Toward Agile Systems Engineering Processes

Dr. Richard Turner
Systems and Software Consortium

Agile software development approaches have been highly successful in a variety of domains. Could they be effective if applied to systems engineering? This article begins a discussion to answer this question by comparing core agile characteristics to those of traditional systems engineering.

The concept of agility is cropping up more and more often throughout the defense and commercial development worlds. It has found its way into the Quadrennial Defense Review, acquisition plans and procurement requests, and even into the language of defense executives[1]. Promises of faster deployment and evolutionary capability, delighted customers and users, and fewer late-occurring acquisition problems are irresistible to the resource-strapped, schedule-limited, and continuously harried program managers and acquisition executives.

However, where can the agile benefit really accrue? Primarily associated with software development, does the concept play into the large systems development that is typical of the defense environment? How does agility apply to the critical systems engineering processes? While research is needed to fully answer these questions, we can begin to identify touch points that on the surface seem ripe for agile approaches.

This article presents some thoughts on agility and systems engineering – how systems engineering can be more agile and how it can support agility in other disciplines. It is a concept discussion, not a specific how-to article. However, looking at systems engineering through the agile lens can extend the dialogue that began between agile and plan-driven software proponents into the systems engineering world [1].

First of all, why should we care about agility within systems engineering? Table 1 identifies some of the changes in the environment facing systems and software developers. The rapid change in threats, requirements, and programmatic parameters has pushed traditional approaches to the limits of their capabilities. As a result, there is a growing zeitgeist that somewhat unfairly casts traditional systems engineering as a holdover from the 1950's and 1960's and as a part of the systems acquisition and development problem. Agilists generally view systems engineering as rigid and waterfall-based, overly process-bounded (MIL-STD-499, MIL-STD-1521, Institute for Electrical and Electronics Engineers [IEEE]-15288). Myopically focused on early correctness, systems engineering can seem to value precision over accuracy and completeness over rapid user satisfaction. Figure 1 shows the traditional systems engineering *V-model* as it was developed for large systems. The model has evolved over time, but the fundamentals still provide a basis for the life cycle used by defense system acquirers. That is, establish requirements, establish an architecture, decompose the system into subsystems, design the subsystems, build the subsystems, test the subsystems, integrate the subsystems, and then test the system.

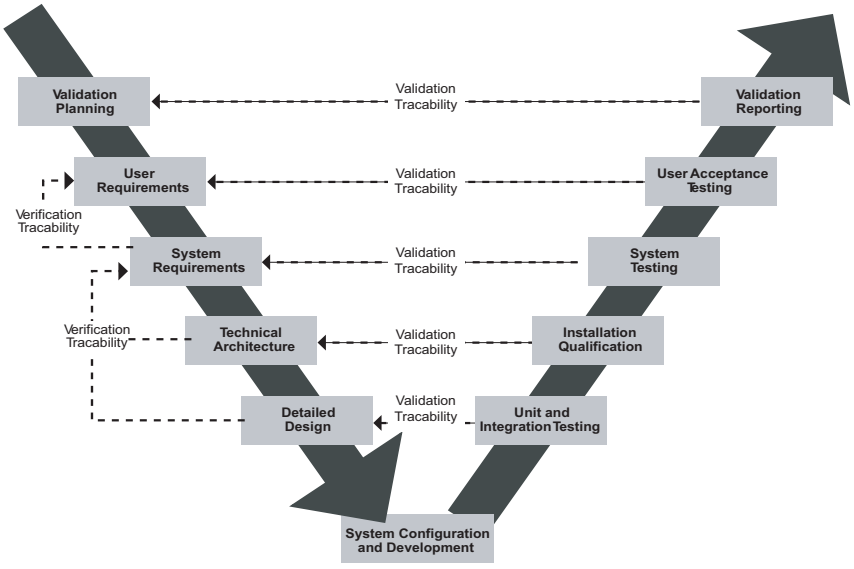
At the same time, agile approaches are portrayed as the promised land. Praised as a panacea for all the developmental ills, agile approaches claim victory over rapid change, increased complexity, emerging requirements, and the ubiquitous schedule-busting integration fiascos. Figure 2 (see page 12) shows a typical agile process. Note the iterative rather than sequential nature. While an iteration could represent a mini-waterfall, that is not always the case, particularly in risk reduction activities.

Of course, neither of the broad characterizations of the approaches is particu-

Table 1: *Some Software-Intensive System Trends*

Traditional Development	Current/Future Trends
• Standalone systems	• Everything connected (maybe)
• Relatively stable requirements	• Rapid requirements change
• Requirements determine capabilities	• Commercial off-the-shelf (COTS) capabilites determine requirements
• Control over evolution of custom systems	• No control over evolution of COTS products
• Enough time to keep stable	• Ever-decreasing cycle times
• Stable jobs	• Outsourced jobs
• Failures locally critical	• Failures broadly critical
• Completely defined systems with specific functionality	• Complex, adaptive, emergent systems of systems
• Repeatability-oriented process, maturity models	• Adaptive process models

Figure 1: *V-Model of a Conventional, Large-System Development Process*



Agility and Process Maturity

It is important to understand that agility is not anti-process, but can conform to Capability Maturity Model Integration (CMMI®) and other process standards. In fact, the Systems and Software Consortium is currently developing a Process Implementation Indicator Description table for CMMI Lead Appraisers to use in appraising agile projects.

Agile concepts in many ways embody Level 5-ness by continuously improving or adjusting processes. By conducting a retrospective/reflective activity after each iteration, recommendations for improvement can be immediately implemented. Agile measures can then confirm or contradict the value of changes within the next few iterations rather than waiting for the next project.

Agile does not specifically address the organizational aspects of many process standards (e.g. Organizational Process Focus, Organizational Process Definition, and so forth in CMMI), but is not a stumbling block to satisfying them. Usually, there needs to be agile instantiations in the set of organizational standard processes to limit tailoring confusion and support agile approaches.

larly accurate as stated, but they do provide insight into the turmoil that has continued to bubble. Regardless of the hype, there is no denying the need for leaner, more responsive development processes. If agile approaches can be harnessed in systems as well as software engineering, they are certainly well worth the effort.

But what, you ask, is *Agile*? There are nearly as many definitions of Agile as there are Agile practitioners. I believe, however, that there are common, key aspects that must be present to capture the essence of Agile. Table 2 captures my own essential list of agile features.

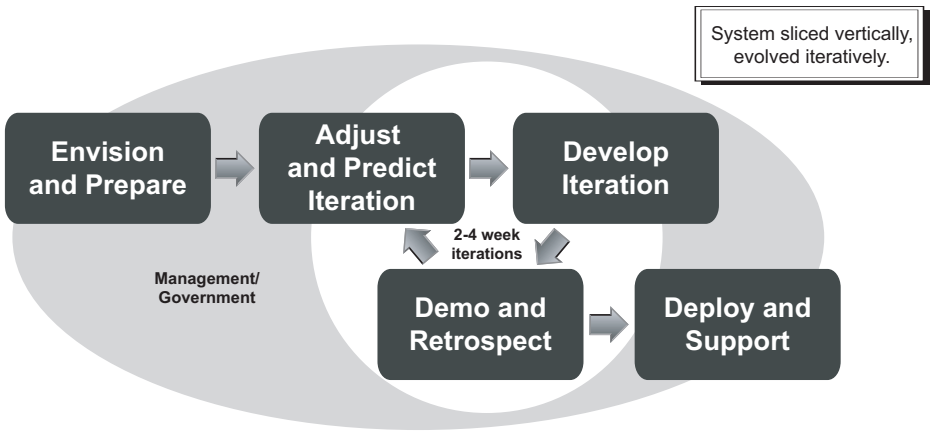
Agility and Systems Engineering Processes

So how do these attributes apply to systems engineering? How can we mature systems engineering to encompass these attributes? Let us look more closely at a few of the attributes that seem to address the engineering process.

Systems Engineering as a Learning-Based Process

One of the characteristics of traditional project management, and by implication much of traditional systems engineering, is the assumption by all stakeholders that foreknowledge is perfect. We can define complete, consistent, testable, and buildable requirements; decompose perfect requirements to perfect specifications; accurately estimate effort, cost, and schedule for the specifications; schedule work according to this information early in the program; and measure progress using earned-value management or similar techniques. While program managers, executives, sponsors and fund providers may believe this, engineers know that with any sufficiently complex system, particularly unprecedented systems, it is unrealistic to assume this kind of knowledge. As Philip Armour said, … *for the most part, engineers do not know how to build the systems they are trying to build; it is their job to find out how to build such systems* [3]. That is why systems engineering can be visualized as a set of tools and approaches that allow us to seek information that fills the gaps in the initial descriptions. By doing so, it adjusts the development to fit the reality of what we have learned. Trade studies, requirements analysis, demonstrations, prototypes, models, design evaluations, allocation analyses, and verification and validations are all ways to learn about the system being developed. So, there is no fundamental reason systems engineering cannot be considered a learning process. Unfortunately, the traditional view of the systems engineering V-model often is interpreted so that it provides only a limited, *one-time through* chance to learn. By reinterpreting the V-model from an agile perspective and using timely iterative feedback, the learning process can be richer.

Figure 2: *Disciplined Agility Process, Basic Model* [2]



Systems Engineers (SEs) as Focused on Customer Value

SEs are often isolated from the customers because their customers are considered fully represented by the predefined requirements and operational concepts. These ostensibly perfect requirements are generally value-neutral, with no sense given to their importance in relationship to each other, save some very high-level key performance parameters or possibly some value thresholds within a particular requirement. This puts the learning systems engineer at a huge disadvantage by debilitating an entire dimension of the trade space: the ability to consider the relative value to the customer of a requirement in deciding to defer or relax it in order to meet some other requirement or for other engineering reasons. The tradeoff between cross-cutting aspects like safety, security, maintainability, and performance has been identified as the number one risk by a University of Southern California survey of systems and software engineers [4]. The relative importance of the requirements must be interpolated using the engineer's experience, physical constraints, and domain knowledge so that fundamental engineering decisions can be made. It would be much easier if the requirements were not only clear and concise but also ranked in terms of importance. There is nothing to prevent including this dimension by having more complete and multi-faceted interfaces with the customer, but the traditional systems engineering requirements activities generally do not support it.

Systems Engineering With Short Iterations

Because systems engineering has been often viewed as a one-pass process (the strict V-model), iterations of systems engineering may sound foreign. However, there are ways to do iterative systems engineering. Prototyping, model-

® CMMI is registered in the U.S. Patent and Trademark Office by Carnegie Mellon University.

ing, demonstrating, and testing can all be iterative within an integrated systems engineering and development cycle. The difference in truly agile iterations is that each of these should describe a complete operable system with functionality that is valuable to the customer. However, in the early systems engineering phases, deployable operational aspects may not be as valuable to the customer as reduced risk, requirements validation, operations concept validation, interface and interoperability verification, or technical feasibility. Systems engineering activities in later iterations are focused on operational capabilities. Development processes where systems engineering is seen as an up-front process and the SEs complete their trades and decomposition tasks and then move on to another program until needed for validation (sometimes referred to as the *do it once and the SEs do lunch* approach) are not conducive to iterative work. One of the most creative ways of envisioning systems engineering iterations is Barry Boehm's characterization of systems engineering as a Command and Control, Intelligence, Surveillance and Reconnaissance (C2ISR) activity (Figure 3), consisting of numerous Observe, Orient, Decide, Act (OODA) loops and ongoing intelligence, surveillance and reconnaissance tasks [5]. This counters the traditional cycle of requirements, delay, and surprise.

Systems Engineering and Neutrality to Change

This involves the architectural and design approach more than pure systems engineering. Unless systems engineering performs its activities and processes with an eye toward supporting change rather than avoiding or denying it, change will become an enemy (rather than an annoying but faithful family member). System engineering can use change as a dimension in its trade studies, evaluate the ease of modification or extension within architectural reviews, and even add requirements and design constraints that support change neutrality.

Systems Engineering, Continuous Integration, and Test Driven Development (TDD)

Once we accept the idea that SE iterations are feasible, then continuous integration and TDD are not as problematic. In order to provide an operable system that demonstrates value, there must be ways to maintain the configuration over time and use it as initial validation of operational capability, interoperability, and interface quality. Most likely done in a completely simulated or hardware-in-the-loop environment, frequent integration and requirements based testing (especially where there are external components that you may or may not control), can identify anomalies, misinterpretations, and downright errors in the interface specifications or implementations much earlier than traditional late-in-the-process integration. This does require a change in the once-through V-model, but can be thought of as concurrent execution of processes within the V-model framework. One way to think of this is to agree that the processes that define the V-model are only required to complete in the order they appear rather than to proceed sequentially.

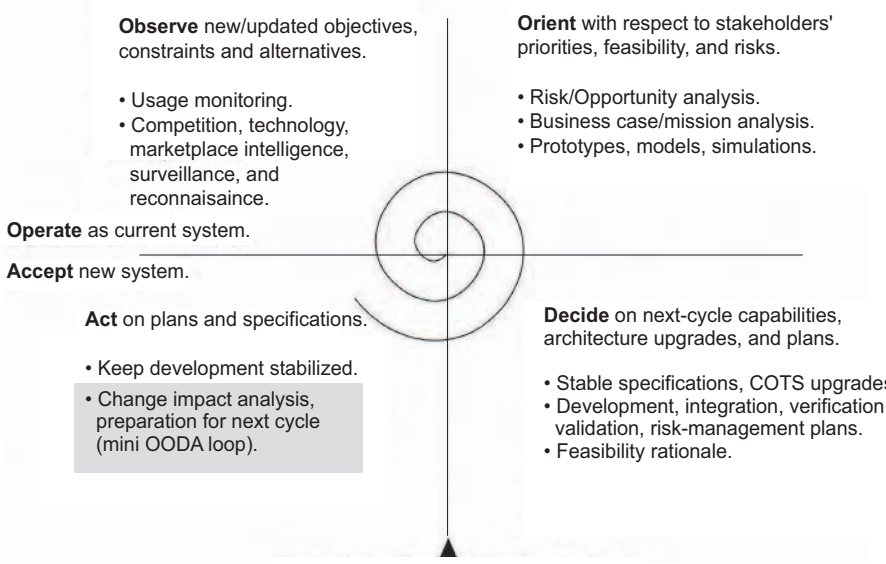
Attribute	Comment
Learning attitude	• Take advantage of lessons learned and adapt both processes and systems to meet customer needs.
Focus on value to customer	• Customer prioritizes requirements and progress is measured by operational features.
Short iterations delivering value	• Goal of each release is a working system. • Rolling planning horizon. • Risk-driven, reality-based iteration planning.
Neutrality to change (design processes and system for change)	• Change is seen as inevitable; ergo *embrace change* applies.
Continuous integration	• Integration is an ongoing activity. • Integration and testing are as automated as possible.
Test-driven (demonstrable progress)	• Tests are written before any other artifacts (design, code). • Capabilities (requirements) are defined by the tests (empirical evidence) that validate them.
Lean attitude (remove no-value-added activities)	• As little ceremony as necessary; just enough (or just too little) process. • Decisions delayed until latest *feasible* time.
Team ownership	• Team has primary responsibility and authority over its own plans and processes. • Quality/performance is everyone's responsibility.

Table 2: *Key Characteristics of Agile*

Systems Engineering and Lean

Lean, as I interpret it here, is the removal of low value or unneeded activities as well as the delay of significant end-user decisions until the latest possible moment. We have talked about rethinking some activities to make them more useful, and certainly most processes have some fat in them somewhere. However, delaying decision making in systems engineering is not easy. There is a drive to complete specifications, finalize allocations, and set architectural structures as early as possible.

Figure 3: *Systems Engineering as C2ISR With Spiral OODA Loop*



This is especially critical when there are long lead manufacturing items in the mix. Remember, though, Lean does not delay all decisions, just those that can have significant impact on operational acceptance or high priority functionality and that can be feasibly delayed. Once you lose the early omnipotence syndrome, delayed decisions can retain design flexibility longer, enabling more rapid reaction to internal or external changes.

Systems Engineering and Team Ownership

This may be the most controversial agile attribute in a process-focused organization. If the systems engineering team owns its own process and can manipulate it to meet its project needs, how can the quality assurance folks ensure that the *correct* process is being followed? This is essentially a management decision to support empowered teams in more than name only. While it may impact the management control residing with some of the stakeholders, providing the systems engineering team with the authority and flexibility of owning their own process could radically improve their effectiveness.

Software Considerations for Agile Systems Engineering

In the introduction, I indicated that systems engineering could support Agile in other disciplines. Software is a prime example. The role of software is a significant systems engineering issue that requires adjustments, if not agility, from systems engineering processes. As systems become less *hardware with some software that helps*, and become more *software with some hardware to run on*, the need for software as a full participant in systems engineering becomes critical. This summer, the National Defense Industrial Association (NDIA) convened a group of industry, government, and academia participants to define the top problems in software-intensive systems (the majority of the systems currently built) [6]. One of the critical findings was that *fundamental system engineering decisions are made without full participation of software engineering.*

Software can no longer be relegated to a secondary activity. The days of software coders carrying out specific instructions from engineers are over. Software is what provides capability, enables flexibility, supports net-centric operations, allows quick response to new threats and environmental factors, and represents the majority of the value of a specific system, even though the hardware production may be the most expensive (and often most profitable) activity. Initial decisions must consider software architecture or they can impact the feasibility of software solutions and result in disjointed, untestable, and unmaintainable software components. The previously referenced NDIA report states the following:

> Complex, distributed, interoperating systems and evolving software capabilities have permanently altered the system level trade space. Key architectural decisions early in the system life cycle have great impact on software capabilities, attributes, and architectural/design approaches, yet the software engineering discipline is not consistently involved in these decisions.

"While it may impact the management control residing with some of the stakeholders, providing the systems engineering team with the authority and flexibility of owning their own process could radically improve their effectiveness."

I like to think of this as software-first engineering. By considering software first, the SEs can take primary advantage of the flexibility and adaptability of software, define the system and its components in such a way that software development is less complex, and the system architecture and design support the effectiveness of software assurance, safety, and security. These are attributes that simply cannot be added on later, particularly in systems of systems or net-centric systems.

Final Thoughts

I have postulated that traditional systems engineering may not fit today's and tomorrow's systems because of its inherent rigidity and its often interpreted waterfall orientation. On the other hand, agility is much more a state of mind or philosophical approach than a set of rules that have to be followed regardless of appropriateness.

Despite the disagreement *from some agile proponents*, process is not the enemy – bad process is. To encourage agility, processes should not be dictated by the *process police*, but be under the control of the actors. Process experts can provide constructive support and guidance when needed, and process asset libraries should include agile or agile-friendly processes that can be used where the development environment or risk profile indicates a need for agility.

The fundamental goals of systems engineering have not changed. However, as systems grow larger and more complex, new ways of dealing with abstraction, concurrency, and uncertainty need to be developed. Agile approaches do offer reasonable and elegant ways of evolving systems and software engineering toward handling these issues.

There are still no silver bullets [7], but we can accept that there are new kinds of regular bullets available, new tactics by which they can be used, and that integrating them into our current operations can significantly improve the capability of our existing systems engineering arsenals.

As I said in the introduction, my intent with this article is to extend the dialogue about innovative ways to consider and apply systems engineering. I have not included examples, but I believe there are many systems and software engineers that have applied some of these approaches to systems engineering. I would be grateful if they joined the conversation by providing their experiences, successful or not, so that we can create better ways to balance the discipline of systems engineering with the agility required to develop today's complex defense systems.◆

References

1. Boehm, Barry, and R. Turner. Balancing Agility and Discipline: A Guide for the Perplexed. Addison-Wesley: Boston, 2004.
2. McCabe, R., et al. "Disciplined Agility Guidebook." Proprietary Internal Report. Systems and Software Consortium, 2006.
3. Armour, Phillip. The Laws of Software Process. Auerbach: Boca Raton, 2004.
4. Boehm, Barry, and Jesal Bhuta. USC CSSE Top 10 Risk Items: People's

Choice Awards. 2006 <http://csse.usc.edu/BoehmsTop10/>.
5. Boehm, Barry, and Jo Ann Lane. "21st Century Processes for Acquiring 21st Century Software-intensive Systems of Systems." CrossTalk, May 2006.
6. NDIA Systems Engineering Division. "Top Software Engineering Issues within Department of Defense and Defense Industry." Aug. 2006.
7. Brooks, Frederick P. "No Silver Bullet: Essence and Accidents of Software Engineering." Computer 20.4 (Apr. 1987): 10-19.

Note

1. Mr. Krieg, Under Secretary of Defense (Acquisition, Technology, Logistics), used agile, agility, flexibility or related words nearly once a minute in a recent presentation to business executives. Mark Schaeffer, Director for Systems and Software Engineering in the Office of the Secretary of Defense, encouraged the process improvement world to become more agile in remarks at the 2006 NDIA CMMI Technology Conference.

About the Author

Richard Turner, D.Sc., a Fellow at the Systems and Software Consortium, is a researcher and consultant with 30 years of international experience in systems, software, and acquisition engineering. He is a frequent collaborator with a wide range of research organizations and system developers to transition new software-related technology to defense acquisition programs. Turner is co-author of *Balancing Agility and Discipline: A Guide for the Perplexed*, *CMMI Distilled*, and *CMMI Survival Guide: Just Enough Process Improvement*.

Systems and Software Consortium
2214 Rock Hill RD
Herndon, VA 22017
Phone: (703) 742-7116
Fax: (202) 390-3772
E-mail: turner@systemsandsoftware.org

Web Sites

Agile Manifesto

www.agilemanifesto.com

On February 11-13, 2001, at The Lodge at Snowbird ski resort in the Wasatch mountains of Utah, 17 people met to talk, ski, relax, and try to find common ground. What emerged was the Agile Software Development Manifesto. Representatives from eXtreme Programming, SCRUM, Dynamic Systems Development Method, Adaptive Software Development, Crystal, Feature-Driven Development, Pragmatic Programming, and others sympathetic to the need for an alternative to documentation driven, heavyweight software development processes convened. Currently, a larger gathering of organizational anarchists would be hard to assemble. The emergence of the Manifesto for Agile Software Development symbolizes the participants' intents.

Agile Advice

www.agileadvice.com

Agile Advice is a blog about agile methods such as SCRUM, Lean, and eXtreme Programming. However, it does not focus on agile software development. Rather, the focus of Agile Advice is on agile methods applied to other types of work such as managing, video-making, teamwork in general, creative working, training, writing, etc. Much of the material here is based on Mishkin Berteig's experiences as an agile coach, consultant or trainer to teams and management in organizations across North America. From time to time, other people contribute articles to Agile Advice. You are welcome to contribute as well, particularly if you have a story about agile methods, agile principles, or agile practices applied outside of software development.

The Agile Journal

www.agilejournal.com

The Agile Journal is an online magazine and monthly e-newsletter focused on providing readers with the need-to-know information and resources they need to develop software for an agile business. Among the topics covered: Open source solutions, service-oriented architecture, globally distributed development environments, Agile and iterative processes, integrated tools, and reuse and collaboration.

Coming Events

May 7-11
DMSC 2007
Defense Modeling and Simulation Conference
Hampton, VA
www.ndia.org

May 7-11
PSQT 2007 West
Practical Software Quality and Testing
Las Vegas, NV
www.psqtconference.com/2007west

May 14-16
SATURN 2007
3rd SEI Software Architecture Technology User Network Workshop
Pittsburgh, PA
www.sei.cmu.edu/architecture/saturn/index.html

May 14-18
STAREAST 2007
Software Testing Analysis and Review
Orlando, FL
www.sqe.com/stareast

May 19-21
ICSP 2007
International Conference on Software Processes
Minneapolis, MN
www.icsp-conferences.org/icsp2007

May 20-26
ICSE 2007
29th International Conference on Software Engineering
Minneapolis, MN
www.icse-conferences.org/2007

June 18-21
2007 Systems and Software Technology Conference
Tampa Bay, FL
www.sstc-online.org

Coming Events: **Please submit coming events that are of interest to our readers at least 90 days before registration. E-mail announcements to: nicole.kentta@hill.af.mil.**