Chapter 45
Information Entropy-Based Complexity Measurement for Systems Engineering and Trade-Off Analysis

Jean Charles Domerçant

45.1 Introduction

Over time, engineered systems have become more interdependent and interconnected due to technological advances in areas such as networking, embedded computing, and communications. While there are many technical and nontechnical factors that influence system design and development, complexity is often blamed for a host of problems including failed system development, cost and schedule overruns, unmaintainable and unrepairable systems, and obsolescence upon delivery [1]. Complexity is not inherently a bad property and helps meet evolving capability needs as more becomes demanded of systems.

While complexity adds value, affordability and complexity are two conflicting objectives that require compromise. Therefore, only value-added complexity must be incorporated into the system architecture and design. The impact of early architecting and design decisions ripple through the entire system life cycle, affecting downstream activities such as manufacture, maintenance, and operations. In some industries, 75–85% of life cycle production costs are determined during *conceptual design* [2]. Conceptual design is the first phase in the design process, where the least knowledge exists but also where decisions that are made have the greatest consequence on cost, schedule, and performance. With this in mind, it is imperative to manage complexity early during this phase of design.

Before system complexity can be managed, it must first be measured. However, a standard measure of complexity for systems engineering is not currently well defined. A recent, comprehensive study of complexity measures reaches the conclusion that "Most conceptions of complexity measures in the theoretical literature could not be applied to systems engineering easily, and many concepts of complexity within engineering do not have a close tie to theory" [3]. The goal of this research

J. C. Domerçant (✉)
Georgia Tech Research Institute, Atlanta, GA, USA
e-mail: jean.domercant@gtri.gatech.edu

S. Adams et al. (eds.), *Systems Engineering in Context*,
https://doi.org/10.1007/978-3-030-00114-8_45

is to develop a standard method of measuring complexity that addresses these shortcomings, enabling more informed trade-offs to meet design objectives.

45.2 Background: Causes of Complexity in Engineered Systems

The first step in defining a measure is understanding the causes of complexity in engineered systems. A system is considered an entity composed of separate yet interrelated system *nodes*. A node may refer to a subsystem or even an entire system that is part of a larger family or system of systems, depending on the scope. Nodes interact through the sharing and processing of resources such as information, energy, materials, and time to perform functions [4] and contribute to the overall system's behaviors, states, and processes. A complex system consists of interconnected parts that as a whole exhibit one or more properties (behaviors among the possible properties) not obvious from the properties of the individual parts [5]. Examples of non-obvious behavior include emergent and adaptive behavior, self-organization, and the difficult-to-predict interactions between elements of a system.

Many of today's systems have evolved from simpler arrangements into the more complex counterparts we see today. A prime example is avionics, which serve as the computing infrastructure aboard military and commercial aircraft. Avionics are responsible for aiding in navigation, flight control, and information display. Two to three decades ago, systems such as these were generally stand-alone, discrete systems or subsystems. Their function could be easily understood and their scope or purpose was very limited [1, 6]. Over time, information sharing between various components increased to reduce the number of black boxes required by each system. For example, "a single sensor providing heading and rate information could provide data to the navigation system, the weapons system, and the pilot's display system." The advent of digital technology and increased computational capability led to the following changes that also caused an increase in complexity [7]:

- Dramatic increases in functional integration
- Added functionality to meet expanding capability roles
- Increased performance requirements
- Increase in the number of subsystems
- Increased functional overlap/redundancies between subsystems and a blurring of functional boundaries
- Increase in the total amount of information/resources processed by the system
- Increased sharing of information/resource among existing subsystems
- Increased physical integration to meet more demanding size, weight, power, and other constraints

As a result, avionics have seen an increase in "performance, sensor types, functionality, cost, integration, complexity, supportability (reuse), software

Table 45.1 Six types of complexity [1, 3]

Complexity type	Example subtype
1. Structural	Size: number and types of elements, size of development process, total number of requirements
2. Structural	Connectivity: number, types, density, and strength of connections, connectivity of development process
2. Structural	Architecture: patterns, chunkiness of connections, inhomogeneity, boundaries
3. Dynamic	Short term: sudden rapid change in system behavior, development system behavior
4. Dynamic	Long term: changes in number and types of things and relationships, evolution of purpose
5. Sociopolitical	Social and political: human cognitive limitations, multiple stakeholders, global context, environmental sustainability, economics, "coopetition," supplier chain depth, distributed development

programs in terms of executable code, memory requirements, throughput, reliability, data handling, data links, and obsolescence" [6]. Simultaneously, there has been a reduction in size, weight, power consumption, and technology windows [6]. This is a pattern that applies to many other types of systems as well. The Defense Advanced Research Projects Agency (DARPA) has recently tried to address the significant growth in development time and cost with increasing complexity for aerospace defense systems [8, 25].

A detailed study of the complexity literature and existing complexity measures resulted in a framework for categorizing complexity types and their impact on system development efforts [1, 3]. A brief summary is provided in Table 45.1.

It is possible to draw lines of cause and effect between the previously mentioned architecture and design changes witnessed in avionics to the many complexity types listed in Table 45.1. Additionally, a complementary view on the causes of complexity in engineered systems is based on a mapping of interactions vs. coupling [9]. Systems such as aircraft (with their onboard avionics) possess both complex interactions and tight coupling. Even here, "the degree of coupling and interaction types have been inferred from a rough idea of the frequency of system accidents in the various systems, rather than derived from analysis of the properties of the systems independent of the nature of their failures" [9]. This further emphasizes the need for an objective quantification of independent, observable system properties that contribute to overall system complexity.

45.3 Technical Approach

45.3.1 Defining the Context

The next step is to understand the context in which systems are architected and designed. For military acquisition, this begins with the systems engineering process (SEP). The SEP transforms the customer's stated needs and requirements "into a set of system product and process descriptions" [10] while generating information for decision-makers. Similarly, a generalized, domain-based mapping of the design process has also been developed that can be applied to many different fields—software, hardware, systems, materials, organizations, and manufacturing systems [11]. The following is a description of the various domains:

- Customer domain: characterized by the attributes (CA) that the customer is looking for in a product or process or system or materials or organizations.
- Functional domain: the customer needs are specified in terms of *functional requirements* (FRs) and *constraints* (Cs). FRs are the minimum set of independent requirements that completely characterize the design objective based on CAs.
- Physical domain: in order to satisfy the specified FRs, *design parameters* (DPs) are conceived in the physical domain.
- Process domain: finally, to produce the product specified in terms of DPs, a process is developed that is characterized by *process variables* (PVs) in the process domain.

During the design process, architectures are generated to better describe and understand the system [12]. An architecture is defined as the structure of components, their relationships, and the principles and guidelines governing their design evolution over time [13]. The following is a generalization of the different architecture views available:

- Functional architecture details the complete set of functions to be performed and their sequence; it identifies and structures the allocated functional and performance requirements [11].
- Physical architecture details how the system is physically divided into subsystems and components [11].
- System architecture identifies all the products necessary to support the system [11].
- Data architecture defines the structure and meaning of data to ensure consistency and proper management; it also defines the approach toward the structure, semantics, redundancy, and storage of data [14].

In contrast to architecting, design is a decision-making process intended to produce technically feasible and economically viable solutions. Each solution represents an integration of system elements under both logical and physical constraints and is often a compromise of competing attributes and objectives according to the

level of technology present at the time. Within this context, the overall approach to measuring complexity will be deemed useful if the causes of complexity can be captured at both the architecture and design levels of abstraction.

45.3.2 Uncertainty, Entropy, and Information Theory

Defining an absolute measure of complexity is not the goal of this research. The diversity of both natural and engineered systems makes defining an absolute measure of complexity difficult at best [15, 16]. Also, many existing complexity measures tend to be very domain specific or too theoretically abstract to usefully apply to real-world systems [15–17]. This research focuses on enabling informed trade-offs during the design process by developing a method to characterize system complexity as it relates to the functions, resources, and interactions between nodes. Information measures prove useful in this regard, as they "provide a precise method of dealing with trade-offs between knowledge and ignorance, and they supply a useful definition of complexity" [18]. In particular, *information entropy* [19] provides a statistical measure of information that relates the fundamental concepts of uncertainty, probability, and entropy, where *entropy* is a measure of unpredictability or the degree of randomness of a thermodynamic system [20]. Equation (45.1) is the mathematical formulation of the *information entropy, H*:

$$H = -K \sum_i p_i \log p_i \tag{45.1}$$

Information entropy "takes the concept of entropy out of the restricted thermodynamic setting in which it arose historically and lifts it to the higher domain of general probability theory" [20]. Maximum entropy occurs when all states are equiprobable. Furthermore, Eq. (45.1) has unique properties that qualify it formally as an information measure, including nonnegativity, symmetry, accumulation, and convexity [18]. The choice of a logarithmic base is arbitrary, as well as the value of the proportionality constant, K. When K is one and the base 2 logarithm is used, the unit of entropy is called a bit [19]. Therefore, a system that can be in two equiprobable states contains one bit of information. Information entropy provides a common basis of measurement across both the architecture and design levels of abstraction, making trade-off analysis easier. The challenge then lies in using information entropy, which is a measure of uncertainty, in a way that captures the fundamental interactions and relationships among nodes [2].

45.4 Architecture Complexity

In order to capture the different sources of system complexity at the architecture level of abstraction, a measurement framework [4] is used that characterizes an architecture according to two principal domains. The identified domains are the *functional domain* and the *resource domain*. Within each domain, there exists a *state* complexity measure, as well as a *processing* complexity measure. Each measure is defined as follows:

- Functional domain:
  - *Functional state complexity (FSC):* The allowable variation in either functional requirements or function outputs that determines the number of distinct functional states the system can inhabit and that must be accounted for.
  - *Functional processing complexity (FPC):* The accounting of all the potential independent process sequence paths that result during the execution of system functions.
- Resource domain:
  - *Resource state complexity (RSC)*: The allowable variation in resource properties that determines the number of distinct values or states these resources inhabit and that must be accounted for. Data/information, energy, time, and materials are examples of different categories or types of system resources that are exchanged and processed by the system.
  - *Resource processing complexity (RPC)*: A measure of the capacity to share, communicate, and process resources between system components.

Measurement within each domain makes use of the tools available to the architect or designer during the conceptual design phase. The following sections illustrate how this is accomplished.

45.4.1 Functional State Complexity

The role of a system function, as in mathematics, is the transformation of an input to a usable output. The functional architecture provides a description of the system in terms of what it does logically and in terms of the performance required when functions are executed [10]. A system will occupy certain functional states at any given time, based on if or how each function transforms given the resources. For example, navigation is a common avionics function where the system relies on information from sensors to determine the position and velocity of the center of mass of the aircraft. If the avionics were only able to compute a single value pair for position and velocity given a range of inputs, then the function would be a constant transform and thus highly predictable and simple. If, however, the position and

velocity output are highly nonlinear and exhibit a wide range of values, this would lead to increased functional complexity, as there would be more distinct possible functional states describing the system.

It cannot be assumed that system functions are well defined during the conceptual design phase since the system components that will carry out these functions may not have been assigned or even exist. Thus, another method of specifying the number of possible states for a given function must be used. With this in mind, FSC is measured using the performance requirements defined in the functional architecture. FSC is calculated using Eq. (45.2) for each performance requirement:

$$\mathrm{FSC} = \sum_i log_2 \frac{|\, b_i - a_i \,|}{r_i} \tag{45.2}$$

The values of b_i and a_i in Eq. (45.1) represent the bounds of different subranges or "bins" within the overall *range* of a single performance requirement. This formulation assumes there is no constraint that bins within a performance requirement must be of uniform size or resolution. Meanwhile, r_i represents the associated *resolution* for each bin or the smallest significant/measurable interval. Further increases in resolution mean comparatively more fine-grained measurements are necessary in order to differentiate between significant functional states, adding to complexity. Intuitively, this matches the effect observed in chaotic systems, where the sensitivity of the parameters plays a key role. Table 45.2 provides a summary of example FSC calculations for a sample avionics terminal electrical requirement [7].

Table 45.2 illustrates how trade-offs in range and resolution of performance requirements affect architecture complexity. Also, the more functions added to the functional architecture, the greater the complexity. The FSC for each performance requirement is then summed to determine the total number of possible functional states.

Table 45.2 Functional state complexity for an avionics performance requirement

Function: changes in input level (transformer coupled)	Range (volts)	Resolution (volts)	Number of function states	FSC (bits)
1. Base requirement	**0.86–14.00**	**0.01**	**1314**	**10.36**
2. Increase in range	0.86–20.00	0.01	1914	10.90
3. Decrease in range	4.00–14.00	0.01	600	9.23
4. Increase in resolution	0.860–14.000	0.001	13,140	13.68
5. Decrease in resolution	0.9–14.0	0.1	131	7.03
6. Increase in both range and resolution	0.860–20.000	0.001	19,140	14.22

The bold values represent baseline values for comparison against increases or decreases in range and/or resolution

Table 45.3 Resource state complexity for various types of resources

Resource type	Range	Resolution	Number of resource states	RSC (bits)
A. Avionics bus Manchester II bit encoding (20-bit command word)	–	–	1,048,576	20.00
B1. Cold, dry cooling air (Mass flow rate – cubic feet per minute)	50–75	1	25	4.64
B2. Cold, dry cooling air (Temperature – °F)	40–60	1	20	4.32
B3. Cold, dry cooling air (Humidity – %)	6.00–10.00	0.01	400	8.64

45.4.2 Resource State Complexity

RSC is measured in a similar fashion to FSC; only now Eq. (45.2) is applied to a resource rather than a performance requirement. The RSC for the entire system is calculated through a simple summation, and RSC increases as more functions and therefore resources are included in the functional architecture. Table 45.3 provides sample RSC calculations for two different types of resources. Resource A is an avionics bus command word that specifies the function that a remote terminal is to perform [7]. Resource B is cold, dry air used to cool electronic components.

Table 45.3 also shows how RSC is determined via the decomposition of a resource into its fundamental physical properties. Digital resources, such as Resource A, are already encoded in bits, so there is no need to calculate the range and resolution.

45.4.3 Functional Processing Complexity

Systems are designed to execute their functionality within a programmatic sequence. The greater the number of possible program paths, the greater the complexity. For example, earlier versions of the Microsoft® Word program for word processing have been estimated to contain more than 2^{64}, or over 1.8×10 [19], separate program paths [2]. This large path size is caused by the presence of multiple feedback and feedforward loops, interdependent tasks that are coupled, and multiple branching points such as if-then statements or parallel pathways. Analyzing, testing, and maintaining software with this type of complexity are difficult. It is hard to trace the sequential program execution or to determine the impact of a change in one part of the program on the functioning of the rest of the program.

Software designers have long used control flow graphs (CFG) to illustrate functional sequences. A CFG is a graph-based visualization where nodes represent either a basic block of code or a branch point [21], and edges indicate the flow of program execution. Example CFG [2] can be seen in Fig. 45.1.

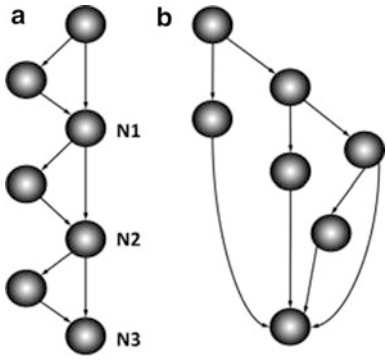

Fig. 45.1 (**a**) Less predictable control flow graph; (**b**) more predictable control flow graph

The CFG is a useful abstraction, and similar types of diagrams found in systems engineering include functional flow block diagrams [10], activity sequence diagrams, and task-based design structure matrices (DSMs) [22]. Any of these diagrams can be analyzed from a graph-theoretic perspective to determine the number of *acyclic paths* present. An acyclic path is defined as a path from program entry to exit that does not traverse an arc more than once. In Fig. 45.1, the left and right CFG possess the same number of nodes but different number of acyclic paths. The left CFG has eight acyclic paths, compared with four acyclic paths for the right CFG. The left CFG is therefore less predictable than the right CFG, and this is reflected in the *in-degree* of the nodes labeled N1, N2, and N3. In-degree is simply the number of edges entering a node. Nodes N1, N2, and N3 will be "the crossroad of more than one path and thus increase the uncertainty of the expected program behavior" [21].

FPC is determined by applying Eq. (45.1), which requires defining the probability or frequency of traversing each path. In the case that the probabilities are equiprobable, this means taking the logarithm of the number of acyclic paths. Under this assumption, the FPC of the left CFG is 3 bits, while the FPC of the right CFG is 2 bits. Various algorithms exist to automatically count acyclic paths for a given CFG, though care must be taken when features such as nesting or different types of conditional and iteration statements are present [21, 23].

45.4.4 Resource Processing Complexity

Nodes exchange resources with other nodes, systems, or the environment in order to function. At the most abstract level, this exchange of resources can be modeled as a communication, where messages originate at an information source, are encoded into a message using symbols, and then transmitted over a channel to a receiver which then decodes the message at the destination. This concept is applied to the exchange of resources between nodes, as different parts of the system effectively communicate via resources such as information, energy, materials, etc. in order for the system as a whole to function. More precisely, this communication is modeled as

a stochastic "*m-order*" Markov chain where the probability of emission of a particular symbol depends on the preceding *m* number of symbols [20].

The first step in calculating RPC is to identify the system interfaces, which define the boundaries at which dependencies and coupling between nodes occur. Next, symbols must be identified for the resources exchanged at these boundaries. Table 45.3, shown previously, specifies the number of possible states for each resource in the avionics example. An arbitrary symbol is assigned to each possible resource state, and Table 45.4 lists example word/messages that are formed when the arbitrary symbols chosen correspond to actual resource values.

A word/message sequence (or even its absence) is a symbolic representation of the current state of a collection of resources being exchanged at a snapshot in time. To calculate RPC, each word/message is assigned a frequency *p(i)*, or the probability of that specific word/message occurring during the resource exchange. Transition probabilities $p_i(j)$ are also assigned to indicate the likelihood of a next word/message occurring in the sequence. Diagram probabilities *p(i,j)* are then obtained with Eq. (45.3):

$$p(i,j) = p(i)p_i(j) \text{ where } \sum_j p_i(j) = \sum_i p(i) = \sum_{i,j} p(i,j) = 1 \tag{45.3}$$

RPC is then equal to the entropy of the *p(i,j)* matrix. Maximum RPC occurs for equiprobable *p(i,j)*, signifying independence among the resource exchanges and great uncertainty in discerning the exact sequence of resource states as they are exchanged over time. Any deviations from equiprobability mean that correlations and dependencies exist in the transmission and processing of resources, helping to drive down the processing complexity. RPC requires the system architect to specify the transmission frequency of each resource, as different resources may be transmitted at different intervals compared to other resources. For example, the command word could be transmitted at twice the frequency as cold, dry air, as seen in example D of Table 45.4. Possible noise sources should also be considered along with the various architectural mechanisms and patterns (such as redundancy, backup systems, check sums, etc.) needed to help ensure error-free transmission of resources.

Table 45.4 Resource word message examples

Example resource message sequence (word/message)	Command word value	Cold, dry air values (CFM, °F, % humidity)
A. 1,001,111,000,001,110,010,165,400,701	10,011,110,000,011,100,101	65/ 40/ 7.01
B. 1,111,111,000,001,110,111,152,510,805	11,111,110,000,011,101,111	52/ 51/ 8.05
C. 00001010110010100101 58530600	00001010110010100101	58/ 53/ 6.00
D. 00001010110010100101000010101100 1010010158530600	00001010110010100101	58/ 53/ 6.00

45.5 Design Complexity

45.5.1 Independence Axiom

In contrast to architecting, design is focused on integration, satisfying constraints, and ensuring the system is physically realizable. An axiomatic approach [11] to design seeks to define scientific foundations for the design of complex systems. The first axiom, the *independence axiom*, states that "when there are two or more FRs, the design solution must be such that each of the FRs can be satisfied without affecting any of the other FRs. This means that we have to choose a correct set of DPs to be able to satisfy the FRs and maintain their independence" [11]. This is represented mathematically in Eq. (45.4) using a design matrix $\boldsymbol{A}$ that provides the mapping between FRs and DPs.

$$\begin{Bmatrix} FR_1 \\ . \\ . \\ FR_m \end{Bmatrix} = \begin{bmatrix} A_{11} & 0 & \cdots & 0 \\ 0 & A_{22} & \ldots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \ldots & A_{mm} \end{bmatrix} \begin{Bmatrix} DP_1 \\ . \\ . \\ DP_m \end{Bmatrix} \tag{45.4}$$

where A_{ij} is a sensitivity coefficient and $A_{ij} = \delta FR_i/\delta DP_j$. Equation (45.4) represents an *uncoupled design* with a square design matrix (where there are m number of FRs and m number of DPs) with values only along the diagonals. A design that violates axiom 1 is either a *decoupled* design or a *coupled design*. A decoupled design has a square design matrix, but there is coupling between DPs due to nonzero off-diagonal elements. Finally, a coupled design has a rectangular matrix where p number of DP components exist with $m > p$. Axiom 1 can be maintained in a decoupled design when the DPs are controlled in some sequence [11, 24]. Equation (45.7) [24] provides an entropy-based formulation of the first axiom, assuming $\boldsymbol{A}$ is a square and nonsingular constant matrix:

$$\boldsymbol{h}(\boldsymbol{f}\{\boldsymbol{FR}\}) = \boldsymbol{h}\big(f_{DP}(\{\boldsymbol{DP}\}) + log_b|[\boldsymbol{A}]| \tag{45.7}$$

where $|[\boldsymbol{A}]|$ is the determinant of the design matrix $\boldsymbol{A}$. The variation inherent in the FRs, DPs, and PVs are typically continuous random variables [24]. If normal probability distribution functions (pdfs) are assumed with parameters (μ, σ^2), and the correlation between DPs is expressed as ρ, the total design complexity is captured by Eq. (45.8), assuming a natural logarithmic base:

$$\boldsymbol{h}(\phi\{\boldsymbol{DP}\}) = \sum_{l=1}^{p-1} \sum_{k=l+1}^{p} \ln\left(2\pi e\sqrt{(1-\rho_{kl}^2)\sigma_l\sigma_k}\right) + \ln\left((2\pi e)^p \prod_{l=1}^{p} \sigma_l^2\right)^{\frac{1}{2}} + \ln|[\boldsymbol{A}]| \tag{45.8}$$

The following components of complexity are captured [24]:

- Variability: the measured entropy of the DPs, captured by the ($\boldsymbol{h}(f_{DP}(\{\boldsymbol{DP}\})$ term in Eq. (45.7) and the normal information sources, $\ln\left((2\pi e)^p \prod_{l=1}^{p} \sigma_l^2\right)^{\frac{1}{2}}$ in Eq. (45.8).
- Vulnerability: three factors related to the design matrix $\boldsymbol{A}$ influence the vulnerability, mapping, sensitivity, and dimension. Mapping is the topological structure of $\boldsymbol{A}$ corresponding to the position of the nonzero sensitivity coefficients $A_{ij} = \delta FR_i/\delta DP_j$. Sensitivity refers to the magnitude and sign of nonzero A_{ij}. Dimension refers to the size of the design problem itself, i.e., the number of the FRs, m. Vulnerability is associated with the $log_b|[\boldsymbol{A}]|$ in Eq. (45.7) and $ln|[\boldsymbol{A}]|$ in Eq. (45.8).
- Correlation: expressed in terms of a jointly distributed pdf between the correlated design variables or via a covariance matrix. Correlation (ρ_{kl} in Eq. (45.8)) is a causal relationship between DPs via some noise factors (e.g., common manufacturing variations).

Thus, the three components are separable and distinguishable from each other. This permits devising strategies for complexity reduction based on the component's significance. However, this might not be the case when the design is highly nonlinear with inseparable components due to the loss of additivity. Consequently, significant components of complexity may not be easily identified [24].

45.5.2 Information Axiom

The second axiom, the information axiom, states that the design with the highest probability of success is the best design [11], even though many different designs may be acceptable in terms of the independence axiom. In the general case of m FRs, the information content for the entire system is:

$$I_{\text{sys}} = -\log_b P_{\{m\}} \tag{45.9}$$

where $P_{\{m\}}$ is the joint probability that all m FRs are satisfied. The probability of success can be computed by specifying the design range for the FR and by determining the system range that the proposed design can provide to satisfy the FR. The area of overlap between the design range and system range indicates the probability of success.

45.6 Results and Conclusions

The complexity measures previously discussed directly address the list of the causes of increased complexity in systems based on the avionics system case study. This is illustrated in Table 45.5.

The aim of this research is to provide a useful measure of complexity for systems engineering that aids trade-off analysis. The approach taken is to identify observable system properties that lead to increased complexity. As part of the approach, notable causes of increased complexity are identified using avionics as a prime example. There is little doubt that systems such as avionics will continue to grow in

Table 45.5 System complexity factors captured by the architecture and design complexity measures

Measure	Complexity factor/trend/sensitive to	Required inputs
Functional state complexity (FSC)	Added functionality/subsystems, functional redundancy, and change in performance requirements	Functional architecture: defined set of atomic-level functions and performance requirements including range and resolution
Functional processing complexity (FPC)	Feedback and feedforward (process execution iterations), coupled tasks, and branching due to different factors: functional overlap, redundancies, function integration, and resource sharing	Functional flow block diagram, control flow graphs/program logic flow (software), design structure matrix, activity sequence diagram
Resource state complexity (RSC)	Added functionality/subsystems, changes in the amount and types (analog signal, digital signal, mechanical, energy, etc.) of resources being shared	Resource architecture: defined set of function resources including range and resolution
Resource processing complexity (RPC)	Increase in the amount of information/resources processed by the system, increased information and sharing among subsystems, dramatic increases in functional integration, increase in the number of subsystems, modularization and coupling and cohesion	Logical/objective architecture: function assignment to logical components and subsystems/modularization/logical interface boundaries, e.g., DSM with clustering applied, noise and loss functions; resource/data architecture: mutual information and resource coupling, redundancy, relative transmission frequencies, communication protocols
Axiomatic engineering design complexity	Increases in physical integration to meet more demanding size, weight, power, and other constraints; the size of the design problem along with coupling and correlation between design parameters; difficulty in meeting design goals and the impact of technology on probability of design success	Functional requirements, design parameters, system range, sensitivity of design parameters in meeting functional requirements, correlation between design parameters

complexity, and an objective measure of complexity provides the basis for evaluating different architectural patterns and design decisions. Future work will focus on applying the developed complexity measures to an avionics case study in order to further develop and assess relevant analyses and their correlation to emergent behavior, difficulty in modeling and simulation, and not to mention cost, schedule, and performance during acquisition. Ultimately, this should result in a useful trade-off environment for systems engineering and aid in the timely and affordable acquisition of systems moving forward.

References

1. Sheard, S. A., & Mostashari, A. (2010). 7.3. 1 A complexity typology for systems engineering. In *INCOSE International Symposium* (Vol. 20, No. 1, pp. 933–945).
2. Minai, A., & Braha, D. (2006). *Complex engineered systems: Science meets technology*.
3. Sheard, S. A., & Mostashari, A. (2013). 5.2. 2 Complexity measures to predict system development project outcomes. In *INCOSE International Symposium* (Vol. 23. No. 1).
4. Domercant, J. C. (2011). *ARC-VM: An architecture real options complexity-based valuation methodology for military systems-of-systems acquisitions*. Atlanta: Georgia Institute of Technology.
5. Joslyn, C., & Luis, R. (2000). Towards semiotic agent-based models of socio-technical organizations. In *Proc. AI, Simulation and Planning in High Autonomy Systems (AIS 2000) Conference, Tucson, Arizona*.
6. Moir, I., Seabridge, A. G., & Jukes, M. (2006). *Military avionics systems*. Hoboken, NJ: John Wiley & Sons.
7. Spitzer, C. R. (2001). *Avionics Handbook* (Vol. 200, p. 158). Boca Raton: CRC Press.
8. Eremenko, P. (2009). *META novel methods for design & verification of complex systems DARPA presentation, December* 22 (p. 10).
9. Perrow, C. (2011). *Normal accidents: Living with high risk technologies*. Princeton, NJ: Princeton University Press.
10. Lightsey, B. (2001). *Systems engineering fundamentals*. Fort Belvoir, VA: Defense Acquisition University.
11. Suh, N. P. (2005). *Complexity: Theory and applications*. Oxford: Oxford University Press on Demand.
12. Maier, M. W. (2009). *The art of systems architecting*. Boca Raton, FL: CRC Press.
13. IEEE Standard 1471 (IEEE 1471, 2000; Maier, Emery, & Hilliard, 2001).
14. Jamshidi, M. (2005). System-of-systems engineering-a definition. In *IEEE SMC* 2005 (pp. 10–12).
15. Kinsner, W. (2008). Complexity and its measures in cognitive and other complex systems. In *ICCI 2008. 7th IEEE International Conference on Cognitive Informatics*. Piscataway, NJ: IEEE.
16. Mitchell, M. (2009). *Complexity: A guided tour*. Oxford: Oxford University Press.
17. Alderson, D. L., & Doyle, J. C. (2010). Contrasting views of complexity and their implications for network-centric infrastructures. *IEEE Transactions on Systems, Man, and Cybernetics-Part A: Systems and Humans, 40*(4), 839–852.
18. Gell-Mann, M., & Lloyd, S. (1996). Information measures, effective complexity, and total information. *Complexity, 2*(1), 44–52.
19. Shannon, C. E. (1948). A mathematical theory of communication. *Bell System Technical Journal, 27*(3), 379–423.
20. Gatlin, L. L. (1972). *Information theory and the living system*.

21. Nejmeh, B. A. (1988). NPATH: A measure of execution path complexity and its applications. *Communications of the ACM, 31*(2), 188–200.
22. Steward, D. V. (1981). The design structure system: A method for managing the design of complex systems. *IEEE Transactions on Engineering Management, 3*, 71–74.
23. McCabe, T. J. (1976). A complexity measure. *IEEE Transactions on Software Engineering, 4*, 308–320.
24. El-Haik, B., & Yang, K. (1999). The components of complexity in engineering design. *IIE Transactions, 31*(10), 925–934.
25. Stuart, D., & Mattikalli, R. (2011). *META II complexity and adaptability*. St Louis, MO: Boeing Co.