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 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 the ability to absorb changes in mission requirements through collaboration between systems engineers and software engineers. The Agile SE Framework defines a way to scale agile from individual agile software teams with a few members to large projects that require a planned architecture and coordinated efforts.
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 methodologies have offered a fast, lean, and 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 the 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 the 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). It 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 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, this paper identifies two general models. The “push” project, typical of commercial product development, is 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

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

\(^2\) This paper addresses agile software development not the development of systems that are designed to have agile capabilities.
This paper examines the challenges and best practices 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 Manufacturing Development (EMD) phase, with large teams.

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. It introduces an Agile SE Framework which consists of architecture, process, and roles describing the changes necessary to align SE and SWE in an agile methodology context. A list of “Challenges” and “Enablers” are identified from the Agile SE Framework. The “Enablers” resolve the identified challenges. A survey given to the INCOSE Agile SE working group provides insight into the level of industry usage of the identified enablers. About ⅔ of the survey participants indicated benefits from using Agile. 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 focuses analysis and recommendations on the role of SE in supporting implementation in the context of an agile software development paradigm. This paper addresses some of the technical processes described in the in *Systems and software engineering — System life cycle processes* (ISO/IEC 2008) standard as used by the INCOSE *Systems Engineering Handbook* (INCOSE 2011). The technical processes addressed are: stakeholder requirements definition, requirements analysis, architectural design, implementation, integration, and verification.

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. 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, the SE obtains and documents 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. This paper focuses on Engineering and Manufacturing Development (EMD) programs, although
recommendations made will work with other acquisition phases or smaller projects as well. “The primary objective of the EMD phase is to develop the product baseline, verify it meets the system functional and allocated baselines, and transform the preliminary design into a producible design” consistent with the Defense Acquisition Guidebook (DoD 2012). Requirements are defined at the beginning of Milestone B for EMD programs as shown in Figure 1.

![System Acquisition Framework Diagram](image)

**Figure 1. System Acquisition Framework**

Traditionally, engineers define and interpret stakeholder needs and develop the system design, which software engineers subsequently use to develop detailed designs and then implemented capabilities. The interactions between SE and SWE are not real-time, iterative or informal, which limits flexibility and responsiveness to change. This paper describes a different approach for systems and software engineers to work together on software intensive projects, called the Agile SE Framework.

**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 an explicit part of the agile software development methodology. When developing small software intensive systems the architecture design 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 developed by different implementation teams. The SE must participate on the Agile SE Framework based Implementation Team(s) to anticipate the architecture support needed. In this Framework it is the SEs’ responsibility to identify and analyze architecture dependencies and create and continuously update an architecture description that will provide the framework to support the software implementation. (Brown et al 2010).

---

3 From Data Acquisition Guidebook published by US Department of Defense. 2012
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.

Architecture Changes
Traditional SE often involves the Big Design Up Front activity. The architecture changes
required to decompose the big design for agile software development teams involve identifying
critical architecture choices that must be made at the start of a project, and creating a flexible
architecture that is amenable to planned refinement as the implementation progresses. Having a
flexible, modular architecture is an important enabler for iterative development. The 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. The 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 to support changes. In the next section we
introduce how 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 to support the next iteration.

Process Changes
An EMD project begins at Milestone B with the definition of capabilities for the system to be
developed. In traditional systems engineering approaches, the handoff from systems teams to
agile software development teams is not typically rapid and iterative. However, in an agile
environment, the SE and SWE need to work together to define, implement, and test the project’s
capabilities to more effectively inspect the results and adapt subsequent iterations to maintain
overall system integrity. The Chaos Manifesto defines the agile process: “is based on iterative
development, where requirements and solutions evolve through collaboration among self-
organizing, cross-functional teams. The agile process encourages frequent inspection and
adaptation, teamwork, self-organization, and personal accountability. The agile process allows
for rapid delivery. In theory, it hopes to align development closer to the users’ needs by
continuous collaboration and delivery of a concrete product.” (Standish Group 2013).

Larger programs that have several agile software development based 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. It is proposed that the use of agile teams, Figure 2, help shift the focus to
"a flexible and holistic” design and development strategy. The team member roles will be
described in more detail in the next section.
Figure 2. Agile Teams

The process flow of the teams introduced in Figure 2 is shown in Figure 3. At the start, (Figure 3, Pre-planning) the planning and the architecture teams define the capabilities for the system to be developed. The pre-planning period includes the technical planning, mission and needs analysis, requirements articulation, requirements management definition, and architecture framework creation. Input into this pre-planning step includes customer needed capabilities and the output is a vision, roadmap, architecture framework, and a prioritized backlog of significant capabilities to be developed. During the pre-planning phase the Planning Team defines the scope and deliverables of the project, while the Architecture Team, establishes the vision, the roadmap, architecture, and a product backlog. 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 since the customer will be involved by providing feedback during the program development cycle. 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, the planning stage outputs could include a Concept of Operations Document (CONOP), planning artifacts, architecture diagrams and models, and a high level list of requirements.
The outputs from the pre-planning phase will flow into the first iteration. The Architecture Team updates the capabilities backlog (Figure 3, Iteration 1) and then prepares materials the implementation teams will develop in the next iteration. These materials can include requirements, architecture, capabilities, and user experience. 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. Concurrently the Implementation Teams work on the highest priority capability product (or software) backlog items and the Integration and Test Team implements the test environment. In subsequent iterations the just completed product from all implementation teams will be tested by the Integration and Test Team, thereby providing the ability to verify the requirements either internally or with the customer, if permitted.

In the Agile SE Framework, Implementation Team members include SE, SWE, Integrators, Product Testers, and other cross-functional team members as needed for the product in development. This cross-functional makeup and purpose of the Implementation Team is similar
to the Integrated Product Teams (IPTs) as defined in the INCOSE *Systems Engineering Handbook*. During iterations, design artifacts or product models are developed and maintained by the SEs on the Implementation Teams. Other responsibilities of the SE are the system capabilities, interface definitions, trades studies, detailed design representations, test procedures, and test plans.

When the Architecture Team determines that changes to the architecture are needed then work will be performed to revise the architecture to support upcoming capability development. This work could include trade studies, architecture revision, prototyping, and database revision, to name a few. The goal is to provide a modular systems architecture that is resilient to change. The iterations continue until the release is complete, (Figure 3, Iteration N, Release 1). The Release is demonstrated to the customer/stakeholder and to the Planning team for review and acceptance. Requested changes may be planned into the next release. At the demonstration of the final Release the SEs verify the requirements are met.

**Role Changes**

The team members described in Table 1 fulfill the various roles listed in Figure 2 Agile Teams. It’s important on self-organizing teams that all needed skills listed in the figure, Program Member Roles, are represented among the Agile Team Members. The team members share the responsibility for products in development by the teams. The Agile SE Framework 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” during pre-planning to provide a clear understanding of key performance parameters and robust system architecture. The pre-planning 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.

A RACI matrix is a type of responsibility assignment matrix which is used to document tasks, activities, milestones, or decisions in the support of a program, project, or task by team members and to clarify expectations on their level of participation. Each team member is designated a role in the RACI hierarchy, an acronym that stands for (Responsible, Accountable, Consulted, Informed). Table 1 is provided as an example of how a typical team self-organizes with example assignments. SE will perform various tasks as an Agile Team Member depending on their skill set and the product in development.
Table 1. RACI Matrix for Agile SE Framework

**PLANNING TEAM**

<table>
<thead>
<tr>
<th>ROLES</th>
<th>Product Owner</th>
<th>Scrum Master</th>
<th>Team Member</th>
<th>Customer/ Stakeholder</th>
</tr>
</thead>
<tbody>
<tr>
<td>Scope</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Define Deliverables (Product Level)</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Technical Management Mission</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Needs Analysis</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Requirement Articulation (Product Capability Backlog)</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Requirements Management</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Meeting Facilitator/ Impediment Remover</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

**ARCHITECTURE TEAM**

<table>
<thead>
<tr>
<th>ROLES</th>
<th>Product Owner</th>
<th>Scrum Master</th>
<th>Team Member</th>
<th>Customer/ Stakeholder</th>
</tr>
</thead>
<tbody>
<tr>
<td>Vision</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Roadmap</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Architecture Framework/ System Design</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Define and Maintain Interfaces</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Architecture Product Backlog</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Concept of Operations (CONOP)</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Perform Trade Studies</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Meeting Facilitator/ Impediment Remover</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

**INTEGRATION AND TEST TEAM**

<table>
<thead>
<tr>
<th>ROLES</th>
<th>Product Owner</th>
<th>Scrum Master</th>
<th>Team Member</th>
<th>Customer/ Stakeholder</th>
</tr>
</thead>
<tbody>
<tr>
<td>Software Backup/ SW Baseline Test Baseline</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>System Integration</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Validation</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Meeting Facilitator/ Impediment Remover</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
The next section documents challenges that some teams have experienced when traditional SE and SWE using agile software development methodologies work together in developing systems.

**Challenges and Enablers**

Attempting to mesh traditional SE methods with agile software development on DoD style (pull) projects often results in challenges, 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 infrequently planned and scheduled meetings. 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 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).** Non-functional requirements or quality attributes (i.e., “ilities” — reliability, speed, usability, flexibility, etc.) may not be completely revealed and captured in agile user stories or backlog items, which leads to a lack of focus on them in design, implementation and test.

**Responding to Change at Scale.** Agile software development methodologies that work well for small projects may fail to scale for large, multi-team projects, leading to ineffective integration of SE activities and products into 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 responses to the challenges described above based on the authors experience and the industry body of knowledge. 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 trade studies 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 team coordination 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 tasks, introduced in “Agile Software Requirements” (Leffingwell 2011) as epics, where the work will be accomplished through multiple releases or the scope affects multiple products, or the architecture work will affect multiple teams or parts of the organization. The management of the architecture work is coordinated through the architecture team. SEs work between the implementation team(s) and the Architecture Team to update the system architecture.

**Enabler. Model Based Systems Engineering (MBSE)** — MBSE replaces the traditional document based architectures and becomes a single source of architecture design and interface information for implementation teams. MBSE uses capability statements and stakeholder needs as inputs to generate requirements, interfaces and design that represent the systems capabilities.

**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.

Survey. About 50% of survey responses evolve the architecture as the Implementation Team works and about 30% of the survey responses develop the architecture up front.

Non-Functional Requirements

The agile paradigm addresses functional requirements as backlog items or user stories. However, common agile practices do not directly address non-functional requirements. When quality attributes (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.

Environment. All lifecycle development efforts benefit from this enabler.

Theory. Studies show that >37% of product development is waste because requirements may be incomplete, i.e. missing NFR. The Standish Group’s “Chaos Report” found that the top three project impairment factors across 352 companies and 8,000 projects were lack of user input (12.8% of respondents), incomplete requirements and specifications (12.3%), and changing requirements and specifications (11.8) a total of 37% (Standish Group 1994).

Survey. About 50% of survey responses actively managed the “ilities” as a backlog item for either the architecture team, the implementation team, or both.

Responding to Change at Scale

When agile software development methods, 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 2. 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 designs evolve.

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 from which the entire team benefits. 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.** 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. Lessons learned reduce waste and educate people on the best use of tools, process, and architecture.

**Survey.** Capabilities are prioritized using a roadmap by 50% of the respondents, backlog lists by 25% of the respondents, or the schedule work breakdown structure by 20% of the respondents.

---

**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. This requires Validation that capabilities satisfy stakeholder needs. Late Validation 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.

**Survey.** In the majority of the survey responses the testing is manual and is done at the end of the sprint. About 30% of the teams practice automated integration and/or test.
Conclusion

Agile software development methodologies offer a fast, lean, and flexible approach for developing software, but have challenged the traditional approach to SE activities. The most successful applications have been on small projects, where the systems engineering activities are handled within the team with minimal formality. When scaling up to large projects with multiple teams, formal milestones and cross team dependencies, integration of systems engineering is more challenging but essential.

The Agile SE Framework described in this paper offers a way for SE and SWE to work together more closely on software intensive projects, evolving the work products iteratively as is typically done on agile projects. This paper proposes that SE develop “just enough” architecture and requirements prior to the beginning of implementation. The cross-functional Implementation Teams includes the SE who maintains the integrity of the requirements and architecture. The requirements and architecture evolve as development proceeds. The role of the SE within the Implementation Teams, the Architecture Team and the Integration and Test Team includes customer and stakeholder requirements definition, requirements analysis, architectural design, implementation, integration, and verification. This approach will deliver the frequent releasable products and flexibility to absorb mission requirements changes enabled by an agile approach. The Agile SE Framework strengthens the focus on requirements, architecture, system design and validation provided by systems engineering on large scale agile projects.

References


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.