Journal of Aerospace Operations 1 (2012) 317–327
DOI 10.3233/AOP-120024
IOS Press

The elephant in the mist: What we don't know about the design, development, test and management of complex systems

Wilson N. Felder[a] and Paul Collopy[b,*]
[a]*Federal Aviation Administration, Wm J Hughes Technical Center, Atlantic City, NJ, USA*
[b]*National Science Foundation, Arlington, VA, USA*

Abstract. The authors survey the literature on complex systems to develop a fundamental critique of the systems engineering methods that value-driven design will attempt to replace. The current approach to systems engineering applies industrial revolution methods to information revolution problems. Not only the flowdown of requirements, but also the ability to verify and validate large engineered systems is brought into question.

Keywords: Design, systems engineering, complex systems, socio-technical systems, validation

1. Introduction

For some time now, the aerospace community has been plagued by a series of failures related to the increasing complexity of the systems we are attempting to design, build, and operate. These failures have been particularly vexing as they come in an era when the availability of powerful systems engineering tools, the application of ever more sophisticated process management concepts, and a highly professional systems engineering and acquisition workforce lead us to believe that "***we know how to do this***!" and therefore we believe our failures must be due to either management incompetence, or perhaps the unwillingness to adhere to the established disciplines. However, it is increasingly becoming clear that most of the failures result not from incompetence, but instead because we did too good a job of following the rules of the game. Thus we are led to the conclusion that there is something inherently different about the systems we are now building which calls for a whole different set of techniques and approaches.

The likelihood that a new understanding was needed has been articulated for some time, beginning with the seminal work of Boehm [6] in software cost estimation; continuing in the rich literature around System of Systems theory (notably Maier [22], Carlock and Decker [7, 8]); and finding strong affirmation in Baldwin and Clark [1], Carlson and Doyle [9], and Miller and Page's [24] outstanding work on complex adaptive systems. The behavior of complex systems has been observed in fields outside aerospace as well, with Taleb's [27] observations from the field of quantitative investment analysis being particularly provocative.

*Corresponding author: Paul Collopy, National Science Foundation, 4201 Wilson Blvd., Arlington, VA 22230, USA. Tel.: +1 703 292 2241; E-mail: pcollopy@nsf.gov.

ISSN 2211-002X/12/$27.50

It is no fluke that the first inkling along this path came from Barry Boehm's attempts to quantify the cost impact of software [6]: the change in the kinds of systems we develop, and the reason for their failure to follow the rules of the game is rooted in the pervasive presence of software as an intimate effector of system behavior. Complex systems, and the bizarre ways in which they behave, are a creature of the Information Revolution, that is, the application of digital technology to just about every facet of our life today. In fact, we now have complex systems not driven by traditional software, but by firmware or even semiconductor gate arrays that suffer the same problems because the gate arrays are developed through a process similar to the way software is developed. Digital technology leads to complexity in two distinctly different ways: first, it causes unpredictable system behavior to begin with, provided the systems in question possess certain architectural features; and second, it permits a much more robust and dynamic interconnection among the humans involved with the design, development and operation of the system, which in turn leads to unpredictable behavior in itself. The conclusion is becoming inescapable:

> *Socio-technical systems, that is, systems that involve groups of people interacting with various engineered systems (an example is the National Air Transportation System) quickly become complex in the formal sense, and will inevitably exhibit emergent behavior, much of which will be undesired.*

In this paper, we will summarize what the attributes of complex systems are, and provide a framework for continued discussion, analysis, and development of understanding about complex systems, their design, development, test, and operation. The task is critical, because the speed with which technology is changing demands our ability to engineer systems capable of taking advantage of new technologies safely and effectively. The pace of technology is an order of magnitude faster than the ability of our institutions to respond: by the time a typical good idea is implemented in a government system, ten years have elapsed: in the same period, there will have been at least a dozen fundamental quantum jumps in technology!

2. Background

So, what in the world is going on? The truth is, we are living through a remarkable period in human history, the largest and most intense shift in human productivity ever experienced by our species. The closest analogy is to the transformation that happened during the middle years of the nineteenth century (and lasting into the mid-twentieth) that resulted from the mining of fossil fuels and the subsequent application of steam power to just about every aspect of human life. In those days, the newspapers were filled daily with some new "invention" (the "app" of the 1840 s . . .) that would have profound consequences for manufacturing. Incidentally, it is interesting to note that the word "manufacture" is a Latin portmanteau word meaning "to make by hand". The cataclysm of the mid nineteenth century forever transformed the meaning of that word into its antonym! The elites who ran the manufacturing sector in England in those heady days simply did what seemed the right thing to do (and thereby gave capitalism and the free market a bad name): they reduced the size of their workforces to take advantage of the muscle available from the steam engine, at an accelerating rate as second and third generation machinery learned to generate and employ mechanical power with ever greater efficiency. Although the resulting spike in productivity, known to history as the Industrial Revolution, increased the English standard of living, millions of cottage laborers and farm workers in England were displaced. The resulting squalor of factory workers living on or below subsistence wages led to the immortal literature of Dickens, but also to Socialism, Marxism, two World Wars and the Great Depression. This impact was not predictable and probably not avoidable. The fact remains that, for much of the nineteenth century in Britain, increasing

productivity uprooted much of society, while it inflicted twelve hour factory workdays on children, and introduced the poor house as a despised repository for the unemployed.

What we are living through today is similar, only one or two orders of magnitude bigger and faster. It is a revolution of information, not energy sources. And as the Information Revolution accelerates through week after week of new advances involving the application of information science to just about every aspect of human life, we find ourselves at a loss to understand how these digital systems interact, and why, after all, they do so in such unpredictable ways.

Meanwhile, the set of best engineering practices enshrined in our systems engineering dogma were developed as solutions to, and are best suited to, industrial revolution problems, not information revolution problems. The tools we use to design, develop, test and operate these Information Revolution systems are tools adapted to the products of the Industrial Revolution: there is a fundamental assumption behind every aspect of our project management and systems engineering approaches and methods that the system we are building responds to Newtonian-type deterministic cause and effect processes, and most importantly, that the user (or observer) stands outside the system boundary. In the digital world, the user-system interaction is co-evolutionary, unpredictable and defies classical analysis. Digital technology, or software as it is currently conceived, fundamentally changes the relationship between the user and the system.

Thus, when we study the performance of a wide variety of system management cases in recent years, we find, in fact, that usage of the most important and prevalent tools for system management (requirements definition, interface control, schedule management, and the like) are poor predictors of system success, and further, that the systems management methods we use to control cost and schedule and assure a successful result, are poorly correlated with keeping programs under cost and on schedule, and poorly correlated with system success [13]. We desperately need new techniques for managing the design, development and test of complex systems, which, as we will see below, define the essential problem we face in the twenty-first century. Industrial Revolution giants such as Henry Gantt built the tools they needed to manage the chaos of their time. They were making it up as they went. Two centuries later, we apply their best ideas to our quite different problems. If they returned today to consult with us, surely they would say: "You idiots! Why would you think that these methodologies we developed to deal with things moving at Industrial Revolution speed would work in your new Information World? They won't. Build your own damn tools!" As a result, Griffin [17] has drawn attention to the impact of large-scale complex systems on our systems engineering profession and called for drastic changes. For example, it makes little sense to establish performance requirements for modern complex socio-technical systems. We instead need to develop the guidance technology (in the sense used by Baldwin and Clark [1]) to direct a (possibly very large) engineering team to design the best system that they can, and cope with the performance that results. We cannot anticipate what the system will need, particularly when we focus on designing the technological system and then mix it in with human users and social organizations in the field. Part of the twenty-first century methodology may be Model-Based Systems Engineering [4, 19], where we model the performance of the technical system and the behavior of the human/social components as the design proceeds, adjusting and redirecting the design to improve the modeled performance.

Northrop et al. [26] discuss ultra-large software systems that engage people and organizations into their operations so intimately that the notions of technology and user become obsolete. Instead, both combine into a complex socio-technical system. Northrop shows how requirements for a large system evolve in use to such an extent that most requirements are unknowable at the outset of design. Requirements may even morph in a contrary response to our efforts to satisfy them. Systems engineering's current top-down requirements-driven processes cannot be effective for these new systems [11].

Meanwhile, with our admirable tendency to review our performance and see what lessons can be learned, we continually insist on learning the wrong lessons from our failures. A poignant example of this habit is related to the Air France 447 crash [29]. Although the fly-by-wire aircraft involved in this catastrophe did not, most likely, constitute a complex system in itself (the engineered portion of the system was complicated, but not complex), the addition of the human flight crew created a complex system. Although the crash apparently resulted from the failure of the crew to realize that they had entered an unfamiliar system state (albeit one appropriately designed into the aircraft system), it is important to realize that the real problems resulted from the underlying complexity of the digital system design interacting with trained-in behaviors and perceptions of the crew. If we are to reverse the unfortunate trend of aerospace system failures, we need to stop looking at systems in Newtonian, Industrial Revolution mode, and start looking at them through new, Information Revolution lenses.

Like other complex systems, complex socio-technical engineered systems are characterized by emergent behavior which cannot be predicted from the deterministic attributes of their components. They also may be adaptive: that is, they may have the ability to change their structure as a result of experience [1]. So far, we only know of complex adaptive systems in nature and naturally evolving in society: they are either organisms as described by Minai [25], the kind of human centered systems described by Miller and Page [24]. The era of intentionally constructing complex systems has just begun.

2.1. System definitions

If we are going to better understand the elephant that is engineered complex systems, it is important to have clear definitions: this discussion is based on the set of system attributes proposed by Boardman and Sauser [5], incorporating the ideas of Maier [22], and extended by Sauser and Baldwin [3]. Baldwin et al. [2] have suggested a taxonomy of increasingly complex system types based on this scheme, which is consistent with anecdotal definitions common in the system literature. An important distinction is the difference between systems and systems-of-systems, because the latter are most prone to evolve into complex socio-technical systems. In Baldwin's taxonomy, a system is considered to be a system-of-systems if and only if it meets the following four criteria:

1) The component systems exhibit autonomy, that is, each component system is capable of standing alone, accomplishing one or more goals for which it has all the required inputs and processes;
2) The component systems are diverse, that is, they are directed toward the accomplishment of different individual goals using different processes;
3) The component systems exhibit belonging, that is, each component system, in addition to meeting its own goals, also contributes to a larger goal set that is distinct from the sum of the goals of the individual components;
4) The component systems are connected to one another dynamically, that is, connections among them are established as required as needed to accomplish the overall goals of the group, and that each component system is capable of establishing connections with more than one other system.

In order for such a system to also be complex, it is necessary that it meet an additional criterion, that of emergence. The interaction among member systems in a system of systems causes emergent features of the overall system of systems that are not predicted by the deterministic behavior of the component systems. This behavior has been observed in all sorts of cases, most certainly in human intensive systems such as the ones described in Miller and Page [24], but also in relatively simple two-system system of systems in which the two systems have a large library of functions.

There are debates as to whether large engineered systems are complex or complicated. While modern aircraft and sophisticated weapon systems are complex in a colloquial sense, because, for example, they cannot be comprehended in their complete detail by a single person, they are not intentionally complex in the way that the word is used in complexity theory – we do not want aircraft to have emergent properties. However, we are observing undesired behaviors emerging from engineered systems, and they are arguably more prevalent in systems of systems. We have even observed undesired emergent behaviors in relatively small engineered systems. Although we do not yet know whether a system of systems is, by definition, complex or not, if we agree that the existence of emergence is the essential feature of a complex system, then it remains to be seen whether there is such a thing as an system of systems that meets all the definitional attributes (autonomy, connectivity, diversity and belonging) but does not display emergence. Perhaps there is: emergence only happens when the number of possible system states is high for more than one of the component systems! In this sense, the "ultra large systems" concept of Northrop et al. [26] does begin to make sense. Anyway, there is work yet to be done in order to establish this point.

We are building systems that spend more time in nominal operation (that is, are generally better behaved) than previous generations, but when they *do* operate off nominal, are much further from the nominal than previous generations. Carlson and Doyle [9] call these systems with "highly optimized tolerance". A good example is the modern car. Automobiles built in 2012 are far more reliable than automobiles built in 1962. However, when they do fail, it is in ways that are often bizarre and unforeseen, such as the Toyota acceleration problem, which was eventually judged to result from a complex interaction of the driver and the car, or the Ford Explorer rollovers, which were compounded of:

- Consumers driving trucks as if they were cars;
- Firestone tires adversely interacting with Ford suspension systems;
- High centers-of-gravity built into vehicles to serve a marketing purpose (off-road capability) that was seldom or never used by the drivers.

This attribute of complex systems is related to the probability distribution that describes the frequency of off nominal conditions: in traditional systems, even very complicated ones, this function follows a Gaussian distribution. In complex systems, however, for a variety of reasons, the distribution is highly leptokurtic: that is, it is much "spikier", with the probability of normal behavior higher than expected, but by the same token, the probability of substantially off-nominal events is much higher (the concept of the "fat tail") and for extreme conditions, the relative probability is much higher still. This leads to the observation that complex systems "[Don't] fail very often, but when [they do], [they] fail spectacularly". (In a conversation with W. Felder at the 2011 Aircraft Technology Integration and Operations Conference, Mr. Capezzuto, at the time the program manager for the FAA's Enroute Automation Modernization (ERAM) Program, commenting about the stability of the system post-operational test, said: "ERAM doesn't fail very often, but when it does, it really fails!", Mr. Capezzuto confirmed this viewpoint in April, 2012, at the AIAA 10th ATIO conference, Norfolk, VA.)

2.2. *Implications for system design*

The need to design systems differently in the face of complex system behavior has been addressed by a number of workers, including DeLaurentis [14, 15]. Minai et al. [25] reviewed a series of new strategies for the design of complex systems including three which bear mentioning here: degeneracy, re-use, and robustness-by-structure.

Degeneracy refers to the use of multiple processes with identical consequences. It was introduced as a key element of engineering complex systems by Edelman and Gally [16] and Cole and Sanders [10]. Degeneracy would be considered an anathema in conventional system design. However, if we are looking at a world in which not only the environment, but in fact the requirements of a system change with time, it makes sense to have at hand a number of tools which act in different ways to produce the same result. Then, in the event that the star pupil is unable to deliver the required result (perhaps because the nature of that result has changed . . .), there will be another mechanism available to get the job done.

When you think about it, this is exactly the situation that existed prior to the Industrial Revolution. Mechanization introduced standardization, but before, in the era of artisanal manufacturing ("making by hand"), there were many options. Tools were general purpose, and a skillful artisan could use a tool designed for one purpose to accomplish some unrelated task. As a matter of fact, saws (in all their manifestations), chisels, axes, spokeshaves, drawknives, planes, and a constellation of other tools, are all variants on a single idea: the knife. The Industrial Revolution changed all that. The Information Revolution asks us to reconsider once again.

Modular re-use is a complementary idea to degeneracy in which a single process is co-opted with minimal (or no) change to perform a quite different task. A good example of modular repurposing was the TRW software built into the Apollo Lunar Module which permitted Apollo 13 to return safely to earth. Such unnecessary redundancy would never have been permitted in today's lean acquisition environment. Fortunately for the Apollo 13 astronauts, it was an era when cost was no object in the space program. It turns out that TRW's scientists and engineers were, in this as in so many other cases, chillingly prescient.

Finally, robustness-by-structure, which is a key element of biological systems, is advocated as a strategy for complex system design by various authors, most eloquently by Ulieru and Doursat [28]. Minai et al. [25] point out that this is a feature of the only successful complex adaptive systems we know of: namely, organisms. For example, once bilateral symmetry had evolved, no branch of that tree, which includes all the most advanced creatures on Earth, ever abandoned it. In the biological world, a central core of successful adaptations provides the platform from which wild experimentation can blossom. Robustness-by-structure provides a similar design precept for engineered systems.

We can well ask whether any of these techniques could be useful in designing systems, whether purely engineered, or the result of a complex human-system collaboration. We believe that the mounting evidence suggests that the answer is clear: at least these, and maybe others yet to be discovered, are valuable approaches to ensure that aerospace systems are up to the task they will face in the coming years.

We should understand the implications of this approach—in the future, it will be possible for an aircrew to enter the cockpit of a commercial airliner, insert a memory stick (or its inevitable mid twenty-first century wireless bionic descendant) into the panel (or maybe by 2050, the Captain will only have to *think* about it to make it happen!), and upload their favorite apps. And do it in a way that does not compromise the safety of flight, FAA regulations, or corporate policy! A shout-out here to the first Chief Technology Officer of the United States, Aneesh Chopra, who famously said: "you're looking at this NextGen thing all wrong: what you need to do is get industry to cloud-source apps for the cockpit!" As in so many other ways, Mr. Chopra was a visionary way ahead of his time. Perhaps it's now our time to step up to the challenge!

3. Implications for system development

Complexity in systems not only affects the system itself, but has profound implications for the management of the development process. The complexity of the system interacts in numerous ways with the

distributed nature of the development enterprise to demand new methodologies for almost all phases of system engineering.

First, we need to consciously avoid complexity wherever we can, both in the system we are designing and in the program that is designing the system. Collaboration tools that provide all the design information to all the designers do not solve systems development problems, they only drown engineers in data. Integrated product teams can provide a valuable conduit for sharing critical information across different program organizations, or they can magnify everyone's communication burden without informing critical decisions. To simplify the designed system and the designing organization, interfaces need to be simplified and easily understood. Work done in software engineering on packaging modules, eliminating interactions, and hiding information within modules is as relevant and useful to mechanical and electrical systems as the it is to software. Griffin's [17] work on elegant design applies here, as well as more general notions of elegance [23]. In essence, the elegant design message is to focus on the product. Will it be effective? Efficient? Robust? Do not depend on systems engineering processes to deliver by themselves. Elegant design emphasizes that the system design activity must be oriented toward making a great product, and processes are merely aids toward that end. In particular, elegant design sharply differs from the requirements analysis strategy found in most systems engineering manuals. Rather than canvassing stakeholders for what requirements they wish to impose on the system, elegant design places the onus on the system designer or chief engineer to select a small number of requirements that the system can perform well, and design to those. The importance of this to program success is verified by Larson and LaFasto [21]. Their research has shown that the single characteristic that is found in every successful team effort is "a clear, elevating goal". Such a goal cannot be expressed as 373 requirements, as are being used to specify the new KC-46 tanker [10]. A clear, elevating goal cannot be spread across ten requirements, or even five. The Apollo program successfully developed a very complex system for lunar exploration in part because the goal was easily understood by every person working on the program: "the goal, before this decade [the 1960's] is out, of landing a man on the moon and returning him safely to the earth" [20].

Second, we need a way to communicate direction to all the members of a very large group of design engineers, who may be spread over several organizations. Knowing that we are going to the moon does not directly inform a design engineer's decision to make the legs of the lunar lander from aluminum rather than titanium. Again, the practice of elegant design comes into play. While our standard requirements flowdown and allocation process propagates more and more requirements at each level of the work breakdown structure (and a complex aerospace system typically has seven to ten levels in this hierarchy), value-driven design [12] provides an alternative method of communicating direction to a very large and diverse design team. Under value-driven design, each design engineer receives an objective function which encapsulates the trade factors necessary to make that engineer's design decisions. Because these trade factors flow from a system level objective function, they directly capture the preferences of the project chief engineer, and make these preferences available to every design engineer in the development program. While requirements become more and more disconnected from system level thinking as they are allocated and re-allocated at each level of the work breakdown structure, value-driven design conveys preferences intact. Preferences are more useful in any case, because they apply to every single design decision throughout the system. Requirements only come into play when they might be violated by a design decision. The vast majority of design decisions do not threaten even component level requirements, so that today they are decided with no guidance whatsoever [11].

Third, we need to take advantage of the more formal design approaches used in the semiconductor and automotive industries. Integrated circuits, cars, and aerospace systems are all becoming much more complex over time. The cost of electronics and automobiles is increasing linearly with complexity. The

cost of aerospace systems, on the other hand, increases exponentially with complexity (according to a comparison by Paul Eremenko at the Defense Advanced Research Projects Agency, in which complexity was measured as the logarithm of the number of parts plus the number of lines of code). Integrated circuits have learned to manage complexity through enforcing a set of design rules. The rules greatly restrict the allowable designs, but in return, a design that follows the rules provably meets its functional requirements. Because a billion component integrated circuit that does not conform to the rules is impossible to debug, the exchange of design limitations for functional correctness is a good deal. It would be very beneficial to build a similar set of rigorous design laws for the cyber-physical systems we deal with in aerospace.

4. Implications for system test

We have a validation philosophy developed for industrial revolution systems in a Newtonian universe. It is unrealistic for complex (or even complicated) engineered systems in the information age.

The number of states in large engineered systems is vast, sometimes astronomical, but to do anything like a traditional failure modes and effects analysis requires comprehension of the state transitions, which are hyper-astronomical in number. For example, when a typical laptop computer experiences a crash, the possible causes are almost uncountable. We expect that a significant number of crashes are unique events which never have occurred at any time or place, and have a vanishingly small probability of ever occurring again, in spite of an installed base of hundreds of millions of systems.

The combination of many component systems (with autonomy, diversity, dynamic connectivity and belonging) and component systems with non-trivial internal complexity (which generates emergence) results in an almost infinite number of possible states for a typical system of systems.

Early in the 20th century, we could validate a system by exercising every possible state and every state transition. By the 1970 s, we were making systems that incorporated embedded software, and completely exercising these systems was impossible. However, we could convincingly exercise every state transition that was likely to occur in a billion hour product life (interpreted as one billion hours mean time between uncovered failures). Now there are so many failure modes that a nearly uncountable number constitute a large percentage, perhaps a majority of the failures we anticipate will occur. That is, we must have a validation process that recognizes that most failures in the field may be manifestations of behavior that was never observed during validation. This will require more than a new process, it will necessitate a new theory, and likely a new philosophy of system validation.

The U.S. National Airspace System is a useful example of this trend. When the system was initially established, it consisted entirely of electro-mechanical components. Testing was accomplished by bringing these components into the laboratory, attaching oscilloscope leads to them, and verifying that they met the specifications. This process held from the establishment of the FAA's National Aviation Facilities Experimental Center in 1958. Starting around 1970, automation was applied to the system with the introduction of flight data processing, and later, digital processing of radar inputs. This automation continued through the late 1980 s, with increasing complexity and pervasiveness. Parenthetically, the increasing level of automation led the agency to realize, around 1982, that the national airspace system was indeed a *system* and needed to be managed as one. This realization did not mean, however, that the right tools and approaches were applied. The agency and its engineering support contractors never did get their arms around the problem, which has continued to eat their lunch ever since. During the 1980 s and 1990 s, however, a great deal of thought was given to the impact of large scale systems with multiple connections [7]. At a minimum, however, the agency did learn to deal with software, by Pareto analyzing the requirements, thoroughly testing the 20% of the requirements that resulted in 80% of the risk, and

letting the remaining low probability/low consequence risks work themselves out once the system had been fielded.

This approach, however, is not viable when we are dealing with complex systems.

As an example, the very first phase of the FAA's Data Communications project, which simply contemplates delivering a flight plan clearance to an aircraft parked at the gate, will require three enroute system labs, two terminal system labs, a communications lab, and other connective equipment. To fully test the overall Data Comm system, in all its forms, will be literally impossible in the traditional mode.

In response to this problem, there is an emerging consensus that a robust virtual/live/constructive simulation environment can be used to explore all corners of the system performance envelope, with the intention of gaining sufficient experience in the virtual operation of the system to drive out the low probability but high consequence events that live in the "fat tail" and cannot be predicted (because of "essential unpredictability", "morphing requirements", "emerging behavior",) as a result of the complex nature of the system under test. These concepts have been advanced by the Software Engineering Institute [26, 18] and form the basis of the FAA's approach to the verification and validation of the Next Generation Air Transportation System.

5. Conclusions (the Elephant in the Mist)

At this point in history, the overall impression is one of looking at an elephant in the mist: we sense that there is a large and complex animal out there, but we are not yet sure of its exact dimensions, characteristics, and nature. However, we can surely discern some very interesting attributes of the beast, as well as knowing how to better ask many unanswered questions, which have been alluded to above. Is a system of systems (defined as above) necessarily complex? Is it possible to build a test process which reliably explores enough of the envelope of system behaviors in a complex system to permit fielding of that system with confidence? What does it mean to say (as Taleb [27] did) that the probability of system failures has a "fractal" distribution, and that this is synonymous with "essential unpredictability"? If requirements management is stupid, ineffective, or impossible, then what discipline should replace it in the new world? Similarly for the management of schedule and cost. And, in a world where technologies change too quickly to survive the length of the most accelerated system acquisition, how do we ensure that the long term objectives of an acquisition are maintained while taking advantage of technology developments in the wider world, as we continue to build our systems? The answers to these questions, and others as yet unstated, will provide us with a sufficiently good understanding of this elephant's anatomy to proceed with confidence as we tackle the system challenges of the twenty-first century.

However, to conclude, the current approach to systems engineering applies industrial revolution methods to information revolution problems. Not only the flowdown of requirements, but also the ability to verify and validate large engineered systems is brought into question and found wanting. Consequently, this reflection on complex systems and a fundamental critique of systems engineering methods concludes that these methods are no longer sufficient and it requires a revolution in methodology, which ultimately value-driven design will attempt to deliver.

Acknowledgments

The authors would like to thank Clifton Baldwin and Michael Griffin for their insights and commentary. Any opinions, findings, and conclusions or recommendations expressed in this material are those of the

authors and do not necessarily reflect the views of the Federal Aviation Administration or the National Science Foundation.

References

[1] C.Y. Baldwin and K.B. Clark, *Design Rules, Vol. 1: The Power of Modularity*, 1st edition, 2000, The MIT Press, Massachusetts, 2000.
[2] W.C. Baldwin, W.N. Felder and B. Sauser, Taxonomy of increasingly complex systems, *International Journal of Industrial and Systems Engineering* **9** (2011), 298–316.
[3] W.C. Baldwin and B. Sauser, Modeling the characteristics of system of systems, *2009 IEEE International Conference on System of Systems Engineering (SoSE)*, Albuquerque, New Mexico, 2009.
[4] M. Belfore, Adaptive Vehicle Make: DARPA's plan to revolutionize auto manufacturing, *Popular Mechanics* 2012.
[5] J. Boardman and B. Sauser, System of systems – the meaning of, *Proceedings of the 2006 IEEE/SMC International Conference on System of Systems Engineering*, Los Angeles, California, April 2006.
[6] B.W. Boehm and R.W. Wolverton, Software cost modeling: Some lessons learned, *Journal of Systems and Software* **1** (1979–1980), 195–201.
[7] P. Carlock and S. Decker, Development and implementation of an architecture for the national airspace system, *Systems and Information Technology Review Journal* (Spring/Summer 1998), 6–16.
[8] P.G . Carlock and R.E. Fenton, System of systems (SoS) enterprise systems engineering for information-intensive organizations, *Systems Engineering* **4** (2001), 242–261.
[9] J.M. Carlson and J. Doyle, Complexity and robustness, *Proceedings, National Academy of Sciences (PNAS)* (2002), 2538–2545.
[10] A. Cole and P. Sanders, Air Force resumes tanker contest, *Wall Street Journal*, 2009.
[11] P.D. Collopy, Adverse impact of extensive attribute requirements on the design of complex systems, *7th AIAA Aviation Technology, Integration, and Operations Conference*, Belfast, Northern Ireland, 2007.
[12] P.D. Collopy and P.M. Hollingsworth, Value-driven design, *Journal of Aircraft* **48** (2011), 749–759.
[13] P.J. Componation, A.D. Youngblood, D.R. Utley and P.A. Farrington, A preliminary assessment of the relationships between project success, system engineering, and team organization, *Engineering Management Journal* **20** (2008), 44–50.
[14] D.A. DeLaurentis and S. Ayyalasomayajula, Exploring the synergy between industrial ecology and system of systems to understand complexity, *Journal of Industrial Ecology* **13** (2009), 247–263.
[15] D.A. DeLaurentis and W.A. Crossley, A taxonomy-based perspective for systems of systems design methods, *IEEE International Conference on Systems, Man and Cybernetics*, October 2005.
[16] G.M. Edelman and J.A. Gally, Degeneracy and complexity in biology systems, *Proceedings, National Academy of Sciences (PNAS)* (2001), 13763–13768.
[17] M.D. Griffin, How do we fix systems engineering? IAC-10.D1.5.4, *Proceedings, 61st International Astronautical Congress*, Prague, Czech Republic, 2010, 2–9.
[18] J. Goodenough and L. Northrop, Software assurance for systems of systems, *Proceedings of the FSE/SDP workshop on Future of software engineering research, FoSER 2010*, 2010, Santa Fe, New Mexico, ACM New York, New York, 133–136.
[19] D.W. Hybertson, *Model-oriented systems engineering science: A unifying framework for traditional and complex systems*, Auerbach Publications, 2009.
[20] J.F. Kennedy, *Excerpt from an Address Before a Joint Session of Congress, 25 May 1961*. John F Kennedy Presidential Library, video clip, accession no. TNC-200-2.
[21] C.E. Larson and F.M.J. LaFasto, TeamWork: What must go right, what can go wrong, Sage Publications, Inc, 1989.
[22] M.W. Maier, Architecting principles for system-of-systems, *Systems Engineering* **1** (1998), 267–284.
[23] M. May, *In pursuit of elegance: Why the best ideas have something missing*, Crown Business, 2009.
[24] J.H. Miller and S.E. Page, *Complex adaptive systems: An introduction to computational models of social life*, (Princeton Studies in Complexity), Princeton University Press, Princeton, New Jersey, 2007.
[25] A.A. Minai, D. Braha and Y. Bar-Yam, Complex systems engineering: A new paradigm, in *Complex engineered systems: Science meets technology*, D. Braha, A.A. Minai and Y. Bar-Yam (eds.), Springer Verlag, 2006, 1–22.
[26] L. Northrop, P. Feiler, R.P. Gabriel, J. Goodenough, R. Linger, T. Longstaff, R. Kazman, M. Klein, D. Schmidt, K. Sullivan, and K. Wallnau, *Ultra-large-scale systems: The software challenge of the future*, Carnegie Mellon University, Software Engineering Institute, Pittsburgh, Pennsylvania, 2006.

[27] N.N. Taleb, *The black swan: The impact of the highly improbable*, Random House, New York, New York, 2007.
[28] M. Ulieru and R. Doursat, Emergent engineering: A radical paradigm shift, *Int. J. Autonomous and Adaptive Communications Systems* **4** (2011), 39–60.
[29] J. Wise, What really happened aboard Air France 447, Popular Mechanics, Digital Edition, 6 December 2011, 3PM, 15 pages.