Systems Engineering In Paradise

Maj. Dan Ward, USAF

Members of the Air Force Studies Board recently wrote a book with the catchy title *Pre-Milestone A and Early-Phase Systems Engineering: A Retrospective Review and Benefits for Future Air Force Acquisition*. It's actually more interesting and readable than the title suggests, and you can download the PDF version for free at <**http://books.nap.edu/catalog.php?record_id=12065**>.

I read the book while sitting on a remote tropical island, sipping a frozen adult beverage of my choice, and enjoying the kind of cheeseburger Jimmy Buffet sings about. As the warm sun turned my skin the color of the tomato on my burger, one line jumped out at me. On page 88, I read, "At least one major prime contractor known to the committee has decided to eliminate the term 'systems engineering' altogether after finding that many of the accumulated documented processes in government, academia, and industry are useless."

Ward, *currently a student at the Air Force Institute of Technology studying systems engineering, holds degrees in electrical engineering and engineering management. He is Level III certified in SPRDE and Level I in PM, T&E, and IT.*

Illustration by Jim Elmore

Because I am about to complete a master's degree in systems engineering, this rejection hit a little close to home. Plus, I wasn't really on a tropical island. I was in Ohio, and I wasn't eating a cheeseburger.

Anyway, the authors go on to talk about "the adverse effects of obsolete and non-relevant process requirements" and the importance of "allowing systems engineering and program management the leeway to tailor compliance with required processes to suit the needs of each specific program." Ah, leeway to tailor compliance—now they're singing my song.

But all this discussion about obsolete and irrelevant processes made me suspect that systems engineering was getting a bum rap in some circles. See, I'm not sure systems engineering is really all about establishing strict, formal processes, despite the best efforts of some to make it so. In fact, while systems engineers certainly need to understand process work and often use a process-driven approach, systems engineering is actually a more organic activity than some people make it sound. With all due respect to my friends at INCOSE (the International Council on Systems Engineering), systems engineering has got to be more than "a structured development process" if it's going to be of much use.

So, in keeping with my preference for principles over rules (see "Socrates in DC," *Defense AT&L*, July-August 2008) and people over process (see everything I've ever written), I pulled together the following collection of systems engineering principles. This grossly incomplete grouping contains a few of the insights the discipline of systems engineering contributes to technology development efforts and perhaps sheds some light on the contributions a systems engineer can make. It may not completely redeem the term systems engineering, but I do hope it helps.

Principle #1: You can't do just one thing

Systems engineering is concerned with the development of complex systems. Accordingly, systems engineers must address the interactions of a variety of entities within their systems, including components, subsystems, and stakeholders. Changes to any one aspect of the system (from funding to function to form) ripple through and affect many, if not most, other aspects of the system.

For example, changing a particular system interface (either internal or external) not only impacts the physical components associated with that interface, but could also have an effect on cost and schedule. It might take time and money to implement the new interface, or the new implementation might save time and money. A new interface might also change the system's performance, maintainability, or reliability. The good news is, it is possible to improve all these things by implementing a dependable, standardized, maintainable interface. The bad news is, it is also possible a new interface will have a negative impact on these factors. The key thing to keep in mind is that we never simply redesign an interface.

Thus, systems engineers can never do just one thing to a system. Every change has more than one implication, and systems engineers must be aware of as many of these implications as possible. A systems engineer's holistic approach involves an awareness of the system's interconnected, interrelated, complex nature.

Principle #2: Complexity and functionality are not always directly proportional

Systems engineers build systems that do things. Whether it is an aircraft, a satellite constellation, or an enterprise information infrastructure, systems engineering projects are designed to accomplish certain functions. The project is deemed a success largely based on whether (or to what degree) the system performs the required functions upon delivery.

However, if we simplistically equate functionality with success, it is easy to fall into the "more is better" trap, and assess the value of a system solely in terms of the sheer number of functions it performs. This approach can lead to over-engineered, excessively complicated systems in which complexity overwhelms functionality.

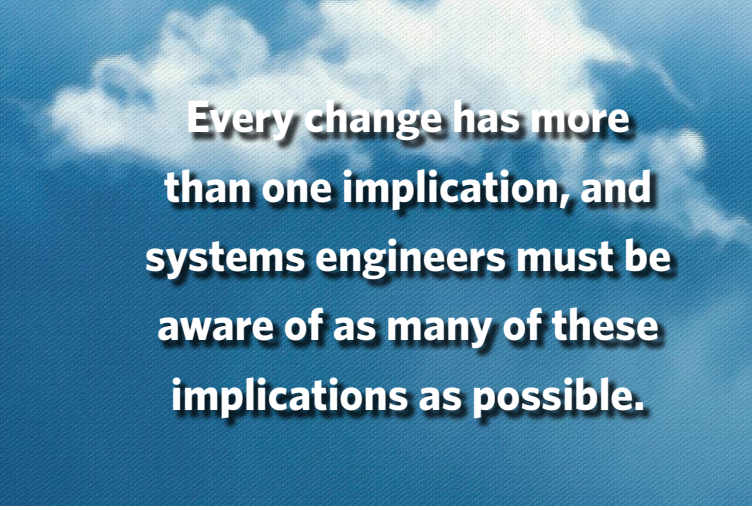
The engineering process might begin with a blank sheet of paper or a collection of legacy systems. In either case, the systems engineer typically begins by adding functions to ensure the system meets the user's requirements. This process of generating new functions is appropriate and necessary ... to a point. Adding too many functions decreases the system's overall value, making it worse, not better.

There are two ways this error can be manifest. First, the system can become too large and unwieldy, making testing, analysis, operations, and maintenance difficult, time-intensive, and expensive. In short, the complexity makes the system difficult to use. Alternately, the conflicting demands of multiple functions might require performance tradeoffs and compromises, which degrade the system's overall utility. In this case, complexity dulls the system's edge.

The end result of this error is either a large, complicated system that makes it difficult to do things well or an overly generic system that does not do anything particularly well. These two outcomes are actually quite similar in that they both result in degraded operational performance, albeit for different reasons. The worst possible outcome is a combination of both—a system that is excessively complicated and not particularly good at any one thing. So, while the systems engineering discipline is concerned with producing complex systems, one of the main objectives is to constrain that complexity and make sure it is not excessive.

Principle #3: Foster common understanding

And the users exclaimed with a laugh and a taunt:
"It's just what we asked for but not what we want."
Anonymous



When we say systems engineering is "multi-disciplinary," that doesn't mean it involves spankings and detentions. Sure, some systems engineers feel the need to act like the vice principal of discipline at an elementary school, reining in the unruly and the truant, but that's not why they're there. The multi-disciplinary nature of systems engineering is actually about providing translations between the various communities and tribes involved in developing a large, complex project, fostering communication and building shared understanding.

Any given systems engineering project inevitably involves a large group of stakeholders, including the people who pay for, design, use, maintain, or dispose of the system. Regularly getting these people together in a timely and meaningful manner and helping them understand each other is one of the key functions of systems engineering.

The various stakeholders each have their own sets of priorities, values, interests, requirements, and talents. These do not necessarily align with those of the other stakeholders—they might even be mutually exclusive—nor are they all defined to equal levels of coherence. Systems engineers need to avoid simply focusing on the loudest, biggest, or most clearly documented requirements and instead consider the full range of inputs. Thus, systems engineering involves a lot of active listening, careful documentation, and extensive networking to establish a shared understanding of what the system needs to do and in what kind of environments (physical, political, and financial) it needs to operate.

Stakeholders even have their own languages, and an apparently clear statement of a requirement might be misleading, misunderstood, or even mistaken. For example, I recall attending a meeting in which a special operations commander stood up, pounded the table, and insisted "We need more training on these systems!" It turns out what he actually needed was a simpler system that required less training, not more. So along with active listening and thoughtful translation, a systems engineer needs to inject insightful and creative alternatives into the discussion, helping to shepherd the stakeholders toward a project that meets their actual needs and not simply their perceived needs.

Principle #4: Iterate, iterate, iterate (aka The SAWABI Principle)

It's not at all important to get it right the first time.
It's vitally important to get it right the last time.
Andrew Hunt and David Thomas

The complexities involved in systems engineering, both technical and political, virtually assure that the first draft and the final product will be different to a certain degree. Fred Brooks, author of *The Mythical Man-Month*, suggests that programmers should "Plan to throw one away. You will anyhow." Other writers have suggested that if we plan to throw one away, we'll end up throwing away two. In any case, the need to throw one (or more) away should not come as a surprise.

The point is that design is an iterative process. This is particularly true for systems engineering design, given the inherent complexities and the large numbers of stakeholders, compounded by the difficulties inherent in communicating complexities across large groups, as discussed in the first three principles.

Good systems engineers avoid becoming overly attached to the initial products, since refusing to discard a failed approach is unwise. Therefore, one of the key tasks for a systems engineer is to plan and coordinate the various iterations of each product (requirements, architectures, budgets, organizations, etc.), to include mechanisms for gracefully discarding initial versions.

In an article for the July-August 2004 issue of *Defense AT&L*, I coined the term SAWABI to describe just such a mechanism. SAWABI stands for Start Again With A Better Idea (not to be confused with Sawabi, Pakistan). The SAWABI principle involves recognizing the need to replace the current version of something with a better version. Depending on the scale and impact of the change, SAWABI might require a large quantity of humility, creativity, honesty, and courage. It might be easy to SAWABI a single requirement, while SAWABIing an entire architecture is probably much harder—but perhaps just as necessary. Good networking and communication skills (see Principle #3) make SAWABI much easier, but we must keep Principle #1 in mind as well and be aware of the potentially widespread implications of any change.

Principle #5: Speed is a virtue

Instability, in all its forms, is one of the biggest challenges faced by systems engineers. Budgets, schedules, and requirements can all change over time, often in inconvenient combinations (i.e., concurrent budget cuts and increased performance requirements) or with unintended consequences (see Principle #1). Stakeholders, team members, critics, and supporters come and go, and their replacements may have different priorities, perspectives, and skillsets. One way to help stabilize the systems engineer-

ing environment, and thus improve the outcome, is to work on a short timeline.

Generally speaking (and perhaps counter-intuitively), speed is a systems engineer's friend. While working on a short timeline injects potentially uncomfortable pressure to deliver, it also reduces the risk of budget cuts or requirements creep, which can be even more uncomfortable. On a short schedule, there simply isn't enough time for anyone to inject significant changes to budgets or requirements. Additionally, a near-term delivery deadline provides a strong justification for systems engineers to resist the introduction of counterproductive change. A short timeline also increases the likelihood of personnel stability, as the project can be completed before too many people move on to bigger and better things. As noted in Principle #1, changes to one element tend to ripple throughout the rest of the system, so stability increases the likelihood the system will be ready when needed and effective when used.

Speed also decreases the risk of delivering obsolete systems because the faster the project moves, the less the technology environment will change. Further, speedy projects tend to incorporate mature technology rather than spend time developing (or waiting for) new, as-yet-undiscovered components. So, speed helps systems engineers avoid the dual risks of bringing obsolete technology forward or expecting to incorporate potentially unavailable technology.

On the other hand, speed introduces a temptation to cut corners, oversimplify, or prematurely optimize a design. These are serious dangers that degrade the system's performance and should be avoided. However, they are no more serious than the risk of requirements creep, personnel turnover, or funding instability inherent in slow, long-term projects. More importantly, project leaders and systems engineers have direct influence over speed-induced risks, while a slow project's risks are largely external and beyond the systems engineer's control. In my opinion, the risks and problems introduced by being fast are preferable to those introduced by being slow.

Principle #6: Talent trumps process

The field of systems engineering has produced a number of methods, processes, tools, and techniques for use in developing complex systems. Those each have varying degrees of utility, and their establishment represents a real step forward in our ability to manage and create big, complex projects. However, the best process or tool in the world is useless in the wrong hands, and a talented systems engineer can deliver a meaningful product despite a bad process or suboptimal tools. Thus, this principle states "talent trumps process."

Systems engineering talent includes, but is not limited to, the abilities to see connections within a system (see Principle #1), to appreciate the value of complexity and distinguish between simplisticness and simplicity (see Principle #2), to communicate and persuade (see Principle #3), and to recognize when to start over (see Principle #4). Talent also includes the ability to work fast and help a team meet a deadline (see Principle #5). And as CalTech's Dr. Joel Sercel pointed out, "Systems engineering without domain knowledge is a net negative." So this entire discussion rests on the assumption that the system engineer knows something about the area in which he or she is working.

While the INCOSE fellows talk about systems engineering as primarily focused on "creating and executing an interdisciplinary process," I think it really comes down to thinking—systems thinking, to be precise—and at this point in history, thinking (systems or otherwise) is a human-only activity. While our tools and processes are useful in accomplishing tasks, tool or process cannot think for us. Thinking skills are, therefore, the ultimate elements of systems engineering talent.

Because talent trumps process, a good systems engineer knows how to unleash talent—both his or her own as well as the talent of others. And ironically, the best way to unleash talent is to not have too much of it. Smaller teams are inherently more streamlined and agile, making it easier for team members to apply their talents. In fact, small teams of talented people generally outperform large committees of similarly talented people because in a big group, it is harder to communicate, harder to see the big picture, harder to inject new ideas, harder to change direction, and harder to be fast. An oversupply of talent is paradoxically counterproductive, so systems engineers would do well to foster and mentor a small cadre of talented people rather than a large stable of mediocre people who basically function as interchangeable parts.

But wait, there's more…

If systems engineering is treated as a formal, inflexible, complexly structured, requirement-heavy development process, more and more enterprises will follow the example of the unnamed "major prime contractor" and eliminate the term altogether. They would not be wrong to do so. But the systems engineering discipline, properly understood, does have some powerfully useful insights and principles for technology development project leaders. It would be a shame to reject the entire concept just because it has been defined too narrowly, misapplied, and generally abused.

The six principles outlined here are only a few of the contributions systems engineering provides. No doubt there are many, many more that could be written, but I'm already over my word limit for this article. Plus, there's a cheeseburger in paradise calling my name …

The author welcomes comments and questions and can be contacted at **daniel.ward@afit.edu**.