Chapter α

Engineering Complex Systems

Douglas O. Norman
dnorman@mitre.org
Michael L. Kuras
mlk@mitre.org
The MITRE Corporation

a.1. Introduction

This chapter motivates the need for, and introduces a formal set of processes that constitute the practice of, "Complex Systems Engineering" (CSE). Our experiences and observations strongly suggest Enterprise Engineering is best approached using CSE to engineer and manage the enterprise[1].

Using the current instantiation of the Air and Space Operations Center (AOC[2]), and the desired evolution of it, the AOC is shown to be best thought of as a complex system. Complex Systems are alive and constantly changing. They respond and interact with their environments – each causing impact on (and inspiring change in) the other. We make the case that a traditional systems engineering (TSE) approach does not scale to the AOC; consequently, we don't believe TSE scales to the "enterprise."

We introduce a new set of processes which complement – and do not replace – the processes that constitute traditional systems engineering. The methods for the engineering of complex systems are based on a view of complex systems as having the characteristics of an *ecosystem*, and the use of processes which take advantage of

[1] Initially, we appeal to the reader's intuition for the definition of an enterprise. The picture in one's mind should be something like a large collection of independent organizations, loosely associated to achieve something in common.

[2] A special thanks to Col Pete Hoene, USAF, and Col Terry Szanto, USAF, the past and current AOC WS System Program Office Director, respectively, good friends both, for their continuing professional engagement in this topic, and their kind reading of this draft along with their helpful comments. Special thanks also to Col Joe May, USAF (ret) who took the time for healthy debate and discussions both as the Director of Operations for the Air Force's C2 and ISR Center, and subsequently. Joe put the operator's "stink" on the thoughts.

emergence and which deliberately mimic *evolution* to accomplish and manage the engineering outcomes desired.

The chapter is structured in four major sections:

- Why Rethink Systems Engineering?
- Complexity and Complex Systems
- Engineering Complex Systems
- Complex Systems Engineering in Practice

We all must come to grips with the non-deterministic nature of enterprises. We hope to extend the concepts and methods of Systems Engineering to complex systems, and to open up the professional dialog so as to codify the engineering and management of complex systems and enterprises.

a.2 Why Rethink Systems Engineering?

First, we should take stock in everything accomplished. We've designed, built, fielded, and operated two Air and Space Operations Centers (AOC) which provided the tools used to plan, task, and monitor all air operations in both Operation Enduring Freedom in Afghanistan and Operation Iraqi Freedom in Iraq as well as the AOC created for NORAD to manage airspace and combat air patrols in the US after 9/11 (Operation Noble Eagle). Other packagings of the functionality are fielded with Special Operations units, and with the Navy and Marines.

Yet, we are having trouble building, integrating, and modifying large "systems." Our difficulties are legion [e.g., see Bar-Yam 03]. Struggling with these failures, as a community, we have continued to refine our notions about systems engineering, and how we define, design, and prepare for systems; but– again, as a community - we haven't changed our underlying "mental model" which informs our general (and specific) philosophy and processes. We continue to view Systems Engineering as fundamentally about allocating desired, known functionality among specific elements of a design; all known *a priori* and stable over time. The users of the functionality built often accuse us, the developers and acquirers, of being "late to need," "unresponsive," and "too expensive."

We respond with a lexicon carefully crafted to put the onus back on the users. We say that the users' requirements are unknown or poorly stated; that, if the requirements are known, there is a requirements drift (i.e. modifiying the requirements), or requirements creep (i.e. adding additional requirements). We suggest that the user can't (or won't) say what they *really* want, nor how they will use that which is to be built and delivered. This situation results in processes which focus on detail: detail of proposed use, detail of environment, detail of design, and detail of planned schedule. The more detail the better.

During the last decade as the world has moved from stand-alone automation which augmented individuals, to networked and shared automation, architectures became the way to deal with the rising complexity. But the early forays into architectural-based engineering didn't seem to pay the dividends anticipated – complaints from users continued, and the most-valued automation tools were ones not built with the same careful attention to detail required by the Architecturally-based Systems Engineering approach practiced. Rather, it seemed much of the architecture-supplied understanding and simplification was implicit. We saw the *technical architectures*, rather than being defined and engineered, were being imposed by the development tools used; and the need for *operational architectures* (what will this system do…and how will it be used) were rendered unnecessary since "experts" were supplying the operational insight directly to the developers. Further, these user/developer collaborations were turning products out quickly; and they were valued by the end-user.

As we attempt to make systems more useful and valued we also start to come face-to-face with the limitations of our current methodologies. For the Air and Space Operations Center, there are two clear forces pressing on it and demanding its evolution: 1) speed of accommodation to new understandings of current missions, or new missions demanding modification of the current AOCs; and 2) application to new doctrinal uses or mission types [Albert 99, 01,03]. In all cases, the manner in which we practice Systems Engineering seems to bog us down; and we're compelled to rethink the practice. This analysis leads us to recognize that there are additional classes of Systems Engineering problems. Each class requires a different qualitative mindset; and consequently, a different set of tools, techniques, and procedures to undertake the task of systems engineering successfully.

Systems engineering, at its simplest, attempts to understand a desired outcome from the interactions among people and things, and is roughly divided into two phases: Analysis and Design. During analysis the context for the desired outcome is explored to uncover the initial conditions and resources available which can contribute to the desired outcome. Then, with this understanding, processes are designed which are able take the initial conditions and transform them into the desired outcome. As the designing proceeds, roles and activities are allocated among people, hardware, software, and organizations. Flowing out from this allocation are the ancillary activities implied and required by the design. For example, if an activity has been allocated to a person – and that person is expected to perform in a certain manner – then the person must understand their role in the system, and must be trained to perform that role. At all times design boundaries and constraints are examined and evaluated to ensure they are not violated. For example, if precision must be maintained, then cumulative error must be watched; or if certain information flows are required, then there must be connectivity and sufficient bandwidth to satisfy the design, etc.

The preceding certainly seems to make sense. And, the more complex or critical the desired outcome, the more important it would seem to conduct this process with more rigor, more information, and with more detailed plans; after all, we have to get it "right."

So what's the issue? In a nutshell: the mindset, skills, methods, and processes used to develop "systems" in this way seem to fail us when we attempt to craft "Systems of Systems." And, the AOC is certainly a System of Systems.

a.2.1 Challenges with Engineering the AOC

The AOC is known as a "System of Systems" (SoS). As such, it is envisioned as a system assembled of other systems so as to offer the capabilities needed to perform roles assigned to an AOC. Implicit in this is the expectation that the systems from which the AOC is assembled can be composed into an AOC *System of Systems*. This has proven harder than anticipated, and it provides insight into the challenges of Enterprise Engineering. While still viewed as a "system" by some, the AOC turns out to be an "enterprise" in the small. As in all enterprises, it is composed of different pieces representing different fiefdoms and principalities; (or "tribes" as Gen J. Jumper, AF Chief of Staff likes to say) Listed and discussed below are those characteristics that have proven to add difficulties to the intended composition of systems into an AOC SoS.

The AOC today is assembled from over 80 elements. There are infrastructure elements, communication elements, applications, servers, and databases. The goal is to compose the desired capabilities from the elements found in, or which can be brought into, the AOC. For the most part, today's systems are **not composable**. The systems:

- Don't share a common conceptual basis.
- Aren't built for the same purpose, or for use within specific (AOC) work flows, or for use exclusively at AOCs,
- Share an acquisition environment which pushes them to be "stand alone"[3],
- Have no common control or management,
- Don't share common funding which can be directed to "problems" as required,
- Have many "customers," of which the AOC is only one,
- Evolve at different rates (as do individual system components) subject to different (generally uncoordinated) pressures and needs.

Because of the above, **ensuring integration and interoperability are unbounded, unpredictable engineering activities**. The following observations clarify this further:

[3] The DoD's acquisition system is built around the concept of a "system" which seeks to separate a given system from every other. This separation extends from the concept through delivery and sustainment. Funds executed on behalf of the system acquisition are, by law, separate from all other monies with Congress carefully monitoring expenditures.

Observation 1: The AOC SoS is an opportunistic aggregation, not a design.

- Only the AOC System Program Office (SPO), which has the acquisition responsibility for the AOC SoS, has a strong interest in an overall AOC design, and has no way to enforce such a design on others who supply the component systems of the AOC,
- Since the AOC SPO doesn't spend its money for many of the component systems of an AOC SoS, the component system-owners have little incentive to comply with, or respect, an AOC design,
- The SPOs for the component systems in an AOC SoS must remain responsive to customers and users with interests other than the AOC, and,
- The need for, and the appearance of, a specific new capability at an AOC is often driven by a new, immediate need not apparent to or felt by the other customers and users of the component systems in an AOC SoS; and to which the AOC SPO would be unable to satisfy within the time-of-need.

Observation 2: Integration-enabling technologies (*glueware*),, are grafted onto the elements (systems) of the AOC, and integration developments are undertaken, after delivery of the component systems to their prime customers,

- Each element in the aggregate is designed and built with its own understanding of the world – around its own set of "conceptual atoms"
    - Integration among these elements requires effort (resources) to understand and bring these potentially disparate "conceptual atoms" in line so they can be composed,
- Integration is a source of work and revenue – using today's dominant business model (employer/contractor) contractors sell engineering hours,
    - "Big Integration" is a potential cash-cow for those who perform it
    - Little incentive to limit the work, or find ways to be more effective
    - Integration of already-developed elements guarantees that the delivery of an integrated, operational AOC will lag behind the availability of the individual elements; however, the expectation from the users is that *general availability* (when the component systems become available to the users) and *integrated* are synonymous. This leads to customer disappointment; and is further compounded by the need to expend additional funds to perform the integration proper.

Observation 3: Funds for integration are limited

- Willingness (and sometimes, the ability) of the user to wait is limited, and accelerating deliver (if even possible) costs additional money;
- Perceived barriers for building automated functionality (in software) are low, setting customer expectations that it's easy, quick, and cheap;
- Integration tends to be built around a defined work flow which implements a specific concept of operation. Integration "glue" which implements the concept of operation binds systems into rigid relationships. This is contrary to achieving "agility" and "netcentricity;" [Alberts 01, 03]

- The ability to conduct tests has an inverse relation to system size;
    - Resources (organizations, staff, time, money) available to conduct large tests are limited;
    - Test coverage plans for large systems becomes unwieldy;
    - Likelihood of finding incompatibilities during large test rises (due to the possibility of uncovering an unexpected transitive affect);
    - Understanding of how to proceed once a problem is uncovered is an open question;

Note: The process of testing of SoS must be rethought; the goals and the uncovering of problems are good things which must be preserved – just not at system test time; especially operational test.

Observation 4: "Value" assessment is not by those who use the capabilities

- The "marketplace" serviced by the acquisition system is the selling of engineering hours through the promise of future assemblies and creations; and the delivery of these creations; not by the assessment of value or utility by the AOC staff (these aspects are supposedly contained in the formal *requirements*),
- Those who use the creations of the formal system have only an indirect influence; any direct influence being the result of heroic efforts on the part of individuals
    - This tends to bring into being a "black market" of applications and functionality – and the hoarding of local "slush funds" which can be directed by the local commanders to satisfy needs as they arise.

Observation 5: Plans (and Planning) as a primary SoS strategy has problems

- Focuses on the future – but is based in the past
- Tends to fix an early (likely incorrect and incomplete) view
- Activities tend to twist reality (subject to unplanned change) to the plan (static, based on past beliefs)
- Imposes expectations, and dependencies, on partially-interested participants
- Design implied in the plan is based on today's understandings. As things change in the world all the elements to be composed are subject to different pressures and decisions which likely will not align
- Assumes a success based on promises (staying on-plan) – not achievements

Additionally, there are new operational concepts being considered, developed and employed. These include over-arching concepts such as Netcentric Warfare (NCW) [Alberts 99], and technical concepts such as the Global Information Grid[4]. For the AOC, new operational ideas such as Dynamic Tasking and Effects-Based Operations are taking hold. Supporting this growth and change, at an acceptable rate and at an acceptable cost, is often described as "agile acquisition." Yet, there are few examples of how to achieve this "agile acquisition."

[4]Global Information Grid Capstone Requirements Document, 5 JROCM 134-01, August 30, 2001

How can these new operational ideas be cast into capabilities which can then be integrated in the AOC? What works? There seem to be characteristics which militate against success, even when carefully practicing Systems Engineering as we know it. Before these characteristics are introduced and reviewed, and the CSE approaches by which complex systems may be engineered are discussed, we present a quick introduction to systems engineering and complexity as it applies to systems (and systems engineering) .

a.2.2 What is Systems Engineering?

Traditional systems engineering (TSE) has its foundations in Linear System Theory (LST). Key ideas are proportionality, superpositioning, and the existence of invertible functions (i.e. $x = f^{-1}(f(x))$). There is also the assumption of repeatability. These ideas, coupled to an attention to detail, explain why traditional systems engineering works as well as illuminating the boundaries of its applicability.

Traditional systems engineering begins with the specification of requirements. Closed and complete, precise and fully detailed are the ideals. Systems are then implemented to comply with or to satisfy exactly these requirements.[5] The practice of TSE is the application of a series of linear transformations moving from the statements of the requirements through to a preliminary design, a final design, the actual development, then testing and fielding. A hallmark of the process is the ability to justify everything built in terms of the original requirements. If requirements change it dislodges the careful scaffolding upon which the system rests. Change ripples through everything; and; therefore, approaches which isolate the impacts of change are sought. Since "no change" is the desired (and expected) state of affairs, engineering efforts shift to development efficiencies.

Decomposition and then integration (or assembly) are the bookends for the implementation that follows specification, both of which depend upon the applicability of LST to a given situation. Because of this, there is a strong preference for hierarchy in both implementation activities as well as in the result.[6]

Traditional systems engineering relies on the making of and the fulfilling of predictions. These predictions are more binding for traditional system engineers than any current realities as is seen (for example) in development of PERTs, and in formal testing procedures which are made independent of implementation but that are predicated on the same requirements. As long as predictions and realities diverge, a preference is given to preserving the predictions. Descriptions of traditional systems

[5] Engineering is always an approximation, however. Traditional systems engineering assumes (or asserts) that the ideal result is "closed" and that it can be completely pre-specified. The appearance of Interface Control Documents (ICDs), for example, is an exception that illustrates (or preserves) the validity of the general rule.
[6] Refer to the brief discussion of multiscale analysis. This preference for hierarchy is actually a result of the fact that LST is limited to a uniscale analysis and synthesis of a problem and its solution.

engineering can be found in many places. A quick "google"[7] of the term brings many hits (millions), including many universities offering systems engineering curricula and degrees.

Systems Engineering is seen as a professional discipline and, as for other professions, has developed professional associations where the practice itself is codified and socialized. Such a professional organization is the International Council on Systems Engineering (INCOSE – www.incose.org). This (traditional) Systems Engineering definition is taken from the INCOSE web site [INCOSE 01]:

"Systems Engineering is an interdisciplinary approach and means to enable the realization of successful systems. It focuses on defining customer needs and required functionality early in the development cycle, documenting requirements, then proceeding with design synthesis and system validation while considering the complete problem:

- Operations
- Performance
- Test
- Manufacturing
- Cost & Schedule
- Training & Support
- Disposal

Systems Engineering integrates all the disciplines and specialty groups into a team effort forming a structured development process that proceeds from concept to production to operation. Systems Engineering considers both the business and the technical needs of all customers with the goal of providing a quality product that meets the user needs."

Figure 1 - INCOSE Systems Engineering Definition

What's not described in this definition is the process, previously outlined, to which all of the activities, areas, and disciplines implied in the INCOSE definition are brought to bear to support.

Fundamentally, the practice of TSE seeks to understand the place of an element within the environment, isolate the element under study from the environment, and then treat the environment as a constant.

Reflect on Systems Engineering. Among the characteristics one would require to have a successful, or at least a low risk outcome, there are a few which are absolutely required to ensure success using traditional Systems Engineering. These serve as boundary conditions for applying TSE:

[7]URL = http://www.google.com. Note the irony of using a complex system to help describe complex systems.

- The specific desired outcome must be known *a priori*, and it must be clear and unambiguous (implied in this is that the edges of the system, and thus responsibility, are clear and known);
- There must be a single, common manager who is able to make decisions about allocating available resources to ensure completion;
- Change is introduced and managed centrally;
- There must be "fungible" resources (that is money, people, time, etc.) which can be applied and reallocated as needed.

Failing to have any of the above raises risk dramatically; and it is unlikely that other mitigation strategies will be possible for the risks introduced. How many of these boundary conditions are found in an enterprise? Our sense is that there seems to be a correlation between small projects which build stand-alone, fairly simple applications/products which are under the complete control and management of a single party, and the likelihood of having these boundary conditions satisfied. Unfortunately, when one considers an *enterprise* every one of the characteristics mentioned above is violated. This isn't too surprising as Systems Engineering has evolved (very successfully) from an industrial, element-manufacturing point of view.

Of note in the INCOSE definition of Systems Engineering is the absence of the concept of the *enterprise*. In fact, the aspects discussed and listed above in the INCOSE definition are appropriate for ***elements*** of the enterprise – and their manufacture- but not the ***enterprise*** itself. Any arguments offered wherein one suggests that one does the same things at a grander scale are wrong; they don't scale; they don't work. Our experience stands in stark contrast to this (sometimes implied) assertion. Notwithstanding this seeming omission, INCOSE identifies the changes needed in engineering education and practice to enable engineering on an enterprise basis. They call out for all engineering curricula to be multi-disciplinary. They hint that Systems Engineering is the place where the currently balkanized set of engineering departments can be brought together, and they suggest a difficult interdisciplinary senior challenge problem and "playground" where the students can learn their trade. We agree wholeheartedly with these observations and recommendations. In a real sense, INCOSE is discovering process and methodological elements which fit Complex Systems Engineering, but has yet to name and describe it.

a.2.3 Can Traditional Systems Engineering be applied to the AOC?

It is clear that processes must be applied where they fit. If boundary conditions for applying a process, or a set of processes, are violated the processes are not really applicable. Is an AOC such a situation?

We can test whether the characteristics we previously argued were required for successful outcomes using TSE fit an AOC:

- *The specific desired outcome must be known* a priori, *and it must be clear and unambiguous (implied in this is that the edges of the system, and thus responsibility, are clear and known);*
    - Test Result: Failed.

    There are expectations expressed in documents known as Block Requirements Documents (BRD) which lay out an AOC's planned functionality over time. The fly-in-the-ointment is that the plan implies a convergent set of developments which would deliver the capabilities found in the AOC BRD. This isn't the case; and it leads to the next characteristic to test.
- *There must be a single, common manager who is able to make decisions about allocating available resources to ensure completion;*
    - Test Result: Failed.

    As observed above, there are many component systems which are managed by many different organizations responding to many constituencies on behalf of a set of users, of which one user community is found at AOCs.
- *Change is introduced and managed centrally*;
    - Test Result: Failed.

    It is certainly the case that senior AF management has (and is) attempting to apply centralized management to bring AOCs under control. AOCs have been declared to "Weapon Systems," The senior acquisition authority has asserted personal control over the official configuration, and detailed configuration control processes and measures have been imposed.

    To date, these measures have not worked; and we suspect they still won't. The only proximal result is a sense of stasis hovering over the AOC formal definitions. This invites the formation of black markets. Each Combatant Commander has funds that can be spent on their AOCs (the OIF AOC was built on Commander's Initiative funds). Bottom line: they have the means; and when a need surfaces, they can fix their own problems. And they are independent of the corporate AF staff.

    Besides, stasis of the AOC definition imposes no stasis on the component systems used to build the AOC, so what does a firm baseline mean in this case?
- *There must be "fungible" resources (that is money, people, time, etc.) which can be applied and reallocated as needed.*
    - Test Result: Failed.

    As mentioned, few of the total set of resources required to produce an AOC are controlled by, or in a way, that renders them fungible.

The conclusion is pretty straight forward. TSE doesn't lend itself to engineering or managing the engineering of AOCs (and by extension, enterprises). Can one take organizational or management steps to bring the characteristics which are outside of the boundary conditions back in line? Perhaps one; but they are all violated. A reasonable guess is that it is not likely that there are management or organizational changes which would allow TSE to be applied successfully.

a.3 Complexity and Complex Systems

To facilitate discussion, terms must be defined. For this discussion there are a few key terms which require definition. While somewhat pedantic, the concepts offered below attempt to define the landscape explored. For those who believe they have a good grasp on the definitions and characteristics of complexity and complex systems, or who have an immediate interest in answering the next logical question with respect to an AOC – is the AOC a complex system? – they might jump ahead to section a.3.4, then return to this point.

"Complex Systems Engineering" contains three terms:

- Complexity
- Systems
- Engineering

Fundamentally, we're talking about an engineering activity centered on complex systems. Yet, to this point the appeal is to a general gut-feel for the concept. But, what is complexity? What are systems? What are complex systems? What is systems engineering? What is complex systems engineering?

a.3.1 Complexity

"Complexity" as a concept is actually rather slippery. For understanding the difficulties "complex systems" present to engineering and management activities, it's worthwhile taking a few moments and exploring this term "complexity." As we discuss complexity, we will use a "progressive formalism" approach which initially appeals to intuition, then fills in the intuition with some formal structures. To set complexity in its proper place, we will also use some forward references – i.e. before we define a system, we will use the common notion of a system to help understand "complexity."

"Complexity" does not mean "difficult to understand" (although it might be the case that something complex is difficult to understand). Reaching into the American Heritage Dictionary,

> **Complex** adj.
> **1.a.** Consisting of interconnected or interwoven parts; composite. **b.** Composed of two or more units

This particular definition is not too useful, since every system (using the working definition of ***system*** below) is "complex" by this definition. The Oxford Dictionary states that something is complex if it is "made of closely connected parts." This definition also does not distinguish between "simple systems" and "complex system." In fact, one could (by simple substitution) quickly create a "simple complex" which seems like an oxymoron.

Bar-Yam [97] suggests that complexity is strongly related to the number of possible states of a collection, or its complete description. That takes the concept closer to a useful understanding for engineering, and borrows in an attractive way from Shannon's Information Theory; but it also seems arbitrary in some ways, as it suggests that a collection becomes more complex when measured with more precision. For example, if one calculates all the possible arrangements of papers on one's desk, the number of discernible possibilities is different depending on the precision of the ruler used. But it's still the same desk and the same set of papers. Arguably, the complexity should be the same. The complexity should not depend on the measuring method. The counter argument is that the use of a different ruler is precisely equivalent to using a different scale; and so finding that the complexity at different scales is different should not be surprising. Nevertheless, the use of a value related to both information theory[8] and entropy[9] remains attractive.

Another aspect to contemplate is the difference between actual number of possibilities and the number of useful possibilities. Consider a spoken language. Is the complexity of a sentence in the language of length 'n' related to the permutations of the number of words in the sentence (O(n!))? Or, is it related to the number of 'useful' arrangements of the 'n' words, which would be significantly less? This is potentially important as one develops metrics to measure complexity and compare complexity levels.

Another view of complexity is Turchin's [Heylighen 95]. He describes complexity in terms of behavior and emergence. He has crafted a theory known as Metasystem Transition Theory which describes interactions within and among *models of meaning*. The creation of each new level of abstraction and complexity he terms a "quantum of evolution."

Whatever model is used to understand complexity, rendering 'complexity' into a useful engineering concept requires metrics. Return to the statement *"Complexity" does not mean "difficult to understand"* above. Since it is easy to assume that the concepts are synonymous, a concept for "difficult to understand" but is not the concept "complexity" must be found. A candidate term offered is *Intricacy*

> **Intricacy**.
> **1.** Having many complexly arranged elements; elaborate. **2.** Solvable or comprehensible only with painstaking effort.

An example may help drive the point home. There used to be a board game called *Mousetrap* played by children (it may still be played!). In the game, players move their playing pieces (colored mice) around a board and in doing so build a Rube-Goldberg mousetrap which one player ends up using to capture the other player's mouse, thus winning the game.

The advertising copy reads as follows[10]:

[8] Shannon's 10th Theorem:
[9] Second Law of Thermodynamics
[10] taken from http://www.areyougame.com

> *"Construct a crazy mice-catchin' contraption piece by piece as you race your mice around the track! Once it's built, turn the crank...that kicks the marble...that rolls down the chute...and sets off a zany chain reaction that just might trap a pesky mouse!"*

It's clear that the bizarre mouse-catching device is *intricate*. However, it is not *complex*. It has only one possible configuration, and it results in only one behavior. Each piece is carefully crafted to fit onto the previous structure which sets up the conditions for the subsequent structure. It also doesn't interact at all with its environment. It assembles the same way each time (in fact it must have this characteristic to be a good toy). It is also clear that the mouse-catching system built is *"solvable or comprehensible only with painstaking effort."* That's the appeal of the game. That's why kids enjoy it. That's not a characteristic necessarily appropriate for our military systems.

Measures of complexity and intricacy may serve as good metrics to understand the relative merits of a system, and may be useful for relative comparisons. Mathematical properties of *complexity* and *intricacy* can be shown to relate to specific mathematical characteristics which we will treat in a subsequent publication. As a precursor to a detailed treatment, it appears that intricacy relates to the number of axes of characterization – i.e. the absolute volume of a hyperspace defined by the axes. Complexity relates to the volume reachable within this hyperspace. Thinking about the mousetrap device, it is a device whose hyperspace has many axes; yet it has a narrow extent along each axis, forming a narrow volume of reachability within this hyperspace. Other models attempt to describe complexity in terms of *variety* and *constraints* [Heylighen 01], which roughly map to Shannon's notion of statistical information entropy and content. Still other formulations of complexity are found in other disciplines.

Christopher Alexander [79] offers well-known (architectural) pattern models for considering complexity and emergence in architecture. Emerging from the repeated application of the principles, Alexander's speaks about spaces, homes, towns and cities which are "alive." His concept of "alive" is a reflection of the interactions among the components in the environment and the people, and the support the environment affords to the repeated patterns and events which make up the peoples' experiences minute-to-minute and day-to-day. He recognizes that there are both patterns formed at higher levels from bottoms-up application of patterns, and there are explicit patterns applied at higher levels – and in this he hints at multiscale analysis. Fundamentally, he is talking about the relations among the entities which interact; and the result of those relations. His work was adopted and interpreted by those who practice "pattern"-oriented approaches to systems and software [Gamma 94]. These approaches certainly seem to have something to say about complexity, and Complex Systems Engineering.

Alexander's notion of complexity seems to align with the notion of "*order*." The utility of a definition (of order now) depends on its alignment with the informal understanding accorded the term. In the informal sense, *order* is almost always associated with organization as well as with the actions or other forms of direction that lead to this organization (rhyme and reason). Order is not simply a passive thing like color (i.e. a state property). It is dynamic; it is associated with doing something – i.e. both form and function. By focusing on the *relationships among things*, not just the state of the things as a result of the relationships, we can understand the reasons for the molar organization and perhaps understand the implications to change – even infer or deduce state elsewhere which may be out of view.

Formally now, the order (of a system) is a measure. The measure is the set of all of the specific and instant relationships among the parts of a system.[11] In many circumstances, the order (of a system) can be quantized and summarized by the cardinality of such a measure-set.[12] This approach, that of focusing on the relations, not merely the state, will likely provide the most useful characterization of complexity since it characterizes things in an active way.

a.3.2 Systems

Here, the term ***system*** denotes a set of parts that have relationships with one another. This is also the preferred definition for this term. From the American Heritage Dictionary, first definition:

> **1.** A group of interacting, interrelated, or interdependent elements forming a complex whole

Not everyone uses this definition, but it is the definition we use. The major stumbling block some have with this definition is that it requires that the parts have some sort of relationship to one another to constitute a system. Some speak of "systems" using a much looser definition. The American Heritage Dictionary supports a looser definition as well; to wit, definition six:

> **6.** A set of objects or phenomena grouped together for classification or analysis.

The issue to consider is which definition fits one's intuitive notion of "system" better. Number one seems to meet the intuition test: it's alive and active, while definition six is a system of academic or conversational convenience. The definition used here insists that a system has multiple parts, AND that those parts have relationships among them.

11 A given relationship can vary over time. The "specific and instant" form of a relationship is to be distinguished from these possibilities. A fuller discussion of the meaning of order would elaborate the definition of "relationship" offered here. It would offer that relationships are patterns in attributes, where attributes define the parts of a system (and sets of "values" define attributes). A relationship allows the inference or deduction of the specific values of an attribute of a part of a system based on other attribute values because those attribute values collectively form patterns

[12] Random is often used to identify the absence of order. If so, random should not be treated as an exact synonym for stochastic – which actually asserts that there are relationships, but that they are not well enough understood to dependably deduce or infer knowledge of specific parts. In fact, stochastic is a telltale for relationships at multiple levels of scale, something that is taken up briefly below.

INCOSE defines a System [INCOSE 01] in the following way: "...A system can be broadly defined as an integrated set of elements that accomplish a defined objective..." They have also been struggling with scaling up, and the implications. They also note "…It is sometimes confusing as to which elements comprise a system…" offering an example of a broad network with independent databases fused, and a desire to print the results.[ibid] Also noted is the presence of multiple "levels" to a system, and the different roles the same "things" are at different levels[13]. Clearly, the limits of the traditional definition of a *system* are being felt, and it too hints at Multi-scale analyses (discussed later).

a.3.3 Complex Systems

A Complex System [Bar-Yam 97; Heylighen 95; Holland 95; Kauffman 93] is a system:

- Whose structure and behavior is not deducible, nor may it be inferred, from the structure and behavior of its component parts;
- Whose elements can change in response to imposed "pressures" from neighboring elements (note the reciprocal and transitive implications of this);
- Which has a large number of useful potential arrangements of its elements;
- That continually increases its own complexity given a steady influx of energy (raw resources);
- Characterized by the presence of independent change agents.

A measure of a complex system (for characterization and comparison) might be based on the balance of *complexity* and *intricacy*. Other corollaries of these measurements might be a measurement of the rate at which the complex system's adapts to required/desired change.

a.3.4 Is the AOC a Complex System?

We can test whether AOCS fit the definition of Complex Systems by comparing the two:
A complex system is a system:

- *Whose structure and behavior is not deducible, nor may it be inferred, from the structure and behavior of its component parts;*
    - Result: Marginal Pass

    The AOC's desired molar behavior is reasonably well known; even it's desired changes, so this characteristic doesn't necessarily fit. However, if we take a broader view of an AOC as an element of C2 (i.e. the enterprise), then this statement becomes more correct.
- *Whose elements can change in response to imposed "pressures" from neighboring elements (note the reciprocal and transitive implications of this);*
    - Result: Pass

[13] "Aircraft, automobiles, and homes are other examples of systems at one level, which can be considered elements or subsystems at another level."; Ibid

This is certainly the case in the AOC. Independently-introduced applications (through independent agents) such as (for example) ADOCS and Falcon-View cause direct "pressure" on those applications which perform similar roles, or which could potentially act in concert with these introduced applications. As an example of resolving the introduced pressures, TBMCS specifically added certain Information Services to interact with ADOCS without the need to own and control it.

- *Which has a large number of useful potential arrangements of its elements;*
  - Result: Pass

  Since the AOC's workflows are numerous, and are in flux due to new missions and doctrine, this fits. IT is aso true, though, that it is the people who supply the flexibility; and they often fight the automation present.
- *That continually increases its own complexity given a steady influx of energy (raw resources);*
  - Result: Pass

  This also seems to be the case. For example, TBMCS re-architected from a monolith to a set of applications riding on a set of Information Services precisely to increase the number of possible connections and relations, and to allow more independence of creation and use of new clients of the services offered.
- *Characterized by the presence of independent change agents.*
  - Result: Pass

  The AOC has upwards of 30 independent agents – in the form of separate Program Elements (PEs). PEs are, by their definition and nature, independent agents.

It seems reasonable to conclude that AOCs are Complex Systems; and, since there is a need to apply a Systems Engineering approach to the AOC which is beyond the traditional Systems Engineering approach (see the earlier discussion in section a.2.3), the AOC might benefit from a Complex Systems Engineering approach which acknowledges the differences between the AOC and other more-traditional developments to which TSE can be applied.

a.4. Engineering Complex Systems

If TSE doesn't scale to AOCs or the enterprise, what does? In the introduction we made the claim that an augmentation of Systems Engineering, to be called Complex Systems Engineering (CSE), should be used to manage and guide the enterprise. As discussed above in some detail, a violation of the boundary conditions required for a favorable application of TSE suggests different tools and approaches are required. In essence, CSE must serve to bring together independent, disparate organizations and entities. It must provide them with a sense of "pressure" that they feel, and a set of processes that can be used to resolve the pressures. CSE must incentivize the partnerships needed; and must compel the engagement of their respective resources to accomplish the integration without resorting to arguments over whose money is being spent, or whether "interoperability" or "integration" is a "requirement" they have.

It's clear that Systems Engineering must extend its philosophic and theoretic foundation to build a consistent (and hopefully complete) framework for the practice of Systems Engineering in general, and Complex Systems Engineering in particular. Within this framework, we describe new roles and responsibilities for new "jobs" which must be performed to do CSE.

Complex Systems Engineering changes the focus from *"…here is the solution designed from the requirements, now go implement it…"* to *"…here are the selective pressures acting on the elements present (likely built using TSE), now resolve or reduce them…"*

CSE does this, and this is the key point, through a deliberate and accelerated mimicry of the processes that drive emergence and natural evolution. Kaufmann [93] noted that complex adaptable systems require both the emergence of novelty and variety as well as selective pressures to account for the richness in ecosystems. We find those characteristics in an AOC. In fact, the AOC (and Command and Control in general) can be thought of as an *Ecosystem*. Bar-Yam [03] has explicitly incorporated this thinking in a recent presentation where he introduces what he calls *Enlightened Evolutionary Engineering*. Using the conceptual models suggested above, one can speculate about niches, selective pressures, competition, adaptation, displacement, etc. It describes a process constantly at work and in line with our daily experience; a process which is alive [Holland 92, 95]. While ecology, evolution, and models of ecology and evolution, are beyond the scope of this paper, an appeal to the reader's intuition will help place CSE in context. Those interested in understanding principles of ecology and evolution are directed to the rich literature available (e.g. Stephen Jay Gould, Richard Dawkins, E. O. Wilson, George C. Williams, Ricard Sole, etc.). Future papers will draw the mappings in a more detailed form.

Complex systems engineering is NOT a new or renewed attention to detail; it is an attention to overall coherence. The two are obviously and will always be related. However, as the actual order and complexity of a system increases, it becomes humanly impractical to address both from a single perspective – in particular from the perspective of ever increasing detail. What complex systems engineering does is to address overall coherence WITHOUT a direct and immediate attention to detail.[14]

Complex Systems Engineering acknowledges the presence and action of "autonomous agents" as important elements of a SoS. These autonomous agents are precisely the effectors which must be (and are) eliminated to apply TSE. Again, to apply TSE one needs to eliminate the independent agents, or one needs to augment the set of tools for dealing with their continued presence.

[14] This is what baffles those confined to a linear theoretic and uniscale (or reductionist) viewpoint.

a.4.1 Comparing Traditional and Complex Systems Engineering

Traditional and complex system engineering can be distinguished by contrasting either their methods or the outcomes resulting from the application of those methods. The following briefly contrasts the outcomes that are obtained using traditional and complex system engineering. The term *product* is used to identify the outcome of traditional system engineering; the term *enterprise* is used to identify the outcome of complex system engineering.

TSE	CSE
Products are reproducible	No two enterprises are alike.
Products are realized to meet pre-conceived specifications	Enterprises continually evolve so as to increase their own complexity.
Products have well-defined boundaries	Enterprises have ambiguous boundaries
Unwanted possibilities are removed during the realizations of products	New possibilities are constantly assessed for utility and feasibility in the evolution of an enterprise.
External agents integrate products	Enterprises are self-integrating and re-integrating
Development always ends for each instance of product realization	Enterprise development never ends – enterprises evolve
Product development ends when unwanted possibilities are removed and sources of internal friction (competition for resources, differing interpretations of the same inputs, etc.) are removed	Enterprises depend on both internal cooperation and internal competition to stimulate their evolution

Table 1 Comparing TSE and CSE

Traditional and complex system engineering can (and should) be applied concurrently in the realization and evolution of a complex system. Traditional system engineering is appropriate for managing the decision making processes of individual autonomous agents in a complex system. Complex system engineering must be added when multiple autonomous agents must be a part of any solution and/or when multiscale analysis becomes essential to a sufficiently complete characterization of an evolving problem and its solution.

a.4.2 The Regimen of Complex Systems Engineering

Complex systems engineering (CSE) operates a bit differently than TSE in that TSE is a practice of direct impact and effects, while CSE tends to be indirect. The goal of CSE is to increase the order of, and the complexity available to, systems. As discussed earlier, there is a practical upper limit to the degree to which this can be done successfully through pre-specification followed by implementation, and all of the other attendant processes of TSE15.

[15] Formally, this can be attributed to the increasing dimensionality of the relational multiscale phase-space that can be used to characterize the actual order and complexity of a system relative to the generally fixed and finite intellectual capacity of any human individual.

To engineer a system beyond this limit it is necessary to combine several related activities into a single continuous "regimen" of engineering and development. This regimen is intended to transcend the boundaries of TSE which have been outlined previously. A regimen is distinguished here from a "recipe," and a recipe here can be understood as shorthand for the cumulative nature of the processes of TSE. A recipe is a tightly and precisely scripted sequence of steps intended to yield reproducible outcomes such as specific kinds of cakes or meat loaves (or cars or planes, or operating systems). In the ideal, every outcome is exactly the same. A regimen is a looser formulation of more generalized steps that can be combined in various ways to yield many different instances of generalized outcomes such as weight loss or increased stamina (or an Air Operations Center, or a Department of Homeland Security). Even in the ideal, there can be no insistence on uniformity, only on acceptability or conformity with broad norms.

The overall regimen of CSE creates and manages an environment16 in which multiple autonomous agents each address a fraction of the relationships that might be involved in an overall complex system. Autonomous agents (independent development tracks in the context of most engineered systems, especially IT-intensive systems) and their creations both operate in this environment and (continuously) interact to explore the utility and practicality of new or modified relationships.17 We have some reasonable hope that establishing such an environment is both practical and doable. Holland [95] points to its natural occurrence when the basic elements are present[18].

In terms of today's IT-intensive systems, it is useful to recognize that most of the increases in complexity (or interoperability, or new or expanded relations) are typically associated with the "run-time" of the system, while most of the collaboration (or interoperation or interoperability) that yields these new or modified "run time" relationships occurs among the people who create the "run time" components. This collaboration is said to occur during the "development time" for the system. As a consequence, complex systems engineering for IT-intensive systems needs to establish and manage an environment for "developers" AND for their "run time" creations. These two aspects (scales) of any IT-intensive complex system are not ultimately independent, but they can be discussed separately and then combined to enhance overall understanding.19

[16] There are many analogs to the developmental environment of CSE. See for example Hayek's general thoughts on monetary and trade-cycle theory. An even more familiar analog is the role of "playgrounds" in child development.

[17] As well as to maintain, to modify, or to discard existing relationships.

[18] Holland [95] talks of the basic elements of: aggregation, tagging, nonlinearity, flows, diversity, internal models, and building blocks.

[19] Continued improvement in the engineering of IT-intensive systems will witness the continued convergence of these "separate" development and run times. This can be colloquially summarized as the emergence of self-programming systems. The compelling "evolutionary pressure" for this emergence is the "expense" of human labor in maintaining and expanding IT-intensive systems. Programmers are expensive relative to their programmed creations.

a.4.2 The Elements of the Regimen

The following very briefly introduces each of the elements that are combined into the Regimen of CSE. Their combination is discussed after the elements are introduced.

a.4.2.1 Developmental Environment

An explicit and conscious attention to a developmental environment (including even a pre-specification of its initial form) is the single most important activity underpinning the deliberate development of complex systems.

This developmental environment can be understood as either a separate and distinct environment in which complex systems develop and operate, or – along with that environment – an overall ecosystem that includes both. As such, this first activity focuses on the completeness of the ecosystem relative to supporting the more focused activities that occur within it. Are a sufficient number of the relevant autonomous agents and their creations present? Can new ones be added? Is the means available for these autonomous agents to interact if they so choose? Are resources flowing through the ecosystem? Are the means for supporting both cooperation and competition among the autonomous agents present? Is the flow of resources modulated by cooperation and competition, or is that flow entirely pre-specified? [20] Are there universal signals that can be interpreted locally (and perhaps differently) that are associated with the whole that cannot be entirely explained by any combination of a subset of the parts?[21]

The developmental environment can not be a one-time thing. It must be, nurtured, and managed so it can evolve itself; even after its initial establishment. Attention to this environment must be continuous, deliberate, and it must be available to all the independent agents; this is why it lends itself to being treated as a separate activity.

a.4.2.2 Outcome spaces

Outcome spaces are identified (or defined) at multiple levels of scale, and from multiple points of view, for a complex system. An outcome space is explicitly distinguished from the many specific outcomes that comprise it. (When very specific outcomes are sought and/or are meant to be exactly reproducible, TSE should be used to achieve them. However, applying TSE becomes increasingly difficult as the number of autonomous agents increases; ideally, only one is involved. All specific outcomes in the outcome space must be viewed as acceptable without there being strong preferences for any of them.[22]

[20] If the flow of resources is entirely pre-specified, then competition cannot operate. The localized decisions of autonomous agents cannot, by definition, influence the flow of resources – although they can still influence the effectiveness of that flow. As a result, complex system development can't occur.

[21] Examples of such signals are the pricing mechanism in a market economy or selective pressures in a biological ecosystem.

[22] This does not mean that outcomes can't be identified as unwanted. Partitioning outcome spaces into wanted and unwanted sub-spaces is one way to do this. This is why outcome spaces are sometimes referred to as the targeted outcome spaces.

When specific outcomes in an outcome space can be realized by individual autonomous agents (or their creations) by themselves, competition is encouraged. When specific outcomes in an outcome space can only be achieved by autonomous agents (or their creations) collectively but not individually, cooperation is encouraged.[23] Network centric operations and Jointness are military domain examples of the latter [Alberts 99]. Such outcomes are best characterized at a scale in which the autonomous agents (or their creations) are not immediately or directly accessible. Complex systems require both competition and cooperation for sustained development, although competition is almost always more important in the short term.

It is sometimes possible to characterize outcome spaces at multiple levels of scale using identical terminology. This often causes confusion and should be avoided. For example, it may be desired that a complex system achieve a reduced footprint (or reduced power consumption, etc.) and/or it may be desired that individual components of that complex system achieve reduced footprint. These are frequently outcomes at different levels of scale. (It is possible that a complex system could achieve a reduced footprint even though individual components do not – or even increase their footprint.) Such outcome spaces should always be explicitly distinguished (for example, by always and explicitly referring to "component" footprints and the complex system's footprint).

The identification of outcome spaces (vice specific, detailed outcomes) focuses attention on explicitly recognizing sub-spaces and partial volumes in a relational phase-space (vice specific phase points or trajectories).

a.4.2.3 Rewards

Autonomous agents (independent development tracks in the case of IT-intensive systems) make the decisions that determine the utility and/or the practicality of existing and new relationships within the complex system. Rewards are structured to motivate the autonomous agents to make decisions that cause the complex system to enter the targeted outcome spaces desired.

Rewards shape the decision making processes employed by autonomous agents. The rewards should be clear and should not be dependent on specific processes of the autonomous agents who are subject to rewards. Since one should not assume that these autonomous agent processes are uniform (or that they should be), rewarding based on a specific process which could be viewed as too invasive by the autonomous agents[24]. Rewards are only one consideration shaping the decisions of the autonomous agents; and should be viewed as incentives only.

[23] There are, of course, many intermediate or blended situations as well in which both cooperation and competition play a part.

[24] Unless the specific outcome space IS a common process. However, insisting on common processes may well stifle innovation and variety needed for evolution.

In the most general case, rewards are access to the energy flowing through a complex system. In the case of IT-intensive system development and within the context of contemporary acquisition protocols, rewards are almost always associated with access to the money flowing through the entire system development environment. There are other forms of rewards (as well as penalties), however, that are neglected only at great peril to the system engineer. Rewards motivate. As long as people are involved, the list of possible rewards is as long as the list of factors that motivate people.

To the degree that rewards can be distributed extra-contractually, that offers motivation to autonomous agents to "keep their eye on" the complex system, even if they are not engaged in a direct manner. Innovations which these agents can bring to the complex system which are shown to bring value should be rewarded. This sets up the potential of new approaches and influences, and avoids stagnation.

a.4.2.4 Developmental precepts

Developmental precepts constitute the "rules of the game," and by doing so stimulate contextual discovery and interaction among autonomous agents. They do this by establishing (for example) certain constraints on how outcomes are achieved by autonomous agents, or how they interact. It's easy to confuse these precepts with rewards, however they are quite different. Like when playing a board game, even if the next space to promises a great, the rule (developmental precept) says you must move the number of spaces found on the die you throw, you can't move that one space if the number is not "1."

Developmental precepts do not specify specific outcomes or even outcome spaces. In contemporary IT-intensive systems in which development time and run time are treated separately, developmental precepts focus on the interaction of the independent development tracks more so than on such behavior among their creations. In contemporary IT-intensive system acquisition and development, these developmental precepts can be contractually fixed since they can be made as specific as desired.

Developmental precepts are initially difficult for most system engineers to appreciate since they shape autonomous decision making leading to specific outcomes rather than the specific outcomes themselves. An example is useful.

In the case of the Air Force, most IT-intensive systems that contribute to Command and Control (C2) are developed under the supervision of (acquisition) Programs. Many of these Programs are physically housed at the Electronic Systems Center (ESC) on Hanscom AFB, Massachusetts. Each of these C2 systems has explicitly designated military "end users."(For example, an automated mission planning system helps a pilot to plan a route to be flown by an aircraft on a military mission. Pilots are the end users of such an automated system. Pilots do not take delivery of such a system, however. Instead, a Commander responsible for many pilots is the notional end user that takes receipt of such a system at a particular Air Force base on *behalf of the Air Force and of the pilots stationed there.) The completion of the acquisition process (the delivery and acceptance of the system) is signified by the signing of a DD-250 form by the end user. The Program uses this signed form to confirm the successful completion of its own (acquisition) mission. Absent that signed form, the mission remains incomplete. Careers depend on the completion of missions, and this is true for the people in Programs as well.*

End users (the Commander in this example) take receipt of multiple systems from ESC – often in the same year. The mission planning system is just one such system. These systems increasingly must interact with one another to fully accomplish their respective purposes. Because such interactions do not fully fall within the scope of any one system, however, the successful realization of such interactions is left to delivery time and is often partially or wholly left to the end users to accomplish.

This is further aggravated by the periodic replacement of an earlier version of a system with a newer (and better) version. This almost always involves moving extensive data bases from the earlier to the newer versions of a system to maintain operations. This is frequently left to the end users to accomplish even though improvements in the newer version of a system often involve the reorganization of the existing information in those data bases.

This is even further aggravated since many improvements to systems impose new or additional burdens on the infrastructures supporting modern C2 operations (power, bandwidth, connectivity, etc.). New or improved systems are not responsible for augmenting such infrastructures since by definition infrastructures are shared. The end users are however constrained by these infrastructures.

As a result, end users increasingly complain of "drive by" deliveries of systems by ESC Programs and the failure of ESC to deliver "integrated" solutions to their operational needs. Although ESC increasingly talks about an "integrated C2 enterprise," it continues to deliver that enterprise in "kit" form – leaving the hard "integration" part to the non-acquisition community to accomplish.

The traditional response to this complaint (thereby acknowledging its essential truth) has been to attempt to formulate master schedules for the delivery of systems to specific locations and the formulation of detailed integration plans to interconnect the independent systems prior to deliveries in the traditional system engineering fashion. This response has failed, been retried and failed again. The specific reasons explaining each failure is now legion.

Treatment of such overlapping and interdependent deliveries as a complex system development would involve (in part) the formulation of a developmental precept. This precept would alter slightly the mechanism already employed to complete the delivery and acceptance process. It would modify the DD-250 form so that an end user could only take receipt of multiple (say two in the simplest case) systems from ESC at one *time. The end user would then have the leverage to compel the acquisition community to address the interconnection of delivered systems. ESC Programs would respond accordingly since a system could no longer be completed by itself. The specifics of which systems to be delivered and how they need to be interconnected, etc. would be left to the Programs (the autonomous agents) to resolve in their own best interests. However, the" global" outcome of more integrated systems from ESC would also be accomplished – even though the specifics of how and when were never explicitly formulated in advance at any "global" level.*

The specification and then the enforcement of such a developmental precept would serve to stimulate discovery and interaction among ESC Programs without specifying what the specific outcomes should be. This is the essential characteristic of a developmental precept.

a.4.2.5 Judging

Judging requires human judgment. Judging associates specific outcomes *achieved* with autonomous agents, and assigns rewards to the autonomous agents accordingly. Rewards are established *prior to* the realization of desired outcomes. Judging, on the other hand, is based on actual outcomes achieved, not before.[25]

Judging for rewards that are associated with outcomes that can be attained directly by autonomous agents (or their creations) is straightforward. Specific outcomes in the targeted outcome spaces are seen to actually occur. They are recognized as such and then the reward is assigned (given) to the autonomous agent(s) responsible.; as quickly as possible following the judgment

Judging for rewards that are associated with specific outcomes that are in outcome spaces that can only be associated collectively with autonomous agents (or their creations) is more demanding. (For example, if a complex system achieves a reduced footprint by virtue of certain components *increasing* their footprint and so allowing others not to be used, or others to decrease their own component footprints so that the net effect is an overall reduction, then judging requires the identification of the autonomous agents responsible and the apportionment of the reward. The actual achievement of network centric operations would be another example.)

[25] Judging removes an important risk attendant to contemporary acquisition protocols. Contemporary contracts are awarded based on what a successful bidder is predicted (in a source selection) to do in the future. Judging assigns rewards based solely on what actually happens, not on what will happen. This has implications for the nature (size, etc.) of rewards since the independent development tracks must assume greater responsibility for the risks attendant in actually achieving desired outcomes. A discussion of such implications is well beyond the scope of this discussion; but it suggests a wider variety of supported business relations should be explored.

a.4.2.6 Continuous Characterization

Outcome spaces, and rewards can (and should) initially be characterized with succinct (even pithy) "bumper sticker" labels.[26] This allows for a maximum role for the autonomous agents in shaping the evolution of the complex system. Because autonomous agents do exactly that -- act autonomously – this maximizes the opportunities for inconsistencies in how these characterizations are interpreted. To the extent that consistency matters, outcome spaces, rewards *and the current condition of a complex system* will benefit from continuous and progressively more detailed and complete characterizations. The characterization of the current situation is crucial in this regard because it permits autonomous agents to independently develop metrics to guide their local decision making in ways that will be broadly similar over time. (Autonomous agents, although autonomous, are never wholly dissimilar.) In this regard, the specific outcomes (as distinct from outcome spaces) used as the basis for judgments should be detailed, as should the rationale supporting those judgments.

[figure 2]

Consistency can never be guaranteed in complex system development (evolution). (It can be, in theory, with TSE.) As a result, these characterizations can never be made too detailed.[27] Characterization refinement can, however, become less than cost effective.[28] Moreover, consistency in this regard will tend to accelerate complex system evolution but in narrower and narrower directions (those explicitly identified and characterized). See the above figure. Therefore, as new outcome spaces become apparent and/or attractive, unless they are explicitly added to the characterizations (initially with limited detail), it becomes increasingly unlikely that the new

[26] The U.S. Army motivated a tremendous spurt in its evolution with the visionary characterization of a targeted outcome space with the exceptionally pithy expression, "Own the Night."
[27] If, however, the detailing of specific outcomes is substituted for outcome spaces (as just one example), the complex system can be caused to collapse even though the complex system appears at first to rapidly accelerate its evolution.
[28] It can even become counter-productive, obscuring with detail rather than further illuminating the essential coherence desired and achieved.

possibilities will be explored even though available. In the extreme, this can even result in stagnation, something that Safety Regulation is used to preclude or at least impede. See the next section. In any case, deliberately stressing complex system development in this way (very detailed refinement of outcome spaces, etc.) should be carefully weighed.

a.4.2.7 Safety Regulations

Safety regulations are aimed at preserving the "stability" of a complex system. Their purpose or focus, is not on the attainment of desirable outcomes but rather on the continued functioning of the other activities that are intended to do that (defining outcome spaces and developmental precepts, judging, etc.). Their scope is wider than that, however. They are, in short, aimed at preserving the developmental environment. They are indirect measures.

The act of applying safety regulations (or equivalently, safety regulation, and performed by safety regulators) can be thought of as policing a complex system. Although this applies to all levels of scale, in contemporary IT-intensive systems, safety regulation is most important during development time. Safety regulation applies to all developmental activities. Since a list of such activities is open ended, any listing of safety regulations is also open ended.

Adding autonomous agents (independent development tracks) to a complex system development is a development-time activity. Vetting such agents (or even preparing them) prior to entry to the developmental environment is an example of safety regulation.

Adding the creations of independent development tracks to the run-time composite of a complex system is roughly analogous to integration in traditional system engineering, except that its goal and its many implementation particulars (defining and refining interfaces, etc.) are the responsibilities of the involved independent development tracks *and not some external integrating agent*. However, progressive steps to regularize this introduction procedure can be formulated and enforced as safety regulations. For example, such "integration" could first be required to happen "offline," and then "online," and finally "inline." Each of these phases would be detailed and enforced by safety regulators. (In this example, offline, online, and inline represent progressively deeper and fuller participation in the run-time composite of the complex system.) The overall purpose of such phases would be to protect the "uninvolved" independent development tracks and their creations from accidental or deliberate rogue behavior by exposing the new components' (admittedly partial) behaviors to independent observers with this goal in mind.

Safety regulation can also be made to apply to the retirement of no-longer-used run-time components in a complex system (when and how this should be done, etc.). This example is cited because it permits attention to be drawn to a role such "soon-to-be retired" components can play as tools themselves in the safety regulation of a system.

In many natural evolutionary complex systems, generations (populations with slightly different capabilities) overlap. Rather than generational replacement, there is gradual displacement with the possibility that older generations (or, equivalently, exact copies of them) can persist. The IT-intensive system analog of this phenomenon is that "older" components remain "on-line" (and in use) while "newer" components are brought "in-line" and then "on-line" as a surety against catastrophic complex system failure. This is illustrative of managed redundancy as a safety regulation in complex systems.

But safety regulation does not have to be entirely *ad hoc*. Safety regulation is about avoiding "collapse" and "stagnation" in overall complex system behavior. These notions can be given rigorous meaning[29] with the use of chaos and catastrophe theory. This meaning can in turn be translated into the specifics of a given complex system's behavior (in terms of its trajectory in its phase-space). These specifics can, in turn, be translated into thresholds used to monitor and control overall system behavior, but only in very specific dimensions. In crude terms, this technique has been in use for a long time in the form of circuit breakers and the like.

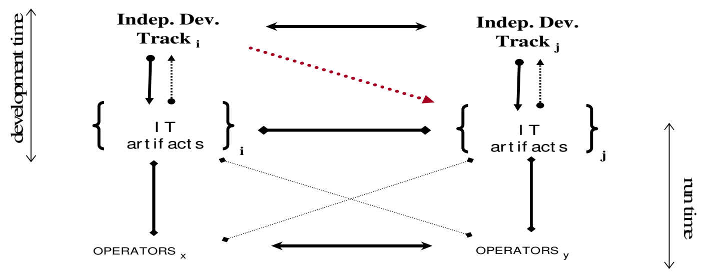
Other safety regulations can and should be directed at the detection of the continued presence of both cooperation and competition since *both* are *always* necessary for the sustained operation of any complex system.

a.4.2.8 Duality

It has already been emphasized that a complex system's "development time" can never be fully separated from its "run time." Complex systems continually change as a natural part of their own operation. This can only be fully appreciated using multiscale analysis.

In the context of IT-intensive systems that also utilize people as operators, it is important to apply multiscale analysis in a fashion suggested in the following figure. Multiscale analysis must be applied to both distinguish between and to understand the relationships among these separate scales in a complex system: the IT-components themselves (application programs, etc.), their developers (groups of people as independent development tracks), and the human operators of the system.

[29] To do so is well beyond the scope of this system engineering focused narrative.



[figure 3]

The IT components in an IT-intensive complex system do not create themselves (yet). Groups of people do that. These autonomous agent developers interact with one another as well as with their creations. There is only very limited (and often strongly inhibited) interaction between such agents and the creations of other such agents. These interactions are frequently identified as occurring during "development time."
The IT components of a complex system also interact with one another. Such interactions can occur during development time, but most of the time such interactions are thought of as an essential element of the "run time" of a system.
However, these are not the only interactions that are associated with the run time of a system. In almost every case there is also a strong degree of interaction between these IT components and their human operators. Moreover, these human operators often interact with one another directly – without any intermediate interactions involving the IT components.

To be fully productive in contemporary IT-intensive systems, the definition of outcome spaces is done (at least) at three distinct scales, two corresponding roughly to "run time" and one corresponding to "development time."

Complex system engineering should take all of interactions into account – not just those involving IT components during run time. This is almost never explicitly acknowledged today – although the growing attention to fostering "user/developer" interaction during development is an implicit recognition of this multiscale reality. Duality is the explicit recognition that development cannot be completely separated from operation in the case of a complex system.

a.4.3 Running the Regimen

There is no single way to characterize how the regimen of complex system engineering unfolds because it is a regimen, not a recipe.

Early on, one should clearly formulate desirable outcome spaces in broad terms (partial volumes and subspaces in the relational phase space for a system). The actual phase points and trajectory of a complex system are determined collectively by the autonomous agents operating within the complex system. In the case of contemporary IT-intensive systems, this is the primary role of the developers – but not exclusively so since operators, for example, can play an important role as well. Recognizing desirable outcomes (actual phase points and trajectories that are in the desired outcome space) when they occur is a primary role of the complex system engineer. In this sense, recognition and continuous characterization augments specification as an engineering activity for complex systems.

Coincident with the identification of outcome spaces must be the publication of rewards available to autonomous agents. These rewards should be expressed in terms that are visible to the autonomous agents – even if the outcomes spaces themselves are not.

Once recognized, desirable outcomes must actually be rewarded. The attainment and recognition of such outcomes does not make this automatic. Human judgment is still required. Complex system engineering can inform this judgment but such judgment must remain the prerogative of those responsible or desirous of the emergent complex system. Once such judgments are made (and rewards and punishments assigned to autonomous agents) the rewards must be restated along with the restatements of desired outcome spaces.

The formulation of desirable outcome spaces should never stop. As new outcomes occur, the desirable outcome spaces need to be restated – along with attendant rewards.

The complex system engineer is responsible for managing the overall developmental (and operational) environment. Key in this regard is formulation of developmental precepts that serve to influence (but not to specify) the decision making of the autonomous agents in the complex system. This requires engineering judgment but judgment that is distinct from the assignment of rewards. These developmental precepts can and should be made "binding" on all developers in the complex system, as should the adherence to safety regulations. Enforcing these precepts and safety regulations are important roles for complex system engineers and are the essential aspects of specifying and managing a developmental environment.

A complex system operates continuously. The complex system engineer is responsible for the overall developmental environment that appropriately mixes operational and developmental contexts. The complex system engineer almost always focuses attention on the developmental aspects of this mixed environment. This includes specifying, operating, maintaining and modifying an infrastructure that supports interactions among autonomous agents and their creations, specifying and enforcing developmental precepts intended to stimulate discovery and interaction among the autonomous agents, and the specification and application of safety regulations.

A complex system operates continuously. In the process of so doing, it is constantly changing and becoming more complex (sic). A complex system engineer continuously characterizes the complex system, emphasizing those aspects that are associated with the order of the system as that order enters targeted outcome spaces. Such system and outcome characterizations become the basis for the complex system engineer's assignment of responsibility for changes in the complex system's order. These assignments become, in turn, the basis for judging – which ultimately assigns rewards to the appropriate autonomous agents. Final judging is always performed by the sponsor or other authority responsible for a complex system.

The complex system engineer assists in the judging process, with the initial formulation of rewards based on targeted outcome spaces, and with their restatements as desirable outcomes are achieved and rewards are assigned.

a.5 Complex Systems Engineering in Practice

Having presented the motivation and the conceptual basis of CSE in the preceding sections, what works? How would one start to apply CSE? The example of the AOC can serve as a template for other analyses, and we welcome and encourage further description of the edges of TSE and the set of techniques appropriate for CSE and its application.

While not explicitly known as CSE, most people have practical understanding of CSE from common experience. Consider how children are raised, or how large organizations exist and evolve. Even a superficial study of these illustrates how CSE can work in practice.

Consider how children are raised. Upon their birth, we do not set out a detailed set of requirements and a schedule for achieving detailed milestones (of course, some parents try, and end up being rebuffed). Rather, we set out our principles, and help them learn what we, as parents, value, then apply guidance as they grow and mature. They come to find their own way in the world. This might be thought of as engineering through indirection. In that sense, it is a practice of CSE.

Consider the behavior of organizations. Seldom is it the case that detailed control is applied top-down through an organization continuously. Leadership tends to exercise control indirectly by publicizing what their values are, what traits they value in others, what their goals are for the organization, and how the organization should run (precepts). They set context and the desired outcome spaces. Periodically subordinates (autonomous agents) are chosen for promotion (rewards) based on what they've done (outcomes) and how well they fit the valued traits. It is those promoted subordinates who determine the day-to-day activities in most organizations.

a.5.1 What are some CS-derived strategies which can be employed within the regimen?

In a real sense, commercial "market places" are complex systems. Commercial practices might be mined for tactics which lend themselves to CSE. Those with particular relevance are those which make technical change easier; those which transmit "selective pressure" easier; those which permit organizations to collaborate; and those which trim the environment selecting "success" and punishing "failure." Table 3 shows a set of strategies harvested from commercial practices which map to aspects of the principles outlined for running a CS regimen. We explore each in more detail below.

	Dev Env	Outcome spaces	Rewards	Dev precepts	Judging	Cont Char	Safety Reg	Duality	Indpnt Agents
½-life Separation	X	X		X			X	X	
Playgrounds	X	X	X	X	X	X	X	X	X
Collaborative Environments	X	X		X		X			X
Partnerships									X
Developers Networks	X	X				X	X	X	X
Branding		X	X	X	X			X	
Co-opetition	X								X
Leveraging others' Investments			X	X		X		X	X
Respect Ricebowls	X	X		X					X
Opportunistic Approach	X	X				X	X	X	X
Advertising and Discovery	X	X	X	X		X		X	X
Value-add business models			X		X	X			X

Experience for test				X	X	X	X	X	

Table 2 - Commercial Practices Embody CSE Principles

As previously discussed, the essence of CSE is the deliberate modeling of the natural processes found in evolution and ecologies. Evolution and ecologies require interaction among entities, and the ability for the entities to change in response to pressures felt from the environment.

To increase the rate of useful change in the enterprise, entities:

- should be in touch with one another for extended periods of time, and;
- the entities' "pulse" time should be reduced as much as possible;
- "value" must be assessed correctly, and by appropriate parties,
- assessed value must impose the selective pressure.

Enabling *ecological* competition and evolution requires that elements can, in practice, rub against one another and allow respective stresses to be resolved as naturally as possible – based on real value. This requires the "systems" which compose the (for example AOC) SoS to build new connections across each other in useful ways.

a.5.1.1 Separation of elements based on anticipated half-life

Architects are very familiar with the concepts of layering systems to separate concerns. This is a central principle, and is a good approach. Yet, the way today's "systems" are tightly-bound into monolithic entities prevents the benefit of this to a large extent. Even within each system, limiting the "pulse" of evolution to the slowest changing element causes evolution to move at the slowest pace, rather than at the natural pace of each of the elements.

Evolution also proceeds at a rate strongly dependent on generation time (spiral time, pulse time, etc.). To increase the rate of evolution, one must shorten the generation time. Therefore, in addition to layering based on functionality, one should separate based on likely rate of change.

a.5.1.2 Playgrounds

How do we come to recognize "goodness," and how is it introduced? Humans, as natural pattern recognizers and problem solvers, learn and innovate through experimentation. We see it everyday among children where they constantly innovate and learn through interaction. "Games" take on an additional dimension when we understand they are using play to prepare for life; they are not merely killing time. "Play" is also a key component of many animals' development. Again, while probably "fun" for the participants, it serves a much more important function. The author Orson Scott Card [77] reflected on the importance of a playground in his science fiction classic Ender's Game.[30]

Within DoD, "games" are recognized for the powerful tools that they are. They are a key way leaders and future leaders get to ply their trade. While the word "playground" can not survive into DoD practice due to its pejorative tone, it serves to make the point by connecting to a truly common understanding, and it is appropriate for these discussions.

Where is the "playground" where technology, doctrine, and *Tactics, Techniques, and Procedures* (TTPs) can come together? Today, the places where technology, doctrine, and TTPs come together are large, carefully-scripted events with carefully (centrally) chosen participants and known answers; they are demonstrations rather than even the experiments they purport to be. Experiments would be fine, but experiments can result in a negative finding; and, experiments are done on a playground. These are not playgrounds. Where playgrounds exist (e.g. C2 Battle Lab), they are somewhat disconnected from the process that gets new elements into the field where that which they construct could (potentially) provide a qualitative edge. The Air Force has a number of labs which investigate technologies and operational needs, but there isn't a good connection between them and the formal acquisition process. Where a connection exists it tends to exist because of the heroic efforts of particular persons. The reason for the "potential" rather than "actual" edge is that the new thing has likely been developed assuming all other elements it may impact or influence have themselves remained static; and it remains to be seen whether the potential is realized. Further, the organizational process is for the keepers of the "Systems of Record" to invite an innovator or new-capability provider into the fold. This puts the identification and valuing of innovation into the hands of the organization least likely to welcome it since, by definition, innovation's appearance is disruptive; and the acquisition community is judged, in part, on the smoothness of delivery.

An important point to remember is that among SoS, no element stands alone. Each element exists within the context of those elements around it; and it supplies partial context back to those elements. Thus any change in any element causes a change in context to all elements which juxtapose the changed element. In this way change flows to neighbors, and they respond, which causes further change, which flows to their neighbors, etc. Generally, the effects and pressures brought by any change can't be predicted, and it occurs independently of any schedule or any *a priori* agreements or expectations, and generally without any insight on how its effects will be felt [Breen 00]. For the enterprise, "change" is constant, unplanned, and unpredictable in its complete effect. This is the essence of a Complex System.

Another characteristic of a playground is that "play" tends to be safe. Ideas can be explored without too much risk. There is a (thankfully) natural reluctance to disturb a

[30] Ender's Game actually is an excellent examination of both the importance of a playground where innovation may be introduced, and an approach to "complex systems engineering" in general.

real-world working process (i.e. the operational system); and if one were to let innovation have free reign within the operational system, the innovation is more likely than not to disturb or interfere with it. This clearly runs counter to the desire to introduce innovation. If one is tempted to suggest that the “new, innovative” element can be developed apart and independently, then introduced into the operational SoS, review the previous paragraph.

A playground is an example of a stigmergic environment [Van Parunak 03] which supports innovation. We’re unaware of any explicit description of an innovation process appropriate to our domain, so we thought we’d supply one which can serve as a point of departure for additional discussion.

On the playground, innovation seems to follow four distinct phases:
- Discovery
- Game (compete)
- Codify
- Practice

Discovery. By whatever means it occurs, something new is found and its potential value is envisioned. We CAN NOT predict where the new killer idea will emerge. Our goal is to capture the idea, not predetermine or restrict the places where the ideas we accept may gestate. What triggers innovation? This question deserves (and has had) many books of its own. Kuhn has written on this topic extensively describing the sociology surrounding scientific paradigm shifts. [Kuhn 96] Innovation and discovery receive much attention in the scientific world, including labs setup to study it explicitly[31], and the literature dates back to Bacon, Descartes, Leibniz, etc., and continues today. Our use of the *paradigm shift* concept is not so grandiose. From time to time a connection will be made by someone new to the AOC, or by someone finding a new solution to a problem previously constraining their performance.

Game. The idea/insight is reduced to a form which can be gamed against others which also occupy the same, or close, space. Within an environment where these ideas can be judged against each other, they compete. The new idea might be found to be a qualitative improvement, it might be refined, or it may be rejected. Alternatively, the old idea might be modified and achieve the benefit of “grinding” against the new idea. This must be done in a non-threatening way; and, we can’t only celebrate success[32]. Like a playground, consequences which differentiate must exist, and some must fail, but a failure can not devastate the failed; it must just remove it/them from the game.

[31] See, for example, Smithsonian Institute’s Lemelson Center for the study of Invention and Innovation

[32] In the commercial world, Venture Capitalists recognize that the one “killer app” often comes after a number of promised, but failed, attempts. Their payback is judged on their portfolio, not on each member of the portfolio.

Codify. During the gaming, the idea/insight is come to be understood better; as is the area it supposedly improves upon. With this understanding, the insight is reduced to a repeatable process or technique. This allows others to learn and use the innovation.

Practice. Once an innovation is reduced to practice and codified into a TTP, it can be taught and practiced.

To the degree there exists a place to play and innovate, the environment will support evolution at a rate faster than would occur otherwise. If there existed a process for guiding and managing the evolution, then the enterprise can move forward based on demonstrated value rather than future promises of value. The development of capabilities in this manner – through discovery - doesn't require the level of detail and *a priori* planning that a pure engineering approach requires.

a.5.1.3 Collaborative environments

The ability to work with others is not an altruistic need. It is purely self-serving if done correctly and effectively. And, it's this self-service that sustains useful collaborations. It may be the case that others may have need for that which I produce. Alternatively, I may be able to use that produced by others, allowing me to concentrate on what I do best (and how I add value) rather than expending resources on incidental aspects which don't discriminate my offering from others.

Additionally, to the degree that what I produce must fit into a bigger whole, if I'm able to easily collaborate with others in that bigger whole, my risk of integration is reduced.

a.5.1.4 Partnerships

The ability to form and sustain partnerships helps to increase the utility of that which the partners produce and offer. Development risk is spread, and understanding of true need is improved. Successful partnerships seek to reduce overlap, and come to rely on each other to play necessary roles. Partnerships rely on effective collaborations.

a.5.1.5 Developers networks – creating opportunities for others

As discussed above, to speed up evolution, one must shorten generation time. Among systems, and with regard to systems engineering, this suggests shortening the feedback cycle, and lowering the amount of code which must be written, by allowing interaction among developers; thereby allowing reuse of useful code and connecting better to the run-time context. This is best embodied in the common understanding found in the software marketplace where developers' networks are a common ploy to getting developers to use a specific platform. A short anecdote illustrates the point.

In 2000 at a meeting of industry with the Air Force acquisition leadership, Paul Maritz[33] was asked his opinion about how it is that Microsoft has achieved such an apparently unassailable presence on desktops. His answer came immediately. It was due, he stated, to their commitment to developers[34]. He said Microsoft's strategy was to create opportunities for others; and use the fact of the opportunities created as a force to ensure that small mom-and-pop developer houses would recommend Microsoft products to their clients. Microsoft even extended their development environment (Visual Basic) down into the Microsoft Office suite, about which Maritiz stated that:

> *Microsoft really supplied word processing, spreadsheet, and presentation graphics, etc. functionality, which, while bundled as useful office applications, were also available as functional primitives with which developers could provide customized value-added functionality to their customers.*

He also noted that Microsoft felt it was necessary to lower the knowledge barriers to developing sophisticated applications.

Maritz' recognition that enabling integration and interoperability required many developers loyal to the Microsoft platform; and winning their loyalty required developer tools and environments which were attractive and compelling. The mechanism for exchange was (and is) Microsoft's Developers Network (MSDN).

This fits the CSE template in that Microsoft was **not** building, or attempting to build, all the functionality themselves, nor were they trying to make a killing on developer tools *per se*. Nor were they trying to get a piece of the action for all the functionality developed using their tools. Instead, they sowed the seeds with their tools, and took advantage of the multitude of applications built on top of their Windows^{tm} platform. They rode the "need" identified by all the thousands of developers who were satisfying their own clients. The developer technical needs also were fed-back to the developer tools developers at Microsoft.

This is not a unique story; it was learned and implemented by others also. For example, Sun Microsystems practiced a similar approach as it brought Java along.

a.5.1.6 Branding

Branding is an interesting concept that we may be able to apply to bring pressure to coalesce. It is best told with an imaginary example. Suppose a potential supplier shows a General a new capability which is especially attractive. The General acknowledges the potential value of this new capability and then asks "… have you

[33] Paul Maritz was a Group Vice President of the Platforms Strategy and Developer Group at Microsoft at the time of his retirement in 2000. He held many different positions there, and was one of Bill Gates trusted advisors.
[34] This was in an unguarded personal conversation between author Norman and Maritz at Lt Gen Kenney's (then Commander of the USAF Electronic Systems Center) first "Presidents Forum" meeting which Norman had defined and helped her put on.

got the '*Ready for the AOC*' sticker yet?...". The potential supplier responses "no" and the General then sighs his disappointment, and states that, had the potential supplier qualified for the *brand*, the capability offered could be moved into consideration for the AOC immediately. Without it, the capability must be subjected to a long process of evaluation and likely rework to ensure it will be able to integrate; and then is integrated. The General, as a proponent of a capability not carrying the '*Ready for the AOC*' brand, would need to advocate for funds to integrate and sustain the capability for its anticipated lifetime.

Brands can be powerful; and they can influence indirectly.

a.5.1.7 "Co-opetition"[35]

This term was coined by Adam Brandenburger of the Harvard Business School and Barry Nalebuff of the Yale School of Management. It described their observation that connectivity among businesses and people require a new way to thin about the business environment. Below is an explanation taken from the preface of their book of the same name:

> *Co-opetition offers a theory of value. It's a book about creating value and capturing value. There's a fundamental duality here: whereas creating value is an inherently cooperative process, capturing value is inherently competitive. To create value, people can't act in isolation. They have to recognize their interdependence. To create value, a business needs to align itself with customers, suppliers, employees, and many others. That's the way to develop new markets and expand existing ones.*
>
> *But along with creating a pie, there's the issue of dividing it up. This is competition. Just as businesses compete with one another for market share, customers and suppliers are also looking out for their slice of the pie.*

Cooperation and competition is well studied. Axelrod [84] wrote about the evolution of cooperation, and showed how it can provide joint benefits. Poundstone [92] also describes mutually-beneficial approaches among autonomous agents.

Co-opetition allows independent parties to cooperate on those elements and aspects which transcend their individual ability to control, while preserving their ability to compete on demonstrated value in their space.

For C2, various persons from traditional DoD contractors, commercial entities, and DoD officials have envisioned a marketplace where firms may specialize and come to dominate a niche. They maintain their dominance in their niche due to their continuing delivery of valuable and valued goods and services, rather than through contractual dictates.

[35] Co-opetition http://mayet.som.yale.edu/coopetition/index2.html (Adam Brandenburger of the Harvard Business School and Barry Nalebuff of the Yale School of Management)

As currently structured, this is difficult to achieve as there doesn't exist an environment for co-opetition. Consortia are often used as a vehicle for broad cooperation, and these options should be examined for achieving a co-opetive environment.

a.5.1.8 Leveraging other investments

Partnerships, collaborations, and other instances of cooperation all attempt to use the investments others have made for one's own benefit. An example (besides the obvious ones of using Commercial Off-the-Shelf technologies) is found in the Family of Interoperable Operational Pictures (FIOP) program. The FIOP program was the first to fund TBMCS's desire to build Information Services – which represented a new way for them to build and deploy functionality. FIOP used a relatively small amount of money to leverage the larger development budget TBMCS had. Essentially, FIOP paid for good behavior on TBMCS's part. Both benefited. And, others (who invested nothing) were able to use the Information Services built by TBMCS.

a.5.1.9 Technical approaches which respect "ricebowls"

There is no doubt that people come to place great value in that which they are personally involved in and responsible for. These apparently parochial interests are often described as "rice bowls." A great source of resistance to cooperation among independent agents is the thought or impression that others will impose themselves on the independent agents in ways and manners which they view are inappropriate. After all, each organization has conducted itself according to its needs, and has made decisions according to its assessment of how best to meet the needs. Additionally, there is often fear that one's activity will be subsumed under another's, and one's contributions will be devalued and possibly ignored.

It's clear that cooperation has great value; and those aspects which interfere with cooperation cause "innovation drag." Technical approaches which tend to respect "ricebowls" remove some of the hesitations for forming cooperative partnerships.
Examples of technical approaches which respect ricebowls include current developments in Web Services. This technology exposes functionality with the minimum requirements for homogeneity. It presents a "virtual homogeneity" within a heterogeneous world. In this way it offers the potentiality for independent agents to offer their services to others; and thereby permits new associations and relations to be exploited – supporting innovation. It also supports the rise of technical structures and approaches which permit the agility needed. Assembly moves out towards the end-users further blurring the difference between development and runtime.

a.5.1.10 Opportunistic approach

An aspect of the usual way in which we do business is to restrict ourselves to a complete capability before fielding. As mentioned earlier this restricts fielding to apparent “complete” sets at a fairly slow rate. If one treated logical sets of users as a unit, and involved them in managing the identification and introduction of functionality and change, then one might be able to be more responsive.

a.5.1.11 Advertising and Discovery

As currently structured, finding useful capabilities and functionality offered by third-parties is not trivial. Potentially useful functionality is not advertised, and there is no well-known place to go looking. Both for the development and the operational environments, achieving transparency for effective advertising and discovery is critical.

A key enabler for evolving and integrating the enterprise is to create opportunities for *small world* phenomena [Barabasi 02; Watts 03] to emerge. The power of loose connections is clear and convincing. New relations possible are likely discovered in areas not previously explored. This will emerge with in both shared information spaces and with shared behavior. Advertising and discovery technologies are key to enable these results.

a.5.1.12 Permitting “value-add” business models

A continuing complaint from users in the field is that they don’t get “a vote” in what is built for them. This is primarily due to the business models employed in acquisition today. As mentioned earlier, the dominant business model used is employer/contractor. In this model, the employer produces a requirements document, and then various potential contractors propose how they will produce the functionality desired. The market place is contract engineering; the selling and buying of engineering hours (perhaps laced with certain processes which can be argued reduce risk). Those who have successful proposals are those who can tell the story of how they are going to produce a requirements-compliant product for the least risk. It is a promise well told, not a demonstration of specific achievement. Success is measured based on compliance with the requirements, and the maintenance of the cost and schedule negotiated. Success is not directly related to the usefulness of that which is produces. The Government’s money is spent for the engineering hours used for development. This is the basis for the complaints.

Assume a by-use payment model. Assume further that there is no *a priori* assumption of the undesirability of redundant functionality[36]. Under such a model money flows to those who produce demonstrated utility to the user. The market now shifts to understanding and satisfying real needs rather than the sale of engineering hours. The acquisition organization’s role, under this model, shifts to verifying compliance with a set of rules under which functionality is built. Under this model the Government’s money pays for demonstrated value.

[36] Ashby’s Law of Requisite Variety [Ashby 56] can be interpreted as explicitly supporting variety as a technique for attempting to supply sufficient potentiality to allow adequate response to selective pressures.

There are additional aspects to such a model: the emphasis shifts from *cost* to *price*. Suppliers (as opposed to *contractors*) attempt to manage their margins; and they may apply their best and brightest (assuming they can control their intellectual property) and be innovative.

a.5.1.13 Analysis, simulation and collecting experiences replace full-coverage testing

How does one test a complex adaptable system? Rather than relying on traditional approaches (which attempt to come as close to full-coverage testing as possible), we might collect and catalog things when they go wrong in the field; analyzing these for insight into subtle transitive effects. We also need to employ better testing approaches to develop some sense of belief about the systems we field before fielding. Phadke's[37] Robust Testing™ approach may provide tools for picking better and smaller test cases.

Additionally, the infrastructure should be tested to failure so we know the boundary. Then it should be monitored in the field to shed light on if and when we approach these limits. This permits time to intervene prior to, not after, problems emerge.

a.6 Summary & Conclusions

The challenge is moving from "things" to "integrated collections of things" which are governed and managed independently. Although we presented the problem as an issue for AOCs, it is not confined to the AF, or to joint forces, or to DoD, or the US Government, or to the US. The observation that one must use methods which respect the characteristics of the enterprise; and which don't require complete control, or complete knowledge in one place.

Our summary would be incomplete if we didn't consider the insights offered by other professions who have faced similar challenges. Architecture has. Christopher Alexander (of architectural fame) talks about the illusion of control; and he observes that attempts to assert control generally has the opposite affect from what is desired; things tend to get worse, more out of control. He notes that the tendency "…to gain 'total design' control of the environment…makes things still worse…"[Alexander 79, pg 238] He points to the need to construct towns and cities using "patterns" which preserved the correct "nature," and became "alive" in their own right. He calls this the "quality without a name." He's talking about complexity and adaptability.

[37] See Phadke Associates' Robust Testing™ at www.phadkeassociates.com/ser/rt.htm

Traditional Systems Engineering has always attempted to understand and deal with complexity; but the nature of that which was being engineered tended to be stand-alone with well-defined edges. Simple rules could tell what was "in the system" or "out of the system;" and the engineering activities started with, or required that, the requirements were well known, understood, and stable. As systems engineering came to deal with collections and aggregations of elements which were to be integrated into definite, well-understood (and understandable) forms-and-function which were to be stable over time. As we scale this approach up to the enterprise and find ourselves dealing with complex systems, we fall directly into the trap outlined by Alexander: things become worse.

Our traditional systems engineering has been concerned with finding those well-bounded subordinate elements, then (in essence) isolating them so they may be "engineered." From this point one proceeds as if the element is isolated and unmoved by other juxtaposed elements. It's this desire to "divide and conquer" which characterizes our tradition approaches. Is this wrong? No! But it's not always correct; nor is it complete.

Consider the richness now possible due to the potential interconnectivity now available, and the interdependence among elements implied. The forms an ecosystem where each element responds to its context through some accommodation - potentially evolving to respond (those elements which are "alive" respond and change). Consider further that each element's context is set by the elements which juxtapose it in almost countless ways (forming a hyperspace of *pressure*). This is certainly an intricate, hard to understand-and-appreciate situation; and in that way, it may be thought of (in the usual vernacular) as *complex*. Using our traditional divide-and-conquer systems engineering (TSE) we would likely measure the external world, then make an assumption of constancy with respect to this external surround. Engineering would proceed on the element from this point of view.

But, the realization that the element under study also forms part of the context for every element which juxtaposes it starts to hint at the limit of the simplifying assumption made to perform the TSE: It imposes a *pressure* (an influence) on its surround in addition to feeling the pressure of the surround.

Note the implications of the transitive nature of these influences. This is what is referred to as *complexity* as opposed to *intricacy* or *difficult to understand*. One could imagine waves and ripples of change flowing through this system of system. Likely, patterns will emerge when viewed from a higher level of abstraction. This is likely where we find ourselves with respect to C2 in a joint and coalition world, and where independent agents can introduce change according to their own agenda and timing.
The big questions: can such an aggregation be engineered at all? Can it even be understood? Will useful patterns present themselves? Should we even bother? We can say that we've failed many times in the past because we've made the simplifying assumptions I mentioned earlier.

One of the principles that pops out right away from taking this *complexity* point of view is the need to have tight control over all (or as many) characteristics of a system to be engineered as possible if one wants to apply TSE. Especially the ability to direct resources to problem areas as they arise. Is this insight new? No; but maybe, where it is impossible to meet that control and authority boundary conditions, there might be other approaches which can be brought to bear. And we believe the codifying of CSE is a step in the right direction.

Understanding complexity, and engineering complex systems is the next step. Systems Engineering is taking the next step. It is maturing past the point where a one-size-fits-all process is what's thought of as "correct." It is finding a new language with which to understand that which it attempts to engineer. This is maturity, and this is its future.

References

Albert, D., Garstka, J., Stein, F.; 1999; Network Centric Warfare; CCRP

Alberts, D., Garstka, J., Hayes, R., Signori, D.; 2001; Understanding Information Age Warfare.; CCRP

Alberts, David S., Hayes, Richard E.; 2003; Power to the Edge;. CCRP

Alexander, Christopher; 1979; The Timeless Way of Building; Oxford University Press

Ashby, W.R.; 1956; Introduction to Cybernetics; Chapman & Hall

Axelrod, R.; 1984; The Evolution of Cooperation; Basic Books.

Barabasi, L-S.; 2002; Linked: the New Science of Networks; Perseus.

Bar-Yam, Y.; 1997; Dynamics of Complex Systems; Perseus;.

Bar-Yam, Y., 2003; When Systems *Engineering Fails---Toward Complex Systems Engineering*; 2003 IEEE International Conference on Systems, Man & Cybernetics, October 5–8 Washington, D.C., USA

Bonabeau, E.;2003; *Swarming Intelligence*; Swarming Network Enabled C4ISR Conference 13-14 January; McLean, VA

Breen, P., Case, R., Kazura, A., Norman, D.; 2000; *DSE – A Decision Support Enviroment*; 2000 Command and Control Research and Technology Symposium, Naval Postgraduate School, Monterey, CA, June 26-28, 2000

Card, O. 1977. Ender's Game; Tor.

Gamma, E., R. Helm R. Johnson, J, Vlissides; 1994; Design Patterns; Addison-Wesley

Heylighen F., Joslyn C. & Turchin V. (eds.); 1995; The Quantum of Evolution. Toward a theory of metasystem transitions, ; Gordon and Breach Science Publishers, New York

Heylighen, F., C. Joslyn; 2001; "Entropy and Information," *Principia Cybernetica Web*, Sep 3,; pespmc1.vub.ac.be/entrinfo.html

Holland, J.H.; 1992; Adaptation in Natural and Artificial Systems; MIT Press

Holland, J.H.; 1995; Hidden Order: How Adaptation Builds Complexity.; Reading: Addison-Wesley.

INCOSE; 2001; Systems Engineering Handbook, v2;

Kauffman S.A; 1993; The Origins of Order: Self-Organization and Selection in Evolution, Oxford University Press, New York,

Krygiel, Annette J.;1999; Behind the Wizard's Curtain: An Integration Environment for a System of Systems; CCRP;

Kuhn, T.S.; 1996; Structure of Scientific Revolutions**,** , University of Chicago Press,

Moffat, J.; 2003; Complexity Theory and Network Centric Warfare;. CCRP

Poundstone, w.; 1992; Prisoner's Dilemma; Anchor Books.

Van Dyke P.; 1997; "*Go to the Ant": Engineering Principles from Natural Multi-Agent Systems*; Annals of Operations Research 75, pp. 69-101;.

Van Parunak, A; 2003.; *Making Swarming Happen*; Swarming Network Enabled C4ISR Conference 13-14 January, McLean, VA

Watts, D.; 2003; Six Degrees: The Science of a Connected Age; Norton.

Douglas O. Norman is the Chief Technologist for USAF Battle Management Capabilities (HQ ESC/AC), and is MITRE Section Leader for Battle Management and C2. Previously, he was the Chief Engineer of the Theater Battle Management Core Systems (TBMCS). He also holds the position of Senior Technical Advisor to the AOC Weapon System Acquisition Group Commander.

Michael L. Kuras is a Principal Systems Engineer in MITRE's USAF Systems Engineering Division with significant interest and experience discovering, defining and applying Complex Systems Engineering principles.

Appendices

A.1 Phase-spaces for systems

A physical ensemble of N parts (an ensemble is something more general than a system) can be characterized by first identifying in three dimensional space where each part is (x, y, z), and then what the momentum of that part is (p_x, p_y, p_z) at that point. A 6N dimensional hyper-cube can be constructed and a point (called the phase point) identified in this hyper-cube (which is called the phase-space). Such a point specifies the position and momentum of every part in the ensemble which is also called the state of the ensemble. The notion of momentum in this original classical model can be generalized to be the "internal" state of the part, and the location can also be made more general. The resulting hyper-cube can then have a substantially higher dimensionality than 6N. Regardless of dimensionality, in the classical approach, the state of the ensemble is represented by the phase point in phase-space which is understood to determine (or to be determined by) the states of the individual parts. In this approach, the state of the ensemble is frequently equated to the order of the ensemble. This is not the definition of order used here.

Another approach is to identify the N_r relationships present in a system, and to use these to construct another "hyper-cube," each dimension corresponding to a relationship in the system. If desired, the individual relationships (functions) can be resolved into partial functions over the fields of the attributes used to characterize the parts of the system (position, color, shape, momentum, etc.). In general, the dimensionality of the relational "hyper-cube" for a system is denoted as δ. Since a metric for these relational dimensions is not necessary, the image of a "hyper-cube" is merely suggestive.

The phase-point (or phase trajectory) in the δ-dimensional hyper-cube corresponds to the order of the system. It specifies (or is specified by) which relationships actually apply to the system and the instant and coincident attribute values of the various parts of the system that are bound together by the relationships.

Not every relationship that might apply to the N parts of a system need be present in a given system (just as not every part need be included in any given system). The (relational) "super" phase-space for a system (as a hyper-cube) would include all such real and possible relationships as the dimensions of the hyper-cube.

a.2 Multiscale analysis

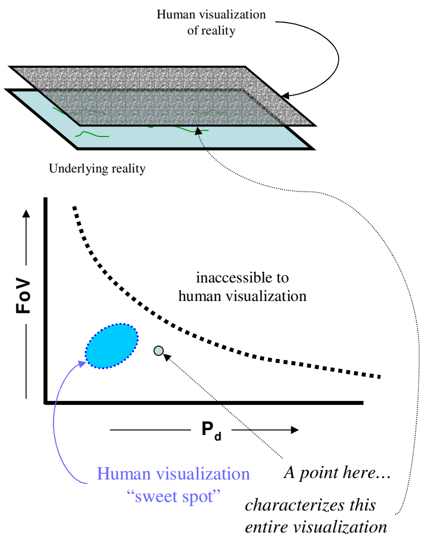
The single most powerful tool in traditional system engineering is the process of decomposition – and its complement of integration.[38] Both of these rely on the

[38] This is inclusive of abstraction, layering, etc. [38] Modular decomposition (at one level of scale) is possible if all of the relationships (including interactions) between the components in a set A with the components in the complement set $\bar{A}$ can be completely restated as the relationships between a single component element a* and the components in $\bar{A}$.

assumptions of "linearity" in the "uniscale" analysis or synthesis of a system. To understand the limits of traditional system engineering it is necessary to recognize the circumstances in which either or both of these assumptions are no longer valid.[39]
"Linearity" asserts that relationships are proportional, can be superimposed, and are reversible. Such relationships can be either static or dynamic.

"Scale" is an attribute of the human visualization of a thing. A simple model of visualization is <FoV, P_d> where FoV is field of view and P_d is "pixel" density. A pixel is a point in a visualization. It is a cellular automaton; it has state. A pixel has no extent (in the FoV) or internal structure (its cellular program is unknown). A pattern in the pixels is what is associated with "understanding." Scale is a point in <FoV, P_d>. It is important not to confuse a scale point with a pixel.[40]

- Scale is an attribute of a specific human visualization of a thing.
- A simple model of scale in visualization is:
  - A visualization is <FoV, P_d> where
    - FoV is field of view.
    - P_d is "pixel" density.
  - A "pixel" is a point in a visualization. It is a cellular automaton; it has state. It has no extent (in FoV) or internal structure (its cellular program is invisible).
    - Patterns in the pixels is what we associate with "understanding."
  - Scale is a point in <FoV, P_d>.
    - Not to be confused with a pixel!

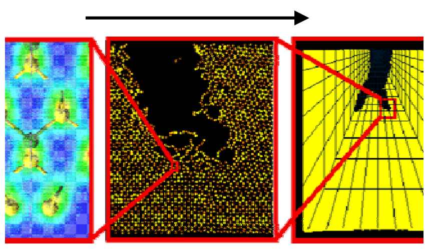

Human visualization of reality
Underlying reality
FoV
P_d
inaccessible to human visualization
Human visualization "sweet spot"
A point here...
characterizes this entire visualization

[figure 4]
As suggested in the graphic, reality is distinct from human visualizations of it. "Breadth" and "depth" are terms that can be used to reference the attributes of the underlying reality that are made to correspond to FoV and pixel density in a visualization of that reality.

[39] The theoretical limits imposed by these assumptions may themselves be beyond reach for practical reasons such as constraints on the time available, or the resources available, for analysis or synthesis.
[40] A slightly richer model of visualization is <FoV, P_d, P_v> where P_v is the state-space of the pixel. The purpose here is to expose the notion of multiscale analysis and the simpler model will do.

A portion of visualization space is inaccessible due to the finite capacity of the human brain. The underlying reality need not be. Rescaling is a way to compensate for this human visualization limitation when attempting to understand all of underlying reality. Re-scaling (e.g., for decomposition or for integration) is not simply about moving from point to point in the visualization space <FoV, P_d>. Rescaling is about changing the association between <FoV, P_d> and breadth and depth in order to access initially inaccessible aspects of the underlying reality. The following graphic suggests one simple example of how this works.

Re-scaling: a simple example



- Trading "depth" for "breadth."
  - Depth and breadth are attributes of the underlying reality, not the visualization.
- Re-scaling here is the expansion of FoV to take in more breadth by reducing the depth (detail in the underlying reality) associated with one pixel.

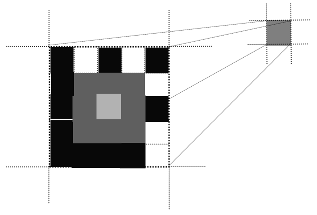
[figure 5]

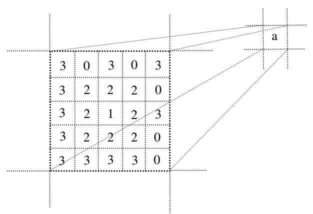
In this example, FoV is expanded in order to take in more of the breadth of the underlying reality while pixel density is reduced (thereby accessing less of the depth in the underlying reality).[41]

In this example, rescaling can be understood as combining many "small" pixels into "larger" ones over and over again simultaneously over the entire FoV. This understanding, however, exposes the limitations and potential pitfalls of rescaling.

[41] As long as the association of either FoV or P_d to breadth and depth remain fixed (even while FoV or P_d may be changed), the analysis a visualization supports remains uniscale.

Re-scaling: a simple example (cont.)





- What should the average be?
    - Arithmetic? (2) Modal? (3) Etc.
- Something else?

- Why is this important?
- **Understanding is linked to recognizing patterns in visualizations.**
- What to do when there is MORE than one pattern to be constructed during re-scaling?

[figure 6]

Even assuming that the sets of "smaller" pixels are all disjoint and cover contiguous underlying reality, and that all "smaller" pixels are combined into "larger" ones in the same way,[42] there can still be more than one way to perform such combinations. If more than one such way is valid (corresponding in some way to the underlying reality by preserving access to existing patterns), a dilemma exists: which ONE way to perform the combining? This is the dilemma of "uniscale" analysis underpinning traditional system engineering. There is, in general, more than one valid way to do the rescaling, but only one way to actually record it. The way out of this dilemma is to acknowledge the validity of "multiscale" analysis.

Multiscale analysis asserts that concurrent and valid rescalings are sometimes necessary in order to preserve or expose all of the patterns necessary for a full understanding of any underlying reality, but that human visualization imposes a limit of one. As a result, the same underlying reality must be visualized (differently) multiple times without there being any way to directly visualize a combination of such visualizations.

Humans have the ability "to walk and to chew gum" at the same time; the human intellect can concurrently maintain to some extent multiple visualizations of reality.

[42] There is, of course, no reason to believe that such assumptions are generally valid.

This ability is extremely limited; moreover, it is poorly understood at present. It is beyond the scope here to explore when multiscale analysis becomes necessary, or how to concurrently apply multiple visualizations to the same underlying reality. What is crucial to understand is that there is NO WAY to “average” or otherwise algorithmically combine multiple human visualizations of reality in a linear theoretic fashion without losing access to some portion of the underlying reality.