Asks the Chief Engineer: "So what do I go do?!"[1]

Douglas O. Norman and Brian E. White
The MITRE Corporation

Several of us in Australia, the United States, and elsewhere[2], have taken on quite a challenge in the past several years. Namely, to take ideas from the realm of complexity theory, the study of complex systems science, etc., and (increasingly) to apply this learning to improving our practice of systems engineering in the most challenging environments that we face today—and in the future. This talk highlights just one aspect of this quest.

We are actively exploring, presenting, and debating ideas and methods beyond the boundaries where systems engineering is currently defined, and more importantly, practiced. Most notably, the ideas that seem to offer the most value are being taken from what is sometimes called "complexity science," and in a language of conception and expression with which many systems engineers are not very familiar. Furthermore, many systems engineers are uncomfortable with adopting a broader perspective of systems engineering and trying to adopt and formalize complementary techniques that may help greatly in practicing systems engineering in the real-life areas within which we are being asked to work. Therein lies the challenge, because at the heart of the matter is the need for ideas, concepts, mechanisms, and processes, in a language that is relevant to systems engineers, by which we can produce useful results at the considerable scales (views: {scope, granularity, mindset, and timeframe}[3]) being asked.

As with any new idea which exposes and articulates potentially new values, there is usually a gap between the expression of the essence of the idea, and the transformation of the idea into sets of common understandings, expectations, and processes by which the idea impacts the real world. Those developing the idea tend to assemble and use a vernacular foreign to that normally used in the particular area of knowledge and practice targeted for application. Usually this is necessary to highlight the idea and sufficiently separate it from the existing conceptual basis from which current beliefs and practices flourish. However, the people mostly for whom the new idea is intended often fail to understand it because of the language used. So, we have a protracted period of time where the new idea percolates around a community finding small "toe-holds" here and there, and morphing to reflect new ways to be understood and expressed. At the same time, the people in the community in which the idea should have traction come to understand it better—and help to sharpen its expression.

---

[1] Based upon an invited keynote talk by Norman, D. O., *"So…," asks the Chief Engineer "What do I go do?,"* Complex07, The 8th Asia-Pacific Complex Systems Conference, Brisbane, Queensland, Australia, 3-5 July 2007

[2] For example, The International Council on Systems Engineering's (INCOSE's) Systems Science Enabler Group (SSEG), http://www.incose.org/practice/techactivities/wg/sseg/

[3] White, B. E., "On Interpreting Scale (or View) and Emergence in Complex Systems Engineering," 1st Annual IEEE Systems Conference, Honolulu, HI, 9-12 April 2007

This process was well-described by Richard Dawkins when he noted that ideas themselves enjoy an ecology-like existence of their own—jumping from mind to mind, being subject to selective pressures, and adapting to accommodate their place among all the other ideas and concepts in the minds which hold them. This was his notion of the "meme" which he compared to genes in the book *The Selfish Gene*[4].

So, new ideas compete for a niche in our thoughts with other ideas. And, in this competition within engineering minds, they must show value and be useful for describing the world, making predictions, and assisting in creating practical outcomes.

This is no different for systems engineering. Systems engineers (and those who practice systems engineering under other names) are being asked to apply their engineering acumen in larger and more complex[5] contexts. As these contexts expand, the (mostly unstated) assumptions of the application of our systems engineering practice are violated more and more. Thus the ideas, methods, and tools, previously unquestioned, have hit some limits, and many complex system failures that have happened can be traced to shortcomings of (traditional, conventional or "reducible"[6]) systems engineering.

Some new ideas have been proffered for tackling systems engineering challenges at these greater scales/views; they are variously embodied in what is called Enterprise Systems Engineering, Large-scale Engineering, Mega-system Engineering, and/or Complex Systems Engineering. As with many things, the labels seem to weaken when examined too closely. For example, what is an enterprise? And, what does it mean to "engineer" it? Notwithstanding these definitional difficulties, there are pretty clear boundaries (as

[4] Dawkins, R., *The Selfish Gene*, Oxford University Press, 1976

[5] We do not mean merely complicated. Complexity here is better defined as a technical term qualitatively describing the ultimate richness of an entity that 1) continuously evolves dynamically by organizing its own internal relationships; 2) requires multi-view analysis to perceive different non-repeating patterns of its behavior; and 3) defies methods of pre-specification, prediction, and control. *Features:* Complex entities possess attributes which cause them to evolve naturally without outside intervention. It is also not possible to pre-specify or predict completely and accurately what will happen with complex entities, even when one intervenes from the outside with a specific purpose. The attribute of complexity is usually associated with the property of instability. Furthermore, it isn't possible to replicate complexity exactly. Each complex instance is unique. Increasing a system's complexity implies its potential behavior will display more variety, nuance, and depth. A system can become so complex that its state approaches that of chaos—and may even transition into chaos—making its nature even more difficult to understand. A system might also evolve with diminishing complexity, trending toward stability. This trend may continue to the point that the system might better be described as deficient in variety and richness, uninteresting, possibly stagnant, or even boring. The challenge in either case is to attempt to shape the environment of a complex system by continually introducing variety and selection (akin to W. Ross Ashby's "Law of Requisite Variety", introduced in Chapter 11 of *Introduction to Cybernetics*, 1956, University Paperbacks, June 1964). This enables a system to become even more complex (e.g., with higher degrees of innovation and integration), yet avoid chaos or stagnation. *Notes:* Many people use the term "complex" as a synonym for anything that is complicated and difficult for a typical human being to understand. Although this is often appropriate in the English vernacular, when used in the context of complex or enterprise systems engineering (CSE or ESE) the term "complexity" implies discerning the matter being considered in much greater depth.

[6] Kuras, M. L., "An Introduction to Complex-System Engineering," 7th International Conference on Complex Systems, New England Complex Systems Institute (NECSI), Boston, MA, 28 October - 2 November 2007

described by Norman and Kuras[7] and others) beyond which the currently accepted systems engineering practices[8] don't carry well. Against these difficulties, concepts drawn from the study of ecology and evolution in the biological sciences, and more broadly the study of "complex adaptive systems," become a useful model for the behavior of these enterprise/ultra-large/complex systems—and hint at the augmentations and complementary approaches needed to undertake formidable systems engineering tasks.

So, what is the language of ecology and evolution, and complex adaptive systems as applied to systems engineering? From ecology we take the conceptual elements of interaction webs, rhizomes[9], niches, and competition. We also recognize energy flow across and among all the interacting elements—and come to appreciate the interconnectedness of things, and the dynamic nature of stability and change.

As we mix in evolution, we add the notions of change and adaptation in "populations"[10]; and we further refine the notions of competition and cooperation into larger aggregates of interacting elements at differing scales/views. This results in an outlook which harnesses variety, selection, and adaptation—and the influence of environmental pressures that impinge upon elements in the interacting mix. Complexity science gives us a language and some additional tools to both qualify and quantify the interactions among the elements we're now in a position to acknowledge as part of our system.[11]

Now we start to talk about systems engineering in terms of "fitness landscapes," "valued elements" forming "attractor basins," and effective capabilities" as "transitive closure" over "element qualities" forming "strange loops" in this "hyperspace." In these terms each system program office is, in effect, engaged in discerning the fitness landscape (i.e., through continuous study and reflection on operational effectiveness and realized operational value of their system elements *in situ*). This analysis and understanding is

[7] Norman, D. O., and M. L. Kuras, "Engineering Complex Systems," Chapter 10, *Complex Engineered Systems: Science Meets Technology*, D. Braha, A. Minai, Y. Bar-Yam (Eds.), New England Complex Systems Institute, Cambridge, MA, Springer, 2006

[8] For a description of these methods, see, for example, the Systems Engineering Handbook, Version 3.1, available from INCOSE at URL: http://www.incose.org/ProductsPubs/products/sehandbook.aspx.

[9] "… Rhizome acts as a web-like structure of connected but independent nodes, borrowing its name from the structures of plants such as bamboo and other grasses. ... Each node ... stands autonomous from the larger structure, but the nodes work together in a larger network that extends benefits to the node without creating dependence. The critical element of a world that focuses power at the level of the individual, ... while providing the flexibility and potential to achieve greater goals, remains the small, connected and relatively self-sufficient node of this rhizome structure. In human terms, such a node represents an economic and a cultural unit at the size preferred by [us]: the household and the tribe. Functionally self-sufficient but not isolated, cooperating but not controlled, the rhizome economy, combined with a self-awareness of control structures, provides the real-world foundation of stability and freedom.", Vial, J., A, Chapter 9: Forward, to Rhizome, *A Theory of Power*, 1 October 1964, http://www.jeffvail.net/2004/10/theory-of-power-chapter-9.html.

[10] Kuras, *op cit.*

[11] Those who do systems engineering always knew these aspects were present, but the "rules of Hoyle" for performing the job encouraged practitioners to view these as external anomalies somehow to be conquered and controlled outside of the systems engineering practice.

predicated on a clear understanding of the various "flows" within which the program's particular elements (e.g., system) are attached to, and engaged within.[12]

On the technical side, we must ensure that our elements are *composable* (capable of being integrated adaptively) with other elements occupying the flow; thus being able to be *composed* into assemblies used to satisfy emergent needs and new operational understandings not previously envisioned. We become sensitive to the selective pressures impinging on our element(s) and seek to adapt them to the current environment at a rate sufficient to ensure their utility, and hence survival (as an evolving population[13]). Not surprisingly, all this usually doesn't communicate very well with the program's chief engineer, who often asks the key question: "So, what do I go do?" Before answering this (properly asked) question, let's make a few observations on what is done today.

What we do today.

Let's recognize that acquisition programs, and the systems they produce, tend to be *insular* (i.e., "stovepiped"). Why is this necessarily so (i.e., isn't it merely an issue of "telling" the programs to cooperate with one another?)? What feeds this insularity? We would argue that it is due to two fundamental conditions—each of which supports the interlocking character of the other. These are, first, *how* good[14] engineers have learned or been taught to think about problems, ways of thinking that the best engineers internalize. Secondly, insularity is aided and abetted by the in-place incentive structures (both explicit and implicit) that penalize rather than reward behaviors embracing interoperability and "horizontal" (as opposed to "vertical") integration. These two conditions have conspired to erect and protect a rather formidable systems engineering methodology worthy of Andre Maginot.[15] These are some strong statements, so let's see if we can back them up.

First, how is it that engineers think "wrong?" Actually, we don't say they think wrong *about a certain class of problems*. But, their success in that context does not prepare them to ask better questions in other contexts—especially at large, enterprise levels (whatever those are). Systems engineers, especially the best ones, tend to ask a series of primary, secondary, etc., questions about any new problem they're given; and it's in the expressions of and answers to these questions that seeds of insularity are sown.

For example, typically there are two first-order questions to answer initially. The first question is: *What are the boundaries for this system?* Clearly, one wants to expend (the usually) limited resources solving the problem at hand, not wandering elsewhere and squandering time, money, equipment, and people on incidental issues. This question is critical for that consideration. Determining the boundaries well provides a set of relatively simple predicates that anyone can use to determine whether an aspect is within one's interest space, or is someone else's problem. It is a simple resource allocation issue

[12] Perhaps now the reader has a better appreciation of the language "barrier" to which we referred earlier.
[13] Kuras, *op cit.*
[14] We're not referring to a small minority of poor performers.
[15] Who, no doubt, produced perfectly fine defensive structures in "The Maginot Line" of World War II fame using an adequate—perhaps excellent—systems engineering process.

when thought about in this way. Recognizing that the essence of engineering is the production of something useful to real people—it's not an academic exercise—the second first-order question to answer is: *What is the most challenging limiting case for this system's use?* By understanding and making explicit the stressing case(s) for the system, we inform all design aspects that follow to help assure that the system will perform well in those cases. One can almost hear the mantra: "Design it right the first time." These two questions are central to the engineering activities of system requirements and design.

Good engineers will also go beyond the aforementioned first-order questions with a series of second-order questions. Three are obvious. They are: *What is expected?* (a question about requirements); *When is it expected?* (a question about schedule); and *What resources do I control?* (a cost question). Answers to these form the essence of program plans. Finally, a good engineer will ask themselves: *How will I be judged?* This goes to the nature of the incentive structures at work.

Let's make some presumptions for our analysis: We know what we want to build. We know the requirements to the level of detail required to answer the two first-order questions. We know what external things our system will touch, and we know what external things touch it. We know the likely stressing environmental challenges.[16] And knowing all this, we've designed strategies and structures, and employed architectures and technologies, to mitigate these issues. In doing so, we've introduced many specializations at many (if not most) of the points of articulation among the various elements. Doing this, elements that must cooperate and interact are tailored for this interaction to prepare for the limiting, most stressing case. It may be that we succeed in providing the needed functionality in a system that will work properly in such an environment. The specialization of the interfaces among the interacting pieces is not an issue because, as an independent, stand-apart system, we are free to do what is needed to meet the *known* requirements, external constraints, and environmental challenges. These latter factors are represented in the predicates we developed to define, understand, and compensate for (in our interfaces and interaction mechanisms) the stress. Thus, the external problem doesn't really exist; Q.E.D.

What we've outlined represents the goals of good (traditional, conventional, or reducible) systems engineering as we understand and expect it to be performed—and one could argue it's necessary to get on with the job. Yet these single system answers, when considered in isolation, i.e., in the context of the system under consideration as a stand-alone entity, tend to act against the needs of the enterprise. And that is precisely what is done today. Why? Because of the prevailing incentive and execution structures, i.e., the "way the world works."

From a practical point of view, not only is there little tangible reward for being mindful of the enterprise(s) in which the system is embedded, there is almost *punishment* for those who pay too much attention to the enterprise. How so? Consider what is <u>not</u>

[16] They might be available communications bandwidth, computing power, number of users, density of transactions, rate of response, etc.

rewarded. It is well understood that relying on an external entity brings a schedule risk that is not under local control for mitigation. Failing to maintain schedule control is one of the “cardinal sins” of program management. Thus, counting on someone else is a dangerous tactic. One is also in danger if external access to the system’s functionality is offered to “outsiders.” This is due to a variation on the preceding argument. If one’s offer to external parties is accepted, then one is on the critical path for another, so how does one respond to technical difficulties which impact the external-facing promised functionality? External promises tend to take a “back-seat” to promises made to the primary receiver of the system. While it makes sense, and seems prudent and correct, it still diminishes the potential value of both offered and received external functionality.

There’s also an issue with whose need is being satisfied when local development is done for an unknown, unidentified external third party. Who “pays the freight” for that one? Who (potentially) benefits? Isn’t this a case of either “gold-plating,” or fraud, waste, and abuse?

We also find issues when some of the presumptions are changed. Suppose it’s the case that one produces a highly-valued and desired element of functionality which is used by a large number of external parties—i.e., it is wildly successful. This sounds like a true success story, yet what is, or could be, done when the usual request for changes arrive? Who pays? How does this impact schedules? Where are the “new” background requirements articulated such that there are justified, approved and valid needs?

We fear these problems have much to do with our notion of “system” itself. Systems are insular because we tend to view them that way, i.e., as isolated and stand-alone, and embedded in an environment that we cannot influence. Systems of systems (SoS), much in vogue right now (and for the past several years) as a definition exercise, discussion topic, and engineering challenge, can’t seem to help being an overt construct or prescribed entity (as contrasted with a discovered assembly). Look at the difficulty the Office of the Secretary of Defense/Acquisition, Technology and Logistics (OSD/AT&L) is having defining an engineering guide[17] for this “animal.” It smacks of “new thing, same as the old thing.” We suspect the issue is (as just stated) the notion of a system as we currently understand it; which we can’t seem to avoid when we go through the engineering process hinted at in the set of questions a good engineer asks.

What we might do differently.

We feel the need to grant that systems (with all their insular characteristics) exist, and will continue to exist. They will be defined for particular needs, and will be subject to particular constraints which will require clever engineering to deliver utility to needy users. The systems that have the development characteristics with which we’ve taken issue in the preceding section have a permanence and persistence *as systems*; not just in name, but in fact. That is, the *system* is what was defined, designed and built. The system elements persistently remain in their relative positions and within their system-defined interrelationships, being used for little or nothing else except in support of the system.

[17] https://acc.dau.mil/CommunityBrowser.aspx?id=147489

The elements were probably, for the most part, built *for* use in the system. Thus, these systems are systems both in name and in fact, over time.

Yet, we can stretch system element utility by augmenting our thinking, analysis, and synthesis so that today's systems present opportunities for assembling and creating tomorrow's systems—which (we assert) generally will have a different quality than today's systems. While tomorrow's systems may be a system in name (and by this we mean a collection of composed elements, having useful functions, and known by an identifying label which connotes the collection), they won't have the permanence that today's systems do. They will be a system only for the duration of use; this is denoted as "late binding." At other times, these same elements will be members of other collections which can be (and already are in some instances) assembled in many different ways for many different purposes. Let us be clear: this is a necessary condition if we are to realize the net-centric goals[18] set out by the senior leadership of the U.S. Department of Defense (DoD); and by others in similar leadership positions in other countries.

How will this come about? We must come to understand how to build with the equivalent of the Lego^{tm} block toys. This is not a unique idea; Lego blocks have been used as metaphor by many. Ease of composition is the principal characteristic that renders them useful for their intended building purpose. Each piece can be affixed easily and in a variety of ways to the interim structure. What one gives up in composing the blocks is the hope of achieving perfect fidelity to the intended structure one is attempting to create. Because of rough edges, etc., the composition only approximates the precise character one has in mind (unless, of course, the targeted ideal can be built within the limitations of the building pieces). But building with Legos doesn't have to end with these approximations. There are many specially shaped pieces and mechanical parts with which one can decorate the approximation to add the specializations and actions desired, or to smooth a discontinuous edge[19]. In this way the approximation has a greater fidelity to the desired end. There are lessons here for us.

From a chief engineer's point of view, a few heuristics can turn the above concepts into operational practice to mitigate the insularity issues often found in systems development.

1) Focus on the *fundamental unique value* (see Appendix) your system offers to the enterprise.
   This core value isn't all the "gingerbread" required to deliver a full system, but its essence, i.e., the essential capability motivating why your system was to be built in the first place. With respect to the Lego model analogy above, here it may seem like we're concentrating on the specialty pieces first; and indeed that is what we're doing. The other run-of-the-mill, commodity pieces are your infrastructure and the composition mechanisms—they are not the fundamental reason that *your* system was

[18] http://www.defenselink.mil/cio-nii/

[19] In fact, these specializations can be adaptations of the standard set which have been subject to particular selective pressures. Thus, they can be understood as outcomes of evolutionary processes acting on the individual blocks. One can also view the thing built—which approximates the desired goal—as being acted upon by evolutionary forces. So the aggregate also adapts. It perhaps presents an "ecosystem" for the blocks with which it is built.

imagined. In effect, you are offering your principal specialty piece for use by others in the enterprise for reasons, and in ways, which are apart from <u>your</u> system's precise purpose. Yet, by providing its essence for composition in new contexts, you are creating new opportunities for others to discover new value-chains—and for the enterprise to achieve the desired *net-centricity*.

2) Develop and use "casual" technical composition mechanisms first.

   A concept seemingly missing from our technical considerations is one which recognizes, explicitly, a continuum of technical relationships from the casual to the intimate. Like casual <u>social</u> contexts which require a minimum number of unique agreements *a priori* and little understanding of local context (generally called "politics"), a casual <u>technical</u> relationship should require a minimum set of unique interface protocols *a priori*, and <u>no</u> understanding of local context (generally called "state").

   A casual technical interaction might be a stateless (independent of present state) query/response interaction. An example would be the use of HTTP[20] as an interaction protocol, and XML[21] for the encoding rules for an exchange over the HTTP protocol. This requires only a low investment yet allows us to test our ideas about the value of a particular collection and set of interactions. We find this today in so-called *Mashups*. Upon validation of the aggregate value of the collection, one can add or modify interactions, relations, and shared context to bring the approximation in line with the goal. This has the immediate advantage of saving resources for use on those things with proven value. It also fits the potential goal of the ultimate characterization of a more intimate relationship (as contrasted with the casual).

3) Know how you will offer access to your elements of fundamental unique value; and what interaction models will be proffered.

   As a first approximation, you know what the system offers, and how that offering can be experienced by a client system. This understanding suggests how the initial offerings will be structured. For example, in Theater Battle Management Core Systems (TBMCS),[22] an information collection called an Air Tasking Order (ATO) is one of the defined products. This product is consumed by many already-existing systems. It seemed reasonable to expect that it also would have utility to others. By this reasoning, it was made available as an XML-structured data payload accessed through a simple query/response mechanism.

4) Provide a mechanism for reducing the integration barrier such as putting in place a *developers' network*, "points-of-presence" with offered functionality exposed as live services.[23]

   Integration is an existing open-ended problem for achieving the agility demanded by the push for net-centricity. To date, when one expects to integrate with an existing

[20] Hypertext Transfer Protocol

[21] eXtensible Markup Language

[22] http://www.marcorsyscom.usmc.mil/sites/TBMCS/Default.asp

[23] Considering how difficult the integration challenges are for one to use others' systems, what would make it easier (and symmetrically, easier for others to use yours)? A proven answer is the creation of a point-of-presence on the net where the offered functionality can be accessed and exercised. Generally, the functionality is packaged as "services," and if structured with casual interaction models (e.g., query/response), potential users of the functionality require little investment in exploring and integrating with the offered functionality.

system, one usually looks to get a copy of that system into one's own development environment. This requires a long-term commitment to this foreign system. In addition to purchasing whatever is required to set it up and get it running, one must develop and maintain in-house expertise in the system itself, as well as keep abreast of changes in the foreign system subsequent to bringing it on-house. Obviously, this is resource intensive. It is untenable for more than a small number of foreign systems. As communities of *developers' networks* are "stood-up" and used, a new dynamic for design, development, and use may emerge. Given the lower barriers to use and integration, this dynamic will support exploration and discovery, and may result in a reinterpretation of what it means to be a system. Systems may be de-constructed into sets of composable parts, and then integrated, and re-integrated into collections which provide capabilities to users. Initially, those elements of unique value, being exposed and made available in venues such as these developers networks, create new opportunities for those close to the network "edges" to assemble the approximations (or realizations) of the needed capabilities.

<u>A large unmet challenge.</u>
We have made a case for why and how to accelerate movement in these network-centric directions technically and operationally, and what one might tell a chief engineer. However, we're not so sanguine to believe these aims will be achieved on a large scale anytime soon since the economic and business structures really don't support them well. Nevertheless, we should endeavor to continue experimenting with these techniques in pockets of opportunity. We believe such efforts will prove successful and will, by example, gradually change the way the world works for the better. To see what might improve, we will elaborate a little more on the difficulties within our present acquisition culture.

Consider the typical system program office and their contractor. There is really no tangible benefit, other than "good will," perhaps, to offering functionality to third (probably unknown) parties. Even assuming that a proffered piece of functionality becomes wildly popular and is composed into a multiplicity of different "systems," what does the offering organization get from this condition? The answer, unfortunately, is nothing of true significance. No additional monies are likely to flow. The program manager may receive a few laudatory commendations from some quarters not in his/her "chain of command." On the other hand, it is likely that there are real negative consequences, if anything. For example, the program manager may be reprimanded for operating outside his/her "lane," and/or penalized for this dissolution of assigned responsibility by not receiving a promotion or a desirable next assignment in their career path. Also, instead of being compensated for good results, the program may have expended funds that could have been used to strengthen the vertical integration of their system. In addition, there is likely to be a barrage of new suggestions, needs, and desires from the new clients of the functionality.

While we've made the argument that casual interaction mechanisms creates an environment which is parsimonious with resources,[24] the source of these additional

[24] "Intimate" mechanisms await demonstrated needs before resources being applied.

resources typically remains unstated or even unknown. In our world of funding developments—not use—there is no revenue flow received by the offeror which equates to the value received by the enterprise. Further, there is no real cultural or developmental benefit to a system program office for using another's functionality.[25] It likely places an area of risk onto one's critical path, while reducing the resources under one's direct control—and likely reduces the revenues available by doing so. This also has the likely impact of reducing the perceived need for the number of persons in a development shop and program office, and thus reduces the likelihood of program manager and staff promotion, etc. There is also the risk of suffering the accusation of "gold plating" a development beyond its stated requirements, as one is building functionality without a particular client being known (and funding the development).

If the above four heuristics do not promise to materially assist chief engineers and their program offices currently, from where would the incentives come to do such things? An answer may lie in the idea that presenting collections of composable elements of valuable functionality is bound to create new opportunities. For whom? Perhaps the end users. Today, end users have two basic options: 1) self-service (i.e., build their own tools and solutions using local talent and discretionary funds); or 2) reciting their perceived needs in the form of written requirements that then enter the formal acquisition system. Providing a third option seems worthwhile: 3) having accessible functionalities easily composed into approximations of (new) systems meeting current needs (or enabling new ways to approach those needs) may allow end-users to satisfy their objectives with real proto-systems. This third option seems to bridge the user's present choice dichotomy, and may lead to resources applied to augmentations that are very likely to deliver real value. This new way of doing business also suggests that such developments will deliver both elements of systems as well as systems themselves. Revenues might be directed (after the fact, i.e., paying for results[26]) to those organizations whose elements are used in the field within real solutions, and for which specializations of the elements are (or were) needed. While this hypothesis keeps the current framework with which we hire contractors, there is still a need to develop a system of revenue flow which rewards actual use instead of paying for development. Gradually, over time—it would certainly years and perhaps even decades—the acquisition system might change in transferring more and more of the up-front monies for development to the back-end, i.e., operational use.

The previous discussion can easily be interpreted within the conceptual framework of ecology and evolution; and with their language of variety, selection, and adaptation. All the characteristics conditions are present. We have also preserved the system-centric viewpoint within which the chief engineer must operate. The bridging concepts are the notions of recognizing and focusing on what each system brings to the enterprise, making

[25] Note that, by our previous argument, one's own system users could benefit operationally from other systems' unique values if they are assembled with our system and, of course, if relevant to their mission space.

[26] This is a fundamental tenet of complex systems engineering, in contrast to paying up-front for perceived promises; for example, see Kuras, M. L., and B. E. White, "Engineering Enterprises Using Complex-System Engineering," INCOSE Symposium, Rochester, NY, 10-15 July 2005

this aspect accessible to potential clients with lower barriers of integration, and then lowering the resources for discovery, exploration, and use.

Is this whole concept sufficient for achieving full net-centricity in the large? Probably not, but it provides an approach which operationalizes the ideas being "bandied about", and provides a useful starting point and some suggested guidelines for moving forward and changing the acquisition world to be more effective in complex environments. And it answers the chief engineer's question: "So what do I go do?!"

Acknowledgment

The authors gratefully acknowledge the contributions of Robert (Bob) S. Miller of The MITRE Corporation, and commend him for visionary leadership supporting new ways of thinking about the systems engineering of large systems.

Appendix

Fundamental Unique Value—what it is, and how to recognize it

As pointed out, the systems engineering "game" has changed. The old game was "certainty and assurance." The new game is "agility," not just any agility, but an agility brings together people, processes, and technology, i.e., software/hardware elements, for creating adaptive solutions to new needs just recognized, or which just came into existence. The elements brought together (i.e., composed) have never been together before, weren't conceived to be together (or brought together), but they need to perform well immediately upon being composed. Isn't this precisely the value proposition for *net-centricity* in the military enterprise?!

Often as systems engineers on insular programs we fail to ask the right questions (from a support-to-the-enterprise perspective) such as: "What elements *should* be brought together?" Or stated in a slightly different way: "What elements, if brought together, would likely provide operational value?" Although some would argue this mischaracterizes things, in that the "what" is exactly that which is supposed to be provided by well-stated *requirements*, as these <u>should be</u> the "what" being asked about. That is, if there was a requirement to have interactions between (say) *this* and *that*, it would be stated. This often articulated thought may miss the point, viz., by definition, such requirements can<u>not</u> be foreseen: *The elements brought together have never been together, weren't conceived to be together (or brought together),* … . What requirements would we use? From whom? One could argue there is an <u>implied</u> requirement which asserts the need to bring any arbitrary element (or piece) of one system together with any other. But, how can we work with such a requirement? It is not assignable or "bin-able" in any particular area or to any particular system; yet it needs to be embraced everywhere. So, does this suggest it is an architectural or infrastructural element? And, if so, wouldn't its scope put it beyond the control and purview of any single project, program, or system? Absolutely! Yet, we've tried many times and in many ways to <u>legislate</u> infrastructural solutions with (stated in the most charitable terms) mixed results. We can never seem to develop the consensus needed to achieve the "everywhere" part.

Perhaps if we come at it from a slightly different way, this notion of agility might reveal some of its mystery and give up some of its recalcitrance. If we transform the above question into "What elements or aspects will it likely be desirable to compose?," we might be able to grasp things better, and in a way which might permit us to "sneak up on" a solution. While we may not have a direct answer to the question posed, we can identify and start to articulate a useful pattern. In fact, in complex systems, where one is not in control and cannot predict or prespecify what will happen as the system evolves, direct questions that presume we can know precise answers just "don't fly." One needs to bring a strong dose of humility to the problem and ask more leading, open-ended, and indirect questions to help discover what may gain traction.

In our world, here specifically the to-be network-centric world of U.S. DoD (and allied) Command and Control (C2), we might characterize those elements that potentially are desirable to compose into new operational value chains as the "elements of unique operational value." That is to say, these aspect(s) of a "system" containing its *fundamental unique value (FUV)* which it purports to bring to the enterprise. But how do we recognize them? While any system likely brings many things to the enterprise, only a very few represent its FUV. This is a point we often lose as we seek to build full, complete, and stand-alone systems. The reason for building the system in the first place probably contained a hint as to the FUV the imaginers of the system thought was needed; and its useful to start here. Ironically, we tend to lose this focus in the day-to-day "hurley-burley" and "brush fires" symptomatic of the underlying political need to prolong the survival of our individual program rather than concentrate on the survival of the population of programs that can contribute to improved net-centricity.

Complete, fieldable "systems" have many "moving parts" that go well-beyond the initial imagining of its FUV. Most systems also yield distinct products as the result of their use and operation. These products may not be synonymous with their FUV; rather, the products are the result of the FUV's application, and we should disentangle such products from the system's FUV.

Understanding the FUV of a system is job-one for a project leader/system manager. As argued, it will not be obvious necessarily, and the articulation of FUV likely will require some thought. Some examples will help. First, consider flight-planning systems. The seemingly straight-forward flight planning system (FPS) of one sort of another is used by every pilot in the DoD. As a system, they offer a couple of products which the pilot needs: a route of flight, and a "data-load" used to initialize the avionics in the aircraft just prior to the flight. Are the flight route and the data-load the FPS's FUVs?" They're certainly valuable, but are they values of the fundamental unique kind? And, are they the elements which will likely be composed with other elements to produce new value chains? Here's a recommended "gut-check" to help in answering these questions:

1. *If the product was altered in some small way by some un-named external entity, would the product still be valid?*

After all, in both cases the product is really just a particular formatting of some particular data. No "magic" there. After the alteration, even if formatted correctly, we likely wouldn't have faith that the product was still valid. Why? Here's a clue: What about "validation?"[27] Hence we discover the FUV of all FPSs used within DoD: the Flight Performance Model (FPM). The FPM ensures that *this* aircraft, in *this* current configuration (constantly changing based on fuel burn, etc.), can fly from here to there within its performance envelope. We would accept the slightly-altered route of flight if it were validated by the FPM. At least we would have faith the route could be flown (whether it is the right route to be flown to achieve a particular operational effect is a separate question). So, here's another gut check:

> *2. If an element were eliminated (as if it never existed) would it need to be re-discovered and recreated?*

Clearly, that would be the case for a FPM.

Of course, FPSs interact with the users in different ways. Some are table-oriented requiring the user to make entries for each successive waypoint in a tabular presentation. Others use a map interface allowing the user to connect waypoints, as dots, by drawing lines on a displayed map background. Both approaches can (and do) produce valid flight plans. Are these differing user-interaction approaches elements of FUV? A third gut-check:

> *3. Does an element or aspect of a system clearly separate itself from all other systems which produce similar products?*

Although almost too obvious to state, the differing user interaction mechanisms represent FUV of each. We can anticipate an objection: "The map-oriented interface obviously is so superior that it should completely replace the table-based one." Two comments on this: First, that it's not an either/or; both approaches have their strengths and weaknesses, and neither is right for all conditions. For example, the map is a poor interface for placing precise locations. It is much easier and more precise to enter specific latitude/longitude values using a software table or a written form than it is to "poke around" on a map attempting to find an exact location. The second point: It is the actual users who should assess actual value received, and while we suspect that users would prefer a map interface for most instances; that wouldn't always be the case. Value offered and value received are quite different, and we will discuss this again below.

By the way, the use of the map interface as a common mechanism for displaying any arbitrary geo-located data is generally known as a Common Operational Picture (COP). A particular flight planning map-interface application (Falconview[28]) has found favor as the preferred display and user-interaction mechanism for a host of uses and users involving geo-located information items. We have failed to recognize it as a visualization mechanism composed with various information streams, and so we treat it as a "thing," and we fuss with one another as to whose COP is best, whose collection of information is best, and whether anyone (and whom) actually has the "right" to produce and manage

[27] Trust is a huge issue in this arena, especially when lives depend upon it. Products from "un-named external entities" may be trusted by no one; even when coming from a known, trusted entity, their stamp of approval is usually necessary to engender trust in the product.

[28] `http://www.falconview.org/`

"the COP" (note the definite article—it's often used). Unfortunately, this "steps on" (countervails) the very characteristic we need, to offer both valuable capabilities to the user, and the variety needed to discover and amplify real goodness.

There are probably a few more elements which represent FUV of FPSs, too. We could argue that the knowledge of particular formats and organizations of information so as to load a Data Transfer Device[29] is an element of FUV, as well, using Gut Check 2 above.

Here's a second example. Consider TBMCS (mentioned in the main body of this paper). This collection of closely-aligned functionality produces a couple of products; the primary ones being ATOs and Air Control Orders (ACOs). These represent the daily orders that describe the air movements and sorties which have been planned and coordinated for a given day across a whole theater of operations. The ATO is a lawful order to take specific action. Considering that the planning process usually contains U.S. allies, the ATO has international scope. The ACO contains the information and definition on the proper designations and use of the airspace during a particular time period and theater of operations. Together the ATO and ACO provide a clear picture of the planned and approved air activities.

Now the question: Are the ATO and ACO products the FUV? They are certainly unique in that one wouldn't want to have a collection of ATOs and ACOs for a day which differed from one another and whose pedigree couldn't be ascertained. Which ones would be "correct?" They may all be at least partially "valid" in that they all corresponded to a known and accepted format for ATOs and ACOs, and could be used as input to other processes which accepted and used them. Clearly, the correct ones would originate from the organizations with responsibilities to produce ATOs and ACOs. They would be presumed to produce products with valid formats, yet even if not correctly formatted (which would be a problem), their ATO and ACO would be "correct" in its meaning (although there may be issues with understanding the meaning given format non-compliance). We say that the ATO and ACO are the products which *result* from *applying* the FUV. What then is the FUV?

The FUV of TBMCS is twofold: It is the understanding and assembly of a "good mission" and the composition of those good missions to produce a "good collection of missions." This seems to beg the question a bit: What constitutes a "good mission?," and how does one compose the same into a "good collection of missions?" Succinctly, a "good mission" is one which satisfies the commander's intent, uses available resources, and can be defined within the operational constraints (such as rules of engagement, air control measures, and valid aircraft routes). A "good collection of missions" is a collection of good missions which is deconflicted in time, space, and effect. Simple to state, tough to do! It is clear there is judgment involved, and it is not likely strictly algorithmic; we'll defer a discussion of this particular issue to what follows below.

[29] The device by which the pilot carries his validated mission plan from the planning machine to the aircraft, and loads it therein.

More *apropos*, at this point, is the recognition that producing a “good mission” requires the ability to know whether it is possible to assign a mission to a particular aircraft, or aircraft type. This suggests an appropriate use of the FPS’s unique value (discussed previously): the FPM. Currently, we don’t do this. We use locally-available people familiar with the flight characteristics to offer qualitative assessments. Therefore, “good missions” in TBMCS are really conjecture on the part of the theater planner; it requires a judgment made by a human with particular knowledge of the various performance characteristics of the aircraft in the inventories of all the participants to render good conjecture in this area. If we wanted to minimize our reliance on such a person, who really may be needed in the cockpit (!)—automate more and reduce the person-count for theater planning—we could substitute the FPM. This bolsters our earlier-claim which stated that FUV is that which would likely be composed to create new value.

Continuing with TBMCS’s FUV: In its current instantiation, TBMCS attempts to lay out a process for building ATOs. Many users feel it falls short, and it’s instructive to look at this a bit. As mentioned, TBMCS produces validated and verifiable ATOs. Yet, there has been a desire to replace TBMCS with another application which also produces validated and verifiable ATOs. If systems which produce the same products are equivalent, then one would <u>not</u> expect such a push. One explanation for this desired change is that the replacement application has a different notion of a “good collection of missions.” This is due to its substitution of a continuous ATO for the more-traditional 24-hour ATO cycle which is implemented explicitly by TBMCS. Thus, by this analysis, the primary reason for a “changing-out” TBMCS is the change in the manner of expressing the FUV of this class of system. If this element were replaceable, then there wouldn’t necessarily be a pressure to change the whole system[30].

One easily can imagine many other compositions of FUV elements which could provide new or augmented capabilities. What we don’t have is a collection, directory, or even a listing, of what elements of FUV we have across the enterprise. A discussion centering around a directory of FUV, when attempted, tends to degenerate into a technical debate of the various discovery, meta-data definition and collection approaches; all interesting, appropriate, and necessary, but secondary to knowing what we actually have in any form that can be assembled or composed into new value. Adopting new approaches often only comes from demonstrated need. In our context here, demonstrated need is driven from the operational edge, not from technology. Not knowing our systems’ elements of FUV limits our ability to postulate new value chains that could satisfy existing or future operational needs. Discover and articulation of our FUVs is the central essence of net-centricity, and we’re not there yet; from our point of view we haven’t even really started.

The approach outlined here is really a generalization of a known pattern from Object-Oriented design which suggests that in a good design one “encapsulates things that change.” Applying this pattern, we could paraphrase it as: “Encapsulate your *FUV*.” But first, of course, identify it.

[30] It could also create a “market” at the replacement point, but that analysis is for another time.