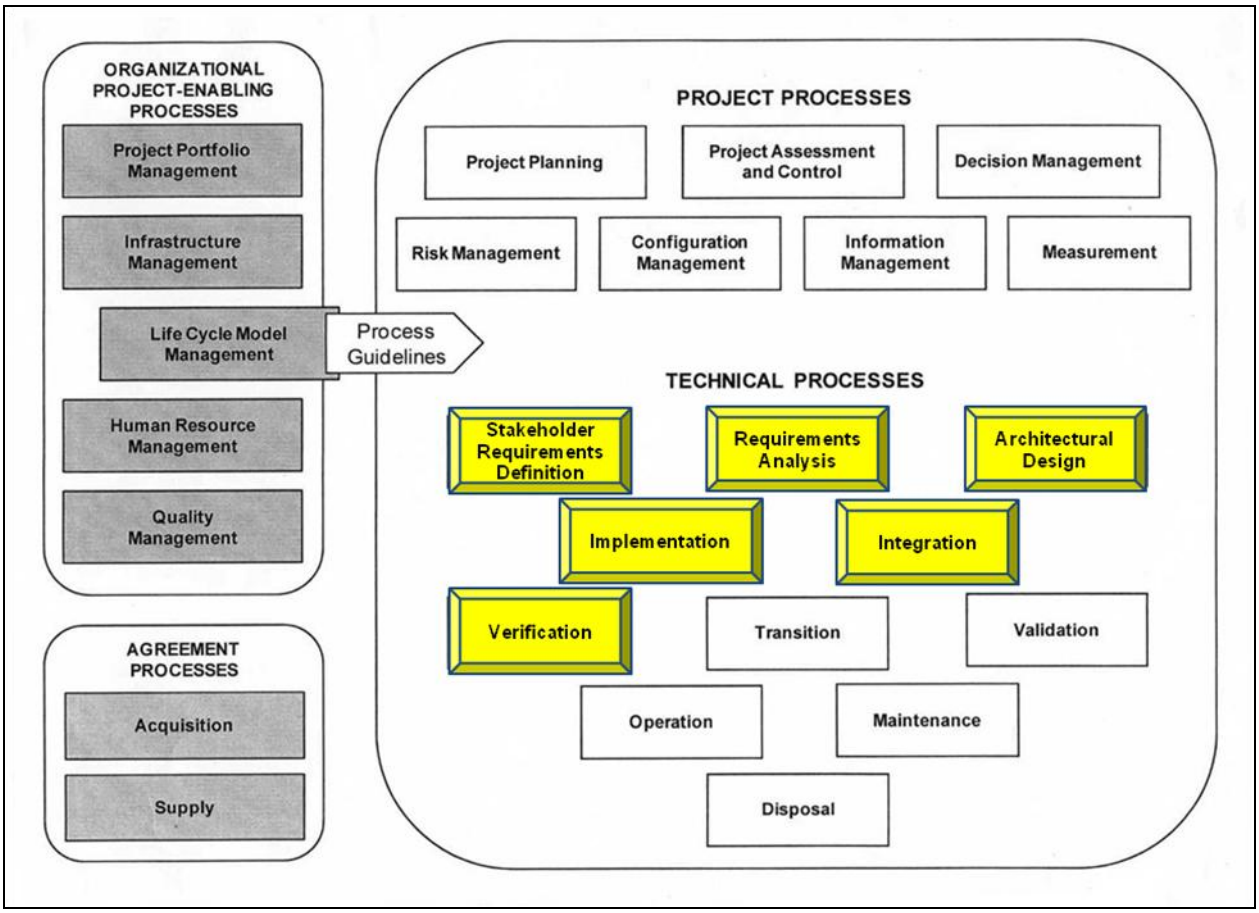

Figure 1. System Life-cycle Process Overview

The traditional DoD program uses a waterfall lifecycle model, in which phases of activity (needs definition, design, implementation, and test) occur sequentially for entire projects or large increments of capability. It is assumed that quality and efficiency are ensured by fully understanding the needs and completely specifying the solution before implementation begins. This approach is sometimes described as "Big Design Up Front" (BDUF). In this paradigm, it is an SE responsibility to obtain and document system needs from the stakeholders (customer, stakeholder, user, etc.) via requirements, operational concepts, workflows and similar artifacts. SE then develops the systems architectural designs and creates software specifications that are derived from the system requirements. EMD programs are the focus of this paper, although recommendations made will work with other acquisition phases or smaller projects as well. EMD programs begin at Milestone B as described in the *Defense Acquisition Guidebook* (DoD 2012) and depicted in Figure 2. At Milestone B requirements are defined for the system to be developed.

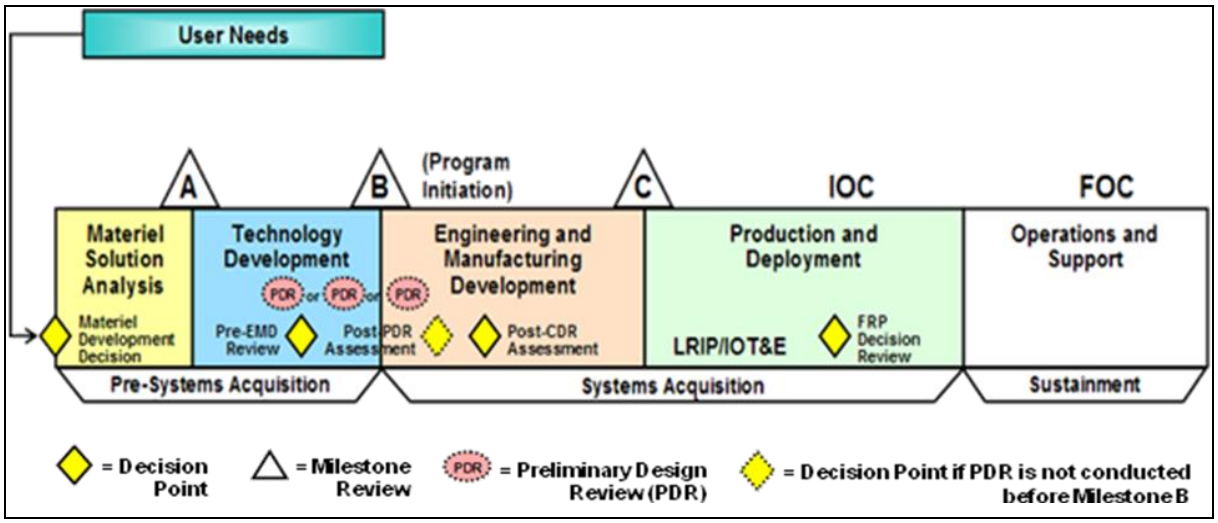

Figure 2. System Acquisition Framework

Traditionally, the systems team is responsible for meeting the stakeholder needs and developing the systems design. The software team is responsible for using the system specifications, architecture, and requirements to perform detailed design and develop the software. Workflows between these teams are defined as interfaces with system and software specifications output from the system team and input to the software team. This has been described as throwing one team's output "over the wall" to become another team's input. Thus feedback and coordination is limited to the detriment of the project. The next section describes a different way for systems team and software team to work together on a software intensive project.

The Agile SE Framework

The Agile SE Framework describes changes to the architecture, process, and roles, required to move from traditional SE processes to a SE process that augments the agile software development teams. An issue with current agile methodologies is that the system architecture is not part of the agile software development methodology strategy. When developing small systems software the architecture is the responsibility of the agile software development team. For larger systems there needs to be consideration for dependencies between the system capabilities and architectural elements. The SE must become a member of the Agile SE Framework based Implementation Team to anticipate the architecture support needed. It is imperative that the SE have the responsibility to identify and analyze architecture dependencies and create and continuously update an architecture that will provide the framework to support the software implementation (Brown et al 2010).

SE working with agile software development teams can apply the Agile SE Framework to select the enablers that best work in their situation. Specific changes to the traditional SE process are called Enablers. The Enablers are described in the Agile SE Framework and are detailed in the "Challenges and Enablers" section, later in this paper.

Role Changes

In traditional systems engineering approaches, the handoff from systems teams to agile software development teams is not always rapid and iterative. In an agile environment, the SWE and SE need to work together to define, implement, and test the project's capabilities. At the present time there is limited guidance from the agile software methodologies on how SE and SWE collaborate to design the systems capability. Therefore it is imperative that SE work as a team with SWE so that the overall system integrity is maintained using an iterative process in order that the SE continue to provide value within the construct of agile software methodologies.

Larger programs that have several agile software development based Implementation Teams working in parallel especially need SE engaged and providing value. The critical message to SE participating with projects using agile software development methodologies is: *the work tempo changes, but the SE work products still matter.* The SE focus on articulation and satisfaction of needs and on verification of capabilities and performance is as important as ever. The challenge is to carry out the essential work in agreement, rather than conflict with the agile software development teams. To achieve this end the systems engineering processes must be adapted to support the agile software design and development methods. In a waterfall model, the design and development teams only have one chance to get each aspect of a project right. In Agile methodologies for SE, every aspect of development (requirements, design, etc.) is continually revisited throughout the development lifecycle (Smith 2008). This paper proposes the Agile SE Framework to help shift the focus to "a flexible and holistic" software design and development strategy.

The team structure, timeline and roles described in Figure 3 and Table 2 illustrate the Agile SE Framework that provides for integrating systems engineering value with an agile software development approach. The Table contents are intended to be illustrative, not prescriptive to provide self-organizing teams the flexibility to manage their artifacts and activities. The desired outcome is to incorporate the value of both systems engineering activities and agile software methodologies into the project approach. This requires "just enough systems engineering" up front to provide a clear understanding of key performance parameters and robust system architecture. This upfront work should guide but not unnecessarily constrain the implementation, and should introduce minimal delays in starting implementation. Additional systems engineering activities should occur as part of the implementation iterations, with SE acting as full participants in the Agile SE Framework. The goal is to mature the requirements and architecture as the project proceeds, taking advantage of early iterations to add clarity before solidifying specific architectural features or sets of requirements. In the next section are documented challenges that some teams have experienced when traditional SE and SWE using agile software development methodologies work together in developing systems.

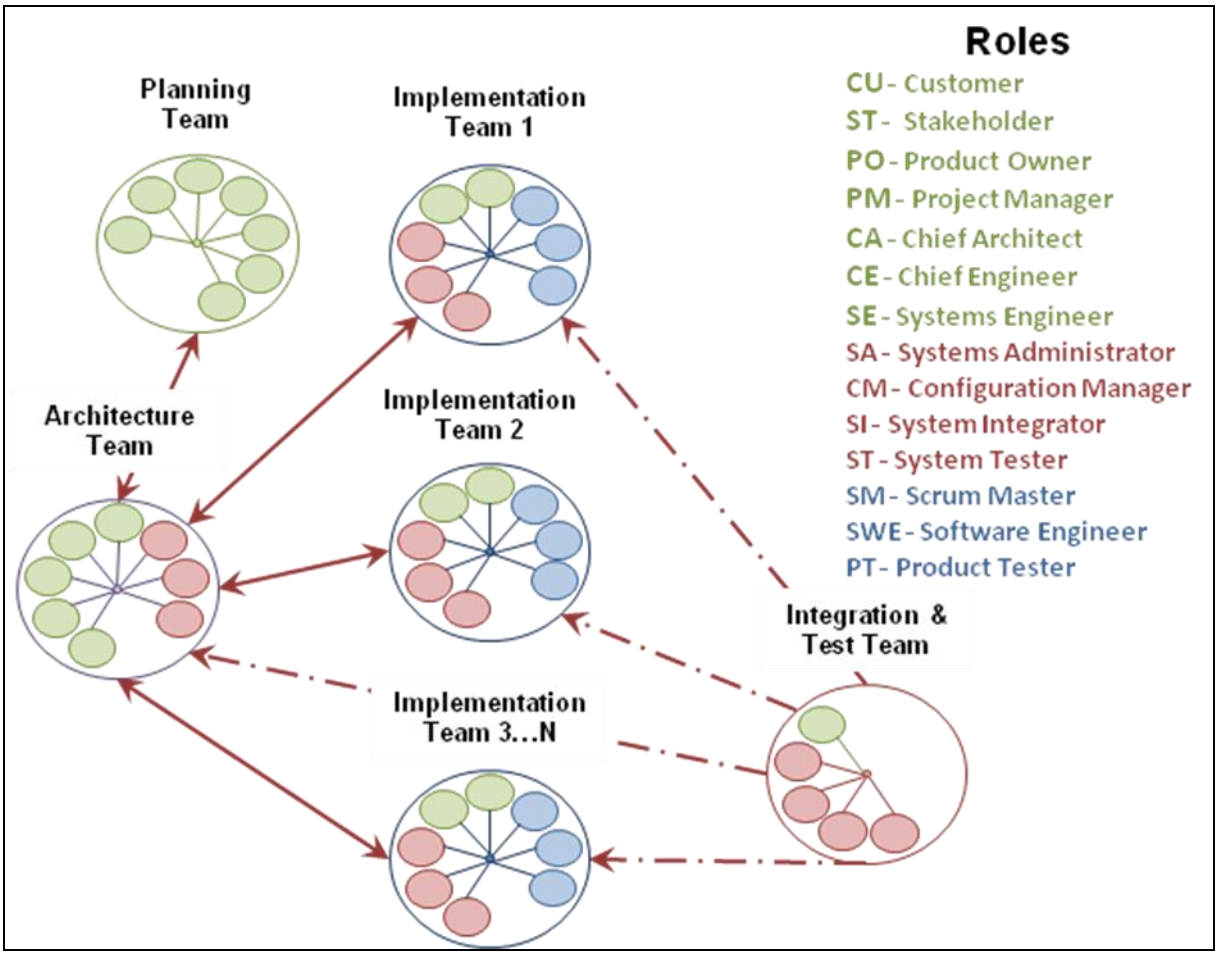

Figure 3. Agile SE Framework

The Responsibility Assignment Matrix also known as the RACI matrix[3] describes the project participants in various role in supporting the program, project or task. Table 1 is provided as an example of how a typical team self-organizes with example assignments.

Table 1. RACI Matrix Legend

Responsible	Does the work
Accountable	Approves and is responsible for assigning the work
Consulted	Subject Matter Expert
Informed	Need to be aware of the work status
Not Applicable	

[3] RACI = Responsible, Accountable, Consulted and Informed.

Table 2. RACI Matrix for Agile SE Framework

PLANNING TEAM							
ROLES	Customer	Stakeholder	Product Owner	Program Manager	Chief Architect	Chief Engineer	Systems Engineer
Develop Requirements	A	R	R	I	I	I	I
Analyze Requirements	I	A	C	I	R	R	C
Analyze Operations	C	A	C	C	R	R	C
Plan Project	A	C	C	R	C	C	C

ARCHITECTURE TEAM								
ROLES	Stakeholder	Product Owner	Chief Architect	Chief Engineer	Systems Engineer	System Admin	Config Manager	System Tester
CONOP	A	C	C	R	C	N/A	N/A	I
Architectural Design	A	C	R	R	C	I	I	I

INTEGRATION AND TEST TEAM					
ROLES	System Admin	Config Manager	Integrator	System Tester	Customer
Software Backup	R	A	N/A	N/A	N/A
System Integration	N/A	N/A	R	A	I
Validation	N/A	N/A	C	R	A

IMPLEMENTATION TEAM								
ROLES	Product Owner	Systems Engineer	Scrum Master	Software Engineer	Product Tester	System Admin	Config Manager	Integrator
Software Design	C	A	I	R	C	N/A	N/A	N/A
Software Implementation	A	C	I	R	C	C	C	C
Integration	I	C	I	A	C	C	C	R
Verification	A	C	I	C	R	I	I	C

Process Changes

In the agile software development process for an EMD project which begins at Milestone B the capabilities are defined for the system to be developed, as in the traditional process. Prior to starting any agile software development effort pre-planning is done, reference Figure 4 Step 1. During the pre-planning phase the Planning Team defines the scope and deliverables of the project. Next the Architecture Teams, Step 2, establish the vision, the roadmap, architecture, and a product backlog. The pre-planning period includes the technical management, mission and needs analysis, requirements articulation, requirements management definition, and architecture framework. Depending on the product in development this pre-planning could require anywhere from 3 days to 6 months or more. The input into this pre-planning step is capabilities and the output is a vision, roadmap, architecture framework, and a prioritized backlog of significant capabilities to be developed. When working with agile software development teams, the level of detail of the design artifacts needed to start the first implementation iteration may be less than what is normally produced on traditional life cycle projects. Some elements of the architecture or requirements may be identified for analysis and elaboration later in the implementation cycle. Depending on the level of formality of the project, outputs might include a concept of operations document (CONOP), planning artifacts, architecture diagrams and models, and a high level list of requirements.

Those outputs from the pre-planning phase will flow into the first iteration where they will be updated as the work is done on the highest priority capability in the iterations (in Scrum[4] Agile iterations are called Sprints). While one or more teams are working on the highest priority capability product (or software) backlog items, the Architecture Team will be working to define the requirements for the next highest level capability that software will develop in the next iteration. The Architecture Team members will also participate on the Implementation Teams to maintain the architecture as the detailed design evolves and help the SWE understand and align the software product to the proposed architecture and requirements.

It the Agile SE Framework, Implementation Team members include SE, SWE, Product Testers, and other cross-functional team members as needed for the product in development. When an implementation iteration is complete (Figure 4, Step 3) and deployed, the Architecture Team reviews (Step 4) the next implementation activity (Step 5) for adherence to the architecture and if needed revises it to provide an architectural framework for the next capability to be implemented. This sequence continues until the customer is satisfied with the capabilities of the system. During iterations, design artifacts or models are developed by the SE in collaboration with the Implementation Teams including: system capabilities, interface definitions, trades studies, detailed design representations, test procedures, and test reports or if MBSE (Model Based Systems Engineering) is being practiced the model products are being updated.

The SE may model the system functionality using *Model Based Systems Engineering (MBSE) Process Using SysML for Architecture Design, Simulation and Visualization* as described by

---

[4] **Scrum** is an iterative and incremental Agile software development framework for managing software intensive projects and product or application development. It is one of most used methodologies and was documented by the Schwaber, Ken; Beedle, Mike (2002) in the book *Agile Software Development with Scrum*. Prentice Hall. ISBN 0-13-067634-9.

(Osvalds 2011). MBSE uses capability statements as inputs and generates requirements, activity, sequence, block and state models that represent the systems capabilities. The model, when executed, can provide visual representation of the system operation. The model can be used for the customer to validate the design before and as the software is implemented. The change from traditional systems engineering is the level of maturity of the artifacts required to start implementation, coupled with planned maturation of the architecture and requirements as implementation progresses.

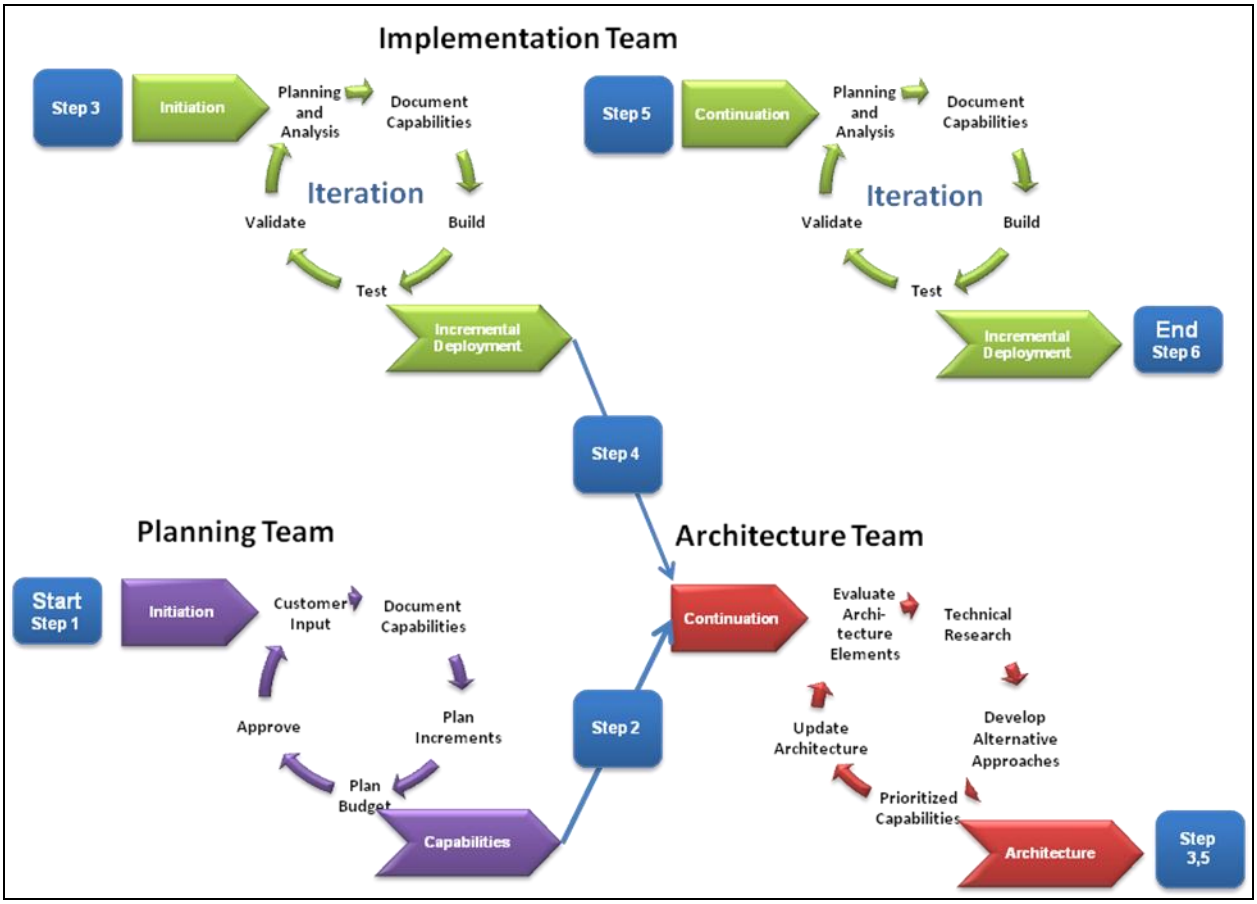

Figure 4. Agile SE Process

Architecture Changes

As described in a previous section, traditional SE often involves the BDUF. The architecture changes required to decompose the big design for agile software development teams involve identifying critical architecture choices that must be made up front, and creating a flexible architecture that is amenable to planned refinement as the implementation progresses. SE should treat this planned refinement as an opportunity to manage technical risk and benefit from technical and user evaluations made on the products of early iterations. SE is responsible for maintaining balance in the key quality attributes of the architecture, and also for adjustments to the architecture to maintain and improve its flexibility. The Architecture Team stays just ahead of the Implementation Team, incorporating lessons learned from the previous iteration as input to

refactor and refine the architecture, followed by developing new SE artifacts needed for the next iteration.

Challenges and Enablers

The traditional systems engineering model described in section "Traditional Systems Engineering and DoD Acquisition" contains some inherent limitations, an overview is described below:

Lack of Rapid Response. Lack of continuous interfacing between groups causes delays in resolving issues that invariably arise when interpreting and implementing the specification, or integrating elements of a system. Many intergroup interfaces, including both communication meetings and integration activities, are planned and scheduled meetings are set to a specific time interval. This can lead to significant delay in identifying and resolving issues.

Big Design Up Front. Creates delay in beginning implementation, and forces design decisions to be made early in the project, often with incomplete information and understanding of the problem space. Project management, may assume that changes in requirements and plans after an initial definition period are bad, and they work hard to limit changes. The risk is that the original specification is incomplete or immature and changes are required to best satisfy the stakeholder needs within the scope of the project.

Architecture Interpretation. The SE develops systems architecture plans which are provided to the SWE as documents. Their interpretation and application to the detailed SW design may vary from the original SE designed intent. Alternatively, additional information may arise within the implementation teams that would suggest changes to the architectural approach that SE is not aware of because they are not present with the implementation team.

Non-Functional Requirements (NFR) Interpretation. The SWE would not consider the quality attributes of system performance or behavior (i.e., "ilities" - reliability, speed, usability, flexibility, etc.) during design and implementation unless the NFR is included into the work planned. Also, if the SWE needs clarification the SE may not be available to help with information in a traditional process.

Responding to Change at Scale. When agile software development methodologies are successfully applied to a small project are then applied to a very large software development effort they may fail to scale, thus SE activities and products are not effectively used in implementation.

Verification, Validation and Test: Traditional SE practice for "pull" programs assumes that sell-off is based on verification of compliance with requirements not stakeholder (customer) satisfaction with deliverable functions which require validation that capabilities satisfy stakeholder needs. This can result in customer dissatisfaction that must be dealt with late in the program, when modification is most expensive.

The subsections below elaborate on the limitations described above between systems engineering activities and an agile software development team that the authors have experienced. The proposed solutions to these challenges are called enablers. The enablers summarize the Agile SE Framework changes previously described.

Lack of Rapid Response

When systems engineering activities are performed in isolation from software development teams, important systems engineering activities such as definition of key performance parameters, testing scenarios, and architecture principals, risk analysis and technical trades are not informed by or responsive to findings from the software development team.

Enabler. *Continual Interfacing* – A cross functional Implementation Team consisting of SE, SWE, and tester(s) co-develop one story/capability from concept through completed customer acceptance testing during an iteration. The cycle time between concept and completed testing is very short. Learning is fast. Risks in incorrect requirements are quickly eliminated. Implementation Teams have a solid foundation to build new capabilities as opposed to abstract changing concepts. Design as needed. Continuous communication through use of Scrum of Scrums meetings (where all teams are represented), internal demonstrations, and other shared events helps ensure rapid response to findings and issues. The integration strategy and a continuous integration environment is also planned and implemented early in the development.

Environment. Projects being developed iteratively.

Theory. Frequent communication during iterations both within and between teams, as well as, frequent builds and integration find errors and issues early. Errors in the definition of one capability do not propagate into other capabilities.

Big Design Up Front

When systems engineering activities are performed on a traditional schedule it is assumed that development will not begin until the Big Design Up Front (BDUF) is released. If the SE is "not finished" implementation is delayed or the software team may start to develop detailed design and code with no input from SE.

Enabler. *Capability Roadmaps* - Create a roadmap of capabilities to implement over time. From that roadmap create a prioritized backlog. Break down the capabilities until each high priority backlog item is sized so that it can be implemented in one iteration. Iterative planning allows the Implementation Team to start into development of the detailed design and coding with input from the SE (who is on the Architecture Team), because the capability roadmap is done and the detailed plan for the first (or next) high priority capability is also done.

Environment. This enabler applies to projects with a significant number of new capabilities or changes.

Theory. The roadmap provides a high level summary of the planned implementation. SE as part of the Architecture Team matures artifacts for each capability in sequence, just before they are addressed by the Implementation Teams. All Implementation teams focus on developing the same capability at the same time. This increases collaborative information flow between the teams.

Architecture Interpretation

When SE as part of the Architecture Team develops a detailed and comprehensive architecture and passes it over to the Implementation Team, software implementation opportunities and constraints are not adequately considered in systems engineering thus limiting flexibility; or, the Implementation Team proceeds without waiting for SE to provide the architecture design, leading to (at best) wasted effort and major variance between documentation and "as built." Furthermore, it could lead to poor implementation that result in excessive defects and a lack of evolvability. Not starting with a well-considered, flexible architecture can lead to suboptimal solutions that miss the benefits of a well thought out architecture.

Enabler. *Architecture Teams* - Architecture modularity and an iterative process requires architecture design effort throughout the development lifecycle. However, for large teams the

integrity of the architecture needs to be maintained as the development proceeds. A modular framework is sufficient to begin development. As the work proceeds there may be architectural epics, introduced in "*Agile Software Requirements"* (Leffingwell 2011), where the epic will be accomplished through multiple releases or the epic scope affects multiple products, or the epic will affect multiple teams or parts of the organization. The management of these epics of work is coordinated through the architecture team. SEs work between the implementation team(s) and the Architecture Team to update the system architecture

Environment. This solution works best when multiple Implementation Teams work in parallel to develop a solution.

Theory. Minimize defects by reducing communication misunderstanding at the handoff.

Non-Functional Requirements Interpretation

When quality attributes of system performance or behavior (i.e., "ilities" - reliability, speed, usability, flexibility, etc.) are not analyzed and tracked through design and implementation then the system may not perform as desired and confidence in the system's ability to perform as desired may be limited.

Enabler. *Include "ilities" into the Product Backlog Items* - Quality attributes are planned into each iterative development user story when a team plans and performs work on agile cross-functional Implementation Teams as described in the Agile SE Framework.

Enabler. *Product Lessons Learned* - After each iteration where design, implementation, and test are completed, the team captures lessons learned on the product. Lessons learned are the result of a completely implemented capability instead of an untested idea. Product lessons learned result in actionable items for a tools team to implement to improve the development and test engineering environment. Product lessons learned result in improved process, metrics, and checklists/job aids for the entire team to benefit from. Product lessons learned result in improved requirements, architecture, or understanding of the requirements or architecture. Each of these lessons learned are applied to the next iteration resulting in improved work environment immediately.

Environment. All lifecycle development efforts benefit from this enabler.

Theory. Studies show that >50% of product development is waste because requirements may be incorrect (Schwaber 2006). Lessons learned reduce waste and educate people on the best use of tools, process, and architecture.

Responding to Change at Scale

When agile software development methods have been used successfully on small projects are applied to a very large effort, the processes fail to scale and SE activities and products are not effectively used in implementation. Requirements may be interpreted differently by different Implementation Teams, architectural principles may not be universally applied, and interface definitions may develop gaps and overlaps.

Enabler. *Agile SE Scalability* - Larger teams need a team to integrate and test the products produced by the Implementation Teams. This team is depicted by the Integration and Test (I&T) Team in Figure 3. Dean Leffingwell, in "*Agile Software Requirements"* (Leffingwell 2011), calls this team the System Team. In the proposed Agile SE Framework described herein, SE are members of the Architecture Team, the Implementation Teams, and the I&T Team so the name

I&T Team is used rather than System Team to minimize the risk of confusion about team membership. In addition to the I&T Team, the Planning Team is needed to identify the prioritized list of capabilities to be developed by the Implementation Teams and the Architecture Team is needed to maintain the overall integrity of the architecture as the product and detail designs evolve.

Environment. This solution works best when multiple Implementation Teams work in parallel to develop a solution.

Theory. The I&T Team works on the same release goals as the Implementation Teams focusing on the highest priority capability being developed.

Verification, Validation and Test

Traditional SE practice for "pull" programs assumes that sell-off is based on Verification of compliance with requirements not stakeholder (customer) satisfaction with deliverable functions which require Validation that capabilities satisfy stakeholder needs. This can result in customer dissatisfaction that must be dealt with late in the program, when modification is most expensive.

Enabler. *Incremental Acceptance* - Leverage the Agile software development practice of continuous integration to create a situation in which stories are demonstrated, tested and even accepted as early as possible in the development cycle. Create tests from use cases, user stories and requirements before the system is designed or implemented. Share the testing artifacts with the customer to ensure a common understanding of the functionality to be developed. Strive to automate testing when each function, feature, and feature set is submitted. This allows standard execution paths of the feature or story to be tested automatically, with each build, ensuring that the feature isn't broken with later development and also freeing human testers to focus on exploratory testing. Test first development results in developing just what is being testing and meets the requirement. This has been found to also improve quality.

Environment. Projects with complex and/or emerging needs/requirements.

Theory. Agreement on test procedures with customers enhances understanding of expectations and customer acceptance of delivered features. Software written to pass an existing test will be more compartmentalized, easier to test and less likely to contain extra features. Incremental testing and acceptance reduces the level of effort required to fix problems late in the development cycle and also levels the effort load for SE, testers and customer representatives.

Conclusion

Over the last decade and a half, agile software development methodologies have offered a faster, leaner, and more flexible approach to developing software. The challenge to complete traditional SE activities has been met most successfully on small, projects, in which the necessary systems engineering activities can be owned by members of the development team, and where definition of needs and determination of system readiness can be handled by the customer and stakeholders with minimal formality. When scaling up to more complex projects with multiple teams, formal milestones and cross team dependencies exist, challenges have been realized. The Agile SE Framework provides a way to resolve the challenges experienced when coupling systems engineering practices with an agile software development approach.

On software intensive projects this Agile SE Framework is proposed as a way for SE and SWE to work together more closely evolving the work products iteratively. This paper proposes that SE

develop "just enough" architecture and requirements prior to the beginning of implementation, and then work on cross-functional Implementation Teams to maintain integrity of the requirements and architecture, while evolving them as development proceeds. The role of the SE within the Implementation Teams, the Architecture Team and/or the I&T Team includes customer and stakeholder requirements definition, requirements analysis, architectural design, implementation, integration, and verification. These duties are performed as part of the Agile SE Framework during the iterations and releases. This approach, for software intensive projects, will deliver frequent releasable software products that result in less waste, lower cost and higher quality. The iteratively developed products enable better customer satisfaction and provide the ability to absorb changes in mission requirements through team collaboration.

References

ADAPT 2013. *"Achieving Better Buying Power 2.0 For Software Acquisition: Agile Methods."* The Agile Defense Adaption Proponents Group of the The association for Enterprise Information. http://www.afei.org/WorkingGroups/ADAPT/Pages/default.aspx

Brown Nanette, Nord Robert, Ozkaya Ipek. 2010. "*Enabling Agility Through Architecture*." CrossTalk. http://www.sei.cmu.edu/library/assets/whitepapers/brown-nord-ozkaya-crosstalk-Nov10.pdf

DoD. 2010. "*Better Buying Power.*" Department of Defense. https://acc.dau.mil/CommunityBrowser.aspx?id=289207&lang=en-US

DoD. 2012. *Defense Acquisition Guidebook*. Department of Defense. https://dag.dau.mil/

Frank M. 2000. *Cognitive and Personality Characteristics of Successful Systems Engineering,* INCOSE International Symposium Proceedings.

Honour, Eric. C. 2004. *"Understanding the Value of Systems Engineering.* INCOSE International Symposium. http://www.incose.org/secoe/0103/ValueSE-INCOSE04.pdf

INCOSE. 2011."*INCOSE Systems Engineering Handbook*."International Council on Systems Engineering, v3.2.2. http://www.incose.org/ProductsPubs/products/sehandbook.aspx

ISO/IEC. 2008. "*15288 Systems and software engineering -- System life cycle processes*." ISO (International Organization for Standardization) and IEC (International Electrotechnical Commission). http://www.iso.org/iso/iso_catalogue/catalogue_tc/catalogue_detail.htm?csnumber=43564

Leffingwell, Dean. 2011. *Agile Software Requirements, Lean Requirements Practices for Teams, Programs, and the Enterprise*. Pearson Education, Inc. , Boston, MA. http://deanleffingwell.com/book-agile-software-requirements/

Osvalds, Gundars. 2011. *"Model Based Systems Engineering (MBSE) Process Using SysML for Architecture Design, Simulation and Visualization.*" NASA, Goddard, Greenbelt, MD. http://ses.gsfc.nasa.gov/ses_data_2011/110301_Osvalds.pdf

Schwaber, Carey. 2006. "*The Root of the Problem: Poor Requirements.*" IT View Research Document, Forrester Research. http://www.techworld.com/cmsdata/whitepapers/5104/ot_forrester_rootproblem_wp.pdf

Smith, Gregory. 2008. *"Agile Methodology Blog.*" CollabNet. http://agilemethodology.org/

Acknowledgements. The INCOSE Agile Systems Engineering Working Group acknowledge the support of all members and especially the reviewers: Mike Coughenour, Lockheed Martin; Rick Dove, Paradigm Shift; Wayne Elseth, CSEP, Praxis Engineering; David Fadeley, ESEP, Henggeler Consulting; Dr. Denise Haskins, CSEP, Kimmich Software; Dr. Suzette Johnson, Northrop Grumman, Robert Angier, IBM (Retired).

Biography

Larri Rosser is a Raytheon Certified Architect in Raytheon Intelligence, Information and Services. She has 30 years industry experience in aerospace, defense and computing technology, multiple patents in the area of human-computer interface and a BS in Information Systems and Computer Science from Charter Oak State College. Currently, she holds the role of Systems Engineering, Integration and Test lead for the DCSG-A family of programs, where she practices Agile Systems Engineering with a cross functional team.

Phyllis Marbach is a Senior Software Manager in Boeing's Defense Space & Security (BDS). Phyllis has over 30 years experience in aerospace programs; including Satellites, chemical lasers, the International Space Station, and various propulsion systems. Currently she is the project manager with Boeing's Lean-Agile Software Services (LASS) and an active BASP Coach for Unmanned Air Systems, Radio, and research programs. Ms Marbach holds an MS degree in Engineering from UCLA.

Gundars Osvalds, CSEP and Certified Scrum Master is a Principal Systems Engineer with over 40 years of experience is currently at Praxis Engineering. He provides systems engineering support to DoD programs ranging from Research and Development to large scale transformation utilizing Information Technology with architecture design, architecture framework development, engineering process creation, model based system design, agile software and engineering design. He is an author of 6 papers and 11 presentations on systems and architecture engineering.

David Lempia is a Principal Systems Engineer in the Engineering Infrastructure Development & Lean organization of Rockwell Collins with over 20 years of experience in systems development. He is currently leading lean for the EID&L organization and working as a skilled modeler. As a skilled modeler he leads workshops, develops best practices and develops training materials for Rockwell Collins. He is an author for papers on Requirements Engineering Management, Model Based Development, and Lean.