26th Annual INCOSE International Symposium (IS 2016)
Edinburgh, Scotland, UK, July 18-21, 2016

Emergent Failure Modes and What to Do about Them

Steven D. Harris
Rational LLC
Contact Information
252-339-3423
sharris@rationalblue.com

Jennifer M. Narkevicius
Jenius LLC
Contact Information
301-904-3631
jnarkevicius@jeniussolutions.com

Abstract. This paper sets out to identify a means to mitigate the effects of unpredictable failures in complex systems. It outlines a formal analysis of complex systems that focuses on emergent system dynamics, some of which may be failure modes that are impossible to predict. The mathematical basis for the analysis, and some real-world implications of the mathematics are introduced. From that same formal analysis, the paper identifies specific characteristics of system architecture to mitigate the risk of emergent failure modes. The paper concludes with an approach to mitigating the effects of EFM predicated on a principle-based process architecture called ***intelligent control theory***.

Introduction

System failures can be highly visible reminders that much remains to be done to achieve strong system design and implementation of complex technology. This section starts with a brief look at several highly visible systems failures, and introduces emergent failure modes (EFM) as a way to frame analysis of those failures. The paper subsequently reviews principles that form the basis for its conclusions.

On 15 August 2015, a significant outage occurred at Leesburg Air Route Traffic Control Center (ZDC), resulting in some 400 commercial flights being canceled in the northeast sector, with impact on traffic at other airports as far away as Dallas and Los Angeles (Ortiz, 2015). The proximal cause of that outage appears to have been an error during a planned software maintenance event. From a commonly held perspective, this catastrophic failure event may be attributable to a conventional software glitch. The approach that evolved in real-time led to a decision to disable the updated functions until a solution could be developed. Subsequent analysis concluded that the system failure during the ZDC upgrade was a function of an unanticipated set of interactions between the software and the user community. It is highly unlikely that the exact conditions of the incident could have been fully anticipated prior to the deployment. An analogous problem emerged recently, when Microsoft introduced an artificial intelligence (AI) "chatbot" system designed to conduct discourse with humans over Twitter (Dobuzinskis, 2016). The product was disabled after only one day of service because it began to veer into vulgar and racist topics.

Preliminary indications from the these incidents suggest that additional testing of software may have been required prior to deployment. It is seductive to treat these and similar events as the result of poor management, however, the details of the analysis below should serve to make the point that no amount of testing can guarantee that a complex system will not fail. This is especially true for testing under conditions that can only be generated with large numbers of humans in the loop. Unfortunately, these examples do not reflect new, or even unique problems. Specific operational examples of interactions in complex systems that result in catastrophic outcomes are readily found in other domains, such as the March 28, 1979, Three Mile Island (TMI) accident in which human operator error interacted with system failures to convert a complex but routine maintenance activity into a lingering national nightmare. A combination of equipment malfunctions, design-related problems and worker errors led to a partial meltdown and very small off-site releases of radioactivity at TMI-2 (Nuclear Regulatory Commission, 2013). Both Space Shuttle accidents, Challenger (Committee on Science and Technology, 1986) and Columbia (Columbia Accident Investigation Board, 2003), were caused by known systemic issues. Neither was a complete surprise to human decision makers. Even as Columbia was breaking up in the upper atmosphere, engineers were discussing the icing incident that occurred at launch, more than two weeks earlier, as the most likely (and ultimately acknowledged) proximal cause.

The partial power outage during Super Bowl XLVII, in New Orleans in 2013, is yet another example: everything worked as advertised, but when the lights went out, 70,000 people were in immediate jeopardy as over 108 million people watched on television. Outages of electric power systems happen locally, occasionally on a national and even international scale. During an outage in 2003, the populous US northeast and the entire Canadian province of Ontario went dark when a software glitch combined with a routine maintenance issue, and the problem propagated rapidly across the grid.

Conventional wisdom is to treat each of these incidents as low probability events that could or should have been foreseen by system operators, designers and acquisition program managers. Often *ex post facto* punitive actions are taken against individuals involved, and/or higher echelon decision-makers. But unforeseen and unforeseeable emergent failure modes (EFM) in complex systems may be responsible for these accidents and incidents. Further, this paper shows that there are specific things the systems engineering (SE) process can do to mitigate the risk and effects of EFM.

Emergent Behavior in Complex Systems

Emergent behavior (EB) is ubiquitous and unavoidable. EB is the evident and inevitable global result of local interactions. Weather is emergent, as are conversation, driving, and molecular gastronomy among other things. These EB are beneficial and in systems may be touted as features. However, EB can also bring forth detrimental outcomes resulting in incidents, accidents and other disruptive effects. The ubiquity of EB in complex systems brings its own complexity. The networking of systems, as well as the development of systems of systems, generates a rich environment from which EB arise. Some of the resulting EB may be failure modes -- EFM. EFM may take a variety of forms; they may put the system into an unknown, perhaps unrecoverable state, or they may allow routine technical operations to cascade to failures. While these failures will appear to stem from hardware or software root causes, their emergent nature is the aspect that leads to catastrophic failure.

Large-scale infrastructure components are not intentionally designed to be unstable or to be vulnerable to catastrophic failure, yet, catastrophic failures do occur. Such failures may not become apparent in design, development, test and evaluation, or even for some time after initial operational capability. Risk of low probability events with high negative consequence is often mitigated during acquisition programs through reliance on favorable probabilities. For example, Failure Modes and Effects Analysis (FMEA), a core tool in the systems engineer’s kit, is predicated on estimating probabilities in so-called “forward logic.” That is, one goal of

FMEA is to ensure that the likelihood of an adverse event is estimated to be low enough to rationalize devoting minimal resources to counteract or mitigate it.

One might argue that loosely coupled Service Oriented Architecture (SOA) of many current-generation and future infrastructure systems – the Systems of Systems (SoS) -- are completely different from the highly integrated tightly coupled systems, such as the Space Shuttle. Nevertheless, in the SOA/SoS context, interactions and connections within an Enterprise Architecture (EA) are no less real. They are simply less visible.

Indeed, it is useful to consider that a taxonomy of complex systems might vary along a dimension of how highly integrated they are, or the extent to which they depend on physical, as opposed to logical connectivity among critical components. We argue here that it does not matter for purposes of systems engineering. All complex systems, no matter their architecture or governance model, are vulnerable to EFM. Systems that are realized by logical connections, i.e., all current- and future-generation systems of any significance, are most vulnerable.

An important distinction can be made between legacy, point-to-point systems architectures and the SOA/SoS approach. Complex legacy systems (e.g. the current-generation National Airspace System) are much more rigid than SOA systems because components are much more tightly coupled by the point-to-point character of their physical interfaces. The very important advantage of a SOA is its inherent flexibility. That flexibility is realized precisely because of the loose, flexible, logic-based (as opposed to physical) connection among components. New interfaces can be defined in software and data structures, rather than requiring the laying of copper or fiber in a conduit. It is relevant to recall that the origin of the terms hard-ware and soft-ware was based in the perceived advantageous flexibility of the new medium for controlling computation. The term firm-ware emerged as a further refinement of that basic notion of flexibility.

Nevertheless, that inherent flexibility has turned out to be a two-edged sword. The proportion of acquisition costs attributable to software (i.e., logical connections) is likely to increase as computing hardware costs continue to decline. Software is also much more likely to result in systemic failures than would the failure of a particular hardware component. Consider that while legacy point-to-point systems have failed, the rate of propagation of failure is often sufficiently slow to allow for real-time response to limit damage. An SOA system such as the emerging electric power Smart Grid, on the other hand, can fail catastrophically, and such failures can propagate across continental boundaries at nearly the speed of light (Harris and Narkevicius, 2014).

An SOA will exhibit interactions that derive entirely from invisible, and often undocumented logic embodied in the software, rather than readily visible (and maintainable) physical connectivity. This problem is particularly relevant because of the logically integrated character of any SOA-type system. The broader enterprise architecture entails significant risks, many known, some unknown, and some unknowable until they materialize in the deployed context. It can be stated, with near mathematical certainty, that complex systems-of-systems will exhibit failure modes that emerge from the architecture of loosely coupled networks. It can further be stated, with absolute mathematical certainty, that complete specification of all such failure modes is impossible. When failures occur in complex networks, rapid, accurate assessment of the failure, its genesis, its trajectory, its consequences, and its correction are essential. The challenge of maintaining system integrity requires a very different set of skills from those required to maintain legacy systems, in addition to new technical resources to collect the best data, synthesize those data, induce the nature of the failure, and implement corrective action(s).

Mathematical Basis

The mathematical basis for our strong assertions about EFM is organized around two key concepts: principles of emergent properties in complex systems and the central role of models and modeling in control systems and processes. These topics are addressed below. The math proves that no amount of engineering and governance can ensure that the emerging SOA-based architectures will never experience a catastrophic system-wide failure. Other solutions will be required to mitigate that risk. Analysis of the central role of models in the control process points to a practical system architecture to mitigate the effects of EFM.

It is generally recognized that complex systems cannot yet be fully represented and analyzed mathematically by simple partial differential equations or other analogous mathematical tools. This assertion may seem obvious, but the implications are important. While such tools are useful for characterizing some aspects of complex system behavior, they are not sufficient. What is needed is an appropriate mathematical formalism for representing such systems. Since we lack such formal tools, complex systems are analyzed through simulation. It is widely recognized within the simulation technology base, that the results of simulation depend heavily on the quality of the models and the information used to set up and run the simulations: i.e., "garbage in, garbage out". Indeed, the very notion of *validating* a simulation is no longer accepted practice in the defense industry, arguably one of the most mature of simulation technology bases. Current practice is to focus on accreditation. This appropriately nuanced approach is a pragmatic response to the fundamental limits of simulation.

Model-Based Engineering (MBE) reduces life cycle costs of major systems by depending heavily on simulation. There is no doubt that simulation is an important tool for systems engineering. The need for improved engineering methods and tools motivates the continuing desire for improved rigor in Modeling and Simulation (M&S) technology. Perhaps the most important recent advances in M&S are concepts and mathematical tools that bring a more rigorous framework for modeling and analyzing complex systems.

The appropriate mathematical construct for analyzing discrete simulations of dynamical systems is called a sequential dynamical system (SDS). SDS is the mathematical formalism that underlies all digital simulation. Several fundamental observations about complex systems can be stated with mathematical precision using the formalism of SDS. For present purposes, we focus on simple properties and observations about SDS (Harris and Narkevicius, 2014).

An SDS can be thought of as analogous to a movie, a sequence of discrete snapshots of system state, played back (computed) to reveal motion (dynamics). The point of watching a movie is not to see the details in each individual frame but to discern the entire story. In fact, human vision is incapable of perceiving each individual frame when the movie is running. Rather, the movie is carefully tuned to properties of human observers to facilitate integrating across the discrete frames to perceive motion. Motion is dynamics; change in state as a function of time. An SDS is a finite state machine that transitions from state to state in a sequence of discrete steps to reveal to an observer the dynamical properties of a system. Analysts can learn things about complex systems in the abstract by exercising SDS constructs. The fundamental properties of SDS are readily extrapolated to complex systems.

Graph Theory, SDS Constructs and Notation

A system is defined (by various authorities) as a collection of interacting elements whose interactions are organized around a common goal or principle (INCOSE, 2015). A network is generally defined in terms of a collection of interacting elements, representing the structure, without reference to the processes that govern behavior. SDS are, in effect, an extension of basic network graph theory to account for individual (local) behavior and provide a basis for examining and predicting global (systemic) behavior. The approach outlined here is derived from a basic text (Mortveit and Reidys, 2008). The purpose of the present treatment is to develop the mathematical basis sufficiently to derive implications and requirements for dealing with EFM. The approach is a logical and provable extension from the foundations, and, as such, it can be tested and evaluated in its own right.

A network is represented mathematically as a graph G = (V, E) comprising a set of vertices V = {v1, v2, …, vn} (also called nodes), and a set of links or edges E = {e1, e2, ..., en} connecting the vertices. The number of vertices n is called the order of the graph, and denoted |V|. The number of edges k is called the size of the graph, and denoted |E|. Two nodes connected by an edge are said to be neighbors.

Figure 1. Illustration of a network G = (V, E).

A network described by the notation G= (V,E) is illustrated in Figure 1. A graph such as depicted in the figure is a static structure. One may easily see that a complex system could be usefully represented as a (very complex) network structure, and that interesting observations might be made on the representation. However, it would be difficult (impossible, actually) to derive and predict system performance solely from such a static representation.

Much research on SDS is focused on formally representing the dynamical properties of systems such as illustrated in Figure 1. This representation indicates both how the system behaves, and how underlying graph structure influences the behavior. Measures of graph structure and behavior (dynamics) can be derived from the SDS formalism.

A particular collection of states of the nodes of the system is called a configuration (analogous to a single frame in a movie) of the system. The set of all configurations into which a system may transition is called the system's phase space (analogous to a particular editor's cut of a movie; other editors might include different frames for an entirely different movie). A phase space can be depicted as a directed network graph (a digraph). The phase space captures some (but not all) important aspects of the dynamical properties of the system it represents. For example, consider the phase space of a dynamical system, illustrated in Figure 2. A more complete treatment of this topic may be found in Mortveit and Reidys (2008).

Each node in the phase space graph (Figure 2) represents a configuration of the system, all of the states of all of the nodes in the graph, taken at a point of time (like a frame in a movie). So, for example, in a 3-node graph, the states of the three nodes might be represented by the node in the far left of Figure 2 as $\{n_1, n_2, n_3\} = \{0,0,1\}$. When the system (i.e., the complete set of nodes n_1, n_2 *and* n_3) transitions to a new state, the nodes might assume the values $\{0,1,1\}$, as illustrated in the node to which the arrow from the first node is pointing.

This figure illustrates all of the possible transitions for a particular 3-node network, where the individual transition functions f_1, f_2, *and* f_3, are defined explicitly, *and the updates to those states are calculated in a particular order*. (The issue of update order is discussed in greater detail below.) An important property of this particular system is that all transitions lead to a cycle. That is, irrespective of the beginning states, if this system is allowed to evolve according to its defined transition functions, updated in a particular order, it will eventually end up in a part of the phase space from which it cannot emerge – the cycle from {0,0,0} to {0,0,0}.

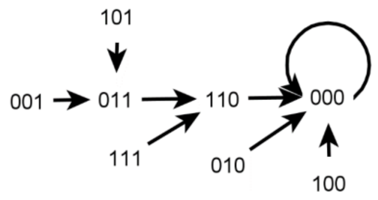

Figure 2. Phase space of a simple 3-node graph.

SDS is a formal extension of graph theory into the realm of system dynamics in an effort to support rigorous analysis of complex real-world networks using constructs that describe and elaborate transitions among states such as illustrated in Figure 2. SDS extends the graph notation introduced above by incorporating a formal definition of state change. An SDS is denoted as a triple (see next paragraph) to accommodate this additional parameter and to expose complex system behavior to formal representation and rigorous analysis. The figure is useful as a tool to visualize the dynamics of the system it represents. Patterns of transitions among configurations are readily evident, such as, the loop in dynamics of the system.

Formally, an SDS ***S*** over a given domain D of state values (i.e., states that can be assumed by nodes) is denoted as a triple *(**G**, F, π)*, where *G(V,E)* is an undirected graph and for each node *i* of *G, F* specifies a local transition function f_i. F, in short, is a set of transition functions *{f_1, f2, …, f_n}*. Those transition functions may be the same for all nodes (defining a homogeneous set of nodes), or they may be defined uniquely for each individual node, thereby characterizing complex systems comprising heterogeneous nodes. Transition functions provide mappings of current states of nodes to new states as a function of the states of neighboring nodes.

The parameter π in the triple *(**G**, F, π)* denotes the set of permutations of order of update for the nodes. Informally, the order of update is the particular sequence of updates of the states of nodes. This parameter turns out to be extremely important when analyzing discrete systems. The significance of update order π for interpreting the results of a simulation run become quite apparent in the analysis of elementary networks such as illustrated in Figure 3. The behavior of the four-node network in the upper left of **Error! Reference source not found.** is illustrative.

In this example, the four nodes of the ***Circ*$_4$** network of **Error! Reference source not found.**, denoted *1, 2, 3,* and *4,* all may assume only one of two states, denoted 0 and 1. They all behave according to the same transition function, when allowed to update, they assume the state defined by the logical *Nor* (*not-Or*) of the states of their neighbors. Therefore, node *1* will assume the state defined by *Nor(1,2,4),* that is, the *Nor* of the states of nodes *2* and *4*.

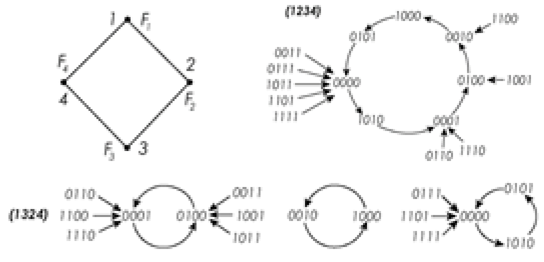

Figure 3. Phase spaces of Circ$_4$ network.

A portion of the phase space of the ***Circ$_4$*** system is illustrated in Figure 3 for two different permutations of update order. Since all four nodes may assume either of two states (0, 1), the total number of possible states that the system might assume is 2^4 (16). For any given permutation, beginning with one state of the system, such as *{0,0,1,1}*, it can be seen that the new states that can be assumed by the system depend on the permutation of update order. If the four nodes are updated in the order 1,2,3,4, the system will transition in the fashion illustrated in the upper right-hand of the figure. Configuration *{0,0,1,1}* transitions to *{0,0,0,0}*, and thence around the loop. If, on the other hand, the update order is 1,3,2,4, the system may assume any of three different phase spaces, depicted in the lower portion of Figure 3. In fact, the state *{0,0,1,1}* is not seen in 2 of the 3 phase spaces.

Moreover, it is not possible for the system to transition from one phase space to another. If the nodes begin life in the states *{0,0,1,1}* they may only transition to the configuration *{1,0,0,0}*. If, on the other hand, the nodes begin life in the configuration *{0,1,1,0}*, the system will remain in the left-hand phase space, and will never transition to the phase in the center of the figure.

For a simple system such a ***Circ$_4$***, it is feasible to calculate all 16 possible states and all state transitions. Since there are four nodes, there are 24 possible permutations π (4!) of update order. As we have seen, each update order π can generate a unique set of phase spaces. Clearly, for a simple system we can just generate all possible outcomes. However, for a just slightly more complex system, say 50 nodes, the problem is unsolvable. The number of permutations would be 50! (Factorial), more than the number of elementary particles in the known universe rendering the calculation impossible: there aren't enough storage places to save the results. Each permutation, recall, prescribes an ordering of step in transition from one configuration (snapshot of states) to another. While many of the phase spaces so generated might be identical, there is (as yet) no way to confirm that except to generate them and see.

There is nothing magical about either the particular transition function in this example (nor), or the particular 4-node network **Circ$_4$**. This example is useful because it illustrates a fundamental limitation to present-day mathematics. We do not now have the mathematical tools to predict the behavior of a simple system such as **Circ$_4$**. The only way to identify the phase spaces of the **Circ$_4$** is to generate them; i.e., to generate the system and observe the result.

The primary motivation for delving into the SDS formalism is the insight it provides to inform the planning for failure modes in complex systems. The conclusion should be self-evident, we do not have the formal mathematical tools to analyze and predict the behavior of complex systems, but the tools we do have prove conclusively that complex systems can enter into regimes that cannot be predicted with any degree of mathematical or statistical certainty.

Consider that the analysis of **Circ$_4$** above is focused on an extremely simple network; it does not speak to the interactions among layers of networks. Ongoing research indicates that the prospect of emergence within a complex network also holds for interactions across and among layers of networks (*cf* Galsyan, Kianercy, and Allahverdyan, 2010). A generally accepted perspective is that most real-world networks are complex dynamical systems in which both nodes and network topology can have inter-coupled dynamics that affect network structure and vice versa. Real-world networks are not static structures. Rather, they change over time. Connections between nodes may be established and destroyed as a function of system states. Such change in structure (e.g., connections between nodes) is another form of dynamics. Changes in structure affect generated phase spaces. Layers of evolving networks influence each other at the points of contact between the layers. Such influence also exhibits dynamics.

These aspects of real-world networks are reflected in the maturing mathematical formalisms built up around SDS and related constructs.

Emergence in Real Systems

The purpose of this paper is to frame an approach for practitioners to confront and cope with systems failures that arise from interactions --- that is, emergent failure modes. The treatment of ***Circ*$_4$** is instructive because it illustrates a how severely limited our understanding is of such phenomena. We can only generate complex systems (we call it simulation) in order to observe how they behave (and fail). We do not have the tools to analyze them rigorously and predict such failures. A rigorous treatment of the notion of emergent phenomena is not yet possible because the concept is not yet fully specified with mathematical precision. We believe the successful approach will be a fusion of the SDS concept outlined above and the intelligent control theory discussed below, but this is an area for continuing research. Instead, we offer an intuitive discussion to set the stage for our proposed approach to redress the effects of EFM.

Consider highway traffic congestion. Congestion in a particular location arises because of local interactions among relatively homogeneous objects. One car slows down (for whatever reason), and the following car also slows down; and the one following that car slows down, ad infinitum. The following car(s) may have no insight into what caused the lead car(s) to slow down; the following driver(s) simply responded to the change in state of the lead driver(s).

Such congestion is illustrative of the concept of emergence. The locus of the congestion and its bounds can be observed from a traffic helicopter. The congestion has physical extent and manifestation; it has causal relations with lower level abstractions and even individual drivers. On the basis of traffic reports from that helicopter, some drivers may elect to exit the highway and search for an alternative route. Drivers caught in a jam are left to their own reactionary approaches to getting out of the jam. Drivers informed about the existence of the jam as they approach, or drivers caught in the jam, can benefit by a broader perspective on the extent and duration of the jam. That perspective is provided by the helicopter.

Congestion may have causal effect over extensive time and space. Policy may evolve to change how the roads are used, or decisions may be made to commit resources to add lanes or move an exit ramp, or change a speed limit. Complex networks can exhibit emergent properties that have causal relations with other emergent properties and patterns of response in a network. The number of possible emergent phenomena that may exhibit causal relations within and among components of a complex system as a function of structure and local mapping functions is simply not known, may not be knowable, and may be effectively infinite. The challenge to systems engineering rigor is quite daunting.

However, challenge to SE should not be dismissed with the argument that MBE and M&S work. It is true that such methods are increasingly critical to systems engineering, and advances in M&S are crucial to improvements in MBE. Below, we even propose M&S as a core configuration item in the proposed material solution to mitigate EFM. But, in truth, we use M&S in this context because we have no alternative. We use it specifically to generate systems dynamics so we can observe them because the formal mathematics do not exist.

The actions taken by any control system are predicated upon an abstract representation (a model) of the system under control. In most cases, the model is implicit in the details of the system design. For example, in an electromechanical thermostat, the spring-like mechanism that controls both the display of temperature and the trigger for a Heating, Ventilation and Air

Conditioning (HVAC) system, entails many abstractions: the concept of temperature, not to mention the relationship between the abstraction called temperature, and the physical distortion of the spring, the scale printed behind the indicator, and the purpose for the system.

The thermostat example represents an abstraction of a relatively static (or at least slow changing) thing we call "temperature." The important point for present purposes is the fact that all control systems predicate action on a model. The character of the model, and the imperfect mapping between features and behaviors of the model and the thing being controlled (e.g., traffic in the NAS, temperature in a room), is the key here. In particular, when modeling a complex system, it is the dynamics (i.e. the behavior) of the modelled thing that matters. Given changing inputs, does the model generate outputs that conform to observations and/or expectations (that is, does the model behave appropriately)?

Dynamical systems can be analyzed by mathematical modeling (e.g., with partial differential equations), or by digital simulation. Math modeling is useful for some purposes, such as describing the behavior of an aircraft in response to perturbations of the air mass in which it is flying. However, math modeling breaks down in one particular situation, when there is uncertainty about the thing being modeled. In that case, we resort to simulation (often with probabilities as a means for "quantifying" the uncertainty). *There are no other practical alternatives.*

Simulation, in effect, embodies the collective knowledge about a complex system in a form that renders it observable. Yet, simulation can only provide an imperfect view of a complex system. Simulation is built up by composing a system from smaller parts that we can represent with some modicum of comfort. The problem is, even for the simplest dynamical systems comprising only a few parts, it is impossible to predict mathematically how the total integrated system will behave. That, indeed, is why we use simulations – to generate behavior and expose it to observation and analysis as a way of gaining insight into the behavior of a system.

Without delving further into the mathematics, two important limitations apply here. These limitations are introduced above, and summarized here for reference:

- There are an infinite number of possible models to account for any finite set of observations. The model of the system that predicates control action has a high probability of being wrong. That is, an infinite number of parameters can be added to a model and still not be able to prove with certainty which parameters matter, or which are just plain wrong.
- The mathematics do not exist to predict the dynamical properties of a composed system…so we cannot know all of the possible behaviors a system might exhibit without generating them and enumerating them. A system with as few as fifty (50) components can generate more possible states than there are elementary particles in the universe.

This last point, the extremely high number of possible states that a complex system can assume, is compounded by the fact that states are themselves abstractions. Moreover, some unknown (and unknowable) number of states are emergent, that is, they arise from local interactions, and they may have causal relations with both local and other emergent states.

The totality of states that a system may assume is known as the system's phase space. A given set of elementary components may be composed in such a way that they will generate multiple discrete phase spaces. Once in a given phase space, it is not possible for the system to transition to a different phase space.

On the basis of the analysis outlined above, it is reasonable to conclude that complex systems will be capable of entering dynamical regimes that were not foreseen when the system was developed. No amount of testing and evaluation of components prior to deployment will obviate the ability to detect, diagnose and redress emergent failure modes (EFM).

EFM: What to Do about It

Because EFM are inevitable, preventive measures are ultimately futile. Exhaustive test and evaluation (T&E) has value, but it cannot eliminate the possibility of EFM. Hence, system design must incorporate features that mitigate the effects. How can that be, given that it is impossible to predict EFM? The only viable option is to implement monitoring capabilities to ensure that the purpose of the system is being served, and to isolate and diagnose deficiencies when it is not.

The concern here is not whether a hard-wired approach is better than an SOA approach; rather the implementation of detailed planning pursuant to any approach. By definition, new systems will embody new, untried architectures and will have new properties. By considering fundamental properties of complex systems we reach a startling conclusion: looking back on how systems operated in the past may not inform how systems will behave in future. Lessons learned will have limited utility when looking forward at the next generation of systems.

As Figure 4 illustrates, a human-machine dyad can be viewed as a nonlinearly coupled set of processes, each of which has input, processing and output components (from Colacioppo, 2015). Human perception, cognition and action is loosely analogous to computer controls, computing, and display. In modern systems, most of the actions from the human-machine dyad are implemented by way of the machine, through insertion of data on a network.

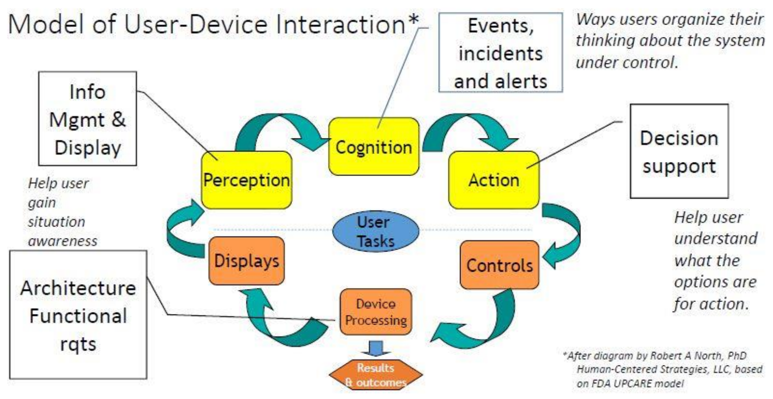
Model of User-Device Interaction*
Events, incidents and alerts
Ways users organize their thinking about the system under control.
Info Mgmt & Display
Cognition
Perception
Action
Decision support
Help user gain situation awareness
User Tasks
Displays
Controls
Help user understand what the options are for action.
Architecture Functional rqts
Device Processing
Results & outcomes
*After diagram by Robert A North, PhD Human-Centered Strategies, LLC, based on FDA UPCARE model

Figure 4. Components of a human-machine dyad.

Control systems are central to new complex designs and can be viewed as networks of human-machine dyads, as illustrated in Figure 4. This perspective should be kept in mind as one considers that information about system health originates as sense data in the infrastructure and propagates through various layers of decision-making authority, as illustrated in Figure 4. The information about infrastructure status is processed in comparison with models. Action plans are developed and implemented based on those results and the available standard operating procedures (SOP) where available. Where SOP is not available or prescribed for a given situation, new procedures are developed and implemented in a trial and error fashion.

Human interaction with Automation is Uncharted Territory

Operations and maintenance activities are control processes. As a system deviates from its design parameters; some manifestation of a deviation becomes apparent (preferably, indicators are displayed to operators and maintainers, and are detected by them); actions are taken for the purpose of reducing the deviations and returning the system to a nominal condition consistent with the design parameters.

Figure 5(a) illustrates a conventional engineering model of a closed-loop control process – the control-theoretic model. Figure 5 (b) shows an extension of that model to accommodate components in the structure of the maintainer's control problem. In Figure 5(a) a forcing function *r* is compared to the output from the controlled plant as sensed by one or more processes *F*, and an error signal (deviation from expectation) *e* is provided to the control process *C*. A command *u* is computed and asserted to the controlled plant *P*. The response of the system *y* is then evident in the behavior of the system as it interacts with its environment, and as sensed in *F*. The component functions represented in Figure 5 (a) may be considered the minimum necessary to construct a working control system. It is useful to note that there is an implicit model of the process captured in the summation of forcing function *r* and system *F*. Indeed, the specific parameters measured, and the specific summation algorithms that collapse those measures into the error signal imply some particular attention for some particular purpose. The issue of purpose or intent of the control process is external to the model in a conventional control process.

In Figure 5 (b), the concept represented in (a) has been elaborated with the minimum necessary and sufficient functions to establish control over a process in which the error signal *e* is not simply a summation, but requires induction to parameters that are not present or known beforehand. (For simplicity, we treat abduction as a special form of induction that involves reasoning from evidence to an hypothesis that did not exist prior to confronting the reasoning problem. The structure of an abduction problem is similar to classicial induction that entails reasoning from evidence to an existing hypothesis. The way induction and abduction are accomplished by a system are probably not the same, since abduction requires the additional step of synthesizing the hypothesis.) In short, the distinction between the control processes in (a) and (b) of the figure concerns the explicit presence in (b) of component processes designed to resolve the uncertainty associated with control to achieve some multi-parametric purpose for which not all parameters are known at the inception of the control process.

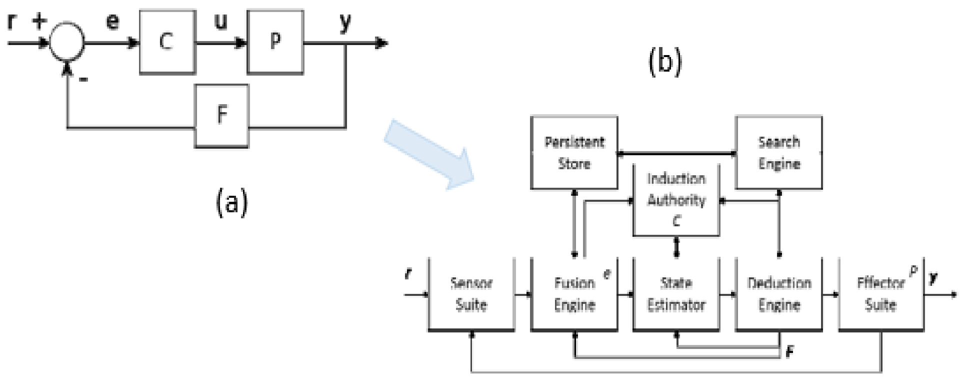

Figure 5. Extension of control theory to intelligent control.

It is informative to analyze the essential abstract character of the control process illustrated in Figure 5 from a mathematico-logical perspective. The goal is to identify the necessary and sufficient components that comprise any control system. We take as a point of departure the perspective of a classical school of psychology – cognitive psychology. After all, psychologists study behavior. Behavior is another term for dynamics. Random behavior we call noise, and often try to minimize it. All non-random behavior serves a purpose. Hence, the study of purposeful behavior should provide insight into dynamics of complex systems. Behavior is controlled change to a purpose. Hence, all behavior is a form of control process.

The logical structure of any closed-loop control problem is illustrated in Figure 6. The top illustration in Figure 6 (a) depicts the most elementary observations about control systems, that they have inputs and generate outputs, and that there must be some boundary between input and output processes, as the energy impinging on the system (the stimuli) differs from the energy emitted by the system (i.e., its responses). The middle and lower illustrations in Figure 6 (a) expand the boundary between input (S) and response (R) processes, to illustrate that of necessity, there must be processes that resolve induction. The middle illustration (S-O-R) expands the boundary between S and R processes in the depiction above, indicating that there must be some intervening organizing (O) processes between the S and the R processes. For example, the same S may elicit a different R as a result of intervening experience and feedback. The bottom illustration of Figure 6 (a) expands the O to reflect that it must comprise distinct components, as there are two separable mappings evident. Mapping from stimulus to an internal representation (often called an interpretation or situation awareness), and mapping from that internal representation to a response (also called decision making). The structure of both of these mappings is of the character of a logical induction.

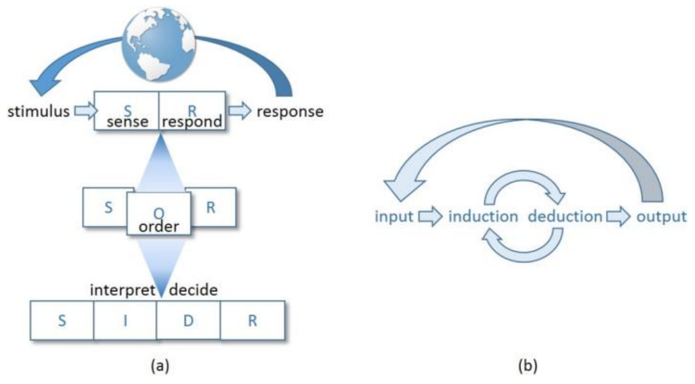

Figure 6. Logical structure of closed-loop control.

Logical induction is the character of the problem of reasoning from a finite set of observations to some general theory to account for those observations, in this case, an internal representation. Evidence for the presence of this inductive process can be found in the transformation of impinging energy. Energy emitted by the control system differs in form and organization from that impinging on the system, yet there are observable regularities between inputs and outputs. Energy in many forms impinges on the system and is mapped more or less systematically to a much smaller set of energetic structures emitted by the system. This systematic filtering and mapping of energy from (perhaps infinite) form on input, to much more limited form on output (the behavior of the control system), indicates that the system is resolving the induction problem (mapping from many to few); from a large number of specific S conditions to a small set of R.

The step from sense data to internal representation is a logical induction. The transformation from stimulus form to response form indicates that there is some sort of internal mapping/representation present in the system. Both humans acting in a control capacity, and automated control systems, resolve this inductive step. How they accomplish that induction is the subject of much research, but it is certain (at this stage) that they solve the problem by different means. That is, humans and machines differ in their ability and approaches to resolve induction problems. As Fitts (1951) observed, humans are better than machines at inductive reasoning. There is no incontrovertible evidence to the contrary in over 50 years of research since Fitts' observation; it can be taken as a principle for analyzing and designing systems.

In addition to Fitts' well-established principle, from the discussion above, a precept can be gleaned. In any control system, human or machine, there is, of necessity, some sort of internal representation (a model) of the control problem. When analyzing a control system, it is reasonable and pertinent to ask where the model resides, how was it developed, and what parameters are represented in that model? If the answers to these questions are unclear, the analyst can conclude with some confidence that the maintenance system or process being analyzed is incompetent. The control system must include appropriate allocation of function and be predicated on an architecture that achives the desired outcome. This allocation can then be captured in requirements for the design team.

Intelligent Control Theory

The approach outlined here is not proposed as a perfect solution to EFM – rather, it is proposed as a framework for the only viable approach to solution. In short, when you cannot possibly eliminate uncertainty, then you must bring humans into the control loop. A control loop that brings input data into contact with the mission or purpose of the system is reasoning beyond the data. That part of the control process is, by definition, an induction. It has been documented for over 65 years that humans are far and away better than machines at solving induction problems. Moreover, because it is not possible to know ahead of time all of the parameters of a control problem in a complex system, there is a continuing requirement to abduce new parameters in the model. There is no rigorous closed-form solution to abduction.

In a conventional control process (Figure 5 (a)), intent is embodied in the summation and the engineered characteristics of the various boxes. In the intelligent control process of Figure 5 (b), intent is introduced into the process by the induction authority. Because there is no closed-form solution to an induction problem, solution always entails the issue of intent. The mathematical basis for this assertion can be traced to research on symbolic logic and the abstraction of mathematics itself. The development of that argument is clearly beyond the scope of the present paper and the reader is encouraged to explore classical work on computability (*cf*, Gödel, 1936), the Church-Turing hypothesis and the development of lambda calculus by Alonzo Church. The gist of this argument is that a solution to induction is not computable from the symbols available solely in the induction problem (Post 1947).

The extension of conventional control theory to accommodate the problem posed by induction (*cf* Boyd, 1977) is referred to here as ***intelligent control theory***. This theory does not posit how intelligence is provided, only that something we choose to call intelligence is the ingredient required to solve the induction problem. In the same vein, the model in Figure 5 does not pretend to account for how intelligence is accomplished, only how that intelligence (however provided by human, software and/or hardware sources) is integrated into a control process.

As illustrated in the figure, all control actions are predicated on a model of the process under control. That is, no actions are predicated on reality, because reality is unknowable to the decision-making process. The internal representation, the state estimator in Figure 5(b), is just that: an estimate of the state of the controlled process. It is instructive to ponder where that state estimate comes from when the topic is maintenance actions. Often, that state estimate resides in the maintainer's head as a so-called cognitive model. If the maintainer has not been in a position to observe the system, as is often the cause with automation, then the development of the state estimate requires access to recent history of the system. Hence, early in a contingency operation, the maintainer will develop a state estimate.

We assert above that the model is usually in the user's head. The process architecture in Figure 5 does not presume that to be the case. In fact, the architecture does not presume anything about how the functions are accomplished; it requires only that a model of the process under control be incorporated into the control process. In truth, at least two such architectures are conjoined in a real application – the human and the machine. Each must resolve the component functions, or obtain the solution from the other.

System architecture and design must be developed to support this process need by assuring the architecture provides the flexibility to support dynamic control based on models held in the system constituents (in human users and maintainers as well as in software). The user's cognitive models should be supported in explicit computer models of the system.

Conclusions

EFM are real artifacts of systems design and implementation. The analysis of complex systems (including SOA/SoS) shows the unpredictable and nearly inevitable character of EFM. The paper outlines a principle-based approach that apportions aspects of control processes to human and machine components in a way that exploits human strengths to detect, diagnose and redress emergent failures provides an approach to solutions. The recommendation is to ensure that a proposed system architecture conforms to the process architecture in Figure 6.

References

Boyd, J. 1977. *Destruction and Creation*, unpublished monograph.

Church, A. 1936. An Unsolvable Problem of Elementary Number Theory. *American Journal of Mathematics*, 58, 345-363.

Colacioppo, T. 2015. Integrated Service and Information Management: NYCT B-Division (ISIM-B). Paper presented at 2015 APTA Rail Conference, Salt Lake City, UT, June, 2015.

Columbia Accident Investigation Board. 2003. *Report on the Space Shuttle Columbia Accident*. Hearing before the Committee on Commerce, Science and Transportation. United States Senate. One Hundred Eighth Congress, First Session. S. Hrg. 108-985.

Committee on Science and Technology. 1986. *Investigation of the Challenger Accident. Report of the Committee on Science and Technology House of Representatives, Ninety-ninth Congress, Second Session*. HR 99-1016. 29 Oct 1986.

Dobuzinskis, A. 2016. Microsoft apologizes for offensive tirade by its 'chatbot'. Thomson Reuters Foundation http://news.trust.org/item/20160326003852-d7mjr.

Fitts, P.M. 1951. *Human Engineering for an Effective Air Navigation and Traffic Control*

System. Washington DC. National Research Council.

Galstyan, A., A. Kianercy, and A. Allahverdyan. 2010. Replicator dynamics of co-evolving networks. In *AAAI Fall Symposium on Complex Adaptive Systems*, November.

Gödel, K. 1962. *On Formally Undecidable Propositions of Principia Mathematica and Related Systems*. New York, US-NY: Basic Books.

Harris, S. and J. Narkevicius, 2014. Appropriate automation: Human system dynamics for control systems. *Control Engineering*, March.

INCOSE 2015. *Systems Engineering Body of Knowledge*. Retrieved from http://sebokwiki.org/wiki/SEBoK_Introduction

Mortveit, H.S., and C. M. Reidys. 2008. *An introduction to sequential dynamical systems*. New York, US-NY: Springer Science+Business Media, LLC.

Ortiz, E. Patel, Y., and The Associated Press. (15 August, 2015). Flight Delays Spread Across East Coast After D.C. Outage. http://www.nbcnews.com/news/us-news/flight-delays-spread-across-east-coast-after-d-c-outage-n410461.

Post, E.L. 1947. Recursive unsolvability of a problem of Thue. *Journal of Symbolic Logic*. Vol 12(1), March.

Biography

Steve Harris is a former Navy Medical Service Corps officer who has conducted research and managed programs in human factors and systems engineering for the Navy, DoD, NASA, Dept. of Energy, FAA, universities and private industry in his 40-year career. He has authored several technical research publications and book chapters. He is now an independent consultant for human factors for government and industry clients.

Dr. Jen Narkevicius brings diverse training, education and experience to complex programs from research to requirements definition, system design and development and test and evaluation. She works across domains including defense, infrastructure, road, air and rail transportation as well as communication and is concurrently persuing identification and development of the linkages between science, technology, engineering, arts and maths.