Systems Engineering Complexity in Context

Sarah A Sheard, Ph.D.[1]
Stevens Institute of Technology
sarah.sheard@incose.org

Abstract. Systems engineering literature identifies many different kinds of complexity. One paper cataloging complexity for the purpose of modifying systems engineering cost estimates identified more than 30 types. The research described in this paper investigates the idea of sorting and organizing the many concepts of complexity and showing the relationships between them on a single two-dimensional chart.

The Systems Engineering Complexity Contexts (SECC) chart was developed to show how a large number of complexity concepts relate to systems engineering and to each other. All but two of the 30 types from the paper mentioned above fit on this chart. Those two appear as aspects that may apply to all other boxes on the chart instead of as orthogonal elements, which was the desired representation.

Introduction

Complexity is often blamed for systems engineering problems, (Britcher, 1998; Calvano & John, 2004; Kurtz & Snowden, 2003; Stevens, Brook, Jackson, & Arnold, 1998; White, 2005) but rarely does the process of fixing blame involve crafting a precise definition of the problem. The systems engineering literature describes a wide range of concepts of complexity, including the number and type of stakeholders, the organization's maturity and scope, and the required product quality. (Maier, 2007) In fact, a paper cataloging complexity for the purpose of adjusting systems engineering cost estimates, Young, Farr, and Valerdi (2010) identified more than 30 types.

It is very difficult to apply such a broad topic as complexity to the equally broad topic of systems engineering. Systems engineering comprises all life cycle phases, from early analysis to anomaly resolution. Various groups, from government acquisition offices to programmers, require many different views of systems engineering. Even if there were only one kind of complexity, it could still apply differently to each aspect of systems engineering. Multiply the number of possible variations by the number of complexity types and it quickly becomes apparent that without a simple framework, describing the effects of complexity on systems engineering would be too convoluted to be useful.

In 1962 Hall stated that the genesis of systems engineering was the need to deal with complexity (Hall, 1962). Perhaps after fifty years it is time for systems engineering to deal with the confusion resulting from the large variety of complexity concepts. Can the various concepts be brought together in an organized form? Is there a way to show their relationships, perhaps on a single two-dimensional chart? Would the chart permit a newly identified conceptualization of complexity to fit within its structure?

The goal of this research is to identify such a two-dimensional structure. Success will be achieved if all or nearly all of the relationships among different aspects of systems engineering complexity can be shown on one chart, without requiring elements that apply to

[1] Now at the Software Engineering Institute, 4500 Fifth Avenue, Pittsburgh Pennsylvania (US).

all boxes (which would represent an additional dimension). Success also implies that when a new typology of complexity arises, its elements can fit in a specific place on the chart.

This paper describes a structure that comes close to achieving these goals. A new chart, the Systems Engineering Complexity Contexts (SECC) chart, was derived from the complexity concepts discussed in Young, et al. (2010) All of the concepts from Young, et al. can be located on the structure, except for two that are captured not as the desired orthogonal elements but as aspects that may apply to all other boxes on the chart. These two non-orthogonal aspects keep the SECC from fully meeting the intended goal of the research.

Methodology

Complexity definitions. An early and ongoing step in this research identified many different ways that systems engineering has encountered complexity. Some of the most often mentioned attributes of complexity are that it makes projects difficult, that it involves large system or project size and high connectivity, and that it arises from many causes, ranging from difficult requirements to project management. The Young, et al. paper was identified as the most complete compendium of complexity definitions and was therefore chosen for the initial test of the structure.

Typology. Another research step, described in a previous paper (Sheard and Mostashari, 2010), created a typology of complexity: structural types (size, connectivity, and inhomogeneity or diversity), dynamic types (short-term and long-term), and sociopolitical types. These types were challenged through application to systems engineering for a survey of complexity vs. project outcomes (Sheard, 2012); the result was the addition of entities (those things that are evaluated as more or less complex). Entities include the *System* itself, the *Project* building the system, the *Environment* (both socio-political and technological), and *Cognition* (a subjective or cognitive type). (Sheard and Mostashari, 2011)

Application to systems engineering. These steps were preparatory to the main goal: organizing the ways that different definitions of complexity apply to systems engineering. The SECC structure followed from describing what happens in the *Environment* entity that leads to a *Project* entity being set up to build a *System* entity. Ways in which each entity can be complex are emphasized in the descriptions. These three entities are linked via the life cycle of systems. The fourth entity, *Cognition*, was added later to emphasize ways in which other activities lead to confusion, uncertainty, frustration, and lack of knowledge. Showing how the activities for each entity relate, the full-page SECC diagram allows each of the 30+ types of complexity from Young, et al. to be located upon it.

Entities

To apply complexity concepts, one must ask is what is the object—the "entity"—that is complex. Within systems engineering there are three to five different kinds of entities that may be complex. Four of these five include the following:

1) the (technological) *System* being designed and built
2) the (socio-technical) *Project* doing the building
3a) the technological *Environment* into which the system will be inserted when built (e.g., the hardware and software technical systems with which this system must interface, for starters)

3b) the socio-political system related to the technological environment, generally system stakeholders (Sheard and Mostashari, 2011)

The last two, 3a and 3b, could be considered separate entities, or grouped together into an entity called the "environment," the "supersystem," or the "complex system encompassing the technological system being built."

The following could be considered a fifth entity:

4) the subjective human experience when thinking about, designing, or using the system, called *Cognition*

These entities are shown in Figure 1, with 3a) and 3b) grouped together into "*Environment*" and subjective experiences (*Cognition*) added.

Figure 1. Entities

The entities are shown as yellow, green, pink, and blue, respectively. To differentiate them when the paper is not viewed in color, the varied border lines indicate the same information. In the rest of this paper, activities related to each of these entities are color- and border-coded as follows:

- *Project*-related activities have yellow fill (dash-dot border)
- *System*-related activities have green fill (solid border)
- *Environment*-related activities have pink fill (dash border)
- *Cognition*-related activities have blue fill (dotted border)

When activities relate to two different entities, both colors appear.

Activities related to the entities

In this section, the entities are decomposed into interrelated parts using the system life cycle as a framework. Although a system is specified prior to being built, specification is *not* the beginning of the life cycle: it actually begins when a problem is perceived.

***Environment* entity, discussion #1.** The perceived problem is expressed under the *Environment* entity, starting as: "Environment is an ongoing system: *the Way Things Are*." The environment has perceived problems. Someone, possibly a politician, interest group, or contractor in a "pre-proposal" effort, envisions a potential solution to one or more of the problems.

The environment also includes stakeholders, some of whom have resources that can be applied to improve *the Way Things Are*. The envisioned solution will provide a desired intervention into *the Way Things Are*. Once the envisioned solution, a budget, and an

organization to do the building have been agreed on, system development begins. These activities are shown in Figure 2.

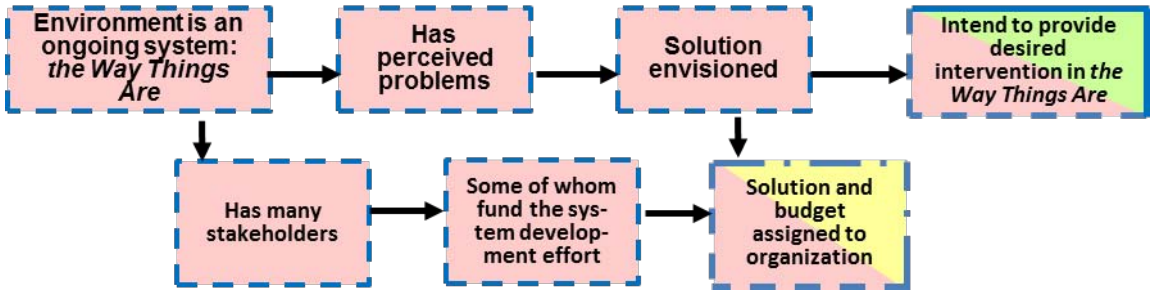

Figure 2. *Environment*, from *the Way Things Are* to Initiating System Development

Note that the yellow fill (dash-dot border) indicates that the last activity concerns the *Project* entity as well as the *Environment* entity. Green/red (solid/dashed border) indicates an interface between the *System* and the *Environment*.

***Project* entity.** Next is the *Project* entity. The project's organization usually exists before a project is started, although not always. At some point the organization establishes a project, which in general consists of many people who interact and form changing teams. Their work consists of many tasks, some of which build system elements. Note that "tasks build elements" represents an interface between the *Project* and the *System* entities. Figure 3 shows this sequence and the interface relationship.

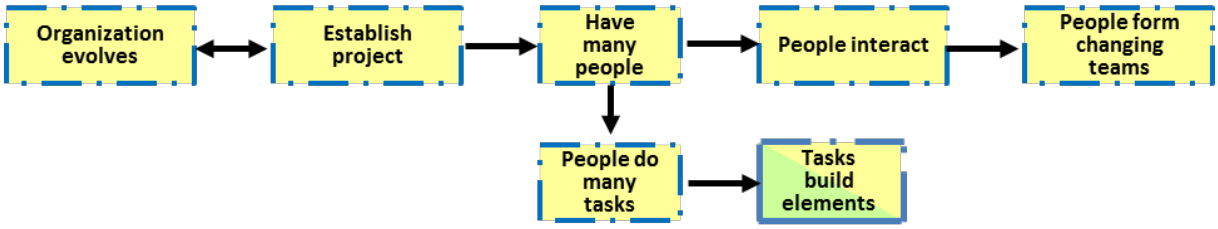

Figure 3. *Project*, from Organization Through Tasks

The project itself is initiated with a plan of how the resources will be spent (i.e., the budget). The immediate goal is to produce the desired system. A better goal is to reduce the problem experienced in *the Way Things Are*, by means of producing an effective system and inserting it effectively into the environment. Organizations that understand how the project goals relate to the environment are better able to produce effective systems. Figure 4 shows this sequence. Many of these activities are shown as interfaces, either between the *Project* and the *System* (green/yellow), the *Project* and the *Environment* (yellow/red), or the *System* and the *Environment* (green/red).

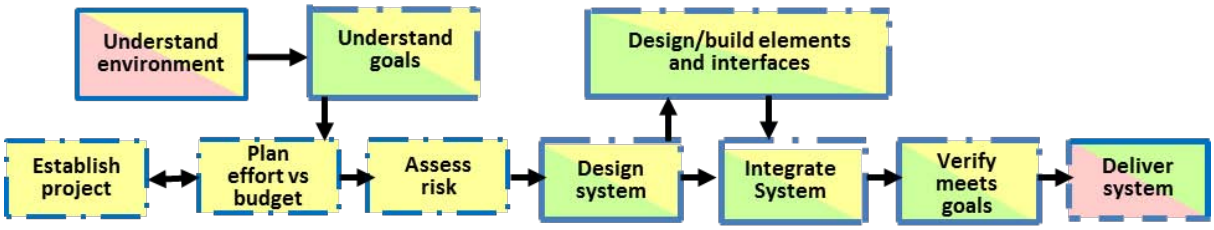

Figure 4. Project Life Cycle

***Environment* entity, discussion #2.** When the system is eventually inserted into the environment, it affects *the Way Things Are*, ideally to the point of resolving the original problem (Figure 5). Thus it causes *the Way Things Are* to change (evolve). Note that *the Way Things Are* was never fully understood, nor were all of its elements or behaviors, and its problems were also uncertain. (Uncertainty, since it resides in the human mind, is shown as blue, the color for the *Cognition* entity.)

This uncertainty and other issues can lead to side effects. Sometimes the system did not do what was intended. Sometimes the system did do what it was supposed to do, but the environment experienced an unexpected change in response to the system's presence. Sometimes the environment had features that were simply not understood and therefore not designed for. Usually, *the Way Things Are* has also evolved because of reasons unrelated to the system. These reasons combine so that the effect of the system is not quite as positive as originally envisioned.

Thus there is a new problem, one that perhaps can be ameliorated by the creation of a new system, given appropriate sponsors and resources, and the cycle begins again.

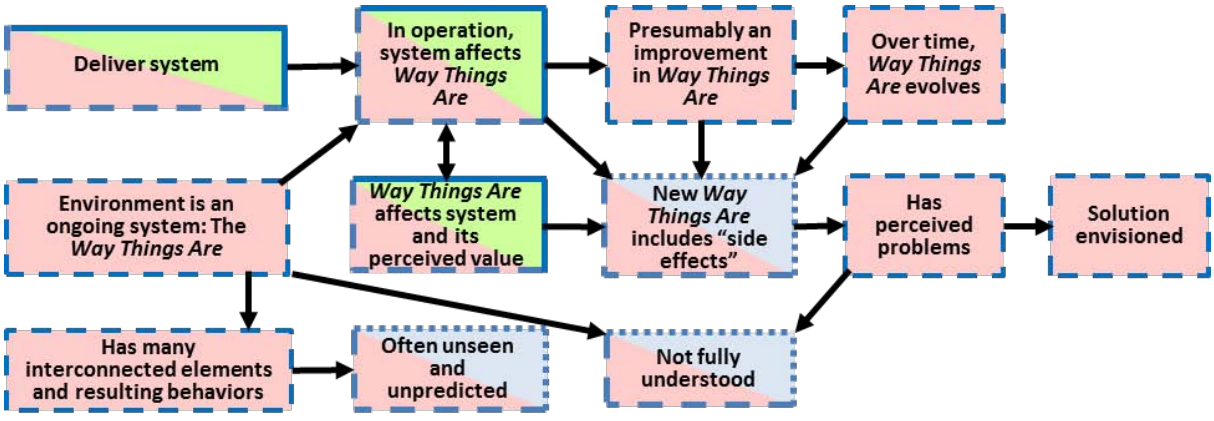

Figure 5. *Environment*: System Inserted Into *the Way Things Are*

***System* entity.** To describe the complexity of a technological system, it is useful to look at how it is composed. The system has many diverse, interconnected elements that are organized in a multi-level structure. Each element has its own structure and behaviors. There are usually many paths through a technological system, especially through its software, that

need to be tested. The system behavior arises from the connected elements and sometimes displays known patterns, but at other times shows emergent behavior that is surprising. Figure 6 shows these aspects of the situation. The blue (dashed border) indicates that the fourth entity, *Cognition*, comes into play where there is significant uncertainty.

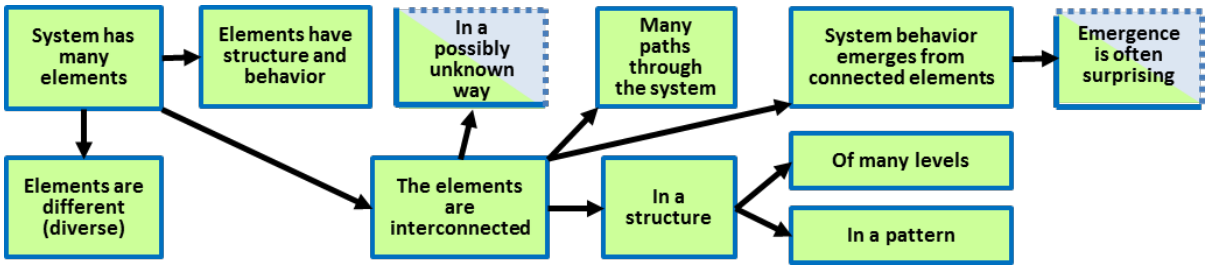

Figure 6. *System* and its Elements

Subjective or cognitive complexity (*Cognition*). People have cognitive limitations. Inability to understand or predict creates uncertainty, which causes risk and makes people uneasy. Therefore, people take steps to reduce uncertainty, by dividing up or sharing tasks, or by creating support tools. This is shown in Figure 7.

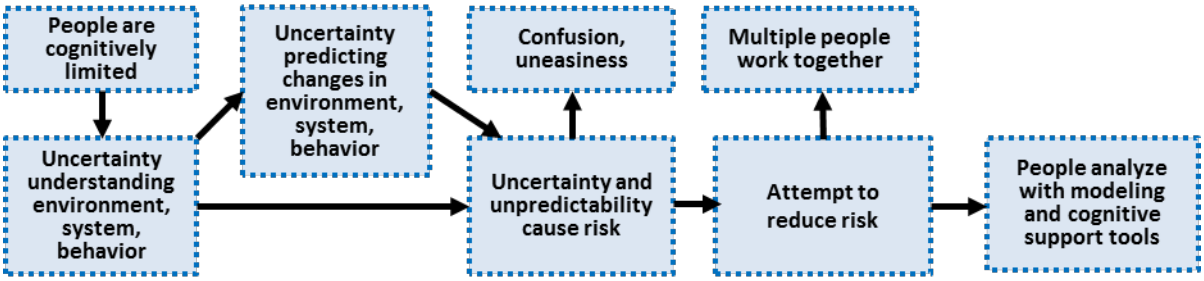

Figure 7. *Cognition* Aspects

In addition to these activities, all activities mentioned so far have some uncertainty; the ones with arguably the most show half blue (although if they already had two colors, blue was not added).

Additional activities. Two other steps are necessary before bringing the above threads together. The first step is to note two generalities: *the Way Things Are* changes with time (as does each activity related to any of the entities), and all of the steps consume and produce information, which can be handled separately from the actual performance of the activity. The second step is to add two activities performed during element design and build: "element technology maturity" and "computer and software expertise needed," which are both *Project*- and *System*-related; see Figure 8.

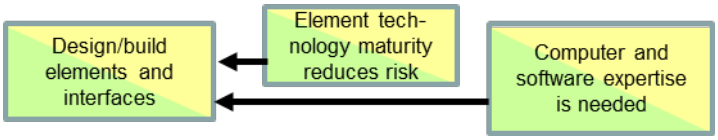

Figure 8. Two Final Activities

Note that computer and software expertise areas were specified because a lack of these two kinds of expertise appeared in descriptions of complexity from the literature. (Complexity as a scientific topic has been applied to computers and software much more often than it has to other kinds of systems.) Clearly other expertise areas are also needed to develop systems, and they also occupy this box.

Systems Engineering Complexity Contexts (SECC) chart

The overall chart that shows all these aspects of systems engineering that may be complex is called the Systems Engineering Complexity Contexts (SECC) chart (Figure 9). The chart is created by orienting the above threads mostly vertically and showing interconnects among them. Because the *System* entity shares many activities with the *Project* entity and many with the *Environment* entity, *System* is placed in the middle. *Project* activities are shown on the left, followed by combined *System/Project* activities and *System* activities. *Environment* activities follow, and *Cognition* is on the right. Arrows drawn between the activities duplicate the above arrows (solid lines) and add new lines (dashed lines), generally between entities.

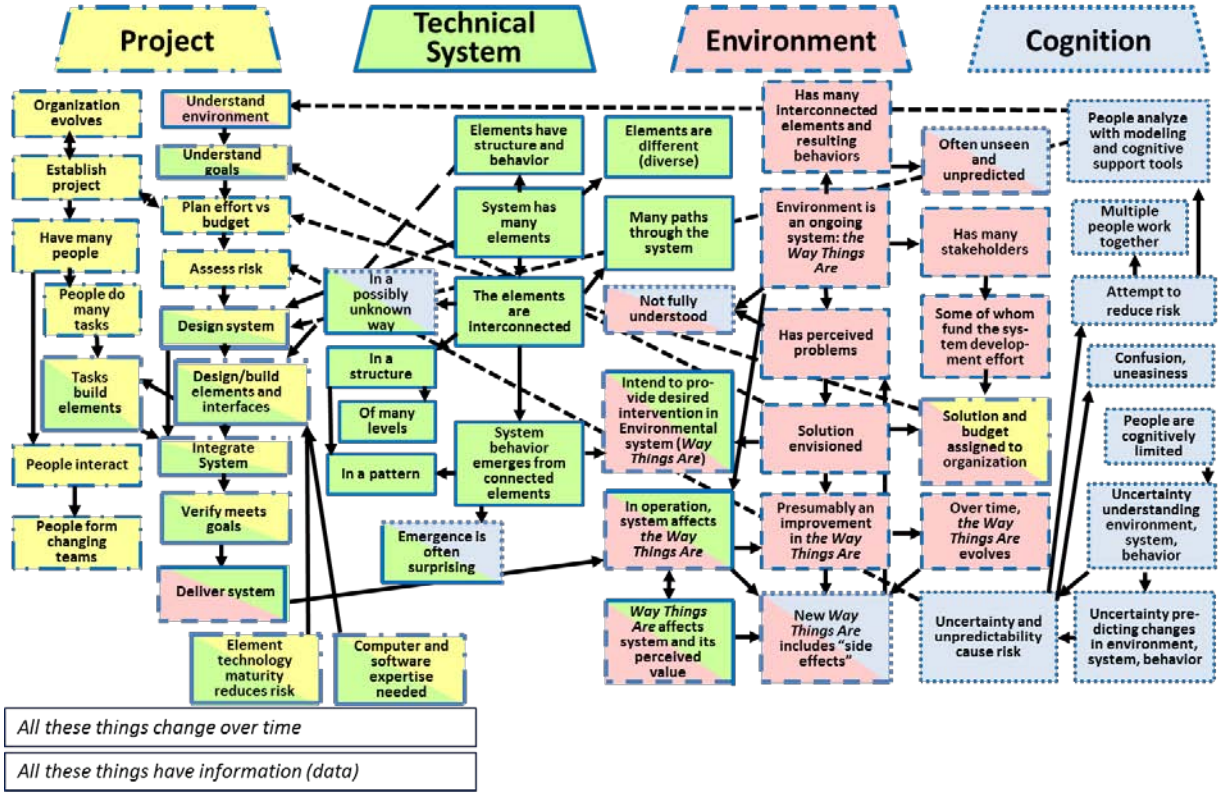

Figure 9. Systems Engineering Complexity Contexts (SECC)

The best way to understand the story is to start with *the Way Things Are*, as has been done in the descriptions above. Other paths are also instructive, particularly looking at the newly-added (dashed) lines. Essentially, the technological *System* is built by the *Project* to solve a problem in the *Environment*. Difficulties arise because of complexities related to all three entities, as well as limitations of people (*Cognition* entity). Uncertainties lead to risks as well as to mitigating activities such as tool development. Risks may turn into project issues or may affect how well the system solves the environmental problem.

If the SECC really addresses the complexities related to systems engineering, then various taxonomies developed to describe complexities in systems engineering contexts can be related to the elements of the chart. The test of this hypothesis is described in the next section.

Verification

Two tests were performed to see whether the SECC chart captures the important kinds of complexity, as applied to systems engineering. The first was to verify that the types of complexity noted in Young, et al. could be located on the chart, since that compendium was the impetus for making the chart. The second was to see whether the chart was general enough to apply easily to other typologies.

First test: Young, et al. types of complexity. Table 1 lists the types of complexity adapted from Young, et al. Their sources are shown in the table, but those sources are not included in the reference section of this paper unless cited elsewhere. Note that their work cataloged the kinds of definitions discussed by different sources without attempting to resolve overlaps. Figure 10 shows that indeed all of these can be assigned to specific locations on the SECC chart. Numbers in the left column of Table 1 correspond to the white numbers in black ovals on the figure.

Table 1. Complexity Types from Young, Farr, and Valerdi (2010)

#	**Type of Complexity**	**Sources**
1	Hierarchical/Structural (# levels)	Ross & Arkin, 2009; Kolasa, 2005; Kitano, 2002; Edelman & Gally, 2001
2	Configuration Complexity	″
3	Complicatedness/ Functional Complexity	″
4	Subjective Complexity	Reitsma, 2003
5	Statistical Complexity	″
6	Algorithmic/Deterministic Complexity	″ ; Manson, 2001
7	Aggregate Complexity (interrelationships)	Manson, 2001
8	Project Complexity (organizational and technological)	Baccarini, 1996
9	Project Complexity (assembly, system, array)	Sauser et al., 2005; Shenhar & Dvir, 1996
10	Product Complexity (physical)	Williams, 1999
11	Structural Organizational Complexity	Xia & Lee, 2004, 2005
12	Structural IT Complexity	″
13	Dynamic Organizational Complexity	″
14	Dynamic IT Complexity	″
15	Inter-Component Complexity (can grow exponentially)	Rumpler, 2006

#	Type of Complexity	Sources
16	Interface Complexity (by component)	"
17	Implementation Complexity (e.g. code)	"
18	System-level Complexity (emergent)	"
19	Structural Complexity (design and structure, persistent)	Laird & Brennan, 2006; Tran et al., 2002; Fenton, 1994; Lew et al., 1988
20	Conceptual Complexity (psychological)	"
21	Computational Complexity (algorithms)	"
22	Structural/Combinatorial Complexity	Mostashari & Sussman, 2009
23	Behavioral Complexity (unpredictability)	"
24	Nested Complexity (technical/socio-technical)	"
25	Evaluative Complexity (multiple stakeholder viewpoints)	"
26	Static* Complexity	Sheard & Mostashari, 2009
27	Dynamic Complexity	"
28	Social-Political Complexity	"
29	Technical Complexity (Systems Integration- based)	Jain et al., 2008
30	Programmatic Complexity (Systems Integration based)	"
31	Configuration Complexity (Systems Integration based)	"
32	Operational Complexity (Systems Integration based)	"
33	Organizational Complexity (Systems Integration based)	"

* This term was used in an early version made available to Young, Farr, and Valerdi, but was changed to "Structural" before publication.

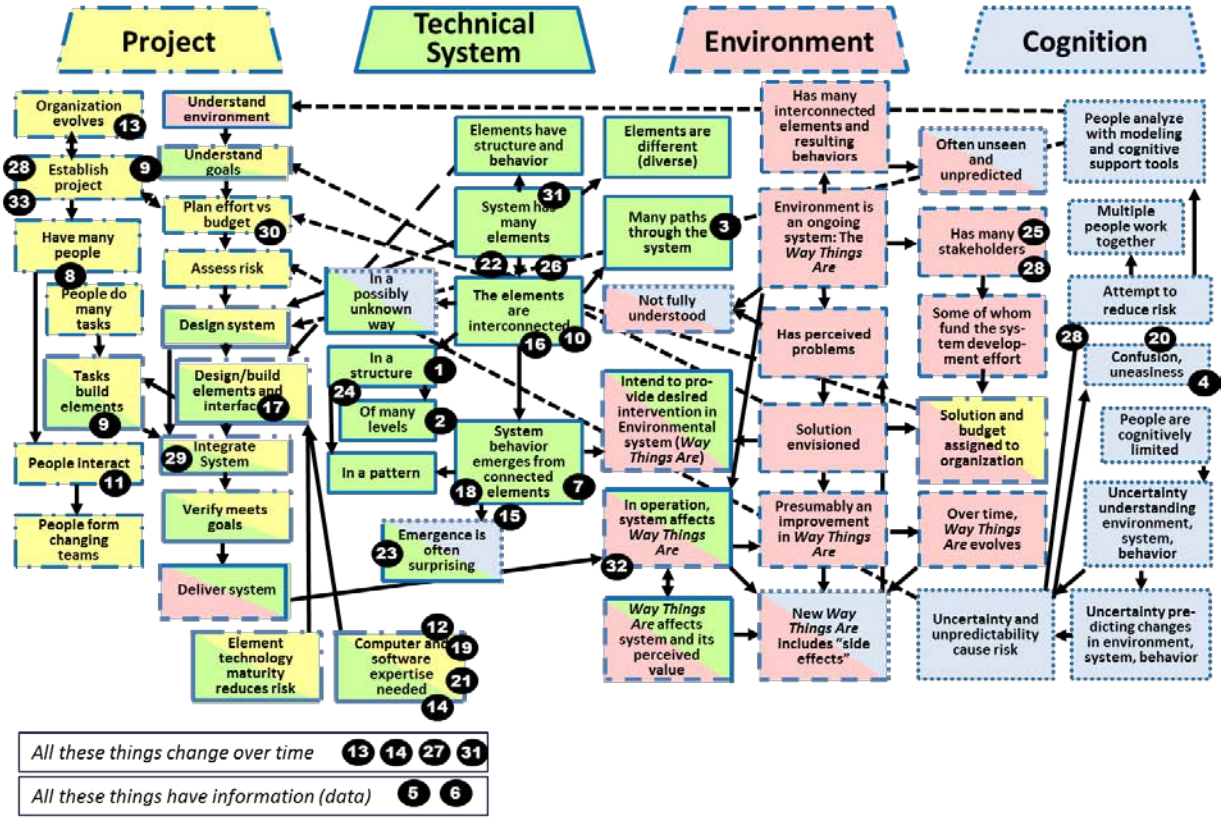

Figure 10. SECC Chart Locates Complexity Types

Second test: two additional typologies.

Sheard and Mostashari's (2010) six types are numbered in the key of Figure 11, which identifies where the six types appear on the SECC chart.

Size refers to extent (often appearing as amount of money dedicated to the project or to maintaining an ongoing system) or other countable items such as number of users, number of hits, number of components or platforms or parts or lines of code. Size appears on the SECC chart wherever a number of things or people are mentioned.

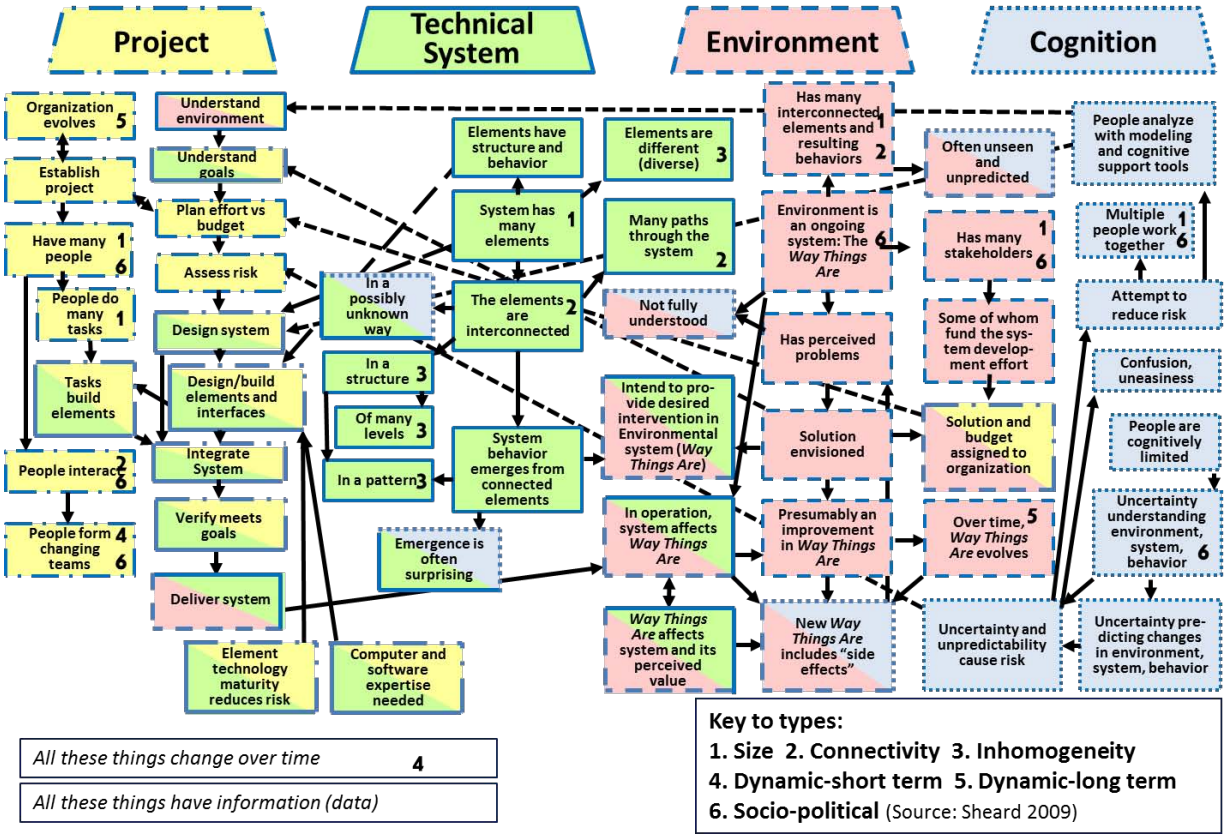

Figure 11. Types of Complexity Applied to These Activities

Connectivity refers to the number of connections among parts, whether physical interfaces, logical interfaces (including buses, protocols, or messages), social connections, or perhaps process sequence. Connectivity appears on the SECC chart when elements, behaviors, or people interact.

Inhomogeneity, also called diversity, includes the number of different kinds of elements or interfaces; it also comes into play when a space or ecosystem has a variety of niches or differentiated areas where characteristics are different from the average. Complexity due to architectural considerations (such as hierarchies, layers, or clusters) is included here as well. Inhomogeneity appears on the SECC chart where diversity or architectural patterns are mentioned.

Short-term dynamics constitute considerations in the operational time frame, for example, the urgency of action to prevent a catastrophic explosion. This appears on the SECC in the box about change, which is one of the cross-cutting factors that applies to most things.

Long-term dynamics considers evolution of a species, state, or configuration into an entirely new system as its pieces adapt, evolve, or perhaps are replaced with a new technology. This appears on the SECC with respect to the evolution of *the Way Things Are* and in the evolution of the organization that forms projects.

Socio-political complexity comprises the non-technical considerations that arise when human agents have opinions, intents, schemes, and plans to obtain resources and incentives (this is often considered the realm of social sciences and liberal arts rather than of hard science and engineering). Socio-political complexity appears on the SECC chart under two of the *Cognition* entity activities, two of the *Environment* activities, and three of the *Project* entity activities. Of course, like dynamics, people are involved in almost every aspect of systems engineering.

Some of the boxes shown on the diagram that do not have indicators of type are descriptions of the world the way it is (*the Way Things Are* box and related boxes), or of how a project works, and do not indicate complexity *per se*.

Maier (2007) addressed 11 different dimensions of complexity, shown in Table 2. Most of these dimensions fit well into the Systems Engineering Complexity Contexts structure, but two do not. His "Situation objectives" and "Feasibility" do not have an obvious location. However, inserting "Feasibility of meeting intervention objectives" next to "Intend to provide desired intervention in *the Way Things Are*" under *Environment* locates the "Situation objectives" dimension. Similarly inserting "Feasibility of meeting project objectives" next to "Understand goals" and "Plan effort vs. budget," between *Project* and *System*, locates "Feasibility." In this case the Systems Engineering Complexity Contexts chart can be extended without significant rewriting to accommodate an additional second set of definitions or dimensions of systems engineering complexity.

Table 2. Maier's Dimensions of Complexity

Factors in Complexity of System Development Efforts (Maier, 2007)	
Factor	Spectrum
Sponsors	One, with funding, ─ Many, without funding
Users	Same as the sponsors ─ Unknown
Technology	Low ─ Super-high
Feasibility	Easy ─ Not
Control	Centralized ─ Virtual
Situation Objectives	Tame ─ Wicked
Quality	Measurable ─ One-shot, unstable
Program Scope	<$1M ─ >$1B
Organizational maturity	High ─ First of kind
Technical Scope	Discrete Product ─ Assemblage of products and enterprises
Operational Adaptation	Stable ─ Full Scope Adaptation

Uses of the SECC chart

The SECC chart describes how normal systems engineering works, with a focus on the ways that systems and systems engineering can be complex. More specific ways of dealing with complexity can be developed if what "complexity" really is can be made clearer. It was difficult to describe the needed tools, training, measures, standards, or research direction when systems engineering was seen as a featureless monolith. Defining a number of systems engineering roles and implementations improved the practice by allowing discussions of processes, tools, etc., to address specific types of systems engineering (Sheard 1996 and 2000). Clarifying the complexity related to systems engineering is expected to have a comparable effect, allowing development of specific complexity remedies for specific types of systems engineering.

In the case of every kind of complexity studied with respect to systems engineering in a doctoral dissertation (Sheard 2012), complexity was associated with worse outcomes for all statistically significant associations. It is advantageous to reduce complexity as much as possible, without creating rigidity or overly restricting the supported operational modes. But to reduce complexity, it is important to know where complexity appears. The SECC provides a structure that can be used to identify a broad range of complexities.

To use the SECC, practitioners should step through the diagram with respect to a situation or system they are involved in. They should identify activities (boxes on the chart) where their situation or system is more complex than normal (at possibly two levels: a stretch, and far worse than anything their organization has done to date). The complexities should be examined to identify risks, which should be analyzed for likelihood and consequence and mitigated so far as resources allow. The worst complexities should be reviewed periodically to ensure the risks are being worked down.

Summary

Systems engineering is a broad process, ranging from early analysis through parts lists and test results. The breadth can be shown schematically by following activity paths that move among entities. The external world entity owns the larger system into which the system must fit; it also owns the stakeholders and their resources that can be brought to bear upon a problem by establishing a project that will create a technological system. The concept of human cognition is represented in a fourth entity that helps capture other aspects of complexity associated with the system, project, and environment. Complexity can apply differently to all these aspects of systems engineering.

Displaying all the entities on one chart shows how various complexities from a broad range of the literature fit into the systems engineering context. Ultimately, types of complexity that are shown by research to be predictive of project or system outcomes can be identified as early indicators of project and system success (Sheard, 2012). Other useful research would include determining where information complexity fits within the Systems Engineering Complexity Contexts diagram and how to tell whether the information is complex.

Acknowledgements

The author acknowledges with appreciation the contribution of INCOSE reviewers who helped improve the paper significantly with their detailed comments and of Erin Harper, an excellent technical editor.

References

Britcher, R. N. 1998. “Why some large computer projects fail.” In R. L. Glass (Ed.), *Software Runaways* (pp. 65-86). Upper Saddle River, NJ: Prentice-Hall, Inc.

Calvano, C. N., & John, P. 2004. “Systems engineering in an age of complexity.” *Syst. Eng*., 7(1), 25-34.

Hall, A. D. (1962). A Methodology for Systems Engineering: Van Nostrand Reinhold.

Kurtz, C. F., & Snowden, D. J. 2003. “The new dynamics of strategy: sense-making in a complex and complicated world.” *IBM Systems Journal*, 42(3), 22. doi: 0018-8670/03

Maier, M. 2007. “Dimensions of complexity other than ‘complexity’”, Paper presented at the Symposium on Complex Systems Engineering, Santa Monica, CA (US), 11–12 January.

Sheard, S. A. 1996. “Twelve systems engineering roles.” Paper presented at the sixth annual international symposium of INCOSE, Boston MA (US), 7-11 July.

——, 2000. “Three types of systems engineering implementation.” Paper presented at the tenth annual international symposium of INCOSE, Minneapolis MN (US), 16-20 July.

——, 2012. “Assessing the impact of complexity attributes on system development project outcomes.” (Ph. D.), Stevens Institute of Technology (Hoboken, NJ, US).

Sheard, S. A., & Mostashari, A. 2010. A complexity typology for systems engineering. Paper presented at the twentieth annual international symposium of INCOSE, Chicago IL (US), 11-15 July.

——, 2011. “Complexity in large-scale technical project management.” *International Journal of Complexity in Leadership and Management*, 1(3), 289-300. doi: 10.1504/IJCLM.2011.042550

Stevens, R., Brook, P., Jackson, K., & Arnold, S. 1998. *Systems engineering: coping with complexity*. Hertfordshire: Prentice Hall.

White, S. M. 2005. “Improving the system/software engineering interface for complex system development.” Paper presented at the 12th IEEE International Conference and Workshops on the Engineering of Computer-Based Systems, Greenbelt Maryland, 4-7 April.

Young, L. Z., Farr, J. V., & Valerdi, R. 2010. “The role of complexities in systems engineering cost estimating processes.” Paper presented at the conference on systems engineering research , Hoboken NJ (US), 17-19 March.

Biography

Sarah Sheard, Fellow of INCOSE and of the Lean Systems Society, earned INCOSE's 2002 Founder's Award and the CSEP certification. Dr. Sheard, a member of INCOSE since 1992, has served in both technical and administrative leadership roles. The most famous of her 40+ papers are *Twelve Systems Engineering Roles*, the *Frameworks Quagmire* and *Principles of Complex Systems for Systems Engineering*.

At the Software Engineering Institute of Carnegie Mellon University, Dr. Sheard researches software engineering process and measurement and brings software engineering tools and technologies to government clients. Previously she was a consultant and teacher at Third Millennium Systems and at the Systems and Software Consortium, and a systems engineer at Loral/IBM Federal Systems and Hughes Aircraft Company.

Dr. Sheard has a Ph.D. in Enterprise Systems at the Stevens Institute of Technology, a master's degree from the California Institute of Technology, and a bachelor's degree from the University of Rochester.