Software Engineering Institute
Carnegie Mellon University

FAA RESEARCH PROJECT ON SYSTEM COMPLEXITY EFFECTS ON AIRCRAFT SAFETY: LITERATURE SEARCH TO DEFINE COMPLEXITY FOR AVIONICS SYSTEMS

Michael Konrad and Sarah Sheard

May 2015

Abstract

This special report describes the results from the first task of a two-year project for the Federal Aviation Administration to investigate the impact of system and software complexity on aircraft safety and certification. The first task was a literature review sampling what is known about complexity. The term *complexity* is often used but not often defined, especially in the context of safety and assurance; our review had to intuit what different sources intended as the meaning by how they used the term in their writings. Essentially, sources described a number of different causes and impacts of complexity. The causes and impacts as described in the literature are presented here in a hierarchy of categories. To use the word "complexity" precisely, the causes and the impacts should be specified. Some sources also discussed measurement and mitigation of complexity; both measurement and mitigation can apply to both causes and effects.

Executive Summary

The Federal Aviation Administration (FAA) is concerned that the growing complexity of aircraft systems and software may cause aircraft to be unsafe, and the FAA may be unable to assure the software with confidence. Systems are increasingly software-reliant and interconnected, making design, analysis, and evaluation harder than in the past. While new capabilities are welcome, they require more thorough validation. Complexity could mean that design flaws or defects could lead to hazardous conditions that are undiscovered and unresolved.

This is the first report in a two-year research project to investigate the nature of complexity, how it manifests in software-reliant systems such as avionics, how to measure it, and how to tell when too much complexity might lead to safety and certifiability problems.

This Report

This literature review offered insights into the *causes of complexity*, the *impacts of complexity*, and practices for *mitigating complexity*. We took a broad approach to the literature search to make sure that when metrics were selected (in the next task) we did not rule out any as not applicable until they

Distribution Statement A: Approved for Public Release; Distribution Is Unlimited

could be truly evaluated. Thus the report captures the breadth of discussion in the literature on complexity. We will fill in details about its application to system safety, aircraft certification, and software and avionics as we work through the rest of the research tasks.

Causes of Complexity

In this systematic literature search, we found that while many problems are blamed on "complexity," the term is usually left undefined. As a result, we modified the focus of the task from simply collecting definitions to describing a taxonomy of issues and general observations associated with complexity.

Our literature review revealed that complexity is a state that is associated with *causes* that produce *effects*. We have a large taxonomy of different kinds of causes and another taxonomy of different kinds of effects. To prevent the impacts that complexity creates, you must reduce the causes of complexity.

Causes of complexity include:

- **Causes related to system design (the largest group of causes)**. Components that are internally complex add complexity to the system as a whole. Also, the interaction of the components (whether functional, data, or another kind of interaction) adds complexity. Dynamic behaviors also add complexity, including numbers of configurations and transitions among them. The way the system is modeled can add complexity as well.
- **Causes that make a system *seem* complex (i.e., reasons for cognitive complexity)**. These causes include the level of abstractions required, the familiarity a user or operator (such as the pilot) has with the system, and the amount of information required to understand the system.
- **Causes related to external stakeholders**. The number or breadth of experience of stakeholders, their political culture, and the range of user capabilities also impact complexity.
- **Causes related to the system requirements.** The inputs the system must handle, outputs the system must produce, or quality attributes the system must satisfy (such as adaptability or various kinds of security) all contribute to system complexity. In addition, if any of these change rapidly, that in itself causes complexity.
- **Causes related to the speed of technological change**. The added pressure that more capable, software-reliant systems place on technologies to accomplish even more also impacts complexity.
- **Causes related to teams**. The necessity and difficulty of working across discipline boundaries, and of creating process maturity in a rapidly evolving change, contributes to complexity.

Impacts of Complexity

Once a system is determined to be complex, no matter the reason, complexity will cause problems. Many consequences of complexity are known, including higher project cost and schedule, lower system performance (features, quality attributes such as adaptability), and lower development productivity.

We found that to address critical quality attributes (e.g., safety versus performance and usability), or to achieve a desired tradeoff among conflicting quality attributes, it is often necessary to make design decisions that increase complexity. For example, to reduce the probability of a hardware failure causing an unsafe condition, redundant units are frequently designed into a system. Then the system not only has two units instead of one, but it also has to have a switching mechanism between the two units and a way to tell whether each one is working. This functionality is often supported by software, which is now considerably more complex than it was in federated or single systems.

Complexity also impacts human planning, design, and troubleshooting activities. Complexity

- makes software planning more difficult (including software lifecycle definition and selection)
- makes the design process harder
  - For example, existing design and analysis techniques may fail to provide adequate safety coverage of high performance, real-time systems as they get more complex.
  - It also may be difficult to make a safety case just from design and test data, making it necessary to wait for operational data to strengthen the safety case.
- may make people less able to predict system properties from the properties of the components because emergent behavior arises
- makes it harder to define, report, and diagnosis problems
- makes it harder for people to follow required processes
- drives up verification and assurance efforts and reduces confidence in the results of verification and assurance
- makes changes more difficult (e.g., in software maintenance)
- makes it harder to service the system in the field

Complexity Measurement

Our literature review also identified what is currently published about the measurement of complexity. There are general measurement considerations such as properties of good metrics; there are specific recommended measures ranging from software design metrics up to top-down abstract measurement concepts; and there are a few measurement concepts in between, including branded characterizations of complexity that are or could be associated with measurement. These are being addressed and the focus narrowed in Task 3.3, which identifies candidate measures of complexity that could apply to software, systems, and avionics safety.

Complexity Mitigation

Our literature review also investigated the ways people mitigate complexity today. We identified three general principles in this area:

1. Assess and mitigate complexity as early as possible (for example, by doing virtual integration prior to building a system).

Distribution Statement A: Approved for Public Release; Distribution Is Unlimited

2. Focus on what in the system being studied is most problematic, abstract a model, and solve the problem in the model. For example, check for consistency in the interfaces of integrated models and simulations.
3. Begin measuring complexity early, and when sufficient quantitative understanding of cause–effect relationships has been reached (e.g., what types of requirements or design decisions introduce complexity later in system development), establish thresholds that, when exceeded, trigger some predefined preventive or corrective action.

Of course, these principles overlap and are expressed and sorted differently by different authors. Table 1 captures what the literature search identified as ways in which complexity is mitigated today. The concepts in this section will be used in work done in the second year of this project.

Table 1. Approaches for the Mitigation of Complexity

Build models whose simplifying assumptions enable answering specific questions or concerns
Diagramming (analogous modeling)
Integrate best-of-breed models from each constituent domain through a single bridging ontology
MBSE, MDE (model-based systems engineering, etc.)
Stakeholder viewpoint-generated graphs, diagrams, and views
Collaborate globally or crowd-source the solution to complex problems
Verify data abstraction and type early (early type checking vs. later verification)
Determine as much as possible statically, then dynamically analyze the rest
Use discipline in system development (clear objectives, processes, measures, planning, training, tooling) to reduce task complexity
Simplify as much as possible, but no simpler (per Occam's Razor) apply designs and justifications for claims
Address both functional complexity and run-time complexity, and do not mix them up
Address synchronization problems by designing in functional periodicity.

Conclusion

We have captured a large number of causes of complexity and impacts of complexity. These lay the groundwork for our future tasks, including defining possible metrics of complexity.

Looking Ahead

Our research is addressing these questions:

- What does the literature say about how complexity is defined and how it may apply to software, systems, and avionics? (This report)
- What metrics might apply to help quantify these kinds of complexity? (CMU/SEI-2015-SR-012)
- How does complexity affect the assurance of software and systems, and the development, validation, and verification of avionics software, hardware, and systems? (Upcoming report)

We will then identify which of our candidate metrics would best measure complexity in a way that predicts problems and provides insight into needed validation and certification steps.

Distribution Statement A: Approved for Public Release; Distribution Is Unlimited

Subsequent questions to be addressed by our research include the following:

- Given available sources of data on an avionics system, can measurement using these metrics be performed in a way that provides useful insight?
- Within what measurement boundaries can a line be drawn between systems that can be assured and those that are too complex to assure with reasonable confidence?

1 Introduction

This special report describes the results from the first task of a two-year project to investigate the impact of system and software complexity on aircraft safety and certifiability for the Federal Aviation Administration (FAA). The focus of the first task was a literature review of what is known about complexity.

1.1 Task Specification

The work plan described the task as follows:

> *… there is a need to ask both "What makes software complex?" and "What makes software complexity manageable?" This task will identify definitions of complexity from the literature of sciences (system science, complex systems, mathematics, computer science, etc.) and engineering (especially software and systems engineering, but also others), including current standards such as DO-178C, ARP-4754A or ARP-4761A, and critique them.*
>
> *Also, from the literature of best design practices for systems and software engineering, this task will identify standard ways of defining and reducing complexity and analyze which definitions are useful for which purposes. Also, from the literature about safety (including accident reports) and about certification, this task will identify the role of complexity in causing safety problems.*
>
> *Deliverables: White paper on the definition of complexity as it applies to software, systems, and flight safety.* [SEI 2014]

1.2 Literature Review Method

The initial method considered was a systematic literature review [Kitchenham 2007] to explore in what ways definitions of complexity were similar and different, toward developing a synthesis of appropriate aspects of multiple definitions. However, after an initial analysis of the recommended references, we discovered that very few references explicitly defined the term. So we sought instead to identify the particular concern addressed in the reference as a way of understanding how a reference's authors approached the topic of complexity. We thus embarked on a grounded theory approach [Glaser 1967] seeking to answer these generative questions:

- Which facets of complexity (artifact, task, or human response) are the focus of the reference?
- What definition (if any) is given? What measures are used or proposed (if any)?
- What causes are identified? What impacts?
- What ways are recommended for dealing with these causes and impacts?

The references to investigate were initially seeded with sources known to us, including those identified in the work plan, those suggested in the kickoff meeting or in emails with technical staff at the FAA, and one author's dissertation [Sheard 2012]. Sources included books, standards, journal articles, more recent research, and even, in one case, the popular press.

As these sources were reviewed, and as specific facets of complexity were identified, the volume of material grew larger and a taxonomy began to emerge. To keep the growing volume of text components, taxonomic categories, and classifications (mapping text components into the taxonomy) navigable and manageable, we used a software tool called NVivo. The grounded theory approach involves coding references against the taxonomy and, as the taxonomy evolves, recoding as needed to maintain coherent and consistent relationships between the references and the taxonomy. To communicate the results of the reference review and coding, customized reports can be generated from the tool that highlight a particular reference or portion of the taxonomy.

In cases where the nature of the source made it impossible to directly enter it into the NVivo repository (e.g., for a book not available electronically), we developed a précis of the document and entered that into NVivo, coding the relevant pieces of précis text.

The resulting taxonomy for the term "complexity" is provided in Appendix A.

Per grounded theory, as we read and categorized, we looked for other sources that would be needed to fill evolving gaps in our understanding of complexity and added them to the literature review reference list. We stopped the approach when the categories in the taxonomy were no longer rapidly changing as additional sources were added—that is, when the categories in the taxonomy generally sufficed to encode what incoming sources had to say about complexity.

Another aspect of grounded theory is the gradual emergence of one core category through which all other parts can be viewed in light of the original generative questions. The taxonomy has five main branches: Definitions, Causes, Impacts, Measures, and Mitigations. The relevant portion of our taxonomy that became the core is the second main branch, labeled "Causes." Why this particular branch? Because the ultimate goal is to help reduce problems caused by complexity, the most important aspect of the definition is what causes the complexity, so that such causes can be reduced.

For each subcategory under Causes, particular impacts might be identified, particular measures might be used to gauge significance, and mitigations might be introduced to reduce effects. Effort was thus spent in re-organizing Causes into a more usable and nearly orthogonal taxonomy. By "usable" we mean that given a piece of text describing some cause, it is possible to readily match that text to an appropriate category under Causes (which also requires that Causes be relatively complete). By "orthogonal" we mean that no or little overlap exists among the categories within the taxonomy.

Figure 1 shows the number of sources of different types that were ultimately used. Appendix B provides a bibliography of complete citations for the sources that were reviewed and coded, which are summarized in Table 1. Because the FAA specifically asked us to look at recent research, we added that line even though all of the references included in that category are also in one of the other categories.

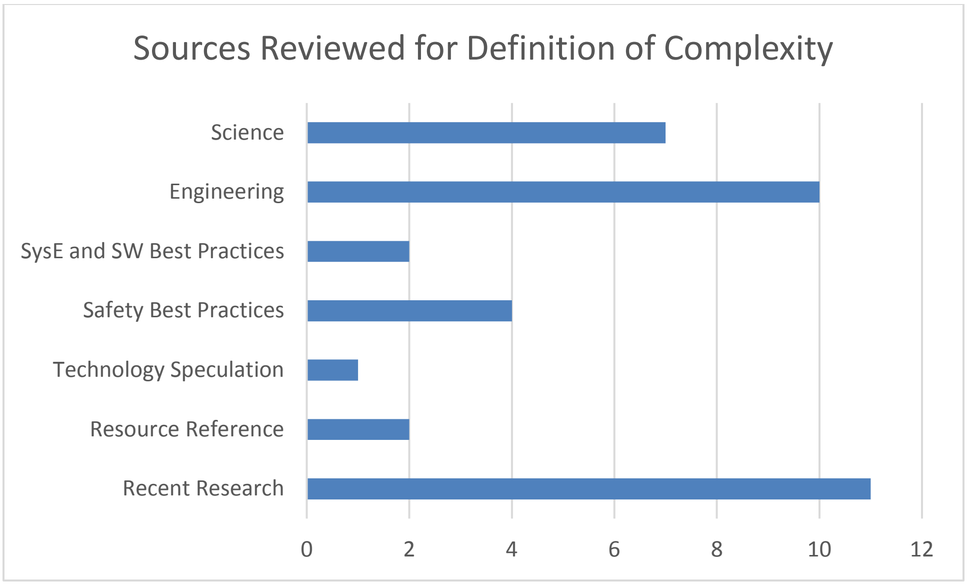


Figure 1: Types of Sources Reviewed

Table 1: Sources Reviewed[1]

Category	Year	Authors	Title
Science	1934	Albert Einstein	"On the Method of Theoretical Physics"
Science	1974	Amos Tversky and Daniel Kahneman	"Judgment Under Uncertainty: Heuristics and Biases"
Science	1995	Murray Gell-Mann	"What Is Complexity"
Science; Recent Research	2010	Jutta Mata, Peter M. Todd, and Sonia Lippke	"When Weight Management Lasts: Lower Perceived Rule Complexity Increases Adherence"
Science; Recent Research	2014	Robert Harper	Structure and Efficiency of Computer Programs
Science; Recent Research	2014	Michael J. Pennock and William B. Rouse (Stevens Institute of Technology)	"Why Connecting Theories Together May Not Work"
Science; Recent Research	2015	Shashank Tamaskar, Kartavya Neema, and Daniel DeLaurentis (System of Systems Laboratory, Purdue University)	Managing Complexity in Model-Based Conceptual Design
Engineering	1976	Thomas J. McCabe	"A Complexity Measure"
Engineering	1993	Robert L. Flood and Ewart R. Carson	Dealing with Complexity: An Introduction to the Theory and Application of Systems Science

1 Summary citations only—see Appendix B for complete citations.

Category	Year	Authors	Title
Engineering	1999	Eberhardt Rechtin	Systems Architecting of Organizations: Why Eagles Can't Swim
Engineering	1999	John N. Warfield	"Twenty Laws of Complexity: Science Applicable in Organizations"
Engineering	2005	Nam Pyo Suh	Complexity: Theory and Applications
Engineering; Recent Research	2010	Alan MacCormack (MIT Sloan School of Management), Carliss Baldwin, and John Rusnak (Harvard Business School)	The Architecture of Complex Systems: Do Core-Periphery Structures Dominate?
Engineering; Recent Research	2013	Sarah A. Sheard (SEI) and Ali Mostashari (Stevens Institute of Technology)	"Complexity Measures to Predict System Development Project Outcomes"
Engineering; Recent Research	2011	Douglas Stuart, Raju Mattikalli (The Boeing Company), Daniel DeLaurentis (Purdue University), and Jami Shah (Arizona State University)	META II Complexity and Adaptability Final Report
Engineering; Recent Research	2013	Dan Sturtevant	Technical Debt in Large Systems: Understanding the Cost of Software Complexity
Engineering; Recent Research	2014	Tilman Frosch, Christian Mainka, Christoph Bader, Florian Bergsma, Jörg Schwenk, and Thorsten Holz (Gortz Hortz Institute for IT Security, Ruhr University, Bochum)	How Secure Is Text Secure?
SysE and SW Best Practices	1995	Center for Systems and Software Engineering, Univ.of Southern California	COCOMO II Cost Driver and Scale Driver Help [COCOMO II 1995]
SysE and SW Best Practices	2010	CMMI Product Development Team	CMMI for Development, V1.3
Safety Best Practices	2010	SAE International	ARP 4754A Safety and Reliability Consideration in Aircraft/Systems Development [ARP 4754A]
Safety Best Practices	2012	Radio Technical Commission for Aeronautics (RTCA), Inc.	DO-178C Software Considerations in Airborne Systems and Equipment Certification [DO-178C 2012]
Safety Best Practices; Recent Research	2012	Cody Harrison Fleming, Melissa Spencer, and Nancy Leveson (MIT); and Chris Wilkinson (Honeywell Aerospace)	"Safety Assurance in NextGen and Complex Transportation Systems"
Safety Best Practices; Recent Research	2014	FAA	FAA, DOT/FAA/AR-xx/xx Draft Final Report for Software Service History and Airborne Electronic Hardware: Service Experience in Airborne Systems (2014)

Category	Year	Authors	Title
Technology Speculation	2013	Thomas L. Friedman, New York Times	“When Complexity Is Free”
Resource Reference	2014	Wikipedia	Complexity
Resource Reference	2014	Numerous online dictionary and reference sources	Common definitions of complexity

Approximately three months into this five-month project task, we developed a presentation package for internal SEI review. Based on the review comments, the taxonomy was updated, additional sources of information were sought, and these sources were coded and added to the repository. The updated taxonomy was then presented to the FAA about four months into the project. This report was written from that presentation.

2 General Observations

In the course of this literature search and analysis, we made some general observations regarding how complexity is addressed in the sources that we reviewed.

2.1 The Implicitness of Definitions

Our initial review of the literature revealed that few sources explicitly defined complexity. Many sources focused on particular causes or impacts, assuming that the meaning of “complexity” was clear rather than trying to define it. Thus, we refocused the literature review from gathering explicit definitions to zeroing in on what was being said about complexity. How was complexity being addressed in the sciences, best practices, and research?

As should be expected, the range of definitions that were implied by the sources was very broad. To answer the FAA’s question of how complexity affects aircraft safety and certification, it was important not to narrow this too rapidly, but to study aspects from the entire breadth of the definitions.

2.2 The Causal Chain for Complexity

Often complexity is approached in terms of the human or task impact that results from some chain of causal factors. General examples of impact are confusion or lack of ability to predict the future. FAA-related examples include inability to determine the state of a component, inability to determine how a system will fail, and inability to test a system exhaustively. Causal chains often begin with changes (in mission, environment, or technologies) to the requirements levied by stakeholders, or from the selections and decisions made during architecture and design. A causal chain often begins with the nature of the problem to be addressed and the design that addresses it, which must be correctly implemented. A causal chain can also begin with cognitive limitations and biases of the designer, with end users (who suffer from the complexity of the system when they operate it), or with demands of other stakeholders.

Distribution Statement A: Approved for Public Release; Distribution Is Unlimited

Causes of complexity related to ignorance about the domain or task were generally excluded from consideration by the sources, with the exceptions of the more process- and team-focused mitigations, such as those of Warfield [Warfield 1999].

2.3 Recursive Depth of Complexity

There is a recursive relationship between some quality attributes and complexity. For example, to meet a quality attribute related to system safety or security, a designer might introduce design constructs (components or connectors) that effectively increase complexity while introducing new threats, which in turn might require more design constructs.

2.4 Complexity in the System Development Lifecycle

It is possible to relate what the sampled literature says about complexity in terms of the system development lifecycle (including understanding requirements, design, implementation, use, and sustainment), thereby highlighting where complexity is typically introduced and removed.

Figure 2 shows that new requirements are introduced due to changes in organizational mission (e.g., broadening mission scope), environmental changes (e.g., addressing new threats or opportunities), new technologies (e.g., offering new mission or development capabilities), and user feedback on current deployed systems. These factors make design decisions and tasks more challenging. In addition, these factors can interact (a new technology might expand the scope of the mission, introducing new threats and opportunities) and may conflict. Some stakeholders might push for changes while others resist. Thus, stakeholder motivations should be understood during system development.

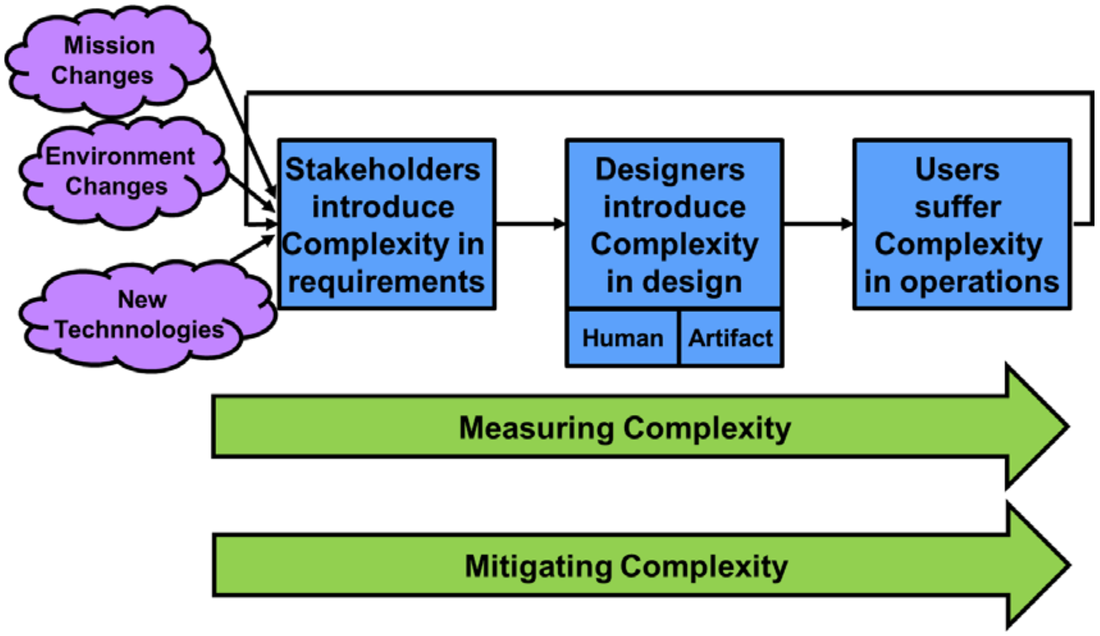

Mission Changes
Environment Changes
New Technnologies
Stakeholders introduce Complexity in requirements
Designers introduce Complexity in design
Human
Artifact
Users suffer Complexity in operations
Measuring Complexity
Mitigating Complexity

Figure 2: Complexity in the System Development Lifecycle

Architects and designers investigate implications of requirements and apply their experience to identify possible system solutions. They build designs or models of these solutions and test them. For some period of time, not only must the relevant requirements be understood, but the architecture, models, and designs intended to satisfy them must also be understood and related to the requirements in order for implications and risks to be understood. Clearly, architecture and designs may introduce new complexity while designers try to understand and address the requirements.

These factors can be measured (e.g., number of requirements, degree of design coupling), and these measurements can help in prioritizing development team attention. As we will see, it appears that errors are more likely to be committed when a team is confronted with complexity (especially when coupled with human fatigue or lagging motivation).

In general, user feedback can contribute to design complexity for the next version of the system. When users learn about and use a new system, implications of new features may be particularly hard to identify and understand. Some features may seem to promise new user capabilities but instead result in user misunderstanding or difficulty in learning the new system. And of course, the user's environment and personal or employer mission color the lens through which the user experiences the new system features.

For the FAA, the certifier is a user whose needs must be considered. It is necessary to determine a line that separates systems that can be certified from systems that cannot. The ability to make this decision is compromised for more complex systems. If there were metrics of complexity that related to the ability to assure software, then designers could design systems that fell below the cutoff line. For example, the system could be architected in such a manner that the safety critical portions of the system resided in a small set of design limits, so that understanding and sufficient testing could be accomplished in a feasible time frame.

In short, complexity can be introduced almost anywhere in the system development lifecycle, and some aspects can be measured and mitigated to help in planning and assuring adequate and early attention to the more challenging aspects of system development.

3 Taxonomy

A key result of this task is the taxonomy, an orderly classification of the different concepts of complexity. Our taxonomy is multilevel. The five main branches (categories) are

1. Definitions and characterizations of complexity
2. Causes of complexity – considerations leading to complexity
3. Impacts of complexity – things that happen when systems are complex
4. Measures of complexity
5. Mitigation of complexity

Distribution Statement A: Approved for Public Release; Distribution Is Unlimited

The taxonomy organizes the text components coded from the various references into categories and subcategories (which the tool we used, NVivo, refers to as "nodes"). Each category represents a subtopic of complexity. When reviewing a source, phrases or paragraphs that relate to a subtopic can be "tagged" with that category, which mechanically involves selecting a block of text with the cursor and then indicating which category that block of text belongs to. For example, in the following excerpt from Section 1 of the work plan, this quotation was selected for tagging:

> *The following are examples of such fields to be entered in problem reporting tools:*
>
> - *The consequence(s) of a fault;*
> - *The root cause(s) of a fault;*
> - *The functional structure of the software;*
> - *The means of evaluating an average execution time of the software functions or operational states, based on the software static and dynamic architecture.*
>
> *Note: the complexity of certain software components as well as the potential impact on timing of abnormal conditions may complicate the determination of such average operating time. However, new information is generally obtained anyway and despite being approximate, is usable.* [FAA 2014]

The quotation was tagged (coded) as "Problem reporting, definition, and diagnosis." The location of this category in the taxonomy is indicated in Table 2, and the category itself is highlighted in gray.

Table 2: Example of Tagging a Text Excerpt with a Taxonomy Term

A. <u>Definitions and Characterizations</u>
B. <u>Causes</u>
C. <u>Impacts of Complexity</u>
Adherence to a task's rules and requirements
Applicability (generalizability) of observations at a particular component level to other decomposition levels
Confusion
<u>Correlated outcomes</u>
<u>Design process</u>
Error possibilities increase, particularly for functions that are performed jointly across multiple systems
Making changes (e.g., in software maintenance)
Over-optimism (bias) in our plans and risk assessments
Problem reporting, definition, and diagnosis
Servicing the system in the field
Software planning (including software lifecycle definition and selection)
Unstable, nonreplicable, unpredictable behavior
Verification and assurance
D. <u>Measuring Complexity</u>
E. <u>Mitigating Complexity</u>

Table 2 shows only the portion of the taxonomy relevant to the placement of the category. The five main branches of the taxonomy are part of an ordered list, and we can see that the “Problem reporting, definition, and diagnosis” category is an immediate subcategory of the third main branch, Impacts of Complexity.

Categories, defined by the user, are hierarchically organized. Thus, when a user selects a parent category in NVivo, it can be opened to reveal its immediate descendent subcategories (representing subtopics of the topic that the category represents). Subcategories appear in alphabetical order. Thus in the portion of taxonomy shown in Table 2, only the main branch “Impacts of Complexity” is open to reveal its subcategories. Two of these subcategories have been underlined to indicate that they are parent categories that can be opened to reveal subcategories, a convention we adopt in the rest of this report. Note that subcategories appear alphabetically, as do the five main branches – enforced by prefixing each with the appropriate letter from A through E. An alphabetical ordering is used to help reinforce the reader’s understanding that there is no significance to the order of the subcategories under a category. When there is an intended order, the subcategories are numbered, as is the case with the main branches of the taxonomy.

The main sections of the taxonomy are described next, and the complete taxonomy, including all levels, appears in Appendix A.

3.1 Category A. Definitions and Characterizations

> *"Increasing Complexity and Coupling: Complexity comes in many forms, most of which are increasing in the systems we are building. Examples include **interactive complexity** (related to interaction among system components), **dynamic complexity** (related to changes over time), **decompositional complexity** (where the structural decomposition is not consistent with the functional decomposition), and **nonlinear complexity** (where cause and effect are not related in a direct or obvious way).*
>
> *The operation of some systems is so complex that it defies the understanding of all but a few experts, and sometimes even they have incomplete information about the system's potential behavior. The problem is that we are attempting to build systems that are beyond our ability to intellectually manage; increased complexity of all types makes it difficult for the designers to consider all the potential system states or for operators to handle all normal and abnormal situations and disturbances safely and effectively. In fact, complexity can be defined as intellectual unmanageability."* [Leveson 2011, emphasis added]

This quote from a well-known system safety author illustrates some of the types of complexity that we address in this report. Definitions for the bolded terms are included in Category A, aside from decompositional complexity, which fits better under system design in Category B6. Because of its specificity to aerospace and software, this quotation supports the need for this project. This quotation also mentions many of the impacts of complexity, which we address in Category C.

There are four main subcategories of Category A: About defining the term “complexity,” Branded characterizations, Aerospace standards and guidebooks, and Ordinary English. The first two require further explanation.

"About defining the term 'complexity'" is a selection of text components from the reviewed sources that discussed the problem of defining complexity. Among these are the many nuances of the various ways authors use the term. Because one effect of complexity is worker confusion and frustration, complexity can be confused with other phenomena and attributes. As a result, some authors take pains to differentiate their use of the term "complexity" from these other phenomena and attributes. While the distinctions made are interesting and somewhat useful, they are not of significant interest to the FAA project. Many uses of the term focus on complexity as something emergent from the interactions of many parts that is generally not easily inferable from a study of the parts or connections individually. This is the primary issue of interest to the FAA.

By "Branded characterizations" we mean that a particular notion of complexity has gained a practice and a following and therefore should be considered important. For example, many books and research efforts have explicitly focused on adaptive systems and cybernetics that address the issue of designing feedback control. Other branded characterizations that appear in the literature are more the studied visions of one individual (which sometimes continued to be pursued by others through both research and consulting practice) such as the Axiomatic Design Theory of Nam Pyo Suh [Suh 2005] and the Twenty Laws of Complexity of John Warfield [Warfield 1999]. We include several different characterization levels here, from single analytical measures such as cyclomatic complexity to design theories such as axiomatic design.

Associated with most branded characterizations is a collection of measures or a general approach to mitigating the causes or effects of complexity. Sometimes the form of mitigation appears idiosyncratic.

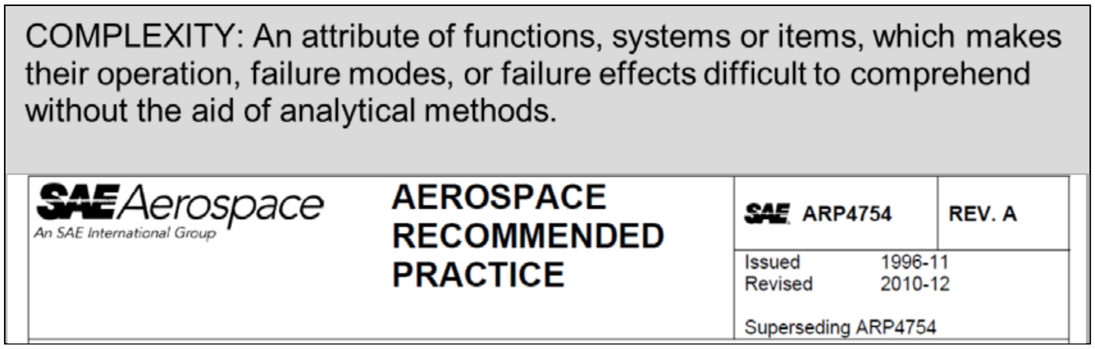
There are a total of 22 subcategories; most, but not all, are shown in Table 3. A few liberties are taken in presenting such excerpts from the taxonomy. Sometime taxonomic elements are truncated (e.g., "Kolmogorov complexity measures…") or left out entirely (e.g., the same taxonomic element has a child taxonomic element that has been omitted from Table 3).

Table 3: Subcategories for the Definition of Complexity

About defining the term "complexity"
General comments
In the mind vs. in the system being developed or used
Vs. chaos, nonlinearity
Vs. complicated
Vs. determinism
Vs. entropy
Vs. randomness
Vs. structured situations
Aerospace standards and guidebooks
Branded characterizations
Adaptive systems and cybernetics

Axiomatic design theory [Suh 2001]
Computational (# of resources consumed by most efficient algorithm as input size grows)
Cyclomatic number of control graph (see D. Measuring complexity)
Frustration at failing to understand a problematic situation (Warfield)
Kolmogorov complexity measures…
Organized complexity
Systems science (see Emergence under B. Cause, System Design)
Ordinary English

The final point to keep in mind about the broad category Definitions and Characterizations is that very few formal definitions for complexity appear. Figure 3 shows an example that presents one such definition.



COMPLEXITY: An attribute of functions, systems or items, which makes their operation, failure modes, or failure effects difficult to comprehend without the aid of analytical methods.

SAE Aerospace
An SAE International Group

AEROSPACE RECOMMENDED PRACTICE

SAE ARP4754	REV. A
Issued 1996-11 Revised 2010-12 Superseding ARP4754	

Figure 3: Example from Category A: Definition of Complexity from ARP4754A

3.2 Category B. Causes of Complexity

Causes of complexity are important because addressing them is the most fruitful way to reduce complexity.

This main category underwent the most redesign during the literature review task. It is a bit easier to tease out particular system factors that are causes when reviewing what particular authors advocate for measures and mitigations (main categories D and E, respectively). It is a bit harder to successfully identify causes when reviewing impacts (main category C), particularly when you consider the human impacts of complexity, which include confusion and fatigue. Therefore, early in the development of the taxonomy, we decided to explicitly include both system-related causes and human-related causes as the two principal subcategories:

- Complexity due to system factors
- Complexity due to human perceptual/social factors

The apparent partial orthogonality of two categories is appealing because, clearly, what is observed (complexity, in this case) is a function of both the system being observed and the perceiver. Eventually, however, we decided to redefine these subcategories into a larger number of subcategories to reduce vagueness. Six cause-related categories were the final result:

1. Cognitive and psychological
2. Stakeholders (external to development team)
3. Requirements
4. Technology change and trends
5. Development team
6. System design

We must elaborate on an earlier observation before discussing these subcategories in turn. The Causes main category increasingly became the core of the taxonomy, in the sense that the term "core" is used in grounded theory: the core consists of the theory (the set of concepts and relationships) that are perceived to be fundamental to understanding the phenomena at hand—complexity, in our case. That Causes could be perceived as core should be no surprise because, as already noted, efforts at measuring or mitigating complexity must begin with some understanding of how complexity arises, and there is a tendency to focus on certain causes that are considered to be of primary interest and to design measures and mitigations around them.

Category B1. Cognitive and psychological causes of complexity

Three considerations affect a task's or system's cognitive complexity:

- ability to reason about a situation requiring multiple abstractions and deductions from cause–effect relationships
- limited opportunities to learn about the regularities in the domain or system
- perceived cognitive complexity of the task (i.e., of a task's rules and requirements, such as what information to gather, how to process that information, and when to process it)

These considerations relate to each other. As Kahneman and Klein have pointed out, human expertise in a particular domain arises from the ability to learn the regularities, which we interpret to be the cause–effect relationships in that domain and, in particular, the effects of human interventions and reasons for those interventions [Kahneman 2009]. Sometimes, the ability to learn such regularities is challenged from the get-go because there are so many such relationships and interacting agents within the domain that it is simply not possible to easily perceive any causality. Another source of challenge to learning regularities is the amount of processing that needs to happen to obtain visibility into those regularities (the third bullet). Without the ability to learn regularities, prediction becomes nearly impossible, or at least, there is very qualified assurance of any predictions.

Category B2. Complexity as a result of stakeholders external to the development team

Four general considerations increase complexity that is due to external stakeholders. Of these, the second one (Political) has a single subcategory.

Distribution Statement A: Approved for Public Release; Distribution Is Unlimited

- Number of stakeholders
- Political
    - Number of layers of management, cultures
- Social context, culture
- User capabilities

Complexity due to stakeholder-related sources appears in the number of requirements, their sometimes conflicting nature, the range of ontologies and values they draw from, the apparent or real contradictions among the requirements, and their volatility.

Mitigations include increasing your understanding of the stakeholders—their likes and dislikes and what drives them—and, sometimes, bringing the various stakeholders together in a joint effort to develop and agree to the requirements. This extends ownership of the result—and the attendant feelings of responsibility and accountability—to the stakeholders.

There may also be multiple end-user types. Mitigations include developing use cases and user personas and referring to these to gauge the effectiveness of the proposed solution through requirements development, design, implementation, testing, and sustainment phases. In addition, multiple types of requirements validation can be pursued throughout system development, and not just relegated to user acceptance testing, when it is generally too late to deal with misunderstood or missed requirements.

Category B3. Complexity as a result of requirements

Quite a few types of considerations related to system requirements can cause complexity, as would be expected. At a top level, these include how much complexity there is in the required system inputs (e.g., in extracting a signal given an unfavorable signal-to-noise ratio) or outputs (e.g., complex, choreographed process control outputs), safety related requirements (e.g., built-in test, redundancy management), a large number of requirements, the number and types of systems that need to be integrated, how much the system is changing (e.g., requirements volatility), and the required quality attributes that the system must satisfy.

The resulting taxonomy for Complexity as a result of requirements follows:

- Input or output complexity (required of the system)
- Number of requirements
- Quality attributes required of the system
    - Adaptability of solution is critical
        - Designed to adapt to changes in its environment (or stakeholder purpose)
        - Product line focus (design for modularity and variability)
        - Reliability needed in solution
        - Robustness needed (instead of highest peaks, look for highest mesas, where performance does not fall off sharply due to small variations in design variables or noise)
        - User-modifiable software (software designed to be modified by its users)
    - Performance of solution is critical

        - Parallelism needed in solution
    - Safety of solution
    - Security of solution (availability, integrity, confidentiality) is critical
    - Systems to integrate with
- Volatility
    - Evolution of system purpose
    - Increasing need to integrate with other systems
    - User role is changing from direct control to a more subtle relationship based on increasing dependence on a system that automates many operational functions and responsibilities

The Quality attributes subcategory deserves more discussion. Quality attributes (also called “ilities”) are notable sources of complexity, driving up the amount of development and assurance case effort. In today’s increasingly interconnected world, it can be very challenging to design a system to be resilient to outside disturbances and malicious attacks. If there are stringent performance constraints, the degree of design challenge can rise enormously. The need to achieve a particular performance is often at odds with the need to achieve security and safety, because the latter are often achieved by adding additional processing elements (sometimes software and hardware) that can slow or interfere with achieving rigid process control deadlines.

Even leaving performance considerations aside, any design activity focused on achieving security and safety often results in a more complex design. Thus, we seem to have a paradox: to make things safe and secure, we add more design, which results in yet more complexity that often introduces new questions regarding how the system is able to address safety and security, which are answered with more design and more verification, in what might seem like an endless loop. At some point, of course, designers stop, and they accept a certain level of complexity and the attendant difficulties of explaining how a system works and why it is safe and secure—to some level of confidence.

Often, the causes for security or safety failure can be found in a premature de-scoping of either who the system stakeholders are (and likewise of the system operational environment) or of the timeline for the system [Konrad 2013]. However, expanding the scope also increases complexity in some sense. It may increase the complexity of stakeholder makeup, or it may affect our ability to reason about how the system will behave and need to evolve (i.e., its adaptability). Note that adaptability is itself a quality attribute that often needs to be considered, and which is linked to our ability to address category B1, Cognitive and psychological causes of complexity. Such broader and longer views come at a price, and future benefits from expanding the scope may be difficult or impossible to discern. Common sense and incentives reward some breadth and longer time horizons when developing requirements such as security and safety so that they can be more credibly assured.

Category B4. Complexity as a result of technology change

As technology advances, more capability becomes possible; therefore, systems with more features and less undesirable behavior are sought. Whether fortunately or unfortunately, the additional capability that technology brings is a new source for complexity in several ways:

- Technology improvements allow development of broader solutions demanded by the market, mission, or societal norms and pressures.
- The increasing capability of software (and underlying hardware and development environments) allows systems to provide broader and more sophisticated operational capabilities.

If the effective use of a new technology becomes critical to a particular organization, that organization will often establish a cadre of specialists (by hiring, recruiting, or mentoring) whose job it is to track the emerging technology, understand it, and determine how its promise can best be realized in different user and product settings. This in turn introduces a new source of complexity: how to apply the evolving understanding of that technology effectively in the development of new systems and system concepts.

Category B5. Complexity resulting from development team behavior or issues

When an equipment manufacturer presents a package for certification, how that package was developed can introduce additional complexity. Often, the development of a system draws on experts or specialists in different application and technology domains. The development team must therefore be capable of effectively working across multiple domain or disciplinary boundaries and in different physical locations. Mastering the formation and support of multidisciplinary teams is especially important when issues and risks require the attention of multiple specialists or team members to jointly better understand how the system should behave and perform (category B1) and to jointly pursue creative solutions (categories B2–B4). The team members' jobs become more difficult as they confront multiple aspects (or dimensions) of complexity of the problem. In such situations, particularly when developing unprecedented systems or interacting systems whose interfaces and behavior cannot be fully understood or trusted, it may be impossible to determine in advance the boundaries between what must be rigorously investigated and what does not need to be known. Thus individual areas of study may be chosen somewhat haphazardly, and the team runs the risk of not identifying a potentially superior solution.

These difficulties are compounded when the team's process capabilities (whether development or manufacturing) are insufficient to meet specific system parameter tolerances [Suh 2005] such as dimensional tolerances; when the team is distributed geographically and must rapidly determine how it will manage both design and production (particularly when partners and suppliers are distributed across the globe); and when a team requires approval of multiple layers of management to obtain needed resources. For designers, all of these contribute to uncertainty, perceived complexity, and, of course, the risk of failure.

Finally, team conflicts and people issues can beset a team that otherwise would be able to productively pursue a solution.

The relevant part of the taxonomy is listed below in detail, indicating causes of complexity:

- Analyses (of system behavior) that necessarily cross multiple domains (e.g., physical vs. social, software vs. hardware)

- Dealing with poorly structured situations (when it is difficult to identify and understand all relevant factors or system dynamics)
- Lack of experience designing and implementing similar systems
- Over-reliance on an unproven technology (whether for a product, development process, or development environment)
- System parameter tolerance vs. capability (voice of customer vs. design [or process] capability) (will be different for different system parameters)
- Team structure (organization of and access to needed capabilities)
  - Learning curve
  - Managing overlap in system design and production across a global network of partners and suppliers
  - Number of layers of management
- Uncertainty (designer side)

The last subcategory, Uncertainty (designer side), is a catch-all for the many development-team-related uncertainties that a designer must deal with. What is the team's capability? Who should be on the team, and who should provide support to the team when needed? How much cross-training should the team receive? How are team members' knowledge of particular stakeholders and their needs developed? How about their knowledge of technologies that may prove key to a superior solution?

Items that did not make it into our taxonomy but could extend the Uncertainty (designer side) subcategory include the following: Are there use cases that have significant architectural significance (e.g., due to unrecognized impact on other quality attributes), and how robust is a solution technology or approach to meeting the unknown or emerging needs of particular stakeholders?

Category B6. System design complexity

System design complexity is the largest subcategory of the taxonomy's main branch and indeed constitutes the most likely contributor to both measurement and aircraft complexity reduction. Of the final 234 total taxonomic categories, 41 are in System design complexity. Its dominance is largely due to the efforts of many researchers to understand how the complexity of a system's design affects other parties and impacts downstream processes, such as verification and, eventually, system behavior and utilization. Within the entire system development lifecycle, design is the one activity and artifact that the system architect has the most control over—almost everything else can be considered as the context in which the designer must work out a solution, or the result of the particular design (e.g., what test cases or assurance arguments need to be developed). Thus, at least within the literature sampled, much attention was given to the complexity of design as an activity but even more so to design as an artifact. In some sense, just as Causes is the core of the taxonomy, System design complexity is the core subcategory of Causes.

Here is the System design portion of the complexity taxonomy in some (but not full) detail

- Component internal complexity (e.g., control structure complexity)
  - Algorithmic

  - Control structure
  - Data structure
  - Numeric
  - Operations at physical IO level
- Dynamics (dynamics of how the system behaves during its operation)
  - Operational modes/state machine transitions
  - Number and variability in policies or rules governing system behavior
  - Number of system configurations
  - Off-design configurations
- Functional coupling complexity (how system components interact to achieve requirements)
  - Cohesion
  - Coupling
  - Data flow complexity
  - Emergence
  - Feedback loops
  - Interplay between subsystems
  - Nonlinearity
- Representational complexity (with what languages, abstractions, vocabulary, fidelity [accuracy, precision], and formality the system requirements, designs, and implementations are expressed)
  - Structural complexity

The first category within System design (which is within Causes) is Component internal complexity (e.g., algorithmic complexity). A component's complexity can be due to its algorithm (Kalman filters are one example, but there can be much more complex algorithms, such as Karmakar's algorithm) but also by implication due to the associated mathematical proof of its soundness and other properties. In fact, multiple fields of mathematics have been developed to explore solutions to fundamental problems that can be translated into computer algorithms (e.g., numerical analysis, nonlinear differential equations) as well as statistics (e.g., logistic regression, Bayesian belief networks, and Monte Carlo methods).

Much of the literature on system design focuses on either the static view of how design components are put together, communicate, and interact, or else on the dynamic view of how the system will behave (time-dependent behavior). (Note that "interaction" includes problems that arise from integration.) There is less focus on the complexity within the components themselves for several reasons. First, a common view is that the architect starts with a palette or toolkit of components to select from or a set of patterns to select from and instantiate, and these are often the focus of the system design literature. However, individual components still must be understood, often to a significant degree, in order to understand how, once included in the system, the overall system's behavior might change. Second, components can be discipline specific, and often techniques developed for one discipline (e.g., for hardware reliability) do not perform well in another discipline (e.g., for software). This part of the taxonomy emerged primarily from our experience with software complexity, but every subtopic in this category was found in the literature.

The second subcategory, Dynamics (dynamics of how the system behaves during its operation), shares some of the same challenges of understanding how organisms behave through study of particular cell types and specific organs: common to both is a need to predict how a collection of components whose individual behaviors are known can produce through interaction a more complex repertoire of behaviors—some of which may be strongly undesirable.

As the number of operational modes or state machine transitions for a system rises, so often does the challenge of understanding and predicting how that system will behave. The possible number of paths through a state machine can be huge, creating a challenge for the designer wishing to demonstrate that particular unsafe system states can never be reached. Mathematics and statistics offer useful abstractions here, as do testing and simulation, but nevertheless our reasoning capabilities are clearly challenged.

The third subcategory, Functional coupling complexity (how system components interact to achieve requirements), is particularly focused on the presence of patterns of how the components are configured to communicate and interact. Perhaps the best-known of these is coupling. An example of coupling is when two components each require the same resource to accomplish their task. Resources could include a database (or global variable), a bus (e.g., CANbus), power, memory, or processor time. Contention for the same resource often must be considered, especially when component performance (or more broadly, system performance) is critical, and these situations can be quite challenging to reason about. An example is using a multi-core processor where the cores share cache, internal interconnects, peripherals, and a controller for external memory: it is very difficult to analyze resource contention, execution times, or functional interference unless the resource allocation is fixed prior to run time; however, this would reduce flexibility and is not often done.

Similar statements can be made about the challenges raised by feedback loops, which make impossible the straightforward reasoning of a system's behavior by considering what each component does to those downstream (in the direction of dependencies). The Interplay between subsystems (one of the bullets of this subcategory) is a kind of coupling and feedback-loop-like situation when regarded at a higher level of abstraction.

To some extent, you can reason about data flows through a system without feedback loops in exactly the way described above: follow an input through the system, tracing out its data path, and follow the direction of dependencies. However, such data paths can become long, they may have multiple branches, and—even more challenging when dealing with periodic processes along the path that have rigid execution times—there can be time lags that make it difficult to predict data currency, especially when periodic processes that the data path flows through must contend with the same processor for their execution. All of these system characteristics introduce a source of complexity, even in the absence of feedback loops.

Strogatz points out that it is possible to predict how linear systems will behave because the mathematics is particularly tractable and you need only to identify the equations correctly and solve them [Strogatz 2004]. But it is an entirely different matter for nonlinear systems, where a small perturbation on an input might not translate to a similarly small perturbation on the output, and there is high uncertainty on how such a system (often also characterized by having high coupling and feedback loops)

Distribution Statement A: Approved for Public Release; Distribution Is Unlimited

will behave, particularly for a significant distance into the future. Suh even advocates a particular pattern when designing such systems: introduce a functional periodicity in the system so that the system resets back to its initial state with regularity [Suh 2005]. Strogatz, Suh, and others point out that many organic systems are periodic perhaps just for this reason: it allows a stable set of dynamic behaviors to arise with predictability within a well-defined time horizon. For example, the human brain gets access to sugar (through regular eating cycles) and rest (through regular sleep cycles). These cycles reset the brain's ability to harness cognitive resources toward situational awareness, communication with others, and problem solving. Behavior of cognitive functions becomes compromised when these cycles are not present.

The fourth subcategory, Representational complexity, acknowledges that sometimes the very terminology and diagram notations that we use—and their consistency—get in the way of our correct reasoning about a system architecture and understanding what it will do. There has been research into whether integrating multiple ontologies might provide a way forward (some conclude "no" [Pennock 2014]). But there are also issues of model fidelity and whether the notations are sufficiently precise to admit some degree of automated reasoning or reasoning support to sufficiently extend human reasoning capabilities as needed to address more layers and iterations of designs consisting of hundreds or even thousands of components.

The fifth subcategory, Structural complexity, simply focuses on the number of items that must be attended to (e.g., cognitively or in verification) in order to successfully reason about a system. This category includes the issue of heterogeneity—we can sometimes extrapolate how a system behaves from an understanding of how a subset of identical components identically connected will behave, but when there are multiple types, complexity rises enormously (consider how easy it is to learn checkers (or draughts, as the British call it) compared to chess.

The sixth and final subcategory, Uncertainty (system side), is, like the similarly-named Uncertainty (designer side) subcategory, a catch-all for the many uncertainties that a designer must deal with, but here the source is what goes into a design. Often, there are uncertainties about particular components or connections, the overall pattern of connections, or possible unidentified interactions within a system (undiagnosed coupling)—all of these can make reasoning about a system's future behavior very challenging.

3.3 Category C. Impacts of Complexity

Once a system is known to be complex for whatever reason, what problems or benefits does the complexity cause? Many consequences of complexity are known and considered negative. Sheard tested whether any consequences of project complexity variables could be considered positive; the answer was that all project outcome variables showed poorer results with higher complexity [Sheard 2012].

However, as we have noted, a common consequence of trying to address critical quality attributes, or to achieve a desired tradeoff between conflicting quality attributes (e.g., safety vs. performance and usability), is additional design complexity. As an analogy, organisms have multiple complex mechanisms involved in metabolizing the food they eat that get variably engaged depending on their overall physical health. There is evidence that complexity can be beneficial to organism survival. On the other hand, that same complexity provides a larger attack surface for a problematic response to particular environmental insults (e.g., sugar metabolism engages a surprisingly long and branchy pathway with feedback loops that results in possible long-term harm from too much fructose or other forms of sugar [Lustig 2006]).

The following categories[2] characterize the types of consequences that complexity can have.

Human activities (perhaps tool-assisted) that complexity makes more difficult:

- adhering to a task's rules and requirements when their complexity is high
- predicting results for a system given knowledge of its components
- the design process, as the existing design and analysis techniques may fail to provide adequate safety coverage (or assurance) of high-performance, real-time systems as they get more complex
- making changes (e.g., in software maintenance), because potential impacts on other parts of the system are not well understood
- problem reporting, definition, and diagnosis, because of the many possible causes that may have led to the problem
- servicing the system in the field, for several reasons, including the large number of changes that might need to be made and the difficulty maintainers have in understanding the system
- software planning (including software lifecycle definition and selection). If it is not harder, it is at least less likely to be correct, because many variables are unknown early.
- verification and assurance, which become more time consuming and costly, with less confidence in the results of verification and assurance

Problems that complexity can cause include the following:

- confusion
- likelihood of error, particularly for functions performed jointly across multiple systems (a function that must work in different contexts may be especially difficult to analyze)
- over-optimism (bias) in plans and risk assessments
- unstable, nonreplicable, unpredictable behavior

Complexity has been shown to correlate negatively with the following:

- adaptability to change
- productivity
- project effort, cost, and schedule performance
- firm performance and survival
- system performance

Complexity has been shown to correlate positively with the following:

2 The wording and organization of this section of the taxonomy has been improved after the initial baselining of the taxonomy as it is presented in Appendix A, which provides an earlier version.

Distribution Statement A: Approved for Public Release; Distribution Is Unlimited

- defect ratios and rates
- staff turnover (i.e., lower staff retention)

We end with a caveat: a few of the reported correlations may be domain specific. Some aspects of complexity typical of systems developed for one domain may not have the same consequences for other domains [Stuart 2009].

3.4 Category D. Measuring Complexity

A complexity measure should be computable and scalable, predict something, and be used to make a decision. Thus, characterizing how complexity is measured in the literature should include the measure's purpose, definition, and how it will be used. We have tried to capture some of what the literature has to say about complexity measurement. However, the next project task addresses measurement in particular, and we thus defer most of the discussion about what measurement is and how to identify good measures of the concept we term "complexity" to that task.

The following categories partially characterize the literature on complexity measurement:

- General topics about measurement
  - Take a lifecycle complexity view (architecture, design process, and manufacturing process)
  - Scalability (in particular, computability of the measure on a graph representation, as the number of nodes and links increases)
  - Properties of a good complexity metric
  - How to select which measures to use
  - Calibration of measures to determine best predictors
- Specific potential measures
  - *Assessing the Impact of Complexity Attributes* top 3 and top 20 [Sheard 2012]
  - CoSysMo model
  - Stuart measures (DARPA META II report)
  - Warfield
  - Software-specific measures such as cyclomatic complexity, function points, and defect count and density
- Measures computable from design
  - From DSM [Stuart 2011]
  - Dynamic complexity [Gell-Mann 2001]
  - Modularity [Stuart 2011, Tamaskar 2014, Zeidner 2010]
  - Visibility [McCormick 2010, Sturtevant 2013]
- Measures related to designer willpower [Warfield 1999, Flood 1993]
- Measures related to the ability to follow and comply with regimens and processes [Mata 2010]

Several references discussed how to measure complexity. Sheard discusses a research project in which a broad range of potential factors of system development project complexity were evaluated by means of a retrospective survey of senior systems engineers [Sheard 2012]; some of these factors correspond to measures. Also, some useful measurement frameworks exist (e.g., CoCoMo and CoSysMo) but do not measure complexity per se. More theoretical measures of complexity exist but are not easily interpretable for system or software development projects [Gell-Mann 1995, Suh 2005]. Software-related measures exist that so far cannot be extended to or be useful for analyzing non-software solutions [McCabe 1976, Harper 2014].

There are two caveats associated with Category D, Measuring complexity:

1. Complexity is a feature of the representation as much as it is a feature of the system; hence, it is hard to measure because representations vary by view and language (see note about Representation complexity, a category under System design complexity).
2. Measurement is a field in its own right. A classic measurement paper not included in the literature review is Basili's Goal, Question, Metric approach to defining useful measures [Basili 1994].

3.5 Category E. Mitigating Complexity

Of course, the desire is not only to identify and measure complexity but to mitigate the causes and impacts described in Categories B and C. What can be done to mitigate the causes and impacts of complexity has been classified in this literature review as Mitigating complexity. There are many subcategories related to this topic. You could make the case that, in general, all practices of systems engineering started from a principle of managing complexity [Hall 1962, Eisner 2005]. Some of the practices mentioned in the literature we searched are listed in Table 4.

Table 4: Subcategories for the Mitigation of Complexity

Build models whose simplifying assumptions enable answering specific questions or concerns (often domain specific)
Diagramming (analogous modeling)
Integrate best-of-breed models from each constituent domain through a single bridging ontology
MBSE, MDE (model-based systems engineering, etc.)
Stakeholder viewpoint-generated graphs, diagrams, and views
Collaborating globally or crowd-sourcing the solution to complex problems
Data abstraction and type systems (move complexity into type declaration and checking vs. later verification)
Determine as much as you can statically; dynamically analyze the rest
Discipline in system development (clear objectives, processes, measures, planning, training, tooling) to reduce task complexity (e.g., clarify task completion)
Everything should be made as simple as possible, but not simpler (Occam's Razor); applied to designs and justifications for claims
Functional abstraction (separate functional complexity from run-time complexity)
Functional periodicity: introduce (functional) periodicity in system design with time periods for system re-initialization to reduce time-dependent combinatorial complexity (increases system reliability and predictability)
General Comments
Incremental assurance (establishing confidence that implementation achieves intention)

Assurance of a system produced by multiple parties; first, ensure overall design will not lead to hazard, then generate specific requirements for each component provider
Assurance techniques (process assurance, validation and verification coverage criteria, analyses) improve visibility of what is or was done and why
Certifying the algorithms used
Evaluate specific attributes of requirements, architecture, and configurations to determine what in-service data is needed for safety certification
Proof of correctness
Interactive management (Warfield 20 Laws of Complexity focused on individual, team, and organizational pathologies)
Interpretive social theory and critical thinking
Mathematically or statistically characterize what you can (e.g., interactions, protocols, data structures, and algorithms)
Modular design (localize impacts and maximize readiness to changes in demand; separate extensional from intentional)
New technology radically shortens test lifecycles (3-D printing, quantum computing)
Optimize (what you can), build for adaptability (where you can), then hedge against multiple futures (in that order)
Prognostics (rather than diagnostics) through sensors, Internet of Things, and Big Data
System boundary identification
Systems engineering (also see MBSE, under E. Mitigating Complexity, Build models…)
Systems thinking-based analysis of system behavior and system design
Use design-based measures early in the lifecycle that are broadly applicable, objective, computable, and validated to guide development
Evaluate design alternatives (or proposed changes) using domain and company-calibrated design measures
Identify design features, patterns, and rules for high performance that have low impact on complexity
Refactor design to reduce complexity and improve modularity (when a complexity measure exceeds some threshold)

Three general principles for mitigating complexity emerge from these subcategories:

1. Assess and mitigate complexity as early as possible.
2. Focus on what in the system being studied is most problematic, abstract a model, and solve the problem in the model.
3. Begin measuring complexity early, and when sufficient quantitative understanding of cause–effect relationships has been reached (e.g., what types of requirements or design decisions introduce complexity later in system development), establish thresholds that, when exceeded, trigger some predefined preventive or corrective action.

Of course, these principles overlap, and to some extent, different authors focus on different stakeholder roles or similar mitigation approaches and thus develop different, though perhaps similar-sounding, recommendations. Additional principles, such as using a safety kernel, have developed specific to software-based systems.

Distribution Statement A: Approved for Public Release; Distribution Is Unlimited

The topic Mitigating Complexity also has its own task in this research project, although its completion is listed as Optional, given allowed time and funds (Design Guidelines, Task 3.9).

Caveats

As has been noted, grounded theory is an ongoing approach that theoretically never ends.[3] The categories given here are not fully orthogonal, even for the core parts of the taxonomy (Causes, System design). For example, "Interplay between subsystems" appears under both B5 and B6.

Often causes and impacts interact. For example, the effects of fatigue and frustration brought about by a first task's complex rule set can be amplified when confronting a second task whose rule set is likewise complex.

Both causes and impacts can be measured, and so can the impacts of mitigations. You would need to know the purpose of the measures (what will be done with them?) and how to reliably measure them with sufficient precision. This is the focus of the next project task.

4 Next Steps and Conclusion

4.1 Next Steps

The task described in this report was Task 3.2 (Task 3.1 is Project management). Task 3.3, which has begun, is "Identify Candidate Measures of Complexity." This task will, according to the work plan,

> *Identify candidate measures of complexity that apply to systems with embedded software and relate to safety, certification, or both. The task will:*
>
> *a. Identify complexity measures that have been used for systems and software, along with general rules for measuring complexity in practice (e.g., number of global variables, uses of inadequate data types, and uses of concurrency mechanisms).*
> *b. Propose a list of common failures in the context of IMA systems (e.g., use of inappropriate design patterns) and how to cope with them (e.g., code analysis, architecture analysis).*
> *c. Use available data sources to determine whether values of these potential measures can be known or estimated for complex systems.*
> *d. Identify the criteria by which complexity metrics will be chosen from the list of potential metrics.* [SEI 2014]

The deliverable for Task 3.3 is specified as:

> *White paper that lists root causes of avionics system complexity and proposes potential metrics to measure software and system complexity relevant to software system safety, including their*

3 The potentially never-ending aspect of grounded research is mentioned here: http://www.socialresearchmethods.net/kb/qualapp.php.

Distribution Statement A: Approved for Public Release; Distribution Is Unlimited

origin, strengths and weaknesses, potential for use as metrics for the purpose of system safety and certification, and criteria for selection [SEI 2014]

The plan for accomplishing Task 3.3 includes the following main steps.

- Identify a broad set (~100) of candidate measures, from literature and experience.
- Identify root causes of avionics system complexity.
- Group lists of candidate measures into categories.
- Perform first down-select (to 25~50) for plausibility.
- Identify criteria for second down-select.
- Build measurement specification.
- Rate selection criteria against specification.
- Perform second down-select for feasibility, availability of data, and prediction ability.
- Perform final down-select for importance and non-redundancy.
- Draft the report.
- Complete reviews and revisions.

Task 3.4, which will start before the end of Task 3.3, is "Identify the Impact of Complexity on Safety." This will finish out the first year of the two-year research project and result in a draft working paper to meet the following requirements:

Identify the impact of complexity on aircraft certification, V&V, and flight safety margins, including reductions in margin occurring because complex system V&V is more problematic. Through interaction with FAA technical staff members, determine top-level impacts that complexity has on issues of concern to FAA.

Deliverable:

A draft working paper that begins to prioritize issues of certification, V&V, and flight safety as relates to the problems that are caused by complexity [SEI 2014]

4.2 Conclusion

In this systematic literature search, we broadened the task from simply collecting definitions to describing a taxonomy of issues and general observations associated with complexity.

Although the literature did not provide crisp definitions of the term "complexity" from which to select a small number of definitions, the readings provided a rich canvas of concepts and approaches that will serve as a solid foundation for defining candidate measures and identifying impacts on flight safety.

The definition of complexity that we will use on this project, which we may evolve as the project progresses, is:

Complexity is that which results from large size, extensive coupling, and behavioral emergence, and negatively impacts the understanding of stakeholders, designers, and users.

Our literature review revealed that complexity is a state that is associated with *causes* that produce *effects*. To prevent the impacts that complexity creates, we must reduce the causes of complexity.

Causes of complexity include:

- **Causes related to system design (the largest group of causes)**. Components that are internally complex add complexity to the system as a whole. Also, the interaction (whether functional, data, or another kind of interaction) of the components adds complexity. Dynamic behaviors also add complexity, including numbers of configurations and transitions among them. The way the system is modeled can add complexity as well.
- **Causes that make a system *seem* complex (i.e., reasons for cognitive complexity)**. These causes include the level of abstractions required, the familiarity a user or operator (such as the pilot) has with the system, and the amount of information and steps required to understand, reason about, or operate the system.
- **Causes related to external stakeholders**. The number or breadth of stakeholders, their political culture, time horizons, and range of distinct capabilities also impact complexity.
- **Causes related to the system requirements.** The inputs the system must handle, outputs the system must produce, or quality attributes the system must satisfy (such as adaptability or various kinds of security) all contribute to system complexity. In addition, if any of these inputs, outputs, or quality attributes change rapidly, that in itself causes complexity.
- **Causes related to the speed of technological change**. The demands on the stakeholder community, organization, and project to explore and mature new technologies and determine what roles they should play in system solutions also increases complexity.
- **Causes related to teams**. The necessity and difficulty of working across discipline boundaries, and of communicating and coordinating work in a rapidly evolving development environment, also contributes to complexity.

Once a system is determined to be complex, no matter the reason, complexity can cause problems, or impacts.

Impacts include higher project cost and schedule, lower system performance (features, quality attributes such as adaptability), and lower development productivity.

We found that design complexity is sometimes increased by a desire to improve critical quality attributes (e.g., safety or reliability). For example, to reduce the probability of a hardware failure causing an unsafe condition, redundant units are frequently designed into a system. Then the system not only has two units instead of one, but it also has to have a switching mechanism between the two units and a way to tell whether each one is working. This functionality is often supported by software, which is now considerably more complex than it was in federated or single systems.

The causes and the impacts are comprehensively described in the taxonomy in Appendix A.

The next steps consist of determining which causes and which impacts to approach with measurement, as well as standard measurement questions such as predictivity of measures and the use of proxies to

Distribution Statement A: Approved for Public Release; Distribution Is Unlimited

approximately measure concepts that are not concrete, followed by selection of which causes of complexity have effects that relate to aviation safety and certifiability. These steps are within the scope of this research project.

Appendix A: Complexity Taxonomy

The Complexity Definition Literature Review Task resulted in a taxonomy of topics related to the definition, causes, impacts, measurement, and mitigation of complexity. The major part of this report characterizes this taxonomy, which can be updated as desired to accommodate new references. Below, the taxonomy appears in full.

A. Definitions and Characterizations
About defining the term *complexity*
General comments
In the mind vs. in the system being developed or used
Vs. chaos, nonlinearity
Vs. complicated
Vs. determinism
Vs. entropy
Vs. randomness
Vs. structured situations
Aerospace standards and guidebooks
Branded characterizations
Adaptive systems and cybernetics
Axiomatic design theory [Suh 2001]
Complexity science
Computational (# of resources consumed by most efficient algorithm as input size grows)
Cyclomatic number of control graph (see D. Measuring Complexity)
Frustration at failing to understand a problematic situation (Warfield)
Kolmogorov complexity measures (shortest program that outputs that artifact) is a member of a larger family of algorithmic-based characterizations of complexity [Kinnunen 2006]
Comment: Measure doesn't credit decisions made during design that lengthen the design time but that reduce operational or repair complexity
Organized complexity
Systems science (see Emergence under B. Causes, System Design)
Ordinary English
B. Causes
Cognitive and psychological
Ability to reason about a situation requiring multiple abstractions and deductions from cause–effect relationships
Limited opportunities to learn about the regularities in the domain or system
Perceived cognitive complexity of task (i.e., of a task's rules and requirements; what info has to be gathered, how processed, and when)
Stakeholders (external to development team) capabilities and structure
Number of stakeholders
Political

Distribution Statement A: Approved for Public Release; Distribution Is Unlimited

Number of layers of management, cultures
Social context, culture
User capabilities
Requirements
Input or output complexity (required of the system)
Number of requirements
Quality attributes required of the system
Adaptability of solution is critical
Designed to adapt to changes in its environment (or stakeholder purpose)
Product line focus (design for modularity and variability)
Reliability needed in solution
Robustness needed in solution (instead of highest peaks, look for highest mesas, where performance does not fall off sharply due to small variations in design variables or noise)
User-modifiable software (software designed to be modified by its users)
Performance of solution is critical
Parallelism needed in solution
Safety of solution
Security of solution (availability, integrity, confidentiality) is critical
Systems to integrate with
Volatility
Evolution of system purpose
Increasing need to integrate with other systems
User role is changing from direct control to a more subtle relationship based on increasing dependence on a system that automates many operational functions and responsibilities
Technology change and trends
Technology improvements allow considering broader solutions with much more sophisticated operational capabilities (driven by market, mission, or societal demand)
Increasing capability of software (and underlying hardware and development environments) allows systems to provide broader and more sophisticated operational capabilities
Development team capabilities and structure
Analyses (of system behavior) that necessarily cross multiple domains (e.g., physical vs. social, software vs. hardware)
Dealing with poorly structured situations (when it is difficult to identify and understand all relevant factors or system dynamics)
Lack of experience designing and implementing similar systems
Over-reliance on an unproven technology (whether for product, dev't process, or dev't environment)
System parameter tolerance vs. capability (voice of customer vs. design (or process) capability) (will be different for different system parameters)
Team structure (organization of and access to needed capabilities)
Learning curve
Managing overlap in system design and production across a global network of partners and suppliers
Number of layers of management
Uncertainty (designer side)
System design
Component internal complexity (e.g., control structure complexity)
Algorithm complexity (video compression; speech and image recognition; and seismic, traffic flow, and molecular models) (not to be confused with algorithmic or Kolmogorov complexity—see A. Definitions and Characterizations)
Low signal-to-noise ratio for system inputs
Need to model highly dynamic situations (e.g., traffic or fluid flow), particularly in real time

Distribution Statement A: Approved for Public Release; Distribution Is Unlimited

Control structure complexity of code
Concurrent process scheduling
Interrupt handling
Many predicates (conditions) for controlling program flow
Data structure complexity
Numeric computational complexity (uses numerical expressions with many levels of nested subexpressions, terms, and numeric data types)
Many nested levels of control
Operations at physical IO level
Dynamics (of how the system behaves during its operation)
Many operational modes or state machine transitions
Number and variability in policies or rules governing system behavior
Nonholonomic constraints
Number of system configurations (driven by external inputs; may involve deactiving or reactiving processes)
Off-design considerations
Functional coupling complexity (how system components interact to achieve requirements)
Cohesion lacking in modules (many unintuitive dependencies)
Coupling (and interactions) between elements of a system, design, or process
Connection and interaction rules and protocols
Number and type of interactions (or interconnections)
Data flow complexity
Emergence
Feedback loops (and number of components in a feedback path)
Interplay between subsystems (e.g., software and hardware)
Nonlinearity
Representational complexity (with what languages, abstractions, vocabulary, fidelity [accuracy, precision], and formality the system requirements, designs, and implementations are expressed)
Architectural or design language fails to represent all component-to-component interactions as explicit structural connections
Structural complexity
Connectivity (number of connections, density of connections, strength of relationships)
Many core components (components that have a high fan-out and activate other components)
Number and types of inputs to, and outputs from, the system
Number of components
Number of functions or features
Number of interfaces
Structural heterogeneity (variety of different component and connection types)
Asymmetry
Uncertainty (system side)
C. Impacts of Complexity
Adherence to a task’s rules and requirements
Applicability (generalizability) of observations at a particular component level to other decomposition levels
Confusion
Correlated outcomes
Adaptability to change
Defect ratios and rates
Domain specific (some aspects of complexity may be typical of systems developed for a particular domain)
Effort, cost, and schedule overruns
Firm performance and survival

Distribution Statement A: Approved for Public Release; Distribution Is Unlimited

Productivity
Staff turnover (retention)
System performance shortfall
Design process
Design and analysis techniques may fail to adequately address (e.g., safety coverage of) more tightly coupled, integrated, high performance, real-time systems
Need to use in-service data to partially satisfy safety objectives for development-design level of product and to claim certification credit
Error possibilities increase, particularly for functions that are performed jointly across multiple systems
Making changes (e.g., in software maintenance)
Over-optimism (bias) in plans and risk assessments
Problem reporting, definition, and diagnosis
Servicing the system in the field
Software planning (including software lifecycle definition and selection)
Unstable, nonreplicable, unpredictable behavior
Verification and assurance
D. Measuring Complexity
Cognitive fog (project members find themselves in fog of conflicting data and cognitive overload)
Requirements difficulty (requirements are considered difficult to implement or engineer, are hard to trace, or overlap with other requirements)
Stakeholder relationships are changing (transitional) or stressed by resistance to change (messy)
COCOMO II Module Complexity Ratings (1) determine type(s) of operation performed (control, computational, device-dependent, data mgt, user interface mgt), (2) determine best placement on the type-associated Behaviorally Anchored Rating Scale (BARS)
Design-derived and computable measures (Comment: Some measures integrate or cover multiple categories of complexity)
[Interaction] Topology
Coupling complexity (accounts for coupling between nodes that are not directly connected—helping assess impact of design change)
Design structure matrix (DSM) (used to measure adaptability as well as complexity)
Comment: Weaknesses with these metrics; don't take interface complexity or product variety into account
Gerhensen's Modularity Index (GMI) (subtract the averaged interactions external to modules from averaged internal within modules)
Singular value decomposition (SVD) of the DSM to obtain singular values whose size and decay rate indicate modularity; e.g., Singular Value Modularity Index (SMI) (SMI could also be used to compute sensitivity to a requirements change)
Ulrich (# of components divided by number of functions they implement)
Whitney Index (# interactions in DSM divided by size of DSM)
Dynamic complexity
(comment) Directivity of interactions unaddressed
(comment) Relative measurement of subsystem complexity unaddressed
Modularity (also use to measure adaptability as well as complexity)
Modified Integration Complexity (CI_M) of a system's subsystems (Purdue University)
Modularity measure [Tamaskar 2014]
Modularity measure [Zeidner 2010]

Distribution Statement A: Approved for Public Release; Distribution Is Unlimited

Comment: Directivity of network connections unaddressed
Comment: Feedback loops unaddressed
Strong's Interface Reuse Metric (IRM) (1 – (# types of interfaces) / (total # of interfaces))
Visibility score (core, periphery, control, utility)
Complex adaptive systems theory–based measures
Information content
Comment: Good for measuring size, heterogeneity, and (process variability) uncertainty, but not (interaction) topology and dynamics
Design process variability, vulnerability, and correlation [El-Haik and Yang 1999]
Entropy-based (e.g., probability of satisfying the functional requirements or of producing a design)
Effective complexity [Gell-Mann and Lloyd 1996]
Network (connectivity-focused) (Comment: Network-based measures tend to conflate complexity and adaptability)
Centrality (e.g., eigenvector centrality)
Clustering of graph nodes (e.g., clustering coefficient, which can indicate robustness or fault tolerance)
Connectivity degree (e.g., node degree, highest degree)
Mobility
Solvability (e.g., Bridge Impact Factor)
Designer willpower-related
Degree of challenge posed by a problematic situation (e.g., Situation Complexity Index (SCI) of Warfield)
Neural stress attendant to observation
Effective complexity (the length of most concise description of an entity's regularities [model size]), crypticity (time it takes to fit model to a given instance), and logical depth (time it takes to use model to predict future instance) [Gell-Mann and Loyd 1996]
Empirical (estimated through historical relationship between performance and design parameters within same domain) [Bearden 2003]
General comments (motivation and approaches)
Calibration of measures to determine best predictors
How to select which measures to use
Properties of a good complexity metric
Scalability (in particular, computability of the measure as number of nodes and links increase)
Take lifecycle complexity view (architecture, design process, and manufacturing process)
Software code-derived and computable measures
Cost semantics (graph) for a program written in a functional language (sequential and parallel time and space complexity)
Cyclomatic complexity of low-level software code (# of control flow graph regions; # of conditions controlling flow)
Essential complexity of a control flow graph (software code)
E. Mitigating Complexity
Build models whose simplifying assumptions enable answering specific questions or concerns (often domain specific)
Diagramming (analogous modeling)
Influence diagrams (a.k.a., stock-and-flow diagrams)
Unit diagrams with feedback loops (cybernetics)
Integrate best-of-breed models from each constituent domain through a single bridging ontology
MBSE, MDE (model-based systems engineering/model-based software engineering, model-driven engineering)
Analyses for consistency, traceability, etc. are automated
Documents design rationale and underlying assumptions
Formally analyzable specification languages
Support for change management

Distribution Statement A: Approved for Public Release; Distribution Is Unlimited

Stakeholder viewpoint-generated graphs, diagrams, and views
Developed for different stakeholder worldviews in soft or unstructured situations
Collaborating globally or crowd-sourcing the solution to complex problems
Data abstraction and type systems (move complexity into type declaration and checking vs. later verification)
Determine as much as you can statically; dynamically analyze the rest
Discipline in system development (clear objectives, processes, measures, planning, training, tooling) to reduce task complexity (e.g., clarify task completion)
Standards for design and coding that help achieve safety, adaptability, and other quality attributes while minimizing use of risky methods and constructs
Integrate key recurring processes (e.g., requirements validation) and analyses (e.g., consistency, completeness) end-to-end into the software lifecycle SLC
Structured programming and defensive programming (avoid certain constructs that make programs difficult to understand, verify, and debug)
Everything should be made as simple as possible, but not simpler (Occam's Razor); applied to designs and justifications for claims
Functional abstraction (separate functional complexity from runtime complexity)
Functional periodicity: introduce (functional) periodicity in system design with time periods for system re-initialization to reduce time-dependent combinatorial complexity (and to increase system reliability and predictability)
General comments
Incremental assurance (establishing confidence that implementation achieves intention)
Assurance of a system produced by multiple parties: first, ensure overall design will not lead to hazard, then generate specific requirements for each component provider
Assurance techniques (process assurance, validation and verification coverage criteria, analyses) improve visibility of what is or was done and why
Certifying the algorithms used
Verification of program efficiency
Evaluate specific attributes of requirements, architecture, and configurations to determine what in-service data is needed for safety certification
Proof of correctness
Interactive management (Warfield's 20 Laws of Complexity focused on individual, team, and organizational pathologies)
Interpretive social theory and critical thinking
Mathematically or statistically characterize what you can (e.g., interactions, protocols, data structures, and algorithms)
Modular design (localize impacts and maximize readiness to changes in demand; separate extensional from intentional)
Create uncoupled or decoupled designs (axiomatic design theory)
New technology that radically shortens test lifecycles (3-D printing, quantum computing)
Optimize (what you can), build for adaptability (where you can), then hedge against multiple futures (in that order)
Prognostics (rather than diagnostics) through sensors, Internet of Things, and Big Data
System boundary identification
Systems engineering (also see MBSE under E. Mitigating Complexity, Build models...)
Systems thinking-based analysis of system behavior and system design
Extend the causality model included in analyses to address non-failure-based component interactions that can introduce hazards (e.g., due to nonlinear, indirect, and feedback relationships)

Distribution Statement A: Approved for Public Release; Distribution Is Unlimited

Use design-based measures early in the lifecycle that are broadly applicable, objective, computable, and validated to guide development
Evaluate design alternatives (or proposed changes) using domain and company-calibrated design measures
Explore performance vs. complexity tradeoff, and select the simplest designs that meet performance requirements
Genetic algorithm to help rapidly search for regions where near-optimal solutions exist in the design space
Identify design features, patterns, and rules for high performance that have low impact on complexity
Refactor design to reduce complexity and improve modularity (when a complexity measure exceeds some threshold)

Appendix B: Bibliography of Sources Used in Literature Review

The sources reviewed as part of the literature review task are listed in this appendix (Sources for the report are shown in Appendix C). In addition to these sources, several definitions of complexity as used in everyday (not necessarily technical) conversation were obtained by googling "define complexity." The intent was to establish a baseline definition to help in identify unique aspects of other definitions.

Bearden, D. A. 2003. "A Complexity-Based Risk Assessment of Low-Cost Planetary Missions: When Is a Mission Too Fast and Too Cheap?" *Acta Astronautica 52* (2-6): 371–379.

CMMI Product Team. *CMMI® for Development, Version 1.3*, Software Engineering Institute, Carnegie Mellon University, November 2010. http://www.sei.cmu.edu/reports/10tr033.pdf

Center for Systems and Software Engineering. COCOMO II Cost Driver and Scale Driver Help, University of Southern California, 2012. http://sunset.usc.edu/research/COCOMOII/expert_cocomo/drivers.html

Einstein, Albert. 1934. "On the Method of Theoretical Physics," *Philosophy of Science 1* (2): 163–169. http://www.jstor.org/stable/184387?origin=JSTOR-pdf&seq=1#page_scan_tab_contents

El-Haik, Basem and Kai Yang. 1999. "The Components of Complexity in Engineering Design." *IIE Transactions 31* (10): 925–934.

FAA, DOT/FAA/AR-xx/xx. Draft Final Report for Software Service History and Airborne Electronic Hardware Service Experience in Airborne Systems (2014).

Fleming, Cody Harrison, Melissa Spencer, John Thomas, Nancy Leveson, and Chris Wilkinson. 2013. "Safety Assurance in NextGen and Complex Transportation Systems," *Safety Science 55*: 173–187. http://sunnyday.mit.edu/papers/ITP-Final.pdf

Flood, Robert L. and Ewart R.Carson. *Dealing with Complexity: An Introduction to the Theory and Application of Systems Science, 2nd Edition*, Springer, New York, NY, 1993.

Friedman, Thomas L. “When Complexity Is Free,” *New York Times*, 14 September 2013. http://www.nytimes.com/2013/09/15/opinion/sunday/friedman-when-complexity-is-free.html?pagewanted=all

Frosch, Tilman, Christian Mainka, Christoph Bader, Florian Bergsma, Jörg Schwenk, and Thorsten Holz. *How Secure Is TextSecure?* 2014. http://www.slideshare.net/TopSecretSpyFiles/how-secure-is-text-secure-42658975

Gell-Mann, M., and S. Lloyd. 1996. “Information Measures, Effective Complexity, and Total Information.” *Complexity 2* (1): 44–52.

Gell-Mann, Murray. 1995. “What Is Complexity,” *Complexity 1* (1):16–19. http://onlinelibrary.wiley.com/doi/10.1002/cplx.6130010105/abstract

Harper, Robert. *Structure and Efficiency of Computer Programs*, 23 July 2014. http://www.cs.cmu.edu/~rwh/papers/secp/secp.pdf

Kinnunen, Matti J. 2006. “Complexity Measures for System Architecture Models.” Massachusetts Institute of Technology.

Leveson, Nancy. *Engineering a Safer World: Systems Thinking Applied to Safety*. MIT Press, 2011.

MacCormack, Alan, Carliss Y. Baldwin, and John Rusnak. *The Architecture of Complex Systems: Do Core-periphery Structures Dominate?* MIT Sloan School of Management Working Paper Number 4770-10, 19 January 2010. http://ebusiness.mit.edu/research/papers/2010.04_MacCormack_Baldwin_Rusnak_The%20Architecture%20of%20Complex%20Systems_268.pdf

Mata, Jutta, Peter M. Todd, and Sonia Lippke. 2010. “When Weight Management Lasts: Lower Perceived Rule Complexity Increases Adherence,” *Appetite 54*: 37–43. http://labes.fmh.utl.pt/obesity/jutta/MataToddLippke2010.pdf

McCabe, Thomas J. 1976. “A Complexity Measure,” *IEEE Transactions on Software Engineering, SE-2* (4): 308–320. http://www.literateprogramming.com/mccabe.pdf

Pennock, Michael J. and William B. Rouse. “Why Connecting Theories Together May Not Work,” *2014 IEEE International Conference on Systems, Man, and Cybernetics*, pp. 373–378, October 2014, San Diego, CA, USA. http://ieeexplore.ieee.org/stamp/stamp.jsp?tp=&arnumber=6973936

Rechtin, Eberhardt. *Systems Architecting of Organizations: Why Eagles Can’t Swim (Systems Engineering)*, CRC Press, July 1999.

RTCA, Inc. *DO-178C Software Considerations in Airborne Systems and Equipment Certification*, January 2012.

SAE International. *ARP 4754A Safety and Reliability Consideration in Aircraft/Systems Development*, December 2010. http://standards.sae.org/arp4754a/

Distribution Statement A: Approved for Public Release; Distribution Is Unlimited

Sheard, Sarah A. and Ali Mostashari. 2013. “Complexity Measures to Predict System Development Project Outcomes,” *Proceedings of the INCOSE International Symposium 23* (1): 170–183.

Stuart, Douglas, Raju Mattikalli, Daniel DeLaurentis, and Jami Shah. *META II Complexity and Adaptability Final Report*, September 2011. http://diyhpl.us/~bryan/irc/darpa/avm/meta/Boeing%20META%20Final%20Report.pdf

Sturtevant, Dan. *Technical Debt in Large Systems: Understanding the Cost of Software Complexity* [Webinar], May 2013. http://sdm.mit.edu/news/news_articles/webinar_050613/sturtevant_050613.pdf

Suh, Nam Pyo. *Complexity: Theory and Applications*, Oxford University Press, 2005.

Tamaskar, Shashank, Kartavya Neema, and Daniel DeLaurentis. 2014. *Managing Complexity in Model-Based Conceptual Design*, *INCOSE INSIGHT 17* (4): 20–22.

Tversky, Amos and Daniel Kahneman. 1974. “Judgment Under Uncertainty: Heuristics and Biases,” *Science, New Series, 185* (4157): 1124–1131. http://links.jstor.org/sici?sici=0036-8075%2819740927%293%3A185%3A4157%3C1124%3AJUUHAB%3E2.0.CO%3B2-M

Warfield, John N. 1999. “Twenty Laws of Complexity: Science Applicable in Organizations,” *Systems Research and Behavioral Science Systems Research 16*: 3–40. http://demosophia.com/wp-content/uploads/Twenty%20laws%20of%20complexity.pdf

Wikipedia, *Complexity*, 2014. http://en.wikipedia.org/wiki/Complexity

Zeidner, L., A. Banaszuk, and S. Becz. 2010. “System Complexity Reduction via Spectral Graph Partitioning to Identify Hierarchical Modular Clusters.” Presented at the 10th AIAA Aviation Technology, Integration, and Operations (ATIO) Conference.

Appendix C: Report Bibliography

The bibliography developed as part of the literature review appears in Appendix B. This is the bibliography for this report.

Basili, Victor, Gianluigi Caldiera, and H. Dieter Rombach. “The Goal Question Metric Approach,” *Encyclopedia of Software Engineering*, Vol. 1, pp. 528–532, Wiley, 1994. https://www.cs.umd.edu/~basili/publications/technical/T89.pdf.

Eisner, Howard. *Managing Complex Systems: Thinking Outside the Box*. John Wiley & Sons, 2011.

FAA, DOT/FAA/AR-xx/xx, Draft Final Report for Software Service History and Airborne Electronic Hardware Service Experience in Airborne Systems (2014).

Glaser, Barney, and Anselm Strauss. *The Discovery of Grounded Theory: Strategies for Qualitative Research*, Aldine, 1967.

Hall, Arthur David. *A Methodology for Systems Engineering*. Van Nostrand, 1962.

Kahneman, Klein. "Conditions for Intuitive Expertise: A Failure to Disagree." *American Psychologist 64* (6): 515–526, 2009. http://www.scribd.com/doc/186002790/Kahneman-Klein-2009-Conditions-for-Intuitive-Expertise-a-Failure-to-Disagree#scribd

Kitchenham, B., and S. Charters. *Guidelines for Performing Systematic Literature Reviews in Software Engineering*, Technical Report EBSE 2007-001, Keele University and Durham University Joint Report, 2007. http://community.dur.ac.uk/ebse/guidelines.php

Konrad, Michael, Art Manion, Andrew Moore, Julia Mullaney, William Nichols, Michael Orlando, and Erin Harper. *Data-Driven Software Assurance: A Research Study* (CMU/SEI-2014-TR-010), Software Engineering Institute, Carnegie Mellon University, 2014. http://resources.sei.cmu.edu/library/asset-view.cfm?AssetID=90086

Leveson, Nancy. *Engineering a Safer World: Systems Thinking Applied to Safety*. MIT Press, 2011. http://mitpress.mit.edu/sites/default/files/titles/free_download/9780262016629_Engineering_a_Safer_World.pdf. Retrieved May 1, 2015.

Lustig, Robert H. 2006. "Childhood Obesity: Behavioral Aberration or Biochemical Drive? Reinterpreting the First Law of Thermodynamics." *Nature Clinical Practice Endocrinology & Metabolism 2:* 447–458. http://www.nature.com/nrendo/journal/v2/n8/ris/ncpendmet0220.html

Sheard, Sarah. *Assessing the Impact of Complexity Attributes on System Development Project Outcomes*, PhD diss., Stevens Institute of Technology, 2012.

Software Engineering Institute, 2014-2016 Work Plan for the Federal Aviation Administration (FAA) PWS 5-427 A1 Version 1.0, 2014.

Strogatz, Synch. *Sync: How Order Emerges from Chaos in the University, Nature, and Daily Life*, Hyperion, 2004.

Suh, Nam Pyo. *Complexity: Theory and Applications*, Oxford University Press, 2005.

Warfield, John N. "Twenty Laws of Complexity: Science Applicable in Organizations," *Systems Research and Behavioral Science Systems Research*, Vol. 16, pp. 3–40, 1999. http://demosophia.com/wp-content/uploads/Twenty%20laws%20of%20complexity.pdf

Acknowledgments

The project team members, in addition to the authors, are William Nichols, Charles Weinstock, and Greg Such. For this task they helped frame the discussion and review the ideas.

Julian Delange, John Goodenough, and Donald Firesmith provided insightful technical review external to the team.

Tamara Marshall-Keim provided superb and timely technical editing.

Contact Us

Software Engineering Institute
4500 Fifth Avenue, Pittsburgh, PA 15213-2612

Phone: 412/268.5800 | 888.201.4479
Web: www.sei.cmu.edu | www.cert.org
Email: info@sei.cmu.edu

Copyright 2016 Carnegie Mellon University

This material is based upon work funded and supported by Federal Aviation Administration under Contract No. FA8721-05-C-0003 with Carnegie Mellon University for the operation of the Software Engineering Institute, a federally funded research and development center sponsored by the United States Department of Defense.

Any opinions, findings and conclusions or recommendations expressed in this material are those of the author(s) and do not necessarily reflect the views of Federal Aviation Administration or the United States Department of Defense.

NO WARRANTY. THIS CARNEGIE MELLON UNIVERSITY AND SOFTWARE ENGINEERING INSTITUTE MATERIAL IS FURNISHED ON AN "AS-IS" BASIS. CARNEGIE MELLON UNIVERSITY MAKES NO WARRANTIES OF ANY KIND, EITHER EXPRESSED OR IMPLIED, AS TO ANY MATTER INCLUDING, BUT NOT LIMITED TO, WARRANTY OF FITNESS FOR PURPOSE OR MERCHANTABILITY, EXCLUSIVITY, OR RESULTS OBTAINED FROM USE OF THE MATERIAL. CARNEGIE MELLON UNIVERSITY DOES NOT MAKE ANY WARRANTY OF ANY KIND WITH RESPECT TO FREEDOM FROM PATENT, TRADEMARK, OR COPYRIGHT INFRINGEMENT.

[Distribution Statement A] This material has been approved for public release and unlimited distribution. Please see Copyright notice for non-US Government use and distribution.

Internal use:* Permission to reproduce this material and to prepare derivative works from this material for internal use is granted, provided the copyright and "No Warranty" statements are included with all reproductions and derivative works.

External use:* This material may be reproduced in its entirety, without modification, and freely distributed in written or electronic form without requesting formal permission. Permission is required for any other external and/or commercial use. Requests for permission should be directed to the Software Engineering Institute at permission@sei.cmu.edu.

* These restrictions do not apply to U.S. government entities.

DM-0004269