29th Annual INCOSE international symposium
Orlando, FL, USA
July 20 - 25, 2019

Agile Systems Engineering Life Cycle Model for Mixed Discipline Engineering

Rick Dove
Paradigm Shift International
Anthem, AZ USA
dove@parshift.com

Bill Schindel
ICTT System Sciences
Terre Haute, IN USA
schindel@ictt.com

Copyright © 2019 by Rick Dove and Bill Schindel. Permission granted to INCOSE to publish and use.

Abstract. This article focuses on six recent understandings that have crystalized from INCOSE's Agile Systems Engineering Life Cycle Model (ASELCM) discovery project. The ASELCM project analyzed effective agile systems engineering processes in 3-day structured analysis workshops hosted by organizations that wanted deeper understandings of what works, why it works, and how to apply those understandings across a broader base. Four case studies arising from these workshops have been published. This article cannot cover everything of interest that was discovered, but will expose six key findings: a problem-space characterization heuristic, an asynchronous/concurrent life cycle model framework, a set of nine systems engineering operating principles, an encompassing pattern of three concurrent behavioral systems operating simultaneously in agile systems engineering, the concept of information debt, and general agile systems engineering response requirements.

Introduction

Systems complexity is increasing and requirements stability is decreasing. Waterfall based systems engineering processes are failing to design, build, and deploy effective systems within time and cost constraints. The requirements for an effective system continue to change during the engineering process; partly because complex systems requirements are insufficiently understood initially, but more so because the target environment and knowledge of it continues to evolve independently during design and development.

The constant evolution of the system's operational environment requires constant coevolution in the system's functional capabilities. Simply put, system development never ends, system operation must accommodate frequent capability evolution, and retirement is now an issue of obsolete functional capability disposal rather than total system decommissioning.

The expectation is that some sort of agile systems engineering is required; but what does that mean? Is there one such sort? Some would have us believe that the Agile (Software Development) Manifesto and the software development principles behind it are the answer. Some others go further down that software path, believing that branded software development practices, such as

Scrum[1] or SAFe® [2] (Scaled Agile Framework), provide a sufficient framework for agile systems engineering – akin to a single master recipe for all manner of dinner preparation, in any cuisine.

This short article provides conceptual considerations for practitioners who will design or improve an agile systems engineering process; and considerations for the next revision of the ISO/IEC/IEEE standards 15288 and 27748-1, and the INCOSE SE Handbook. A more detailed treatment awaits in-process completion of the formal INCOSE product unconstrained for length.

Codified systems engineering currently relies on the 15288 standard (ISO/IEC/IEEE 2015), the V diagram (Forsberg, Mooz 1991), defense acquisition policies (country specific), and companion guidance-for-practice in the INCOSE SE Handbook (INCOSE 2015). As most frequently interpreted, these have provided a somewhat unified *one-sort* of systems engineering.

Can agile systems engineering coexist compatibly with these current mainstays of understanding and guidance? Before this question can be answered, an understanding of what the phrase *agile systems engineering* means is necessary – at the encompassing conceptual level, not at the procedural or best practice level. This starts with succinct statements of need and intent.

Need: Effective system engineering in the face of uncontrolled change.

Intent: Effective response to a systems engineering operational environment that is capricious, uncertain, risky, variable, and evolving (CURVE). This intent defines agile systems engineering.

Uncontrolled change encompasses uncontrollable change, but is not limited to that which can't be controlled – just what isn't controlled regardless of reason. The word *effective* means that value is realized from resource employment. At one extreme, a project canceled before completion should provide valuable learning and artifacts for later employment. At the other extreme, a deployed system should provide sustainable relevance beyond a break-even ROI. In the middle, responding to changes in the engineering operational environment should sustain innovative forward progress. The word *should* is used to indicate a necessary objective for an agile systems engineering process: timely and sustained value delivery.

Incremental and iterative development alone do not provide the necessary agility in the system of interest. Incremental implies successive feature development, iterative implies successive improvement cycles on a feature. Both of these techniques occur in waterfall. Both of these techniques can be done with little or no effect on agility in the system of interest. They *can* contribute greatly to the agility of the system engineering process, if their practiced purpose and outcome is timely learning applied to work in process.

This article is focused on six key discoveries that have crystalized from INCOSE's Agile Systems Engineering Life Cycle Model (ASELCM) project. The project analyzed effective agile systems engineering processes in 3-day structured analysis workshops. Four case studies arising from these workshops have been published. (Dove, Schindel, Scrapper 2016, Dove, Schindel 2017, Dove, Schindel, Hartney 2017, Dove, Schindel, Garlington 2018). This article cannot cover everything of interest that was discovered, but will introduce six primary findings for agile systems engineering:

---

[1] Schwaber, Ken and Jeff Sutherland. 2013. *The Scrum Guide*. www.scrum.org.

[2] Leffingwell, Dean. 2007. *Scaling Software Agility: Best Practices for Large Enterprises.* Addison-Wesley Professional. SAFe and Scaled Agile Framework are registered trademarks of Scaled Agile, Inc.

1) a driving problem-space characterization, 2) an asynchronous/concurrent life cycle model framework, 3) a set of operational principles, 4) a pattern of three behavioral systems operating simultaneously, 5) the concept of information debt, and 6) general systems engineering response requirements.

Before discussing these six findings, some relevant background knowledge is necessary.

Prior Relevant Knowledge

This article is focused on life cycle operational aspects of agile systems engineering, which rely on capabilities enabled by a common conceptual architecture for both the systems engineering process and the system being engineered. As a prelude, a review of enabling architecture is presented.

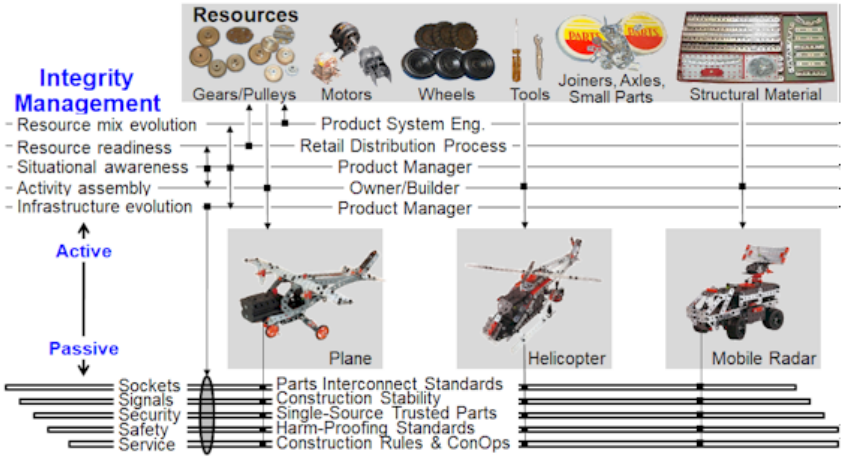

Figure 1. Agile Architecture Pattern (AAP) construction kit concept

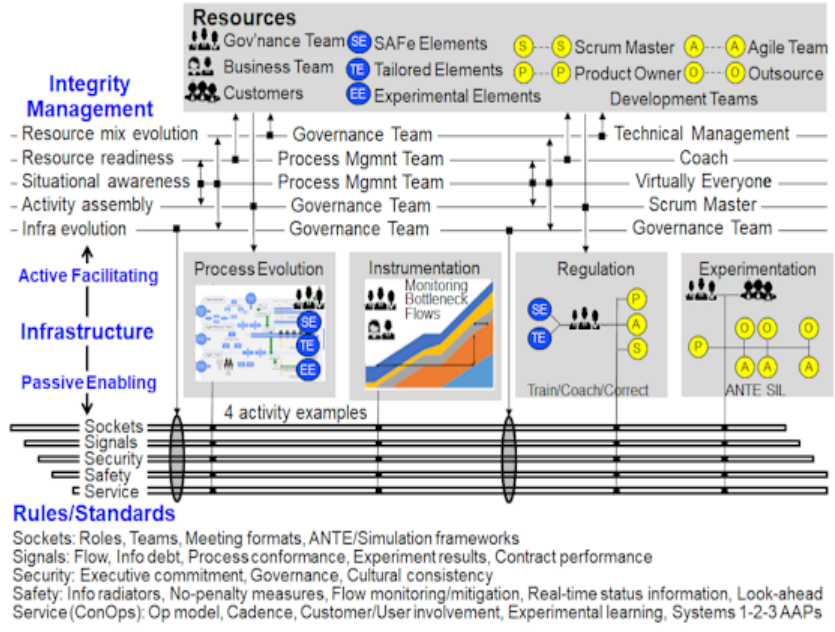

Figure 2. Lockheed Martin Integrated Fighter Group (Dove, Schindel, Garlington 2018)

We are all familiar with architectures that accommodate and facilitate structural change. Think of the construction sets we grew up with: Erector set®, Tinker Toy®, and Lego®, to name some of the classics. Each of these construction sets consists of different types of components, and constraints on how these components can be connected and interact. Though each construction set is better suited to some types of constructions than others, they all share a common architectural pattern, shown as Figure 1 Erector Set example.

Figure 2 depicts the AAP shown in the case study from Lockheed Martin Aeronautics Integrated Fighter Group (Dove, Schindel, Garlington 2018). See paper for details.

There are three critical elements in the agile architecture pattern (AAP): a roster of drag-and-drop encapsulated resources, a passive infrastructure of minimal but

sufficient rules and standards that enable and constrain plug-and-play interconnection, and an active infrastructure that designates five specific responsibilities for sustaining agile operational capability.

It is important to understand and respect the intimate dependence an agile systems engineering process has on the agility of the system it builds. Operating in an uncontrolled environment is necessarily about real-time awareness, learning, and employment of that learning. If employment of that learning is inhibited by a non-agile system, the agility of the process is diminished. This requires a target system amenable to change and evolution as it is being developed, as well as after it is deployed.

Both process and product need AAP.

Agile SE processes deal with and learn knowledge and environment changes.

- This learning is employed during SE process operation.
- Product work-in-process is modified/augmented.
- The product's AAP enables affordable work in process modification/augmentation.

Agile software development processes (silently) rely on product AAP.

- Program code development employs an object-oriented AAP development platform (e.g., C++, Java, Eclipse).
- Web code development employs a loosely-coupled modular AAP inherent with hyper-linked web-pages.

Agile hardware development doesn't have off-the-shelf AAP tools – but:

- Proprietary product-line-engineering employs AAP.
- Proprietary Open System Architecture (OSA) employs AAP.
- Proprietary Live-Virtual-Constructive platforms employ AAP.
- FPGA technology provides electronic hardware realization from virtual models.
- "It from bit" of additive manufacturing translates virtual models to material things.

Central to agile systems-engineering is continuous learning and subsequent modification of both work process and work in process, as new knowledge is revealed. Consequently it is counter-productive to employ an agile systems-engineering process that doesn't develop and evolve an agile system that can facilitate time- and cost-affordable modification of work in process.

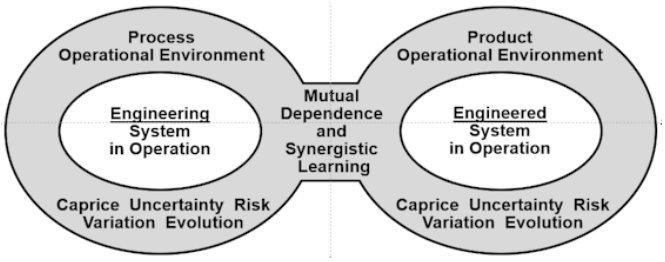

Figure 3. Two different operational environments requiring agile counterpoint for the systems they encompass.

Figure 3 depicts this mutual dependency of agile systems-engineering and engineered agile-systems. The family of agile software development practices typically obtain this necessary relationship by employing an off-the-shelf object-oriented development platform, such as Eclipse[3]. Although hardware development practices have not yet embraced an equally extensive

[3] Eclipse (software). Wikipedia, https://en.wikipedia.org/wiki/Eclipse_(software).

change-enabling development platform, related opportunities are available in early commercial interoperable model-based systems engineering platforms from PTC and Dassault (and others), in prototype research sponsored by the Systems Engineering Research Center (Blackburn 2018a, 2018b), and in variations of so-called live-virtual-constructive (LVC) platforms that can intermix human, simulated, prototyped, and instantiated system components (Dove, Schindel, Garlington 2018).

Finding 1: CURVE Framework Characterizing the Problem Space

What follows is a heuristic framework for characterizing uncontrolled internal and external environmental forces that impact process and product as systems. This framework, referred to as CURVE, is useful for characterizing the problem space in which either system will exist, and for which the systems should have appropriate solution-space capability.

- Caprice: Unknowable situations. Unanticipated system-environment change.
- Uncertainty: Randomness with unknowable probabilities. Kinetic and potential forces present in the system.
- Risk: Randomness with knowable probabilities. Relevance of current system-dynamics understandings.
- Variation: Knowable variables and associated variance ranges. Temporal excursions on existing behavior attractors (a reference to complex system behavior trajectories).
- Evolution: Gradual successive developments. Experimentation and natural selection at work.

You might wonder how a solution space can be prepared to respond to a capricious (unpredictable) situation. Think about the ingredients available in a well-stocked kitchen run by a creative chef. The employment combinations of these ingredients and the amounts of each used in any combination are uncountable. Most possible combinations will never be used. But the possibility to concoct an appropriate combination is available when the unanticipated situation arises. This may be an opportunity for the proactive creation of a new signature dish; or a need for creative reaction when a diner with an unanticipated but declared allergy attends; or the Queen comes to dinner with an out-of-repertoire request. Note that it is not necessary to characterize the environment comprehensively or precisely – if enough characterization is present to allow preparation sufficient to accommodate that which is not itemized.

Fleshing out the uncontrolled problem space in the CURVE framework is a necessary first step toward developing effective agile response requirements. Developing response requirements uses another heuristic framework, referred to as Response Situation Analysis, which will not be detailed here as it has been around for some time and is adequately covered in (Dove, LaBarge 2014).

The four ASELCM case studies provide the CURVE environment characterized by each workshop Host. An example from the Lockheed case study (Dove, Schindel, Garlington 2018) provides an example:

Caprice: Unknowable situations

- Urgent pre-emptive customer needs, sometimes called Quick Reaction Notice events
- Changes in business environment, e.g., congressional funding, legal requirements
- Project scope change

Uncertainty: Randomness with unknowable probabilities

- Effectiveness of process tailoring
- Contract/customer compatibility with agile approach
- Management support/engagement in agile approach
- Team-member engagement with agile approach

Risk: Randomness with knowable probabilities

- Cultural incompatibility
- Ability to keep and attract talent
- External stakeholder schedules (e.g. certification)
- Systems of Systems requirements changes

Variation: Knowable variables and ranges

- Multiple-project resource conflicts (e.g. test facilities, key people)
- Subcontractor development compatibility
- System of Systems integration integrity
- Requirements of differing importance levels

Evolution: Gradual Successive Development

- OSA/OMS emphasis
- Customer mission needs
- New compelling technology availability

Finding 2: Agile Systems Engineering Life Cycle Model Framework

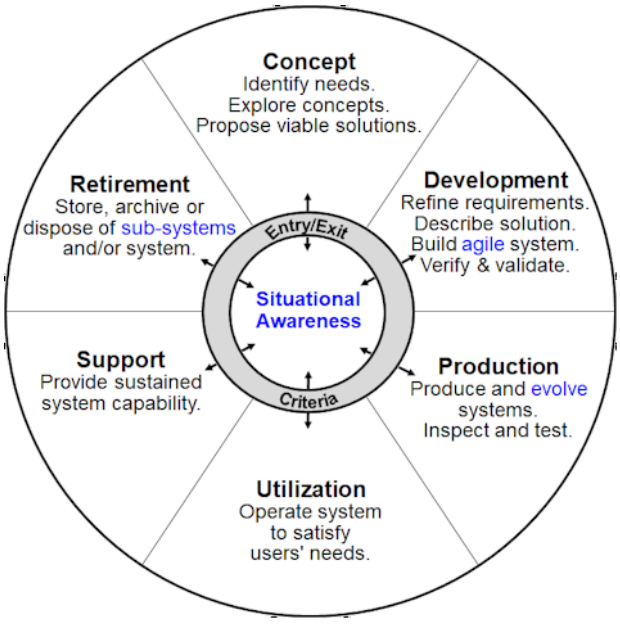

Figure 4. Purposes for each Life Cycle Model stage, adapted from (ISO/IEC/IEEE 2018, p 17), adding Awareness stage and blue highlights.

Asynchronous and concurrent life-cycle stage and process activity, as shown in Figure 4, is a hallmark of effective agile systems engineering processes.

ISO/IEC/IEEE 24748-1 Systems and Software Engineering — Life Cycle Management — Part 1: Guide for Life Cycle Management (ISO/IEC/IEEE 2018) recognizes six "commonly encountered" system life cycle stages. Figure 4 adds a seventh life-cycle stage, *Situational Awareness*, as a critically necessary element of an effective agile systems engineering life cycle model, as discovered in the ASELCM project.

Counter to the implication that a progression through stages is sequentially expected, the 24748-1 standard (ISO/IEC/IEEE 2018, p 36) states clearly that asynchronous and simultaneous activity in any and all stages is

within expectations: “Each system life cycle process ... can be invoked, as required, at any point, and at multiple points, throughout the life cycle and there is no definitive order or time sequence in their use. ... An individual system life cycle is thus a complex system of processes that will normally possess concurrent, iterative, recursive and time dependent characteristics. These statements are true for both a system-of-interest and any enabling systems. ... The changing nature and complexity of the influences on the system (e.g. operational environment changes, new opportunities for system element implementation, modified structure and responsibilities in organizations) requires continual review of the selection and timing of process use. Process use in the life cycle is thus dynamic, responding to the many external influences on the system.”

Unlike sequential-stage waterfall, agile systems engineering processes may be conducting activities in any and all of the stages concurrently without progressive sequencing.

The agile life cycle model framework features the addition of a central Situational Awareness seventh stage. The Awareness stage attends to active external and internal situational awareness – and engages all of the other stages. A distinguishing feature of effective agile systems engineering, previously unrecognized explicitly for its fundamental driving role, is proactive awareness of situational opportunities and risks to process and product alike, throughout the systems engineering process and the target system’s life cycle.

This proactive awareness and subsequent mitigation activity breathes life into the agile systems engineering process, taking it beyond a repetitive execution of traditional development increments fulfilling a backlog of planned features.

The agile life cycle model framework accommodates perpetual evolution beyond initial delivery, and requires that the Development stage produces an agile system-of-interest in order to support evolution. The Situational Awareness stage is the starting point for all other stage activity, and includes the original awareness that a new system is needed. Situational awareness is closely related to Attention Management and Optimal Estimation and Control, discuss later.

The Situational Awareness stage has no entry or exit criteria, as it should, in principle, be a continuous activity. Entry criteria for all of the other stages begins with a decision to act upon specific triggering awareness, and may require prudent, or contract required, minimal engagement criteria for a stage or stages to be entered. An agile systems engineering process is predicated upon real-time experimentation and learning in all stages, and as such, the entry criteria may be as simple as the decision to enter a stage for experimental knowledge development that may or may not produce artifacts for use in other stages. On the other hand, exit criteria for a stage that produces artifacts for use in other stages should have some fixed requirements for satisfactory stage completion, with recognition that the outcome of stage activity may simply be valuable learned knowledge that aborts the need for producing artifacts of use in other stages.

Another consideration, the 24748-1 standard presents the Retirement stage as a terminal stage: “Store, archive, or dispose of system (ISO/IEC/IEEE 2018, p 17).” The agile life cycle model framework expects that subsystems and older system versions are retired frequently, as the “current” system evolves. This has new implications for maintenance, disposal, and reversion processes.

Fleshing out a generic Agile SE Life Cycle Model likely starts with default processes in each stage, per (ISO/IEC/IEEE 2018), tailored and augmented for specific agile SE differences generally noted earlier. Adapting the generic model to a specific organization's process will tailor and augment the generic model as the organization's evolving document of record.

For a software system application example see the (Dove, Schindel, Kenney 2017) case study that generally runs around the circle sequentially every six months with the next evolution of the system-of-interest, while simultaneously and asynchronously invoking specific stage processes in production, support, utilization, and retirement for system versions in certification and operation. For a mixed-discipline example see the (Dove, Schindel, Hartney 2017) case study viewed from the product line engineering perspective.

Finding 3: Operational Principles

Fundamentally the purpose of agility is to survive and thrive in a CURVE environment. Doing so requires that a CURVE event is sensed and responded to in a timely fashion, and that both CURVE characterization and SE process capabilities evolve with the environment, The architecture and structural principles that enable agility have been covered in detail elsewhere (Dove, LaBarge 2014). Here the concern is with principles that facilitate operational agility in action. Current knowledge has three categories in an SRE framework that encompass nine principles:

Sensing:
- External awareness (proactive alertness).
- Internal awareness (proactive alertness).
- Sense making (risk analysis, trade space analysis).

Responding:
- Decision making (timely, informed).
- Action making (invoke/configure process activity to address the situation).
- Action evaluation (verification and validation).

Evolving:
- Experimentation (variations on process ConOps).
- Evaluation (internal and external judgement).
- Memory (evolving culture, response capabilities, and process ConOps).

Sensing and Responding mirror John Boyd's OODA loop (Boyd nd), with Observe and Orient encompassed by Sensing, and with Decide and Act encompassed by Responding. The Evolving category wasn't overlooked by Boyd, but rather the ultimate purpose of his OODA loop. He valued the necessity to learn and improve the practice of OODA looping. It is instructive to understand that the accomplished OODA loop practitioner is not running through a sequence of four activities in incremental repetition, but rather engaged in all four activities simultaneously.

In Boyd's application of the OODA loop to fighter pilot dog-fight engagement, he recognized a human cognitive activity, and expected an increasing learned intuition. Putting this roughly in neuroscience terms, the brain is a pattern learner and recognizer that drives motor action. Increased learning experience drives these functions away from reasoning and closer to direct and immediate motor control. Action becomes systemically autonomic – the ultimate objective of Evolution in the SRE operational principles.

Note the difference between Plan-Do-Check-Act (Deming nd) and OODA. PDCA has a sequential procedural feel, OODA has an in-the-moment dynamic-engagement feel. OODA is focused on awareness-driven re-evaluation of the changing problem space, rather than marching to convergence on a plan.

Explaining how things work has a reductionist quality to it. It can be off-putting to think that a new model needs to be learned and an old comfortable model abandoned. There is nothing new here. It is the natural way we navigate through life. It is, however, a new way of appreciating where the cul-de-sac of artificial, seemingly logical, approaches have taken us.

Finding 4: ASELCM Pattern of Three Concurrent Systems

Agile systems engineering encompasses three nested concurrent systems, depicted in Figure 5 as an iconic pattern (Schindel, Dove 2016).

The ASELCM Pattern establishes a set of system reference boundaries which are independent of whether the systems of interest are small or large, human or inanimate, flying through the air or performing business processes.

This ASELCM Pattern in Figure 5 particularly refers to three major system reference boundaries, and within those, six subsystem reference boundaries. These are all logical boundaries (defined by the behavior, not the identity, of systems), and are depicted by the iconic diagram in Figure 5.

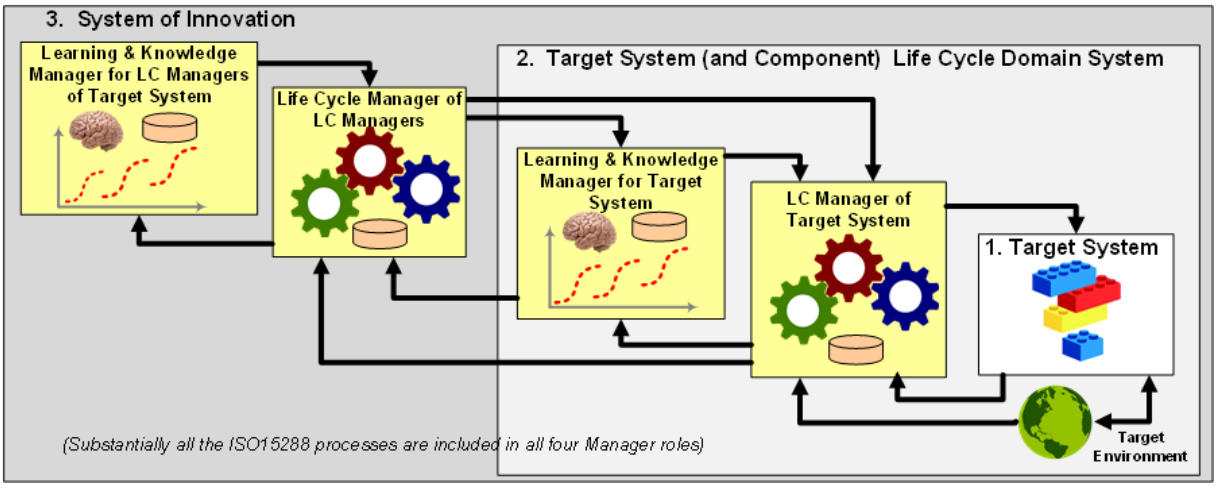

Figure 5. Iconic view of the ASELCM Pattern reference boundaries (Schindel, Dove 2016).

- System 1: The Target System, the subject of innovation over managed life cycles of development, deployment, and support.
- System 2: The Target System Life Cycle Domain System, including the entire external environment of the Target System—everything with which it directly interacts, particularly its operational environment and all systems that manage the life cycle of the Target System. This includes the external environment of the operational target system(s), as well as all the (agile or other) development, production, deployment, support, security, accounting, performance, and configuration management systems that manage System 1.

- System 3: The System of Innovation, which includes System 1 and 2 along with the systems managing (improving, deploying, supporting) the life cycle of System 2. This includes the systems that define, observe, analyze (as in agile software process retrospective), improve and support processes of development, deployment, service, or other managers of System 1.

System 1 is contained in System 2, which is contained in System 3. All are (or at least should be) performing simultaneously, effectively an organic complex system motivated by self-preservation to evolve suitably in an uncontrolled operational environment. Think of the arrow-pointed pipes of Figure 5 as a circulatory system – not as channels for intermittent communication; but rather as pipes with constantly circulating information fluid. This circulatory system brings nourishing information and also provides regulation – policing the effectiveness, removing the dysfunctional, correcting the crippled.

Finding 5: Concept of Information Debt

The ASELCM Pattern differentiates *adaptation* (managed change in the performance) and *learning* (information gain by a learner). Adaptation can occur (within limits) without new learning, as when a flexible aircraft systems architecture is already in place and is exploited by a managing system, to rapidly adapt performance characteristics in response to environmental changes. By contrast, learning is illustrated by accumulating new information about potential aircraft system architectures and environmental threats. The same difference can be seen in even single cell living systems, whose existing DNA is variably expressed to adapt to environmental changes, without additional "learning" accumulated in changes to DNA over evolutionary time.

By "learning", we mean accumulation of experience in the form of information, and by "adaptation" we mean change in performance using only what is already known. So, an already well-informed system may, without learning new things, demonstrate agile adaptation within a given envelope, but advancing beyond that envelope demands learning new information – a form of "debt" analyzed in the ASELCM Discovery Project and described in the next sections.

Managing Information Debt: Balance Sheet Model of Learning

"Debt" has a specific meaning in finance, while use of the term in agile methods (technical debt) has been quantitative but not the whole picture of the "balance sheet" analogy across life cycles.

Information Debt was added to the ASELCM Pattern's stakeholder features (Schindel and Dove 2016), expressing the difference between the information currently available and the information needed to (not only deliver but also) support the life cycle of a system, and this includes uncertainty. It is in contrast to the better-known technical debt - "the extra development work that arises when code that is easy to implement in the short run is used instead of applying the best overall solution (Techopedia nd)." Information debt as an explicit concept helps us address the perceived tension between Agile Software Development methods and traditional Systems Engineering methods—but also an earlier and more basic challenge of justifying systems engineering of any kind.

Figure 6 (a) reminds us of the familiar (to systems engineers, if not others) fact of life—during the early project stages of lower accumulated cost, most of the future costs of a project become committed, by decisions (explicit or implicit) of a systems engineering nature. This is one of the

traditional arguments for early stage systems engineering investment. Figure 6 (b) adds the idea of information debt, which is the not-yet-generated information necessary to deliver and sustain the system, and illustrates three different scenarios of information debt reduction scenarios. As pointed out by (Thomas 2016), there are effective "interest" costs paid by projects that don't pay off their information debt early enough, and the higher the risks involved, the greater the interest rate penalty to be expected. Scenario 3 of Figure 6 (b) illustrates a case of particular worry to traditional systems engineers considering agile methods: Does the Agile Manifesto mean that the project will end with remaining information debt outstanding, leaving us with a "working system" but an ongoing interest penalty caused by a shortage of needed information?

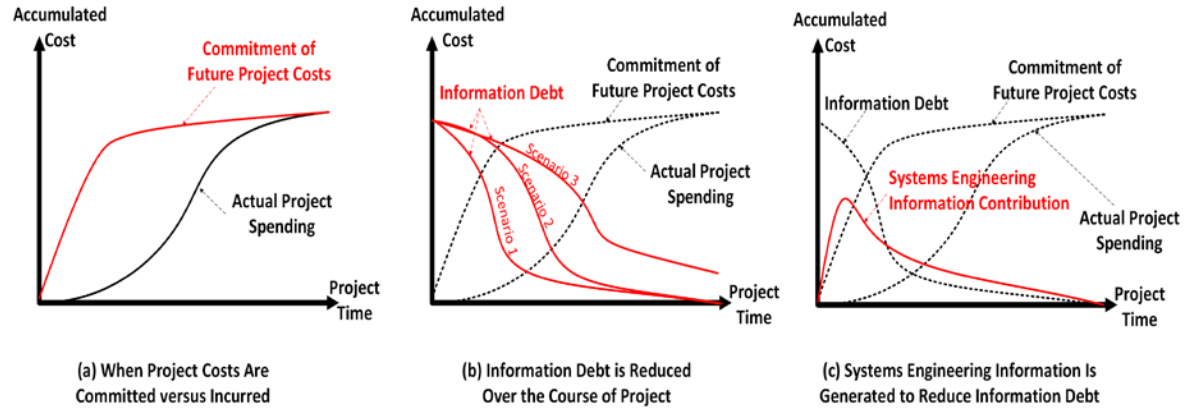

Figure 6. Financial flows—accumulated project costs, information debt, and SE information contribution.

Figure 6 (c) illustrates the idea that systems engineering information must be generated (e.g., requirements, design architectures, risk assessments, etc.) early enough in the project to drive down information debt early enough and completely enough. The other side of the related controversy is the agile community's concern that top-down documentation generation they associate with systems engineering can have its own risks, in too-late discovery of misunderstandings concerning stakeholder needs and expectations, the efficacy of design approaches, etc. Both of these opposite concerns are valid, and an objective means is needed to find the right middle ground—that is the purpose of the concept of information debt. It forces us to decide what information is really required by the subsequent life cycle of a system. It also sets the stage for recognizing that there are both real Balance Sheet (asset and liability) and Income Statement (revenue and expense) issues at stake, described further in the next section.

System 2 Learning Observed: Explicit System 1 Patterns as Balance Sheet Assets

Learning can be seen as discovery of regularities (patterns) that apply repeatedly over otherwise varying instances. The ability to rapidly develop and support ASELCM System 1 aircraft configurations that dynamically respond to a range of "different" instance conditions is improved when ASELCM System 2 recognizes and exploits these underlying patterns. For an aircraft OEM and other enterprises, this takes the form of System 1 platform architectures that provide a framework and component family, which became a "learned" part of the formal models discovered and maintained by System 2, describing System 1, shown in Figure 7.

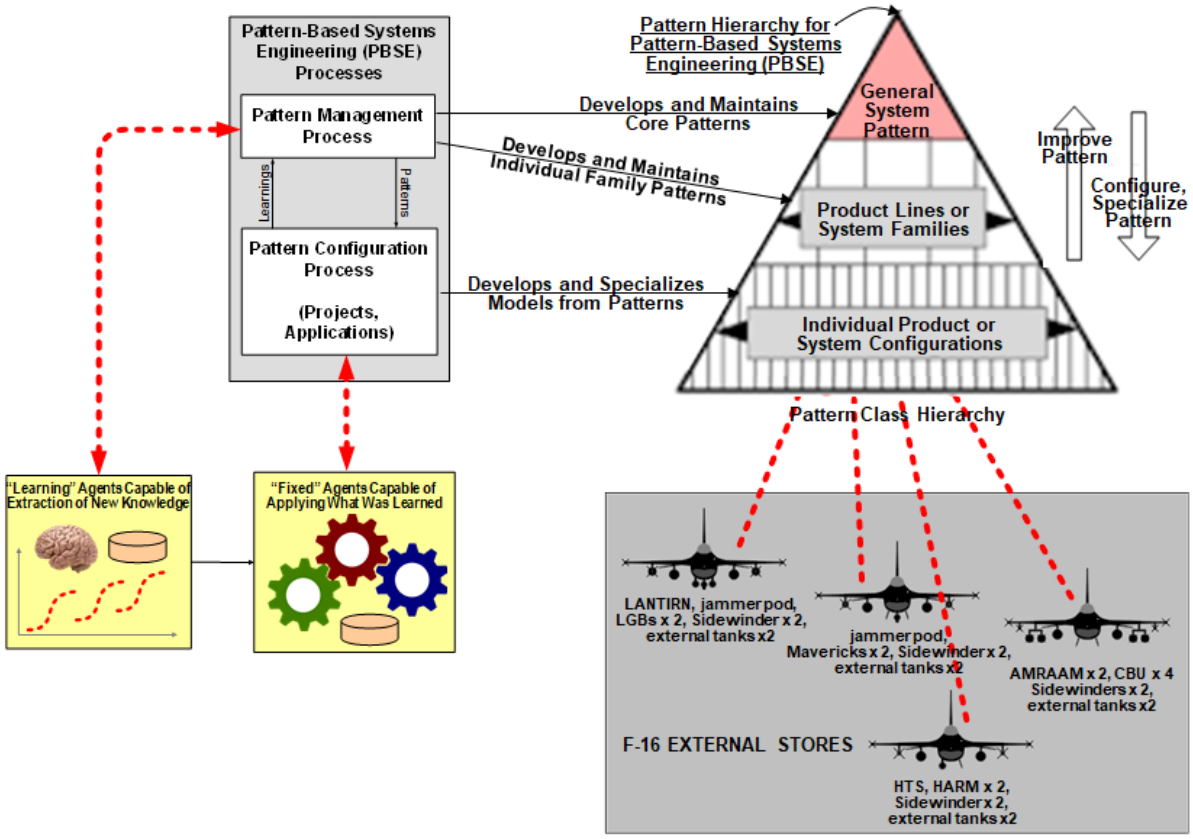

Figure 7. Platform architectures increase agility.
(Goebel, G. F-16 Variants <www.airvectors.net/avf16_2.html>)

Both traditional and agile systems engineering appear to build in enough "process" to address a "green field" or "clean sheet" situation, in which a project begins with no prior knowledge of requirements, design, or otherwise, and processes are provided to discover that information. In practice this is rarely the case, because nearly all system projects begin in the context of a large existing base of knowledge. Until recently, both traditional and agile SE methods offered scant theory in how these existing "assets" (prior knowledge) should be used, other than general guidance to consult and make use of standards, technical readiness levels, architectural frameworks, product lines, etc.

Historically, agile methods in particular emphasize learning by humans, but focus more on optimizing for human learning, not a general theory for accumulation and use of what is learned, and the sharing of this knowledge across a learning organization. The ASELCM Pattern recognizes prior knowledge in both human and other (e.g., stored data) forms, as learned System Patterns, whether in informal human expertise or formal representations shared between humans and information systems—in both cases, these are subsequently applied when the past learning is needed. Figure 8 is the subset of the ASELCM Pattern recognizing those aspects.

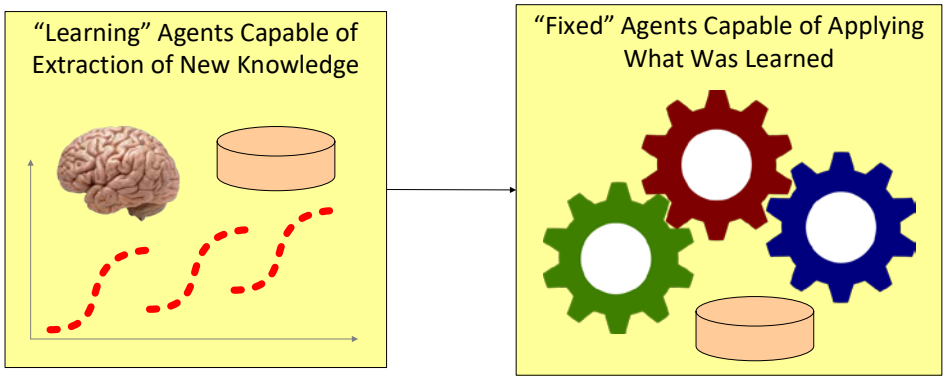

Figure 8. ASELCM human or other learning processes, learned assets, and their use.

Having identified information debt as a "balance sheet" entry, separate from the revenue and expense ("income statement") view of development, we can now turn to the positive side of that balance sheet. We observe that learned system patterns can be viewed as capital assets. In fact, this intellectual property (IP) can be used to offset information debt on the balance sheet.

This approach can be used to greatly strengthen the argument for early stage systems engineering during projects, because the information contribution curve of Figure 6 (c) can be generated without an equivalent surge in systems engineering expense, an income statement variable. This is accomplished by discovering and maintaining system pattern assets, then applying them during the early stage of a project as IP assets to generate information and pay off information debt—analogous to paying for a new house by using an existing asset.

This is the approach that was observed during the ASELCM Discovery Project, where architectural platform patterns were used to effectively increase rapid response flexibility by lowering the cost of early stage Information Debt reduction, using this asset.

Financial standards (e.g., Financial Accounting Standards Board) do not typically provide for capitalization of human expertise, but patterns that are learned and explicitly stored are effectively software IP, which can be capitalized financially (Sherey 2006). This moves us from a metaphor to an actual financial model portion of the ASELCM Pattern. It provides a powerful shift in the methods of financial management of projects, programs, and enterprises, offering an alternative to the traditional question of "up front expense" of systems engineering, while using traditional capital expense accounting methods.

Information Uncertainty, Model VVUQ, Optimal Estimation and Control for Projects

The concept of information assets and debt are not as simple as merely possessing or not possessing components of information that represent certainties. In general, information (e.g., a model) comes with degrees of uncertainty as to its veracity. The theory and practice of determining, representing, communicating, and managing this uncertainty is directly addressed by the work of the ASME standards committees on verification, validation, and uncertainty quantifica-

tion (VVUQ) of models. This work has been joined in by INCOSE, as reported in (Schindel 2018), extending it to more general classes of system models.

Once the representation of information has migrated into formalized models having appropriate completeness as well as minimality, a Hilbert space results, and classical mathematical and engineering methods and tools can be applied. Among these are powerful methods of optimal estimation and control in the presence of incomplete and uncertain information. The big difference is that they are applied not just to System 1, but also to System 2, as described in (Schindel 2017).

Finding 6: General Agile SE Response Requirements

A framework for Response Situation Analyses that guides the identification of agile system and process response needs in eight different response domains has been employed effectively since the mid-nineties (Dove, LaBarge 2014). From ASELCM case study reflection, this framework is used to identify general agile SE process response needs as follows:

Table 1: General Response Requirements

Domain		General Response Requirements
Proactive	Creation & Elimination	What will the process be creating or eliminating in the course of its operational activity? • Opportunity and risk awareness/knowledge • Acculturated memory • Response actions/options • Decisions to act
	Improvement	What performance will the process be expected to improve during its operational life cycle? • Awareness/Sensing • Effectiveness of response actions/options • Memory in acculturation, inventoried response options, and ConOps
	Migration	What major events coming down the road will require a change in the process infrastructure? • New fundamentally-different types of opportunities and risks
	Modification (add/subtract capability)	What modifications in resources-employed might need made as the system is used? • Response action appropriate for specific response need • Personnel appropriate and available for a response action
Reactive	Correction	What can go wrong that will need a systemic detection and response? • Insufficient/inadequate awareness • Wrong decisions • Ineffective response actions/options
	Variation	What process variables will need accommodation? • Effectiveness of response actions/options • Effectiveness of response evaluation
	Expansion & Contraction	What elastic-capacity ranges will be needed on resources/output/activity/ other? • Capacity to handle 1-? critical response actions simultaneously
	Reconfiguration	What types of resource relationship configurations will need changed during operation? • Elements of a response action • Response managers/engineers

Discussion

Our usage of the phrase “life cycle” began with a cognitive connotation of sequential stage progression through a single cycle of system maturity from birth to death. Major systems have often needed to find ways to lengthen life with aperiodic upgrades, but these are expensive and time consuming if not anticipated and facilitated by an underlying agile architecture.

Webster’s definitions of life cycle include “a permutation of a set of ordered elements in which each element takes the place of the next and the last becomes first,” as well as “a course or series of events or operations that recur regularly and usually lead back to the starting point.” Both of these expect a repeating rather than a single cycle.

The Cambridge Dictionary defines life cycle more generally as “the series of changes that a living thing goes through from the beginning of its life until death,” without requiring any order to the series of changes.

The Agile SE Life Cycle Model Framework brings new connotations to “life cycle” for systems usage, as it explicitly models asynchronous re-cycling of all stages without a need for ordered sequence. Some order is rationally prudent for disciplined engineering, but realty often ignores that in practice and patches it up afterwards to make a rational engineering story. Many problems in need of a system solution, for instance, are defined after a desired system is perceived and in need of justification.

The prime purpose of systems agility has been described by one of the authors as risk management for product and process in CURVE environments. Inclusive, but perhaps a better worded purpose, would be sustainable coevolution in the face of an evolving environment; with the cycle time of responsive evolutionary increments the principle metric of effective process capability.

There are no conceptual disconnects between agile systems engineering and common agile software development practices, though the different time frames for incremental development affecting integrated system testing and demonstration in multidiscipline projects needs to be facilitated. Two methods are discussed explicitly in the ASELCM case studies: 1) Rockwell Collins (Dove, Schindel, Hartney 2017) employs an asynchronous alignment methodology, where software short development increments are integrated with the most recently completed staged increment of firmware and hardware, and 2) Lockheed Martin (Dove, Schindel, Garlington 2018) employs a preliminary systems integration lab, where device simulations, prototypes, low-fidelity COTS stand-ins, and incremental completions are integrated with the latest increments of software development for total system incremental testing and demonstration.

Many people looking for agile SE, especially with hardware content, tend to expect it to sound and look like agile software development processes. This leads them to look in the wrong places, and look for the word “agile” as a qualifier. There are good agile SE processes that don’t use the word agile: Product Line Engineering, Wave, Spiral, Open Systems Architecture (OSA), and Live, Virtual, Constructive (LVC) – are five that come to mind.

Live-Virtual-Constructive is an architecture-enabled operational concept, associated by many with DoD training and testing environments that mix simulations, people, and operable equipment. But the general concept of mixed LVC environments has employment and history in other areas, most

notably for System Integration Laboratories that start with simulations and people and incrementally replace simulations (and some people) with proxy or finished equipment. Notably, the Lockheed Martin ASELCM case study (Dove, Schindel, Garlington 2018) discusses an LVC-like preliminary systems integration laboratory they call the Agile Non-Target Environment (ANTE). The ANTE is used to compose integrated systems consisting of real devices, simulated devices, software work-in-process, and operators. The ANTE also employs lower-fidelity open-market devices with similar capability but lower performance than what is eventually expected from subcontractors. Subcontractors are required to provide device simulations to Lockheed ANTE specs. In contrast, the target system testing environment includes both traditional SIL and test-aircraft platforms employed at the end of program increments. The ANTE concept evolved over a few years and has been declared a successful experiment based on customer feedback that values the early and incremental demonstration of working concepts and advanced exposure to difficulties in need of attention.

An informative spiral hardware development process at Space X, which recognizes agile systems engineering as a goal, is available at (Muratore 2012) – with a testament to innovation through experimental learning: "Because we can design-build-test at low cost we can afford to learn through experience rather than consuming schedule attempting to anticipate all possible system interactions."

Agile hardware systems engineering doesn't look much like agile software engineering, though the two can usefully share general agile project management concepts. Differences include development platform availability and needs, hardware lead times, certification time, integration and test facilities, mixed disciplines with different time frames and documentation requirements, DoD acquisition stage-gate needs, and perhaps most striking, fabrication in software is done by the designer rather than handed off to another group with sufficient documentation to preserve design intent.

Hardware systems have more diversity to their nature than software – a variety of electronic approaches, a variety of materials, a variety of fabrication techniques, and very different safety, security, reliability, and retirement concerns. One proscribed method doesn't work. Needed rather is one set of principles to satisfy in whatever must be dealt with to fit the situation.

Summary

Darwin didn't have a model of evolution that he tried to prove or force fit. He observed, and asked, "What's going on here and how does it work?" From that he iterated on model refinement until he could find no exceptions. In a similar fashion, the six findings reported in this article are the result of observations and model refinement that fit what was seen in analyzing a variety of effective agile processes. Theses findings are supported, in retrospect, with similar discovery work in the nineties (Dove 2001) that analyzed over a hundred cases for common structural fundamentals that enable product and process agility. We believe these six findings are timely for today's practitioner. Time will tell if they are timeless.

The definition of agile systems engineering is rooted in what it does, not how it does it. The how can be satisfied many ways, some already chronicled in the four ASELCM case studies to date. As discussed in this article, agile systems engineering responds effectively in CURVE environments, operates asynchronously and potentially simultaneously in at least seven life cycle stages, appears

to behave according to nine operating principles, and melds target system, development system, and learning system into one interdependent system.

The Agile SE Life Cycle Model Framework, as depicted in this article, is compatible with ISO/IEC/IEEE 15288 and 24748-1 standards, and the venerable V diagram, with some suggested minor augmentation and the addition of the Situational Awareness stage. At heart, an effective agile systems engineering process is an organic complex system with many specialized processes working and reacting in mutually dependent concert – of particular interest when systems engineering agility encompasses multidiscipline projects.

The ASELCM Pattern accommodates both V-diagram practices and 15288 processes throughout, as well as the guidance-for-practice in the INCOSE SE Handbook. This article gives a different way of viewing what is going on and a different way of appreciating how value is generated. "An individual system life cycle is thus a complex system of processes that will normally possess concurrent, iterative, recursive and time dependent characteristics. These statements are true for both a system-of-interest and any enabling systems." (ISO/IEC/IEEE 2018: 36).

As to compatibility with defense acquisition policies: all four ASELCM case studies are defense projects, all employ agile systems engineering, three of the four deliver hardware as well as software, and all include preliminary and critical design reviews and other typical contract gates. The case studies show how, over time, program management and contracting offices participated in some adjustments to contract terms as they came to appreciate the benefits that could be realized.

So "yes" to the question posed earlier: the ASELCM project findings can exist compatibly with the current mainstays of systems engineering understanding and guidance.

References

Blackburn, M. 2018a. Transforming Systems Engineering through Model-Centric Engineering-NAVAIR. Technical Report SERC-2018-TR-103. 28 February. <*https://web.sercuarc.org/documents/technical_reports/1527015991-A013_SERC-RT-170_Technical-Report-SERC-2018-TR-103.pdf*>

———— 2018b. Transforming Systems Engineering through Model-Centric Engineering-ARDEC. Technical Report SERC-2017-TR-111. 8 August. <*https://web.sercuarc.org/documents/technical_reports/1536335935-A013_SERC%20RT%20168_Technical%20Report%20SERC-2018-TR-111.pdf*>

Boyd, J. nd. OODA Loop, <*https://en.wikipedia.org/wiki/OODA_loop*>

Deming, W. E. nd. Plan-Do-Check-Act, <https://en.wikipedia.org/wiki/PDCA>

Dove, R. 2001. *Response Ability – the Language, Structure, and Culture of the Agile Enterprise*. John Wiley and Sons.

Dove, R., W. Schindel, C. Scrapper. 2016. Agile Systems Engineering Process Features Collective Culture, Consciousness, and Conscience at SSC Pacific Unmanned Systems Group. Proceedings International Symposium. International Council on Systems Engineering. Edinburgh, Scotland, 18-21 July. <*www.parshift.com/s/ASELCM-01SSCPac.pdf*>

Dove, R., W. Schindel, R. Hartney. 2017. Case Study: Agile Hardware/Firmware/Software Product Line Engineering at Rockwell Collins. Proceedings 11th Annual IEEE International Systems Conference. Montreal, Quebec, Canada, 24-27 April. <*www.parshift.com/s/ASELCM-02RC.pdf*>

Dove, R, W. Schindel. 2017. Case study: agile SE process for centralized SoS sustainment at Northrop Grumman. Proceedings International Symposium. International Council on Systems Engineering. Adelaide, Australia, 17-20 July. <*www.parshift.com/s/ASELCM-03NGC.pdf*>

Dove, R., W. Schindel, K. Garlington. 2018. Case Study: Agile Systems Engineering at Lockheed Martin Aeronautics Integrated Fighter Group. Proceedings International Symposium. International Council on Systems Engineering. Washington, DC, 7-12 July. <*www.parshift.com/s/ASELCM-04LMC.pdf*>

Dove, R., R. LaBarge. 2014. Fundamentals of Agile Systems Engineering – Part 1 and Part 2. International Council on Systems Engineering, International Symposium, Las Vegas, NV, 30 June-3 July. <*www.parshift.com/s/140630IS14-AgileSystemsEngineering-Part1&2.pdf*>

Forsberg, K., H. Mooz. 1991. The Relationship of Systems Engineering to the Project Cycle. Presented at the joint conference sponsored by: National Council On Systems Engineering (NCOSE) and American Society for Engineering Management (ASEM). Chattanooga, TN, 21–23 October. <*www.damiantgordon.com/Videos/ProgrammingAndAlgorithms/Papers/The Relationship of System Engineering to the Project Cycle.pdf*>

ISO/IEC/IEEE. 2018. Systems and Software Engineering — Life Cycle Management — Part 1: Guidelines for Life Cycle Management. ISO/IEC/IEEE 24748-1:2018.

ISO/IEC/IEEE. 2015. Systems and Software Engineering — System Life Cycle Processes. ISO/IEC/IEEE 15288:2015(E) first edition 2015-05-15. Switzerland.

Muratore, J. 2012. System Engineering: A Traditional Discipline in a Non-traditional Organization. AIAA Conference. Pasadena, CA, 11-13 September. <*www.aiaa.org/uploadedFiles/Events/Conferences/2012_Conferences/2012-Complex-Aerospace-Systems-Exchange-Event/Detailed_Program/CASE2012_2-4_Muratore_presentation.pdf*>

Schindel, W., R. Dove. 2016. Introduction to the Agile Systems Engineering Life Cycle MBSE Pattern. Proceedings International Symposium. International Council on Systems Engineering. Edinburgh, Scotland, 18-21 July. <*www.parshift.com/s/160718IS16-IntroToTheAgileSystemsEngineeringLifeCycleMBSEPattern.pdf*>

Schindel, W. 2017. Innovation, Risk, Agility, and Learning, Viewed as Optimal Control & Estimation. Proceedings International Symposium. International Council on Systems Engineering. Adelaide, Australia, 17-20 July.

Schindel, W. 2018. INCOSE Collaboration in an ASME-Led Standards Activity: Standardizing V&V of Models. MBSE Workshop at INCOSE International Workshop, Jacksonville, FL, 20-23 January. <*www.omgwiki.org/MBSE/lib/exe/fetch.php?media=mbse:patterns:standardizing_v_v_of_models_iw2018_mbse_workshop_report_01.21.2018_v1.2.1.pdf*>

Sherey, J. 2006. Capitalizing on Systems Engineering. Proceedings. International Symposium. International Council on Systems Engineering. Orlando, FL, 9-13 July.

Techopedia. nd. Technical Debt – Definition. Retrieved 7 March 2019. <*www.techopedia.com/definition/27913/technical-debt*>

Acknowledgements

The ASELCM project leadership team participated in, and contributed to, the discoveries observed in the project's workshops, and consists of INCOSE Fellows: Rick Dove, Kevin Forsberg, Jack Ring, Garry Roedler, and Bill Schindel. Numerous other INCOSE members and non-members also participated and contributed, and are each acknowledged in the published case studies accordingly.

Biography

Rick Dove is CEO of Paradigm Shift International, specializing in agile systems research, engineering, and project management; and an adjunct professor at Stevens Institute of Technology teaching graduate courses in agile and self-organizing systems. He chairs the INCOSE working groups for Agile Systems and Systems Engineering, and for Systems Security Engineering, and is the leader of the INCOSE Agile Systems Engineering Life Cycle Model Discovery Project. He is an INCOSE Fellow, and author of *Response Ability, the Language, Structure, and Culture of the Agile Enterprise*.

Bill Schindel is president of ICTT System Sciences. His engineering career began in mil/aero systems with IBM Federal Systems, included faculty service at Rose-Hulman Institute of Technology, and founding of three systems enterprises. An INCOSE Fellow, Bill co-led a project on Systems of Innovation in the INCOSE System Science Working Group, co-leads the Patterns Working Group, and is a member of the lead team of the INCOSE Agile Systems Engineering Life Cycle Model Discovery Project