System Qualities Ontology, Tradespace, and Affordability (SQOTA)

Phases 1-7 (RTs 46/113,137,160,181,209)

Technical Report SERC-2019-TR-012

August 23, 2019

Principal Investigator:

Principal Investigator: Dr. Barry Boehm, University of Southern California

Research Team:

Air Force Institute of Technology

Georgia Institute of Technology

Massachusetts Institute of Technology

Naval Postgraduate School

Pennsylvania State University

University of Southern California

University of Virginia

Wayne State University

Sponsor:

OUSD(R&E)

The Systems Engineering Research Center (SERC) is a federally funded University Affiliated Research Center managed by Stevens Institute of Technology.

This material is based upon work supported, in whole or in part, by the U.S. Department of Defense through the Office of the Assistant Secretary of Defense for Research and Engineering (ASD(R&E)) under Contract [HQ0034-13-D-0004, TO#0427].

Table of Contents

List of Figures

List of (Tables, Sequences)

Executive Summary

Systems and software qualities (SQs) are also known as non-functional requirements (NFRs). Where functional requirements (FRs) specify what a system should do, the NFRs specify how well the system should do them. Many of them, such as Reliability, Availability, Maintainability, Usability, Affordability, Interoperability, and Adaptability, are often called "ilities," but not to the exclusion of other SQs such as Security, Safety, Resilience, Robustness, Accuracy, and Speed.

As compared to functional requirements, NFRs have been underemphasized in project management, and serious sources of project shortfalls and overruns. They are often late in being thoroughly reviewed, being preceded by reviews such as the System Functional Requirement Review. They do not have a place in function-oriented management aids such as Work Breakdown Structures and traceability diagrams: the NFRs generally trace to the whole system. Their requirements are often easy to specify and hard to validate: one classic case was a project in which changing one character in the NFR for system response time in seconds from a 1 to a 4 in a 2000-page specification reduced the cost of achieving the system from $100 million to $30 million.

This report begins with a summary of the origin of the SERC project as the result of SERC universities' participation in two 2012 workshops that addressed the challenges of achieving one such NFR: resilience, in support of one of DoD's high-priority initiatives on Engineered Resilient Systems (ERS). It turned out that the existing ERS research underway was primarily directed at field testing, supercomputer modeling, and resilient design of physical systems, and that the SERC could best complement this research by addressing the design and development of resilient cyber-physical-human (CPH) systems.

Some of the SERC universities involved in the workshops were performing such research, such as AFIT, Georgia Tech, MIT, NPS, Penn State, USC, U. Virginia, and Wayne State. These universities have been addressing aspects of this research area as a team since 2013, culminating in Phase 7 in August 2019. This final report summarizes their overall achievements, in the alphabetical order above.

Project PI: Prof .Barry Boehm, USC
AFIT Co-PI: Prof David Jacques
Georgia Tech Co-PIs: Drs. Tommer Ender and Valerie Sitterle

MIT Co-PIs: Profs. Donna Rhodes and Adam Ross
NPS Co-PI: Prof. Raymond Madachy
PSU Co-PI: Prof. Michael Yukish
U. Virginia Co-PI: Prof Kevin Sullivan
Wayne State U. Co-PI: Prof. Gary Witus

Origins of the iTAP-SQOTA Project

Tradespace and affordability analysis can involve numerous combinations of options for decreasing life cycle costs and increasing life cycle effectiveness. Evaluating these combined options often involves complex tradeoffs among affordability and other SQs such as reliability, availability, maintainability, usability, adaptability, interoperability, scalability, and others such as safety, security, reliance, and timeliness. For software systems, a considerable literature on SQ tradeoffs has evolved, from (Boehm-Brown-Lipow, 1976) through (Gilb, 1988), (Chung et al., 1999), to the (Clements-Kazman-Klein, 2002) Architecture Tradeoff Analysis Method (ATAM) approach. Current systems engineering approaches include real options analysis (Baldwin-Clark, 2000; Mun, 2005), total cost of ownership analysis (Boehm-Lane-Madachy, 2011), incremental-commitment decision-space narrowing, (deNeufville-Scholtes, 2011), and physical tradespace analysis (Ross-Hastings, 2005). All of the approaches face significant challenges in multi-criteria decision analysis, but the need for improved capabilities continues to increase.

The SQ activity resulted primarily from a strong DoD emphasis on Affordability via the DepSecDef Better Buying Power memoranda, and on ilties tradespace analysis via one of DoD's seven Priority Science and Technology Investment Areas: Engineered Resilient Systems (ERS). One of the ERS five Key Technology Thrust Areas is Data-Driven Tradespace Exploration and Analysis. In July 2012, ERS held a workshop that focused on tradespace analysis in general, followed by a SERC workshop to establish the near-term SQ direction.

1 The ERS Tradespace Exploration and Analysis Workshop Results

The July 17-18, 2012 ERS workshop on the topic primarily addressed physical tradespace analysis, but one of its working groups also addressed SQ trades. As it included DoD, industry, and academic participants, its assessment of the SQ tradespace state of the art and state of the practice are well balanced. Its findings include:

- Data characterizing the SQs of tradespace options is often missing or inadequate, even though the quality of those SQs has a material impact on the relative value of the options to the stakeholders.
- The means to adequately express and analyze SQs are lacking when compared to physics-based characteristics such as weight and size.
- Needs include agreed upon terminology to express SQs, and cross-impact models and relationships that quantitatively determine the impact of changing one SQ on the other SQs
- The values of most SQs are scenario-dependent. Anticipated and possible future operational environments and user scenarios are usually not stated in sufficient detail to support tradespace evaluation.
- Languages and technology are lacking to cost-effectively and rapidly express vivid complex operational environments and user scenarios for a range of stakeholders.
- Affordability, as an SQ worthy of special note, is often not addressed during tradestudies, which often focus on technical aspects of trades apart from the cost implications across the full life cycle, including manufacturing, operations, and maintenance.
- Needs include involving SE and affordability analysis in very early phases of the life cycle in parallel with S&T activities; and applying SE cost models to justify greater expenditures early in the life cycle to reduce Total Cost of Ownership (TCO).
- Needs also include better TCO methods and tools that account for SoS, that avoid suboptimization on colors of money across the life cycle, and that avoid suboptimization around different phases of the life cycle such as S&T, development, manufacturing, and life cycle support.

2 THE SERC SQ TRADESPACE AND AFFORDABILITY WORKSHOP RESULTS

The SERC SQ workshop immediately followed the ERS workshop, being held on the afternoon of July 18 and all day on July 19. Both workshops included lead and sponsor personnel from each others' organizations. The SQ workshop also included presentations and participation from two FFRDCs: Aerospace Corp. and MITRE.
The workshop addressed both the challenges and the opportunities of strengthening the ability to perform SQ analysis in concert with complementary initiatives to improve the ability to perform physical, functional, and overall Tradespace and Affordability (T&A) analysis. It assessed current SQ research capabilities, and concluded that the SERC could play a significant research leadership role in strengthening SQ analysis capabilities and in integrating SQ with other T&A analysis capabilities.

Highlights of the ITAP workshop include the following:

- Several SERC universities had already produced useful SQ capabilities. Some had been on SERC projects such as Valuing Flexibility, Next-Generation Software Cost Models, and Expedited Systems Engineering. Others have been on DARPA, other DoD, NASA, or NSF projects.
- The SERC SQ capabilities are not as scalable or interoperable as those of MITRE and Aerospace, which are also representative of several large aerospace companies. Those capabilities include extensive infrastructure and support organizations that are beyond the direct reach of SERC universities and likely SERC SQ budgets.
- Several frameworks were presented that could provide organizing principles for an SQ initiative. They included an incremental commitment epoch-era temporal framework for reducing a tradespace's Cone of Uncertainty; an architecture-based framework for indicating impacts of architecture choices for achieving one SQ that reinforce or undermine other SQs; a means-ends framework for improving system affordability, and a value-based framework for reasoning about and reconciling system stakeholders' SQ value propositions.
- Value-based approaches have the potential to integrate the other frameworks. They would involve monetization of SQs where feasible, but with clear alternatives or proxies where monetization would not apply well to DoD mission effectiveness considerations.
- During the workshop, a further SQ analysis process framework (setup; share; analyze; act) was explored, which generated 10-20 desired capabilities and research challenges for each process step. The exploration also generated similar numbers of desired capabilities and research challenges for an SQ and associated analysis support environment, and for ITA analysis setup capabilities.

The workshop Conclusions included the definition of several key strategies involved in a long-term vision for SQ research. A primary conclusion was that a SERC SQ research program should be pro-active in helping to define overall physical, informational, and SQ analysis environment architectures, and to focus on research and exploratory development of interoperable, service-oriented, plug-in SQ capabilities for use in multiple DoD T&A analysis environments vs. developing a separate SQ analysis environment.

Resulting SERC Early Research Results

The early SERC research focused on providing a stronger scientific basis for reasoning about SQs and their relationships. Taking "resilience" as one example, our Phase 3 research found that the definitions of "resilience" have significant differences between and within the domains of Ecology, Energy Development, Engineering and Construction, Network, Organizational, Psychological, and Soil (Wikipedia, 2014). Within the Ecology and Society domain, an additional set of definitions includes Original-ecological, Extended-ecological (several definitions), Systemic-heuristic, Operational, Sociological, Ecological-economic, Social-ecological, Metaphorical, and Sustainabilty-related (Brand and Jax, 2007).

The differences in these definitions are non-trivial. For example, the variants in resilience outcomes include Returning to original state; Restoring or improving original state; Maintaining same relationships among state variables; Maintaining desired services; Maintaining an acceptable level of service; Retaining essentially the same function, structure, and feedbacks; Absorbing disturbances; Coping with disturbances; Self-organizing; Learning and adaptation; and Creating lasting value. Trying to get interdiscplinary teams to create consistently resilient systems across such a variety of implicit understandings presents a formidable challenge.

It did not help that most sources of SQ definitions limit them to a single hopefully one-size-fits-all definition of each SQ. This was particularly the case in our evaluation of the primary standard in the area: the ISO/IEC 25010 Systems and Software Quality Requirements and Evaluation (SQuaRE) Standard (ISO/IEC, 2011). As a representative example, it defines Reliability as the, "degree to which a system, product, or component performs specified functions under specified conditions for a specified period of time." As a standard, this is supposed to hold for any definition of "specified functions" and "specified conditions." However, for agile methods, in which "specified functions and conditions" are sunny-day stories or use cases, a system will be judged to be reliable if it satisfies the specified sunny-day conditions, but fails on the rainy-day conditions. Further, the definition only focuses on performing functions and not on satisfying quality levels.

Another difficulty with one-size-fits-all definitions is that they do not accommodate multiple stakeholders with multiple value propositions. For example, if a system specification defines reliability as liveness to satisfy one stakeholder, it will be judged to be reliable if its liveness is very high, even though it may deliver garbled and inaccurate output and not satisfy other stakeholders whose value propositions emphasize intelligibility and accuracy. ISO/IEC does not define Resilience, but if it also defined it in terms of "specified functions and conditions," as it does for other ilities such as Effectiveness and Usability, a system specified to be Resilient under one

of the over-15 domain-varying definitions of Resilience above, would be judged to be resilient even though it did not satisfy the resilience definitions of success-critical stakeholders in other domains. Such one-size-fits-all definitions (also employed in other quality attribute guidance descriptions) thereby presented hindrances to effective interdisciplinary collaboration.

Such considerations highlighted the need to address quality factors important to the full range of DoD stakeholders. The previous ERS focus on physical systems emphasized such qualities as Speed, Endurability, Maneuverability, Accuracy, Impact, Capacity, Transportability, Size, Weight, and Power Consumption. For other stakeholders, we were fortunate to encounter similar lists of needed qualities in the JCIDS Manual for the Operation of the Joint Capabilities Integration and Development System (JCIDS, 2012). For Battlespace Awareness, these were Comprehensive, Persistent, Survivable, Integrated, Timely, Credible, Adaptable, Innovative, and Interoperable. For Command and Control, they were Interoperability, Understanding, Timeliness, Accessibility, Simplicity, Completeness, Agility, Accuracy, Relevance, Robustness, Operational Trust, and Security. For Logistics Supply, they were Responsiveness, Sustainability, Flexibility, Survivability, Attainability, Economy, and Simplicity.

References

1. B. Boehm, J. Brown, and M. Lipow, Quantitative Evaluation of Software Quality, Proceedings, ICSE 2, pp. 592-605, October 1976.
2. T. Gilb, and S. Finzi, Principles of Software Engineering Management. Addison-Wesley, 1988.
3. L. Chung, B. Nixon, E. Yu, and J. Mylopoulos, Non-Functional Requirements in Software Engineering, Kluwer, 1999.
4. P. Clements, R. Kazman, and M. Klein, Evaluating Software Architectures, Addison Wesley, 2002.
5. J. Mun, Real Options Analysis: Tools and Techniques for Valuing Strategic Investment and Decisions, 2nd Edition, Wiley Finance, 2005.
6. C. Baldwin, and K. Clark, Design Rules, Vol. 1: The Power of Modularity, MIT Press, 2000.
7. B. Boehm, J.A. Lane, R. Madachy,Total Ownership Cost Models for Valuing System Flexibility, Proceedings, CSER 2011, March 2011.
8. R. de Neufville and S. Scholtes, Flexibility in Engineering Design, MIT Press, 2011.
9. A. Ross, and D. Hastings, The Tradespace Exploration Paradigm, Proceedings, INCOSE 2005.

10. S. Koolmanojwong, J. Lane, Enablers and Inhibitors of Expediting Systems Engineering, Proceedings, CSER 2013
11. N. Kukreja; S. Payyavula; B. Boehm; S. Padmanabhuni, Value-Based Requirements Prioritization: Usage Experiences, Proceedings, CSER 2013
12. B. Boehm, J. Lane, S. Koolmanojwong, R. Turner, An Evidence-Based Systems Engineering (SE) Data Item Description, Proceedings, CSER 2013
13. B. Boehm, J. Lane, S. Koolmanojwong, An Orthogonal Framework for Improving Life Cycle Affordability, Proceedings, CSER 2013
14. A. Ross, "Contributing toward a Prescriptive 'Theory of Ilities'," Presentation to the SE410 Graduate Seminar, Missouri University of Science & Technology, Rolla, MO, April 2014.
15. Sitterle, V., Curry, M., Freeman, D., and Ender, T. "Integrated Toolset and Workflow for Tradespace Analytics in Systems Engineering," 24th Annual International Council on Systems Engineering (INCOSE) Symposium, Las Vegas, NV, June 30- July 3, 2014.
16. Wikipedia, *Resilience (disambiguation)* http://en.wikipedia.org/wiki/Resilience_(disambiguation), accessed 09/29/2014.
17. F. Brand and K. Jax, Focusing the meaning(s) of resilience: resilience as a descriptive concept and a boundary object. *Ecology and Society* **12**(1): 23.; 2007.
18. ISO/IEC, *Systems and software engineering – SQuaRE system and software quality models standard 25010,* ISO/IEC; 2011.
19. JCIDS, Manual for the Operation of the Joint Capabilities Integration and Development System, 19 January 2012.

Early Results: The System Qualities Ontology

Given the chaotic nature of the variety of quality definitions such as Resilience, the variety of stakeholders having different priorities and contexts for candidate system qualities, and the weakness of the main standard ISO/IEC 25010, in defining and organizing the qualities, USC led an effort to produce an ontology for the system qualities. Drawing on MIT's semantic ontology of change-oriented qualities, AFIT's definitions of flexibility attributes, and U.Virginia's formalization of dependability-related qualities, and initial ontology was developed. After evaluating a number of ontology structures, the IDEF5 ontology structure was adopted. It organized the qualities into a class hierarchy, with specific definitions of the relations up and down the hierarchy. For the qualities ontology, we defined the relations as means-ends:

the lower quality was a means of realizing the upper quality. The IDEF5 ontology structure also includes sources of variation in a quality's value, such as variation by stakeholder of Reliability, in which some stakeholders rely on a system's liveness, and others rely on its ability not to fail in delivering or garbling messages.

An initial version of the ontology was presented at INCOSE IS 2015. Subsequently, refinements were made and the varied contributions of Maintainability were documented in a revised paper presented at INCOSE IS 2016. The revised structure is shown in Table 1.

Table 1 Upper Levels of Revised Stakeholder Value-Based SQ Means-Ends Hierarchy

Stakeholder Value-Based SQ Ends	Contributing SQ Means
Mission Effectiveness	Stakeholders-satisfactory balance of Physical Capability, Cyber Capability, Human Usability, Speed, Endurability, Maneuverability, Accuracy, Impact, Scalability, Versatility, Interoperability, Domain-Specific Objectives
Life Cycle Efficiency	Development and Maintenance Cost, Duration, Key Personnel, Other Scarce Resources; Manufacturability, Sustainability
Dependability	Reliability, Maintainability, Availability, Survivability, Robustness, Graceful Degradation, Security, Safety
Changeability	Maintainability, Modifiability, Repairability, Adaptability
Composite QAs	
Affordability	Mission Effectiveness, Life Cycle Efficiency
Resilience	Dependability, Changeability

The main classes of success-critical stakeholders are the system's mission operators, end-users, usage managers, interoperators, and support personnel with respect to the system's operational mission. They will be most concerned with the components of Mission Effectiveness above, but also with domain-specific SQs such as Auditability for finance, In-Process Visibility for supply chains, and Health Maintainability for medical domains.

Another class of success-critical stakeholders is those who are investing key resources (funds, property, materials, personnel, services, etc.) to define, develop, operate, and evolve a system. They will have high priority value propositions not only on mission effectiveness but also the relative returns on their investments. The combined SQ is often called Cost-Effectiveness and the resource expenditures often called Affordability, but the ontology follows INCOSE, NDIA, and MORS in interpreting

Affordability as a cost-effectiveness composite SQ, and categorizes the expenditures as Life Cycle Efficiency, which includes both Development and Maintenance Costs. The mission stakeholders are also concerned about Dependability, but so are a further class of success-critical stakeholders: people who are not involved in the system's definition, development, and evolution, but who are depending on the system to avoid their loss of property and quality of life during its operational performance (driving a car, boarding an airplane, using a chain saw, providing credit card and social security numbers, etc.). The means for achieving this Dependability end include protection from vulnerabilities from system adversaries (Security); from natural causes, defects and human errors (Safety); from failure to deliver needed capabilities (Reliability), and from persistence of such failure for long periods of time (Availability, and its enabler Maintainability). Graceful Degradation has been added to indicate that Survivability can be full or partial.

Finally, all of these stakeholders are concerned with the system's ability to continue to provide or improve the desired levels of Mission Effectiveness, Life Cycle Efficiency, and Dependability as the world around it continues to provide increasing sources of change in technology, competition, market demands, organizations, and leadership. We have generalized Flexibility to Changeability to be more consistent with the MIT Changeability ontology. Its mutually-exclusive means are internal change (Adaptability) and external change (Maintainability), which includes Modifiability to support Changeability and Repairability to support Dependability.

Elaboration of Maintainability Relationships. In pursuing the various contributions and types of Maintainability as a critical but complex SQ, we have developed Figures 1 and 2 as elaborations of Maintainability relationships. Figure 1 focuses on the role of Maintainability in support of Changeability. It shows that Maintainability and Adaptability are two alternative means of achieving Changeability, external and internal to the system. It also shows that Modifiability and Repairability are two alternative means of achieving Maintainability, one for changes and one for defects. It also shows some enabling (and), but not alternative (or) subclasses of Modifiability, Repairability, and support of verification and validation, called "V&V-ity." These latter are Automated Analysis, Peer Reviews, and Execution Testing; their rating scales are provided in (Boehm & Kukreja, 2015), where they were taken from the Constructive Quality Model for estimating delivered software defect density (Boehm *et al*., 2000).

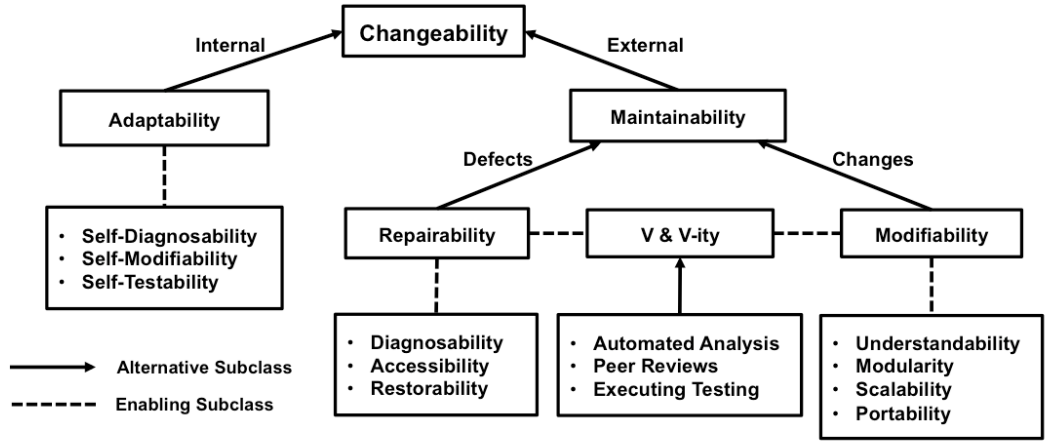


Figure 1. Role of Maintainability for Changeability

Figure 2 further elaborates on how Maintainability supports both Changeability and Dependability, and elaborates on how these two combine to support Resilience. It also elaborates the means-ends subclasses of Reliability, and of Testing in support of V&V-ity.

The strength and value of the System Qualities Ontology, and some confusion about whether security, safety, speed, accuracy, resilience, robustness, etc. were ilities, caused us to change the project title from ilities Tradespace and Affordability Project (iTAP) to System Qualities Ontology, Tradespace and Affordability (SQOTA).

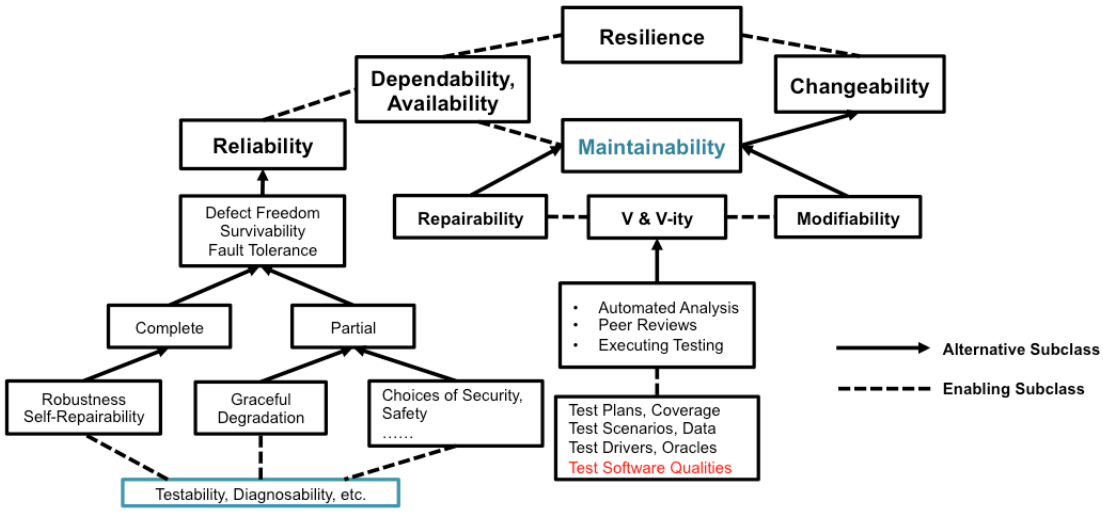


Figure 2. Role of Maintainability for Changeability, Dependability, and Resilience

Figure 2 also emphasizes the many-to-many relationships of Testability, as compared to the one-to-many relationship it has in ISO/IEC 25010, where Testability only supports Maintainability. It is clear that Testability is also important to the achievement of numerous other SQs, such as the many aspects of Dependability in Figure 2. It also highlights the recursive complexities of the SQ hierarchy, as Testability also involves the test software, such as its classes (performance, stress, regression, etc.) and capabilities (test drivers, monitors, diagnosis aids, data managers, etc.) and the need to ensure its full set of SQs as well, as shown in red in Figure 2.

In summary, Table 1 and Figure 1 summarize the class hierarchy of the primary stakeholder SQ end value classes of Mission Effectiveness, Life Cycle Efficiency, Dependability, and Changeability, along with their primary means-ends subclasses, and the primary composite SQs of Affordability and Resilience. The over-20 definitions of Resilience discussed earlier can generally be traced to variations in their lower-level enablers. Each of the SQ classes and subclasses must also be characterized by the variation of their values with respect to their variation by Referents, States, Processes, and Relations, as will be described next in the context of Maintainability.

Maintainability Referents, States, Processes and Relations

All too often, SQs are specified as single numbers, when in general their values vary by Referent (primarily by stakeholder value proposition), by State (the system's internal state and its external environment state), by Process (internal procedures, external operational scenarios), and by their Relations with other SQs. These are discussed below in the context of Maintainability.

Referents. For Maintainability, all stakeholders will prefer shorter above longer mean times to repair or modify, but their duration values will vary by stakeholder. A Meta-Group industry survey reported in the year 2000 that the average cost per hour of downtime ranged from $1 Million per hour for Pharmaceuticals to $2.8M/hour for Energy-sector enterprises. More recent 2015 surveys of the cost of data center downtime have ranged from $500K/hour to $1M/hour, probably across a wider sample. On the other hand, users of many agile Scrum-based systems have been satisfied with the agile proposition that most defects in the current 30-day Scrum can be fixed in the next 30-day Scrum.

States. The time and effort to effect changes or repair defects will vary by the system's internal state, such as the brittleness of its point-solution architecture, or its weak or strong modularity. The system's degree of Maintainability would also depend on aspects of its external state, such as its need to support multiple systems of systems with different stakeholders, each with different priorities and independently-evolving interfaces.

Processes. Maintainability values will also vary by the nature of the external processes or operational scenarios that the system needs to be involved in, such as changes across systems of systems. The average time to process changes across two large systems of systems ranged from 27 workdays for changes within a system, to 48 workdays for changes across a group of systems, to 141 workdays if a contract change was involved.

For variation by internal processes, a good checklist is summarized in the book Maintainability. It identifies the process steps Detection, Preparation for Maintenance, Localization and Isolation, Disassembly (Access), Repair or Removal and Replacement, Reassembly, Alignment and Adjustment, and Condition Verification (Checkout).

Relations. The 7x7 matrix of SQ Synergies and Conflicts in the INCOSE-IS 2015 paper identifies a number of relations between Maintainability and other desired SQs. For example, tightly-coupled High Performance Computing architectures and software improve computational Speed but reduce Maintainability. Easiest-first agile methods often improve initial Usability, but make architectural commitments that often subsequently degrade Scalability, Safety, and/or Security. On the other hand, investments in nanosensor-based monitoring systems improve both Dependability but also improve Maintainability. For example, aircraft autonomic logistics systems can communicate maintenance needs during flight and communicate them to the landing field, where the maintainers can prepare to turn the aircraft around in hours vs. days.

We try to quantify such relationships where possible. For example the 2015 GAO report cited in the Abstract identified annual US Government information technology (IT) expenditures of $79 billion, of which $58 billion were in operations and

maintenance (O&M). Table 2 provides more detail on fractions of hardware and software costs from recent studies.

Table 2. Percentage of Post-Deployment Life Cycle Cost

Hardware	**Software**
• 12% -- Missiles (average) • 60% -- Ships (average) • 78% -- Aircraft (F-16) • 84% -- Ground vehicles (Bradley)	• 75-90% -- Business, Command-Control • 50-80% -- Complex platforms as above • 10-30% -- Simple embedded software

The key roles of Maintainability in supporting Life Cycle Efficiency, Dependability, and Changeability caused USC to do a deep dive into Maintainability, including ways of measuring it, such as Technical Debt; Opportunity Trees for improving it, and a counterpart of Technology Readiness Levels called the Software Maintainability Readiness Framework (SMRF) for assessing management, staffing, and maintenance tools support readiness levels at milestones prior to system operation. These are described under the USC section of this Final Report.

AFIT Final Report

AFIT Contribution to Final Report

AFIT's participation in research tasks 18, 46, 113, 160 and 209 has resulted in several methods for evaluating early conceptual designs for a wide variety of "qualities" such as flexibility, robustness, affordability and mission effectiveness. A common theme for AFIT's approach is the tie-in between system architecture and concept evaluation. While this may seem ordinary at first glance, numerous examples from across the DoD suggest that architectural depictions are often disconnected from the analysis even though a good architectural model should contain the very elements needed to evaluate the anticipated value and qualities associated with proposed concepts. The following sections will review the three main areas of development for AFIT's contribution to the subject RTs.

Valuing Flexibility in Design (RT-18, 46, 113)

Often DoD leadership is less likely to attribute cost and schedule overruns to flaws in the acquisition process, but instead to the inherent lack of flexibility in the systems being developed. If weapon systems were to be designed in such a way that they are able to more readily respond to uncertainty (either in the field, or in the form of less complex modification programs), then it stands to reason that when requirements change (as they inevitably do), the impact to the program will be lessened. Among policy makers and acquisition professionals, there is broad consensus that infusing greater flexibility into DoD systems is essential to improving mission effectiveness and reducing life cycle costs. And yet, true flexibility is rarely achieved. The problem is that flexibility necessarily incurs additional investment costs. The most obvious is the direct cost associated with implementing the flexible design. There is often a direct monetary and schedule cost related to pre-provisioning (i.e., "scarring" or "stubbing") the system with nascent capabilities that can be matured to full implementation at a later time, but it may also involve developing more capability than currently required (i.e., "gold-plating"). The less obvious cost of flexibility

pertains to the tradeoffs that must be made against other performance attributes. The notion of designing to the "bleeding edge" of performance requirements is antithetical to the aims of flexibility, as it consumes engineering tradespace. An inherently flexible design cannot, axiomatically, achieve the same level of technical performance along every dimension as the performance-optimized design (think *modular* vs. *integrated*). This "capability cost" of flexibility can serve as an especially strong deterrent in DoD's contemporary, requirement-driven, performance-dominated mindset. In such an environment, the costs associated with more flexible design solutions must be assiduously justified in order to have any hope of being implemented. At present, there is no such method within the DoD.

To capture the value of flexibility in design, AFIT developed an augmented approach to designing for Life Cycle Cost, termed *Current Expected Value Life Cycle Cost Curve* or CEVLCCC (pronounced *kev' lik*). The name is intended to convey a couple of key distinctions from both the standard LCC and the notion of a stochastic LCC. The "Expected Value" phrase discriminates CEVLCCC from the standard LCC as a more probabilistically accurate measurement of system cost; whereas the word "Current" is intended to connote the fact that the CEVLCCC tool is intended to be employed as a continually updated decision analysis tool. The notion that an LCC estimate might be applied dynamically, and at lower levels of system design, is distinct from Brown's view that the stochastic LCC could only be useful for "preliminary trade space exploration" and not for value determinations "below the architectural level" (Brown & Eremenko, Application of Value-Centric Design to Space Architectures: The Case of Fractionated Spacecraft, 2008). Finally, "Curve" denotes that the output is a cumulative distribution function (CDF) of potential costs, not a single point estimate, devoid of the information depth that accompanies a probability distribution.

Under the CEVLCCC approach, the "expected value" concept is essentially a penalty that attempts to capture the anticipated cost impacts related to future baseline changes. The more cost-effectively a given design can respond to these changes, the lower the penalty. Given the inherent cost accounting methodology of the CEVLCCC approach, as long as each design is capable of achieving "some assured minimum capability," then the *corresponding military capabilities and political outcomes need not be valued*. The relative value, which is more important than the *absolute* value in the system design decision, can be inferred solely from each design's expected life cycle cost, with the best value option presumably the one with the most favorable LCC CDF.

In addition to these benefits, the proposed methodology is also highly straightforward, consisting of the following four steps:

1. Establish the System Design Options. First, the user identifies the candidate designs to be evaluated. Each design must be of sufficient maturity that its traditional life cycle cost can be reasonably estimated, and cost impacts can be estimated should there be changes related to the assured minimum capability of the system.

2. Construct Time-Phased CDFs. The user then creates CDFs to characterize potential changes to the assured minimum capability of the system. In practice, this means estimating the probability that the threshold value of existing schedule or technical performance requirements will change at various time points in the future, as well as estimating the probability that specific new requirements (with associated thresholds) will be imposed.

3. Estimate LCC Impacts. Next, the user estimates life cycle cost impacts associated with the potential changes in the assured minimum capability of the system. As part of each estimate, the user specifies a minimum and maximum cost along with an associated confidence that can range from 50 to 90 percent.

4. Select Most Favorable CEVLCCC. The CEVLCCC tool then outputs a probability curve in the form of a CDF of expected life cycle costs associated with each design. If the resulting cost curve of one design is perceived to be more favorable than the other(s), then the user now has a quantitative rationale for choosing among the candidate designs.

The CEVLCC methodology was initially demonstrated using a notional fighter aircraft example. A more extensive follow-on example was sponsored by the T-X (Fighter-Trainer) program within the Air Force's Life Cycle Management Center. The T-X application demonstrate a methodology combining CEVLCC with an Epoch-Era Analysis to quantify and estimate the value of design flexibility early in the Department of Defense's (DOD) acquisition life cycle. This method was implemented using a possible replacement to the Air Force's fighter-trainer aircraft as a baseline and a set of future requirements that would change the baseline, possibly to include Navy utilization and bomber and multi-engine requirements. The results indicated that this methodology can quantitatively measure design flexibility using existing tools when key assumptions are made. The methodology exists as a proof of concept within the domain of aircraft to quantitatively measure design flexibility early in the acquisition life cycle.

3 PREDICTING INFLUENCE OF REQUIREMENTS CHANGE (RT-137)

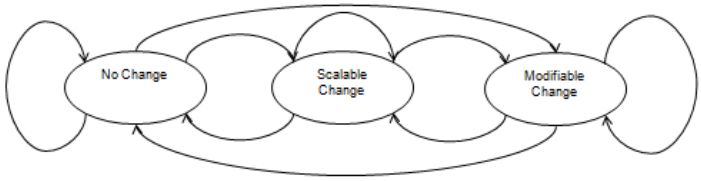
Building on the idea of flexibility in design, an effort to quantitatively predict the influence of requirements change was conducted under RT-137. Military systems have traditionally been designed to satisfy a small set of initial requirements. This approach often causes an inability for the system to easily meet future requirements. This research introduced a modeling method which uses architecture analysis to

assess the impact of change on a system module or component due to a specific requirements change. It attempts to inform the designer/decision-maker as to strategies that are more likely to mitigate the impact of potential requirements changes. The method modifies the Generational Variety Index [Martin, M. V., & Ishii, K. (2002). Design for variety: developing standardized and modularized product platform architectures. *Research in Engineering Design, 13*, 213-235] methodology by including uncertainty in (1) the likelihood of a specific requirements change, and (2) the impact of that requirements change on a system module. In addition to requirements changes, per se, we consider two types of design changes that can result from a requirements change—scalable changes, which are incremental in nature, and modifiable changes, which are more radical. Our probabilistic model examines (1) how a specific requirements change impacts a module, (2) how a given module is impacted by a combination of requirements changes, (3) how a specific requirements change impacts the entire system, and (4) how the system is impacted by a combination of requirements changes. Examples of architectural information used by this approach include requirements traceability to system functionality, allocation decisions tying functions to components, and identification of interfaces and/or design dependencies between components or modules. Results obtained from our method can further inform decisions regarding resource allocation and system design in the face of uncertainty regarding future system requirements.

Due to the uncertainty of the probability and impact of future requirement changes in complex systems, uncertainty is included in the model to measure the impact of functional change over the life of a system. Within the system architecture, functionality can be directly traced to system requirements. The model implements uncertainty into the likelihood of functional change in peacetime; likelihood of functional change in wartime; type of functional change (scalable, modifiable, or no change); and the direct impact of change on components. Direct and indirect propagation impacts are also sources of change and output on the same scale as the impact values of the direct impact of functional change. A direct impact is defined as the impact of change that occurs on a component from the initiating source (requirements change). A propagation impact is defined as the impact that occurs on a component from another component in the system undergoing change. The model includes both the direct and propagation impacts as sources of functional change. The term “impact” represents the cost and time required to implement the functional change. The metric is unitless and can be weighted based on cost and schedule preferences defined by a program. The use of multiple time periods is similar to epoch era analysis [9] where the system lifecycle is broken into intermediate chucks (time periods) to generate the output impact data. In this model, the impact value is accumulated at each time period and is labeled based on the source and type of

change. This labeling informs design decisions associated with the various functions and components of the system. The following paragraphs describe how the model generates impact values.

The Figure below is a state diagram that describes the possible types of change for a function. Each line represents a probability distribution of a functional change and a direct impact distribution. In this example, each probability distribution for a given type of change is the same regardless of the type of change recently implemented. However, the model could support a difference in these inputs if subject matter experts (SMEs) or historical data indicated otherwise.



Types of Change

The model generates impact data through the use of the random numbers and input distributions as previously described. Due to the military application, there is an input distribution for each function defining the likelihood of change in peacetime and wartime. The user specifies the expected probability of the system experiencing war within the time period specified. This current example used a notional input of 10% probability of war in a 5-year time-period. This type of modeling is commonly referred to as a "jump" step.

Each function has a likelihood of change based on the current state of change of the evaluated component. If a change does not occur then no impact on the system occurs. If the function is determined to change, then the next step is to determine what type of change is occurring. Once a change is determined, an input distribution is used to determine if the change is either scalable or modifiable. In the model, it is assumed that both types of changes cannot occur at the same time (iteration and time period) for a function. After the type of change is determined, the direct impact of change is obtained from the scalable or modifiable impact distributions of the function.

The Table below describes the probability inputs necessary for one function in the model. In this demonstration, triangular distributions are utilized as is often done in the absence of historical data. Other distributions could be used as appropriate. Once the acquisition environment is determined (peace or war), the probability of change for the present state is drawn from the associated distribution. If a change is present,

then the type of change is determined from the probability of scalable change distribution. The probability of a modifiable type change is the complement of the probability of a scalable type change.

Probability of Change Model Inputs

		Probability of Change						Probability of Scalable Type Change		
		Peace			Wartime					
Function	Present State	min	max	mode	min	max	mode	min	max	mode
i	no change	0.05	0.4	0.15	0.05	0.55	0.3	0.05	0.45	0.1
	scale	0.05	0.4	0.15	0.05	0.55	0.3	0.05	0.45	0.1
	modify	0.05	0.4	0.15	0.05	0.55	0.3	0.05	0.45	0.1

The direct impact of change is determined by specifying the distribution of "intensity" of change in a function and the weight of change. The function weighting is based on the type of change and other functions in the system. The impact weights were determined using the value increment weighting method. This method is common in determining value functions and value weights in decision analysis. The weights are then normalized between 0-10. The inputs to the model are computed by multiplying the intensity distribution values by the normalized weight. Separating the functional impact inputs allows for a SME to relate each function in the system, the type of change, and the amount of change within the function.

After an impact is determined to occur due to a functional change, the propagation impact is calculated at one and two steps from the initiating function using the functional DSM as input. Any number of propagation steps can be implemented as necessary. To implement this propagation, the model considers two DSMs that vary based on whether the initiating functional change was scalable or modifiable. A DSM is used as input for the intensity of propagation impact from the initiating functional change.

The model outputs are composed of several different data sets based on the cause of the impact (direct and propagation) as well as the types of impact (scalable and modifiable). The direct and propagation causes of impact can be summed to determine the total impact. The total impact can then be analyzed and broken into the cause and type pieces to determine possible design strategies that will lower the impact of change on the system. The model produces outputs in a way that describes the *source of change* (direct or propagation) and the *type of change* (impact of functions scaled

or modified). A determination of the overall impact of change on the system for each time period is a useful starting point for analyzing the data. Beginning at the system impact level and decomposing the data to the sources of change allows for useful change mitigation strategies to be developed to implement in design of the system.

The total impact of system change data enables an understanding of the sources (direct and propagation) as well as the types (scalable and modifiable) of change on the system. It answers the question of how the system is impacted by a combination of functional changes. The figure below, from left to right, displays the total impact of causes, total impact, and total impact of the types of change occurring on the system over time.

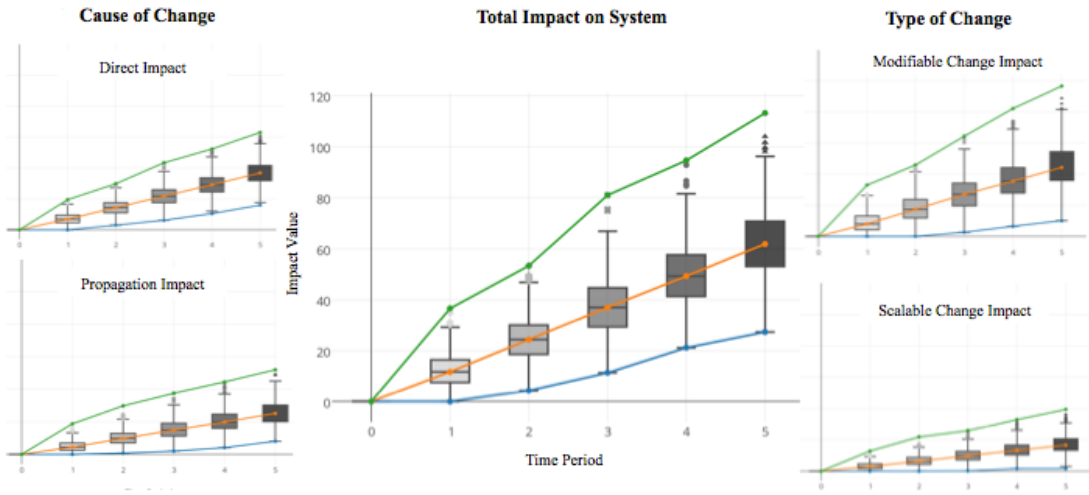
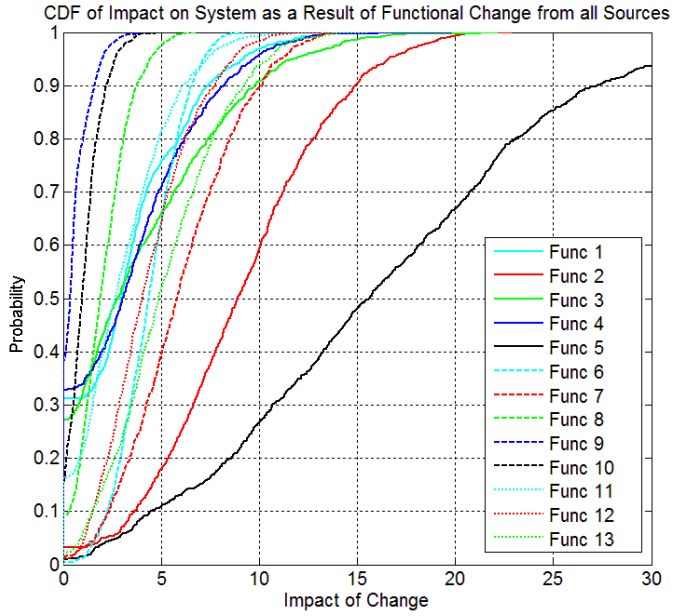


Figure 1: System Change Total Impact, Cause, and Type

From these charts, one can see that the direct and propagation impact sources of change are somewhat similar in contributing to the total impact, with propagation being slightly less. On the far right, the modifiable change is contributing significantly more than scalable change. This was expected as all the functions have a higher impact input weighting for modifiable change.

An empirical cumulative distribution function (CDF) plot of the impact data is a useful chart in understanding the change occurring in the system. The CDF chart in below describes the probability of the amount of change occurring in the system due to a function. The interpretation of the CDF plot is that shallow lines in the middle are

associated with functions with the highest probability to cause the most impact due to all sources of change. The lines in the upper-left portion of the plot are associated with functions that have the highest probability to cause the least amount of system impact due to change. The CDF plots allow for a direct comparison of the probability of change and the impact of change. From **Error! Reference source not found.** the highest and lowest functions contributing to system impact are fairly easy to determine; however, the functions with similar CDFs are difficult to discern as many lines cross.



CDF of Functional Impact on System

To better evaluate the data, the area to the left of the curve (ALC) can be calculated to rank impact of the system functions or components. The area to the left of a CDF curve represents a specific point value that can be used to evaluate and compare each CDF.

Model Centric Analysis in Early Conceptual Design (RT-137, 160, 209)

Initiated under RT-137, AFIT has been developing and demonstrating methods for creating executable models from MBSE architectures to provide early concept evaluation for mission effectiveness. Behavior and performance analysis are evaluated in the face of requirements changes and System of System architectural variations. This effort also aims to use the architectural definition to support

modeling inputs for improved cost estimation. Decisions made early in the system development cycle determine a majority of the total lifecycle costs as well as establish a baseline for long term system performance and thus it is vital to program success to choose favorable design alternatives.

The approach used is to first develop operational and system architectures to capture sets of military scenarios. A UAV based ISR concept was initially used to demonstrate the approach. These architectures are transitioned to the MBSE environments to design and demonstrate the UAV ISR tradespace. The UAV ISR concept utilized one or more vehicles to find and engage targets. Varying levels of autonomy and cooperation among vehicles were included in the architectural variations, and varying levels of target recognition quality and warhead lethality were included as system design parameters. The model was built using the Innoslate software tool, but other MBSE tools (CORE, Cameo System Modeler) were evaluated as well. A Designed Experiment was utilized to evaluate the causal relations between architectural variations, system design parameters and mission effectiveness, and statistically significant results were achieved. MBSE modeling tools are constantly improving and many are now supporting executable model, making this a promising approach to early concept evaluation.

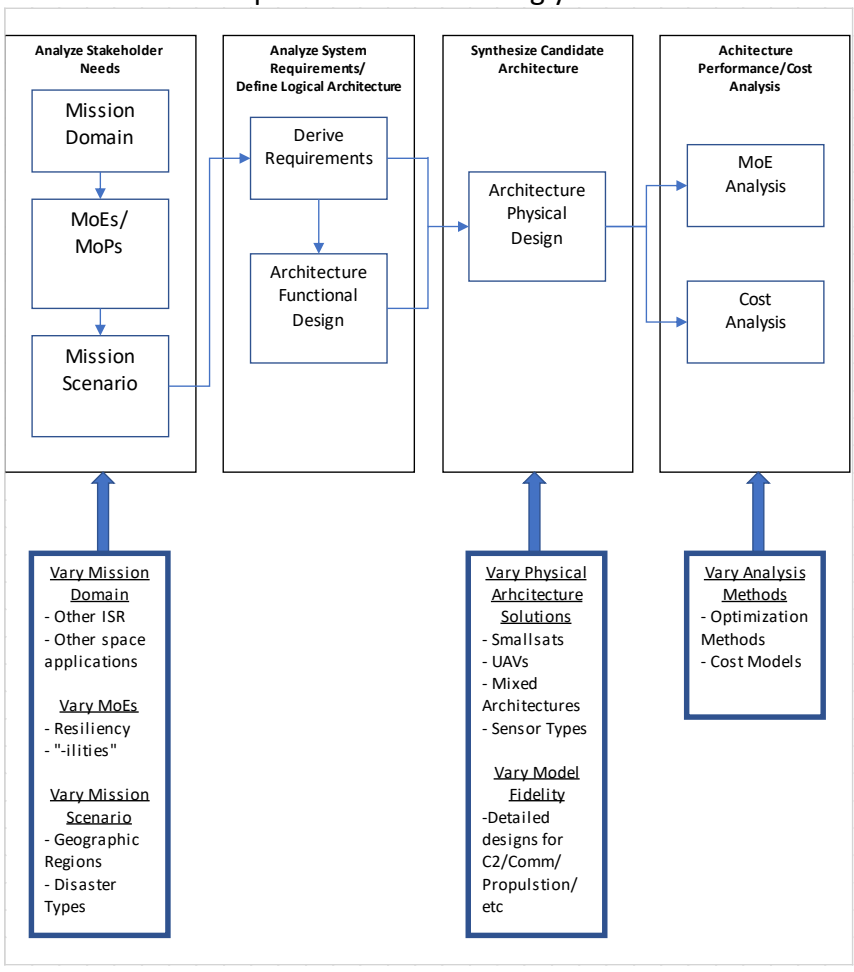
Affordability is of paramount importance in the system tradespace. DoD organizations generally separate systems engineering technical analyses from financial-community cost analyses, often leading to technical specifications with unnecessarily expensive or simply unaffordable costs. The goal of Total Ownership Cost (TOC) modeling is to enable affordability tradeoffs with integrated software-hardware-human factors. The same architectures developed for design and effectiveness analysis also support parametric software and system cost estimating approaches. Working with NPS, cost model interfaces were developed for components of the architectures in order to evaluate cost effectiveness in an uncertain future environment. We have been developing translation rules and constructs between MBSE methods, performance analysis and cost model inputs. We have been assessing SySML for extracting cost attributes and Monterey Phoenix (MP) for automatically providing cost information from the architectural models. MP can be used to extract software sizing information. It generates function point measures which are inputs into COSYSMO. More of this effort is described in the NPS section for these same Research Tasks.

Under RT-160, AFIT's architectural based cost effectiveness analysis was applied to a second domain – that of space based remote sensing. Historically, operational space missions have been performed by large, highly-capable satellite systems. While these

traditional satellite designs meet or exceed very demanding performance and reliability requirements, their drawbacks of expense and potential vulnerability have incentivized consideration of alternative architecture types. The trade space for alternative architectures is limited only by creativity; architectures for a given mission could vary by satellite size, number, orbit, payload types, design life, or any other number of parameters or concepts. Evaluation of concepts and options within this trade space is helped by a structured and systematic approach; Model-Based Systems Engineering provides such an approach. This research explored the practical usage of Model-Based Systems Engineering to assess one possible alternative to traditional satellite architectures. Since their inception in 1999, CubeSats have been used for education and technology demonstration. More recently, commercial companies have begun to deploy constellations of CubeSats for Earth imaging and other purposes. CubeSats have the potential for being both cheaper and potentially less vulnerable, though with sacrifices in capability or performance. As an example of how MBSE can be used to assess solutions from a vast space architecture design trade space, this research developed static and dynamic architectural models using MBSE and SysML comparing traditional and CubeSat architecture performance in a disaster response imagery scenario. The scenario modeled was based on the Hurricane Maria landfall over Puerto Rico in 2017.

The model developed for the traditional architecture, and its derivative CubeSat model, were used for one limited application. The figure below illustrates where “tap points” for varying trade space exist in the four steps of the OOSEM System Specification and Design Process. It would be relatively straightforward to incorporate SysML models of other design solutions for this scenario, such as UAVs or a mixture of different sensors and satellites. Specific to the CubeSat/Traditional architecture comparison, this research involved performance evaluation and relative cost estimation. The performance evaluation demonstrated the ability to meet threshold requirements using the CubeSat architecture, but fundamental limitations associated with small aperture cameras on the CubeSats did not allow it to meet objectives associated with ground resolution. The cost estimation effort, performed by NPS researchers, utilized our previous approach to parsing the architectures for cost parameters (scenarios, requirements, blocks/algorithms and interfaces) used as inputs to COSYSMO. Results from the COSYSMO analysis were inconclusive for this initial investigation as they did not reflect major differences in the cost estimates based on the two very different system scales considered; the sheer size and complexity differences associated with traditional and CubeSat architectures. This indicates the need for additional Cost Estimating Relationships (CERs) associated with the size of the physical system. These CERs are available specific to the domain of

application, but they are not fully incorporated within COSYSMO. This highlights an important area of research to be pursued in the coming years.



A final effort begun under RT-160 and continued through RT-209 involves the use of Reference Architectures to support rapid prototyping and system evaluation. Several SysML MBSE tools have become full featured modeling environments, moving beyond static architecture depictions to now include parametric trade space analysis, verification of design requirements, and evaluation of behavior and performance of the design. This enables early evaluation of a concept based on an architectural definition, well in advance of a detailed system design. AFIT has been exploring the use and reuse of MBSE architectures based on SysML. A reference architecture for sUAS has been developed in Cameo Systems Modeler, a SysML MBSE tool developed by NoMagic, Inc. The reference architecture includes a library of predefined components for common sUAS components, defined in terms of technical parameters, interfaces and behaviors. The component library includes air vehicle, ground station and mission payload components. Parametric diagrams allow

evaluation of common performance metrics such as vehicle range, mission endurance, communication range, image ground resolution, and area coverage rate. sUAS concepts can be evaluated against mission needs by developing Use Case models and Activity models in the tool to determine functional requirements for the concept. Physical architectures can then be built from the library components, with subsequent evaluation using the parametric diagrams. Multiple models can be constructed with different choices of components and/or different parameter values, thus facilitating trade space analysis.

There have been several evaluation efforts associated with AFIT's sUAS Reference Architecture. An initial effort was conducted based on a Remote Targeting System (RTS) concept previously considered by AFIT and NPS. Prior evaluation of the RTS concept was done using a non-SysML tool for performance analysis, and cost parameters were exported to COSYSMO for subsequent evaluation by NPS researchers. Results of this analysis were presented in November, 2017 at the SERC Annual Review. Since then, the RTS concept was re-created using the sUAS Reference Architecture, and a method for direct machine parsing of the architecture for the cost parameters was developed.

A second application of the sUAS Reference Architecture involved evaluation of vulnerability of sUAS. An MS student from the National Air and Space Intelligence Center (NASIC) used the architecture identify possible avenues of approach for performing counter UAS activities. sUAS rchitectures commonly used by rogue operators and lower-tier adversaries were defined using the library components in the reference architecture. These architectures included fully autonomous vehicles (no communication between vehicle and ground station), semi-autonomous control, and flight conducted via First Person Video (FPV). Attributes of the system were identified based on their ability to impact detection, tracking/orientation, and or direct attack options. The analysis will be used to define counter-UAS strategies, and will be validated with upcoming flight tests. This research was documented in a limited distribution thesis, published through the Defense Technical Information Center (DTIC).

A third application of the sUAS Reference Architecture involved a class project. AFIT offers a three course graduate specialization covering sUAS design, development and evaluation. The courses utilize a project based approach, sponsored by another AF organization, whereby a mission problem is given to the student groups, requiring them to execute a rapid prototyping process to conceptualize, design, build and flight test an sUAS concept to address the mission need. As part of the project, the students were provided with the sUAS Reference Architecture, and asked to use that

to develop their design, provide requirements traceability, and facilitate verification and validation of their approach. Students have been using the parametric diagrams to perform tradespace analysis. Use of the sUAS Reference Architecture significantly improved architecture and assessment over past years, and students have indicated that it saves cycle time by allowing them to rapidly evaluate candidate vehicle designs. The results of this evaluation will be presented at the 2019 NDIA Mission and Systems Engineering Conference.

Georgia Tech Final Report

4 BACKGROUND

This final technical report summarizes GTRI's support of the System Qualities Ontology, Tradespace and Affordability (SQOTA) effort, originally called the -ilities Tradespace and Affordability Project (iTAP), funded and executed through the Systems Engineering Research Center (SERC). Its main objective has been to provide DoD-community systems engineers with stronger foundations and methods, models, processes, and tools (MMPTs) for dealing with the complex and system-critical interactions among a system's quality attributes, also called "-ilities" or non-functional requirements (NFRs). System Qualities (SQs) are weakly and inconsistently defined, often underemphasized in DoD acquisition reviews and guidance, and end up being a major source of system acquisition and support shortfalls and overruns. In all, the SQOTA effort was executed across 7 distinct phases from 2012 through 2019.

SQOTA research was loosely composed of three distinct thrusts across team members:

- *Task 1. Research and Develop SQ Scientific Foundations* focused on developing of the -ilities ontology and associated foundations.
- *Task 2: SQ Methods, Processes, and Tools (MPTs) Piloting and Refinement* focused on piloting the development, refinement, and application of SERC methods, processes, and tools to DoD-system SQ tradespace and affordability issues.
- *Task 3: Next-Generation, Full-Coverage Cost Estimation Model Ensembles* focused on addressing future cost estimation challenges through research efforts based on strength of DoD needs and availability of DoD-relevant data, unified when possible through MBSE methods.

As illustrated in

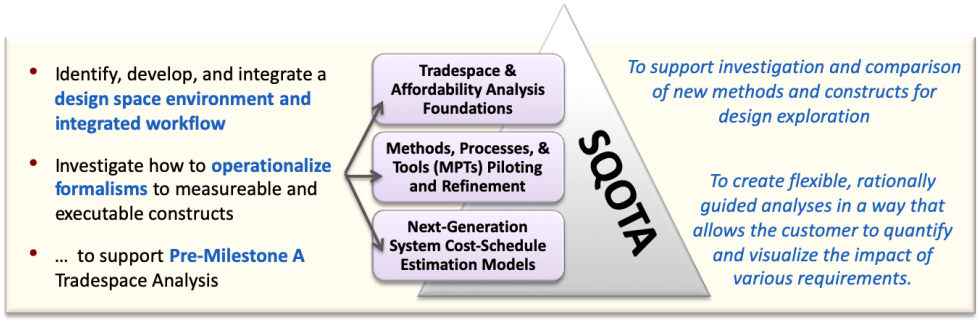


Figure 2, GTRI's efforts have focused on supporting SQOTA Task 2 and Task 3: the development of methods, processes, and tools (MPTs) necessary to realize executable evaluation of the SQs and integrating system engineering cost models developed by SQOTA collaborators (University of Southern California (USC) and Naval Postgraduate School (NPS)) with model-based systems engineering (MBSE) frameworks such as SysML and OpenMBEE. As this effort matured from its inception, and as the DoD has emphasized emerging priorities, GTRI's work supporting SQOTA similarly evolved.

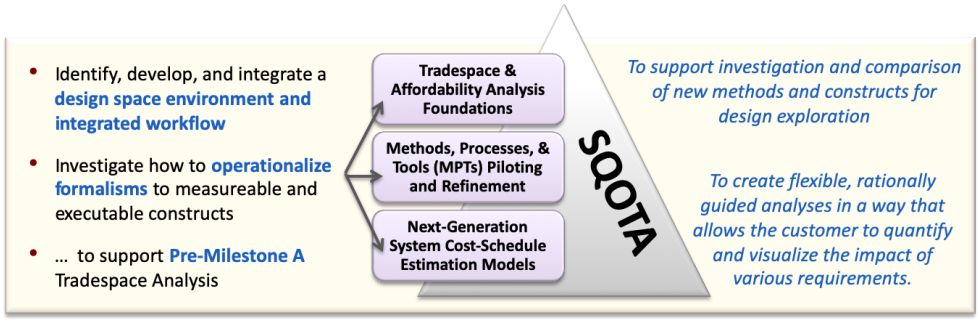


Figure 2. GTRI Objectives in support of SQOTA

5 SUMMARY OF GTRI'S CONTRIBUTIONS ACROSS SQOTA PHASES 1 - 7

The unifying theme across all phases of GTRI's SQOTA support has been a focus on defining methods, processes, and tools (MPTs) to support the evolving nature of -*ilities* (which matured into the term *System Qualities*) definition and analyses relevant to early-stage evaluation of DoD systems. Specifically, GTRI has sought to pilot proof-of-concept implementations and associated modeling constructs that can support more effective, efficient, and scalable tradespace exploration in a computational environment. A summary of GTRI's activities across all SQOTA phases is presented below:

- **Phase 1 (January 2013 – May 2013)** – GTRI laid the foundation posited the initial framework, for proof-of-concept development to further analysis of systems from a System Qualities (SQs) perspective based on SERC team member's existing capabilities.

- **Phase 2 (May 2013 – December 2013)** – GTRI extended the prior investigations to develop, open-source, web-based analytical tools outside of the SQOTA effort as proof-of-concept implementations. The goals were to identify the best MPT components and their integration to develop a flexible, integrated framework that could support analyses from various starting points in a workflow. GTRI also collaborated with SQOTA team members USC and NPS to lay the foundations that could bridge MBSE methods via SysML with cost modeling as one aspect of affordability.

- **Phase 3 (January 2014 – December 2014)** – GTRI extended Phase 2 investigations to more rigorously evaluate the analytical constructs from the prior phase and developed, open-source, web-based analytical pilot implementations as proofs-of-concept. GTRI also continued cost modeling collaboration with USC and AFIT, maturing the SysML cost modeling building blocks created in the previous phase, leveraging the COSYSMO-SoS/COCOMO legacy and experiences, and validating the approach on two case studies.

- **Phase 4 (January 2015 – December 2015)** – GTRI investigated MPT approaches that would enable SEs to add environmental dependence to (i) tradespace generation, and (ii) analyses. As part of the cost modeling collaboration, the team focused efforts to allow reuse of validation data from previous models, building on the modular SysML blocks from the prior phase, and began to investigate technology extensions to promote future reuse and extensibility

- **Phase 5 (April 2016 – April 2017)** – GTRI focused on development of a framework and associated data pipeline that would enable systems engineers (SEs) to answer unique questions, architecting the necessary analysis with respect to regions of a tradespace specific to those questions. In parallel, GTRI continued support of SQOTA Task 3 in collaboration with USC and NPS by investigating OpenMBEE to help fulfill the model interoperability framework vision.

- **Phase 6 (June 2017 – June 2018)** – GTRI focused on methods and processes that could be employed within the previously piloted frameworks and tools as well as capture different aspects of design and synthesis with operational context that would be highly relevant to DoD materiel design and development. Specifically, the effort sought to address contextual and other non-simple sources of uncertainty in tradespace analysis.

- **Phase 7 (October 2018 – August 2019)** – GTRI investigated modeling methods for use in analysis tools addressing current and newer SERC priorities established by the DoD. Specifically, GTRI evaluated metrics that would be relevant to the understanding of function (i.e., dynamic performance) of cyber-physical systems.

Further discussion may be found in the sections that follow.

GTRI Phase 1

GTRI's work in Phase 1 (January 2013 – May 2013) laid the foundation posited the initial framework, for proof-of-concept development to further analysis of systems from a System Qualities (SQs) perspective based on SERC team member's existing capabilities. GTRI investigated relevant, existing tools and their ability to capture SQs in a tradespace environment. The initial investigations were limited to those toolsets developed by SERC members involved in the SQOTA Phase 1 effort. Toolsets, frameworks, associated modeling concepts that could result in more effective and efficient tradespace exploration in a computational environment to support SQOTA types of analyses were identified and defined.

Of these, the GTRI-developed Framework for Assessing Cost and Technology (FACT) was identified as a promising toolset example that integrated a model based systems engineering (MBSE) approach and methodology to enable tradespace analysis [Browne et al., 2013; Ender et al., 2012; O'Neal et al., 2011]. GTRI investigators therefore identified key aspects of FACT that could be used across future MPTs to

help capture and analyze SQs as their definitions might mature during the course of the multi-year SQOTA effort:

- Open architecture web-based tool to enable collaborative tradespace exploration for early phase design of complex systems (initially military ground vehicles, but since extended to other applications).
- Web services based environment that enables models to be interconnected, providing a rapid exploration of the design tradespace in support of systems engineering analysis.
- MPTs such as FACT that would be model agnostic and capable of linking disparate models and simulations of both government and commercial origin through the application of community established data interoperability standards.

The Phase 1 evaluation found that a FACT-like framework and methodology might well incorporate extensions to the SERC team's methods, especially as they are defined to capture -ilities tradespace of interest. The team determined that a flexible and scalable integrating toolset based on many of the FACT tenets might be better suited to support research and integration of –ilities for DoD acquisition and design processes. Through these insights, GTRI laid the foundations for piloting proof of concept implementations in subsequent phases.

GTRI Phase 2

GTRI's Phase 2 activities (May 2013 – December 2013) extended the prior investigations to more rigorously evaluate and define the analytical constructs from the prior phase and developed, open-source, web-based analytical pilot implementations as proofs-of-concept. The goals were to identify the best MPT components and their integration to develop a flexible, integrated framework that could support analyses from various starting points in a workflow. In so doing, GTRI's contributions for the Phase 2 effort focused on two primary fronts: (1) integration of a toolset and workflow process based on open-source technologies to guide early stage design refinement, and (2) directly using this toolset and workflow to enable designers to begin assessing some key aspect of resiliency as related to evaluating early-stage design alternatives.

Recognizing widespread use of SysML across the DoD, GTRI leveraged previous research for authoring SysML models and using those models to execute design tradespace exploration [Browne et al., 2013]. The integration of these capabilities was extended to allow feedbacks within the design process and allow compatibility with NASA's OpenMDAO framework, an open-source Multidisciplinary Design

Analysis and Optimization (MDAO) framework developed by NASA Glenn and Langley Research Centers [Gray et al., 2019; https://openmdao.org]. GTRI developed an initial proof-of-concept as an open source, web-based toolset to couple a rationally guided workflow to existing analysis methods for design tradeoff evaluation. The toolset leveraged existing open source web frameworks such as Python based Django[1] and Javascript-based D3[2] to enable the rapid development of a complex, database-driven website. Including the OpenMDAO framework facilitated complex analysis by linking together the separate models describing the behavior and performance of the system of interest.

To test evaluation of SQs within the proof-of-concept toolset and workflow, GTRI implemented a streamlined analytical method inspired by aspects of MIT team member's Epoch Era Analysis (EAA) [Ross and Rhodes, 2008]. The resulting analytical basis and workflow were derived from concepts espoused by [Hazelrig, 1998], conceptually expanded by [Schultz and Fricke, 1999; Fricke and Schultz, 2005], and further delineated by [Ross and Rhodes, 2008] as EEA. These concepts were altered to support and streamline computational implementation and scalability as well as align with DoD Key Performance Parameter (KPP) requirements.

GTRI developed a critical insight that differed from prior Multi-Attribute Utility Theory (MUAT) work. Instead of scaling value to the tradespace attribute levels as traditionally done, GTRI expressed value of a given system attribute scaled against objective and threshold requirement levels using the KPP concept. This offered direct comparability from one tradespace to the next, or even from a larger tradespace to any subset. The resulting Needs Context construct was defined on flexible subsets of performance attributes relevant to the stakeholder(s) and ranking of those attributes within each. The Needs Context methodology adds a dimension of analysis to the classical utility representation as shown in Figure 3. The toolset, workflow, and example operationalized construct were described and published as part of INCOSE 2014 [Sitterle, Curry, Freeman, and Ender, 2014].

[1] https://www.djangoproject.com/
[2] http://d3js.org/

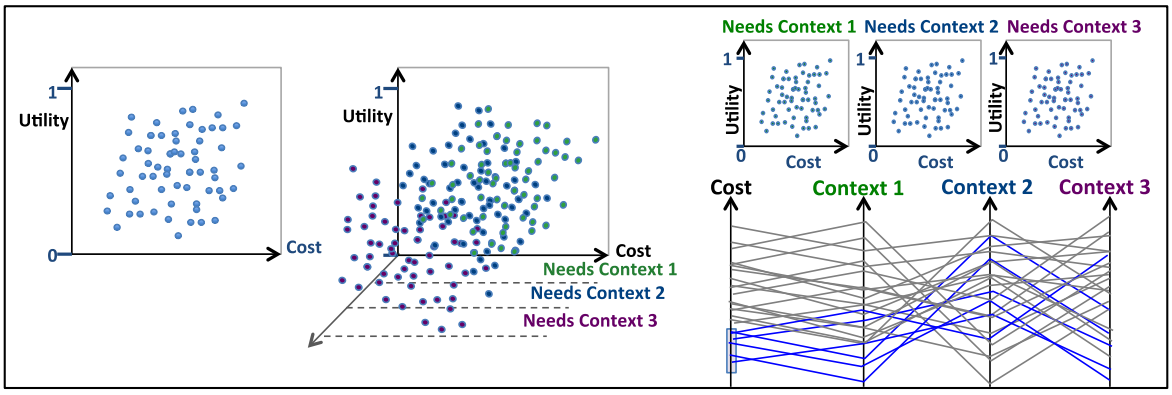


Figure 3. Classical 2D utility analysis compared to 3D Needs Context implementation

GTRI also collaborated with SQOTA team members USC and NPS to lay the foundations that could bridge MBSE methods via SysML with cost modeling as one aspect of affordability. The team took the COSYSMO-System of Systems (SoS) cost modeling concepts described in [Lane, 2009] captured them as general-purpose SysML building blocks, and applied them to a case study. This work provided the insights necessary for more rigorous implementation in the following phase.

GTRI Phase 3

In Phase 3 (January 2014 – December 2014), GTRI extended this work to (i) investigate a more rigorous grounding as well as augment and mature the Needs Context method from Phase 2, and (ii) focus on more robust integration of the processes and tools to support future extensibility to additional methods and –ilities-related analyses. GTRI also continued cost modeling collaboration with USC and AFIT, maturing the SysML cost modeling building blocks created in the previous phase, leveraging the COSYSMO-SoS/COCOMO legacy and experiences, and validating the approach on two case studies.

To more clearly understand nuances of Needs Context implementation, help the SE community understand when such methods could or should not be used in conjunction with other constructs, and the context within a larger workflow, GTRI sought to more fully define its basis in existing ontologies. GTRI conducted a review of systems evaluation according to requirement needs and changes in those needs. This included a review on flexibility performed by SQOTA collaborators at AFIT that placed many concepts relevant to resiliency into an ontological framework [Ryan et al., 2013] and new work by DoD researchers on ERS concepts defining a system being "trusted and effective in a wide range of contexts" and exhibiting "broad utility" [Goerger et al., 2014]. This established the Needs Context analytical construct, designed specifically to help evaluate robustness in the face of changing or competing

stakeholder needs, to be consistent with as robustness as defined by [Ryan et al., 2013] and the ERS notion of broad utility, leading to an expanded description of "*Robustness of Fielded System Capabilities and Capacity with respect to Operational Requirements*".

The GTRI team also investigated different methods that might help modify this approach if the attributes specified as part of the value equation were not truly independent from a stakeholder perspective as assumed in MUAT. GTRI found that the exponential complexity of evaluating a design alternative "valuation" using fuzzy integrals and other methods designed to cope with potential interactions makes these methods exceeding difficult to use for moderate to large numbers of contributing attributes. Further, many preference ranking and information theoretic measures rely on correlation measures expressly in a given data set. This would eliminate the direct comparability of one tradespace analysis to the next or even across another analytical "filtering" method applied earlier in the process.

In light of these findings, the GTRI team next considered how uncertainty could be brought into the additive function for broad utility and carried forward as part of the analytical process. Two distinct cases were considered: (i) ordinal ranking of attributes was provided but without certainty, or (ii) no ordinal ranks at all were provided. Adding uncertainty to weights for MUAT problems was defined previously in the literature, and a discussion may be found in [Boehm et al., Dec 2014]. GTRI built from this work to apply uncertainty to the weights for an additive value function, creating a stochastic version of the analysis achieved via an efficient Monte Carlo on a specified random distribution for (i) and constrained according to an ordinally defined convex polytope for (ii). The benefit of including uncertainty in this manner is that it allows SEs to move away from deterministic evaluations without requiring in depth knowledge about models or other sources of uncertainty that may not be available to the analyst.

On the Cost Modeling task, GTRI continued collaboration with USC and AFIT, maturing the SysML cost modeling building blocks created in the previous phase, leveraging the COSYSMO-SoS/COCOMO legacy and experiences. The team successfully validated this knowledge capture via two healthcare SoS case studies: base complexity (from [Lane, 2009]) and increased complexity (from [Lane, 2010]). The approach enabled better knowledge capture than previous, more ad hoc methods. It proved to be more modular, reusable, precise, maintainable, and complete (e.g., units). It also offered better avenues for verification and validation than traditional use of spreadsheets. The patterns were easy-to-apply with practically any system or SoS, with an easy ability to swap in/out alternative subsystem designs.

The work provided a basis to integrate with existing body of system models, including executable system models represented in SysML and/or DoDAF/UPDM.

GTRI Phase 4

For Phase 4 (January 2015 – December 2015), GTRI investigated MPT approaches that would enable SEs to add environmental dependence to (i) tradespace generation, and (ii) analyses. GTRI chose this focus because the model of a system alone is insufficient to produce all quantitative system attributes important to the decision making process; many system attributes critical to comparative analyses of operational performance are environmentally dependent.

These efforts helped GTRI articulate how to define an operational scenario in terms of two classes of parameters necessary to evaluate the performance/ other measures of a system: operational environment (physical environment parameters) and how a system is used (i.e., Concepts of Operation (CONOPS)). This delineation served as intuitive scaffolding for modelers to specify parameters needed for tradespace generation with operationally-specific attributes, enabling parts to be defined, reused, and expanded upon for creation of new operational scenario blocks. GTRI investigated various degrees of modularity in implementation and how this may impact reusability, the backend data model, and interface semantics necessary to facilitate subsequent analyses. For this work, GTRI built on open source technologies (i.e., Python, OpenMDAO) to enable future interoperability and extensibility. GTRI also investigated orchestration of the analytical components in such a way as to recognize data dependency and preserve the "behavior" of the analysis through a data pipeline. During this work, the team evaluated various insertion points to this process within the framework to broaden the usability for SEs.

In collaboration with GTRI, USC, and NPS cost modeling team members, a high-level evaluation for how COSYSMO cost estimation drivers might be pulled from a SysML description of a system via a script (i.e., in an automated manner) was conducted. This work determined that further maturation of the analogous parameters and an evaluation of feasibility would need to be developed for a specific problem type, i.e. be domain specific.

The cost modeling team also developed a version of the cost model that used the Bloom taxonomy to completely revise the "understanding" cost drivers. This seemed to improve the definitions of these cost drivers. However, it was realized that validation data from previous versions of the model could not in general be carried forward to envisioned COSYSMO 3.0 validation. The team consequently constrained remaining efforts to allow reuse of validation data from previous models, building on

the modular SysML blocks from the prior phase and beginning to investigate technology extensions to promote future reuse and extensibility.

GTRI Phase 5

GTRI's Phase 5 (April 2016 – April 2017) activities focused on development of a framework and associated data pipeline that would enable systems engineers (SEs) to answer unique questions, architecting the necessary analysis with respect to regions of a tradespace specific to those questions. In parallel, GTRI continued support of SQOTA Task 3 in collaboration with USC and NPS by investigating OpenMBEE to help fulfill the model interoperability framework vision.

The Phase 5 goal was to enable an SE to evaluate and specify the order in which various analytical and/or filtering methods are applied. This resulted in a Pipelining Analysis and Workflows (PAW) framework, implemented as a systems engineering (SE) toolbox using open source technologies. The PAW framework tied analytical components (building blocks such as sensitivity analyses, regression models, etc.) to data pipelines relevant to the question SEs are trying to ask, especially with respect to operational context (as investigated in Phase 4). The PAW framework enabled SEs to evaluate specific regions of a tradespace (as opposed to traditional, monolithic tradespace analysis) and architect an analytical workflow (what methods/ math to apply in what order). The open source general-purpose programming language Python was used as the foundation for PAW development; Python packages Luigi and Pandas provided the core mechanism for component (analysis) execution and pipelining in the concept framework and the basis for data structuring and core operations in the concept framework, respectively. The complete development is discussed in [Boehm et al., Apr 2017].

The initial framework build included a base of simple components built to demonstrate examples: basic column operations (filter, join, merge, add, multiply, etc.), some statistical analysis components (normalize, correlate), and other components created for example workflows. Example ordinary linear regression response surface and neural network-based components used to develop surrogate models were also created, as were a few components to utilize the models (i.e., Monte Carlo simulation, and Sobol sensitivity analysis). The components made for the proof-of-concept were utilized to create several example workflows for some example systems engineering and data science problems. The core PAW framework and its functionality are shown in Figure 4. At the end of Phase 5, GTRI stood PAW up on a temporary virtual server to share with other SQOTA team members for their exploration.

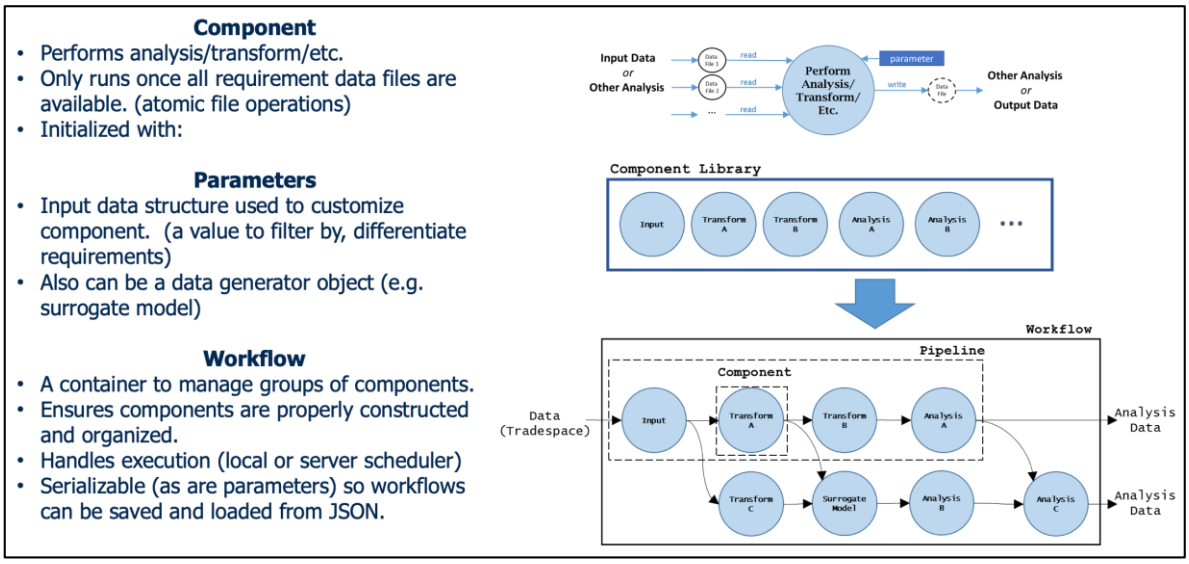


Figure 4 Core PAW Framework

For the SQOTA Task 3 portion of Phase 5, the GTRI-USC-NPS team investigated implementing an initial OpenMBEE "model-based wiki" frontend for the cost modeling building blocks. The team envisioned an easy-to-use web browser frontend providing rich SysML-based backend capabilities to a general audience (i.e., they can take advantage of SysML without having to learn SysML) as the ultimate goal of implementing OpenMBEE for cost modeling. Model-based document generation (such as pdfs) was another benefit from the same technology implementation. Therefore, the team also implemented an initial version of templates to create typical cost modeling summaries as pdf documents for multiple purposes such as distribution to people not familiar with SysML, long-term archiving, usage in traditional document-based processes, and so on. This allows users to take advantage of the rich modeling in SysML as well as the generation of familiar and ubiquitous pdf-based artifacts. The GTRI-USC-NPS team outlined a SysML cost modeling roadmap in the 2015-2016 report.

GTRI Phase 6

For Phase 6 (June 2017 – June 2018), GTRI focused on methods and processes that could be employed within the previously piloted frameworks and tools as well as capture different aspects of design and synthesis with operational context that would be highly relevant to DoD materiel design and development. Specifically, the effort sought to address contextual and other non-simple sources of uncertainty in tradespace analysis.

The work in Phase 6 centered on how to choose a "best design" based on highly uncertain evidence. To capture uncertainty in inputs to a system model and also in the model itself (and its simulation) respectively, the approach focuses on uncertainty in the operational scenarios a notional ballistic missile defense (BMD) system might face as well as uncertainty associated with the technological and system performance. The former is defined as inputs based on expert elicitation while the inherent uncertainty associated with the latter is modeled via stochasticity and non-deterministic system behavior. In the adaptation of methods described by [Argarwal et al., 2004] created for this effort, simulated "experts" provide basic probability assignments (BPAs) over a set of intervals for each uncertain input variable. In the case of cyber-physical system security, this problem is analogous to representing some potential unknown attacks to the system based on expert elicitation. The work demonstrated that Dempster Shafer Theory (DST) can be used in tandem with non-deterministic system modeling to account for uncertainty across both system performance and operational scenarios that might be encountered as a tool to explore and refine system design parameters.

GTRI Phase 7

In Phase 7 (October 2018 – August 2019), GTRI investigated modeling methods for use in analysis tools addressing current and newer SERC priorities established by the DoD. Specifically, GTRI evaluated metrics that would be relevant to the understanding of function (i.e., dynamic performance) of cyber-physical systems.

The SERC's 2019-2024 Technical Plan strongly emphasizes improving Security, Safety, and Interoperability of systems participating in multiple, increasingly rapidly evolving systems of systems. Consequently, SQOTA team members were asked to incorporate concepts of Security, Safety, and Interoperability to the research focus in Phase 7 as related to the acquisitions process and design phases, with associated decision analysis included in that process. GTRI supported that direction by investigating modeling methods most suitable to capture functional capabilities of cyber-physical systems, specifically viewing security as an application, that can in turn produce abstract views of dynamic processes of those systems in a computational environment.

The work sought to discover analytical constructs that would be agile with respect to being applicable to different systems within the specified domain of cyber-physical systems and scalable with respect to the size of problems that may be encountered in the future. This led to a focus on identifying analytical foundations to advance trade-based analyses for systems requiring a dynamic simulation to reveal capabilities and performance associated with their design.

6 ACTIVITIES

Prior SERC investigations have tended to converge on the notion of representing cyber-physical systems as graphs or directed graphs. GTRI's Phase 7 work consequently started from this view and began with a literature review on representation of cyber-physical systems as graphs (preferable directed graphs) and associated metrics. This review included the System of Systems and Analytic Workbench effort led by Purdue, the System Aware Security work led by the University of Virginia, and a recent pilot investigation lead by Georgia Tech.

GTRI specifically sought to identify and evaluate (i) what metrics would relate to/ reveal functional behavior or properties of such systems, and (ii)how these metrics could be used in tandem with dynamic simulation to reveal changes in functional state (i.e., to show whether function is preserved in these systems given some disruption).

7 INSIGHTS

Numerous static graph metrics are available in the field of network science and may be found in most textbooks on the subject. There are directed graph analogs for most but not all undirected graph measures. Static measures reflecting whole-graph statistics can be highly misleading for evaluation of functional graphs of cyber-physical systems meant to help elucidate preservation of performance. A review of work on critical infrastructure and cyber networked systems [Li, 2007] showed the danger of relying on static graph measures to reflect the function and potential vulnerabilities of a cyber-physical functional graph. [Li, 2007] demonstrated that graphs with identical degree distributions could present with significantly different structures and consequently completely different engineering performance. The performance-based topology metric developed by Li was more suited for router-based technology networks, but the lessons were relevant: static metrics alone could be very un-revealing of the performance dynamics associated with cyber-physical systems that will be required for meaningful evaluation.

Even so, static measures of in-degree and out-degree can be helpful on a node basis. In-degree is the number of graph edges directed into a given node, reflecting its susceptibility. Out-degree is the number of graph edges directed out of a given node, indicating how many other nodes it can immediately and directly influence. Local measures of in-degree and out-degree highlight "transmitter functional nodes" and "receiver functional nodes". Specifically, nodes with the highest in-degree measures will be those at a high risk of disruption from any one of a number of influencing functions. Similarly, if a given set of nodes have a substantially higher out-degree

than most nodes in a graph, those nodes may be thought of as those most able to transmit failure or corruption due to some disruption or threat. Protection measures should be sure to account for these functional elements.

The clustering coefficient of a node is the measure of edges between the node's direct neighbors relative to the total number of possible edges between its direct neighbors, where 0 indicates that the node is a star (no direct connections between neighbors) and 1 indicates that the node is the center of a local clique (all neighbors are directly connected to each other). Depending on the types of intrinsic system functions that exist and potential or likely threat types, nodes in a tight cluster are likely to be affected by the threats in similar ways; the majority of nodes in a tight cluster could all be degraded by a particular threat, they could be deceived, or they could be resistant to the threat's influence. When a functional graph has a high average clustering coefficient, rapid near-homogeneity is expected in response to threats, whereas a low average clustering coefficient may indicate that threat effects follow a clearer path.

Similarly, graph modularity is another way to identify the degree to which groups of nodes tend to respond to threats. A graph's modularity indicates how tightly knit nodes are within clusters and how sparse the connections are between clusters. Unlike the clustering coefficient, modularity takes into account longer paths than just direct neighbors, and it's dependent on the relative isolation of the clusters from each other. Graph modularity is based on how an analysist chooses to partition (i.e., cluster) a graph, resulting in different modularity outcomes for different graph cuts. A specific method or set of methods specifically relevant to the problem of cyber-physical system functional behavior will need maturation alongside of maturing graph representation and simulation of these systems. Once clustering has been performed, the modularity can be established by determining the number of connections within each cluster relative to the total number of connections.

When analyzing functional graphs, distinct networks (revealed as parts of a system's graph that are not connected to other parts of a given system's graph by any edges) indicate that the networks have absolutely no influence on each other's functionality. The number of these sub-graphs should be determined prior to dynamic simulation. In many cases, disconnected networks are the result of an erroneous assumption or modeling mistake (such as forgetting an input, output, or an entire functional node), as functionality is usually fully connected through some path. In the rare case of a system or system-of-systems with multiple unrelated capabilities that do not utilize any overlapping functionality, multiple networks may be appropriate in a functional graph.

For a directed graph-based representation of the functional architecture of a cyber-physical system, the analysis will focus on dynamics *on* the graph, i.e., what dynamic processes take place and are preserved or compromised over time. This is distinct from dynamics of a graph where the entire graph structure changes over time. This understanding suggests a model for "if/then" nodes in a system graph model where functional flow proceeds on any one of two or more potential pathways. All possible edges may be drawn a priori, with default edge weights equal to 1. For if/then nodes, outgoing edge weights will be binary, switching to 1 or 0 depending on the outcome. This could produce a scalable, fast implementation of more complicated behavior dynamics, It also suggests caution for whole-graph metrics and a focus more toward time series analysis.

The intent of dynamic simulation of functional system behavior using directed graphs as a model, is to determine if the overall state space is preserved. That is, the goal is to reveal of critical system capabilities continue to function and needed despite disruption or threat occurring somewhere within the functional structure. This may be represented as changes in node state, each node having a state that may change across time steps depending on the nature and/or values of its inputs. Node states may therefore be analyzed across time steps of a dynamic simulation to look for both global and local state patterns. These patterns can reveal whether functional states are stable, vary significantly, gradually trend toward capability, restored capability, or failure, or even rapidly fail.

Examples of time series measures relevant to these problem types are auto-correlation, cross-correlation, changes in slope over time, and recurrence quantification analysis. Auto-correlation is the measure of similarity of a signal with itself. In this case, it would measure the magnitude of time dependency between changes of state for a given node state, revealing the extent to which given states (i.e., behaviors) are related to previous states in that node's time series. Cross-correlation is a measure of similarity of two different time series as a function of the displacement of one relative to the other. This can reveal whether certain behaviors tend to lag behind which other processes or whether they oscillate together. Changes in slope with time, itself a change in state with time, can also reveal stability or failure patterns and how quickly they manifest. Recurrence quantification analysis (RQA) includes a variety of measures to look for patterns of recurrence in a dynamic data set. This approach can help determine whether a system's functional states are stable, periodic, drift over time, become suddenly disrupted, etc. This is especially helpful for understanding how a system responds to threats over time.

These time series measures may be evaluated over a whole graph, for nodes identified as critical "transmitter" or "receiver" nodes via measures described

previously, or over sliding Markov blankets to reveal a neighborhood (systemic but also local) behavioral view. A given node's Markov blanket consists of its direct parents, direct children, and the other parents of its direct children. A Markov blanket therefore provides a local perspective of what influences a node and the effects of the node's influence.

GTRI's research under this last phase of SQOTA shifts the MPT work to more specifically focus on the needs facing analysis of dynamic processes on cyber-physical systems, in line with SERC strategy. It will serve as foundational work to connect to and advance the MPT framework development in future follow-on efforts from RT-204. Phase 7 has provided paths forward to further the dynamic analysis within the computational framework pilot established in Rt-204.

Overall Insights

Typically, decision making is viewed strictly as an analytical problem. The process defines alternatives, evaluates them according to predefined criteria and requirements or objectives, and compares the alternatives to find the best performers. This process is well-structured and executable in a computational environment. It can, however, miss insights that different methods can reveal, especially when dealing with uncertainty. Tradespace analyses are needed to support key decision makers and are dominantly based on multi-additive value decision criteria or a variant of this approach. While well-understood and embraced, some aspects of the problem critical to inform design and stakeholder decisions are not well-addressed via these traditional analyses where uncertainty is usually propagated through as simple distributions on input variables.

Over the course of SQOTA, GTRI's work sought to develop and mature methods, processes, and tools that would enable an SE to architect tailored analyses based on defining and redefining requirements and associated preference hierarchies, specifying different regions of the tradespace data, and defining the specific analytical tools and their order of implementation (in terms of how the data is treated). GTRI focused on modular, reusable, scalable implementations of the necessary building blocks for these analyses. Some key insights gleaned during the course of this work include that the model of the system alone is insufficient to produce all quantitative system attributes that are typically important to the decision making process, and this has implications for structuring an integrated toolset for tradespace exploration. Performance-based attributes (e.g., braking distance), for example, require a system model to couple with basic engineering representations of the system's operation and/or operational environment.

Another key insight was that most standard forms of utility evaluation derive from normalizations of the current design space with a single value function for each performance attribute. That utility is then not comparable from one analysis to the next when different performance attribute ranges are generated from differences in input variable ranges or system architectures. Similarly, using single value functions for each performance attribute implicitly assumes non-competing preferences across different stakeholders, mission profiles, etc. GTRI's adjustment to use a KPP concept to scale value (as a non-formal proxy for utility) to defined objective and threshold requirement levels avoids both of these limitations and allows for simultaneous visualization and evaluation of competing objectives.

Conclusions

Tradespace analyses are needed to support key decision makers, and some questions critical to informing these decisions are not well-addressed via traditional, more globally focused analyses. Systems engineering questions unique to a given system or problem will often require similarly unique analytical workflows supported by contextually relevant data. Multiple specific systems engineering insights can be gleaned from exploration of specific analysis pathways rather than over-simplified global analysis. To address this issue, GTRI has sought to tie analytical components (building blocks such as sensitivity analyses, regression models, etc.) to data pipelines relevant to the question we are trying to ask.

The primary, unifying objective of GTRI's prior efforts was to develop an integrated workflow process to guide design exploration. Further, this workflow was designed to explicitly consider how the context in which a system is used influences its overall value to stakeholders and, importantly, across simultaneously competing or sequentially changing needs of these stakeholders. As it is used here, context can refer to how the additive value of a system varies between stakeholders, or temporal differences in a system's application over its lifecycle that impact its perceived usefulness. Further efforts investigated how to facilitate modeling in support of tradespace generation and analysis to capture specific operational scenario needs, hence increasing operational relevance of these approaches.

GTRI also decomposed the analytical constructs into smaller reusable functional building blocks implemented in Python programming language as steps in a data pipelining tool. This laid the foundation for GTRI's Phase 2 and 3 work to be matured under another DoD effort funded through the SERC on the Engineered Resilient Systems program, performed with and for the US Army Engineer Research and Development Center. This maturation and transition of the analytical portion of the work was published in [Sitterle, Freeman, Goerger, and Ender, 2015].

A major understanding from GTRI's SQOTA efforts is that generating a tradespace from various models is not a trivial task if the goals are to achieve flexibility, scalability (often via properly orchestrated modularity), and efficiency of the process. Also, a use case has a specific path through a networked workflow. In addition, -ilities are often defined according to life cycle stage or blur across several; care must be taken to operationalize appropriately. Specifying the precise way in which any analytical construct applies to tradespace analysis and also its specific lifecycle context is critical to future synthesis with other methods. Composability and traceability of constructs is key to future maturation using other methods in tandem. Driving the tool development with a generalized workflow helps ensure MPTs are developed in such a way to meet the requirements of future use cases. GTRI discovered through this work that the degree of modularity and the extent of the abstract description necessary to define the problem in a way that is directly executable are strongly linked and tremendously important to usability by a person and reusability in a computational environment.

GTRI built on the foundations developed through SQOTA and transitioned them under a different effort for US Army/ERDC, Engineered Resilient Systems that began with funding sent through the SERC. This work resulted in further carry over of concepts developed initially under SQOTA, matured within a DoD application, and further disseminated collaboratively with the sponsor, US Army/ERDC. This work is cited in published literature listed in Appendix A.

GTRI's final SQOTA work had helped develop an understanding of the next evolution of these MPTs for the SERC's focus in Security, Safety, and Interoperability. GTRI will propose that portion of the effort be transitioned and matured in collaboration with Stevens and the University of Virginia under the next phase of RT-204.

Additionally, the cost modeling work in collaboration with USC and NPS effectively demonstrated how the basic approach incorporate COSYSMO-SoS and related concepts in a modular SysML building block fashion. This puts COSYSMO-SoS capabilities within the broader SE context that can be built upon and tied together with other tradespace analysis tools for larger-scale case studies involving multiple "–ility" considerations.

GTRI's overall goal in support of SQOTA has been to develop and mature methods, processes, and tools whereby Systems Engineers can intelligently create the data we need to answer SE questions critical to understanding what is driving decisions for DoD decision makers. Together, these capabilities will help the DoD community address more complex environments, operational scenarios, and multi-criteria analyses within an executable toolset.

Appendix A: List of Publications Resulted

Boehm, B. et al. (9 July 2013). Tradespace and Affordability – Phase 1, Technical Report SERC-2013-TR-039-1.

Boehm, B. et al. (31 December 2013). Tradespace and Affordability – Phase 2, Technical Report SERC-2013-TR-039-2.

Boehm, B. et al. (31 December 2014). -ilities Tradespace and Affordability Project – Phase 3, Technical Report SERC-2014-TR-039-3.

Boehm, B. et al. (16 February 2016). System Qualities Ontology, Tradespace and Affordability (SQOTA) Project – Phase 4, Technical Report SERC-2016-TR-101.

Boehm, B. et al. (30 April 2017). System Qualities Ontology, Tradespace and Affordability Project, Phase 5. SERC-2017-TR-105.

Boehm, B. et al. (21 June 2018). System Qualities Ontology, Tradespace and Affordability (SQOTA) Project – Phase 6, Technical Report SERC-2018-TR-108.

Sitterle, V. B., Curry, M. D., Freeman, D. F., & Ender, T. R. (2014, July). 4.3. 3 Integrated Toolset and Workflow for Tradespace Analytics in Systems Engineering. In *INCOSE International Symposium*, 24(1), 347-361.

Sitterle, V. B., Freeman, D. F., Goerger, S. R., & Ender, T. R. (2015). Systems engineering resiliency: guiding tradespace exploration within an engineered resilient systems context. *Procedia Computer Science*, 44, 649-658.

GTRI built on the foundations developed through SQOTA and transitioned them under a different effort for US Army/ERDC, Engineered Resilient Systems that began with funding sent through the SERC. This work resulted in further carry over of concepts developed initially under SQOTA, matured within a DoD application, and further disseminated collaboratively with the sponsor, US Army/ERDC, specifically Dr. Simon Goerger:

> Sitterle, V.B, Freeman, D.F., Goerger, S.R., and Ender, T.R. “Systems Engineering Resiliency: Guiding Tradespace Exploration within an Engineered Resilient Systems Context,” at 2015 Conference on Systems Engineering Research (Hoboken, NJ), *Procedia Computer Science*, 44: 649-658, 2015.

Sitterle, V.B., Gorger, S.R., Freeman, D.F., and Ender, T.R. “Measuring and Analyzing Multiple Perspectives of System Resilience in a Tradespace Tool,” 83rd Military Operational Research Society (MORS) Conference, Arlington VA, June 2015.

Sitterle, V.B., Gorger, S.R., Ender, T.R., and Freeman, D.F. “Defining Resilience for DoD Systems,” Industrial and Systems Engineering Conference (ISERC), Nashville TN, June 2015.

Ender, T.R., Gorger, S.R., Sitterle, V.B., and Freeman, D.F. “Tradespace Analysis - Tools and Processes for Engineered Resilient Systems,” Industrial and Systems Engineering Conference (ISERC), Nashville TN, June 2015.

Sitterle, V.B., Brinhall, E., Balestrini-Robinson, S. Freeman, D., Goerger, S., and Ender, T. “Bringing Operational Perspectives into the Analysis of Engineered Resilient Systems,” at 2016 Conference on Systems Engineering Research (Edinburgh, Scotland, UK), *Procedia Computer Science, 2016.*

Sitterle, VB., Freeman, D.F., Balestrini-Robinson, S., Arruda, J., Goerger, S., Ender, T.R. “Tradespace Tools for Engineering Resilient Systems,” for the Military Decision Analysis session at The Institute for Operations Research and the Management Sciences (INFORMS) Annual Conference, Nashville, November 2016.

Sitterle, V.B., Brimhall, E.L., Freeman, D.F., Balestrini-Robinson, S., Ender, T.R. and Goerger, S.R. (2017). “Bringing Operational Perspectives into the Analysis of Engineered Resilient Systems,” *INSIGHT*, 20(3), 47-55.

Appendix B: Cited and Related References

Agarwal, H., Renaud, J. E., Preston, E. L., & Padmanabhan, D. (2004). Uncertainty quantification using evidence theory in multidisciplinary design optimization. *Reliability Engineering & System Safety*, *85*(1-3), 281-294.

Browne, D., Kempf, R., Hansen, A., O’Neal, M., & Yates, W. (2013). Enabling systems modeling language authoring in a collaborative web-based decision support tool. *Procedia Computer Science,* 16, 373-382.

Ender, T. R., Browne, C. D., Yates, W. W., & O'Neal, M. (2012). FACT: an M&S framework for systems engineering. In *Interservice/Industry Training, Simulation, and Education Conference (I/ITSEC)*, 3-6.

Fricke, E., & Schulz, A. P. (2005). Design for changeability (DfC): Principles to enable changes in systems throughout their entire lifecycle. *Systems Engineering*, 8(4).

Goerger, S. R., Madni, A. M., & Eslinger, O. J. (2014). Engineered resilient systems: A DoD perspective. ARMY CORPS OF ENGINEERS VICKSBURG MS ENGINEER RESEARCH AND DEVELOPMENT CENTER.

Gray, J. S., Hwang, J. T., Martins, J. R., Moore, K. T., & Naylor, B. A. (2019). OpenMDAO: An open-source framework for multidisciplinary design, analysis, and optimization. *Structural and Multidisciplinary Optimization*, 59(4), 1075-1104.

Hazelrigg, G. A. (1998). A framework for decision-based engineering design. *Journal of Mechanical Design*, 120(4), 653-658.

Lane, J. (2009, April). Cost model extensions to support systems engineering cost estimation for complex systems and systems of systems. In *7th Annual Conference on Systems Engineering Research*.

Li, L. (2007). *Topologies of complex networks: functions and structures* (Doctoral dissertation, California Institute of Technology).

O'Neal, M., Ender, T. R., Browne, D., Bollweg, N., Pearl, C. J., & Bricio, J. L. (2011). Framework for assessing cost and technology: an enterprise strategy for modeling and simulation based analysis. In *Proceedings of MODSIM World 2011 Conference and Expo*, Virginia Beach, VA.

Ross, A. M., & Rhodes, D. H. (2008, June). 11.1. 1 Using Natural Value-Centric Time Scales for Conceptualizing System Timelines through Epoch-Era Analysis. In *INCOSE International Symposium*, 18(1), 1186-1201.

Schulz, A. P., & Fricke, E. (1999, October). Incorporating flexibility, agility, robustness, and adaptability within the design of integrated systems-key to success?. In Gateway to the New Millennium. 18th Digital Avionics Systems Conference. Proceedings (Cat. No. 99CH37033) (Vol. 1, pp. 1-A). IEEE.

Sitterle, V. B., Curry, M. D., Freeman, D. F., & Ender, T. R. (2014, July). 4.3. 3 Integrated Toolset and Workflow for Tradespace Analytics in Systems Engineering. In *INCOSE International Symposium*, 24(1), 347-361.

Sitterle, V. B., Freeman, D. F., Goerger, S. R., & Ender, T. R. (2015). Systems engineering resiliency: guiding tradespace exploration within an engineered resilient systems context. *Procedia Computer Science*, 44, 649-658.

MIT Final Report

Massachusetts Institute of Technology – TASK 1

8 ILITIES PRESCRIPTIVE SEMANTIC BASIS

One of the fundamental challenges for developing a clearer understanding of the semantics of "ilities" (sometimes referred to as system qualities) is continuing ambiguity in these terms. This SERC task has investigated use of a *prescriptive semantic basis*, resulting in a framework for formulating ility "definitions." The research has evolved the semantic basis construct, generated a concept for a translation layer for ease of use, and formulated ilities metrics. Synergies with other relevant research were investigated, and potential applications of the semantic basis approach were explored. Several future directions for ilities research within the systems community have emerged from the project.

BACKGROUND

One of the fundamental challenges for developing a clearer understanding of the semantics of "ilities" (sometimes referred to as system qualities) is the current ambiguity in these terms, which are often used colloquially and therefore inherit informal meaning. Use of "ilities" in some technical disciplines extend from well-accepted prescription; however this has not yet occurred in the systems community as evidenced by the abundance of definition offering papers with conflicting meanings. Additionally, these terms display both polysemy (having multiple meanings that are semantically related) and synonymy (multiple terms having similar meaning). An example of polysemy is two different, but related meanings for flexibility: "able to be changed" and "able to satisfy multiple needs". An example of synonymy is the interchangeable use of terms such as flexibility (able to be changed) and changeability (able to be changed or change itself). One root cause of the ambiguity in technical usage of ilities is that typically ilities are considered one at a time in the literature. Recent research suggests considering sets of ilities has merit (Ross, Rhodes & Hastings, 2008; Ross, Beesemyer & Rhodes, 2011; de Weck, Ross & Rhodes, 2012; Ross & Rhodes, 2015). Further, there is growing recognition of the importance to consider ilities in relationship to other ilities and larger context (Lee & Collins, 2017; Daclin et al. 2018; Lowe, 2018).

APPROACH

A prior ilities study looked at co-occurrence of ilities terms in the literature, with implied dependence amongst terms, resulting in a directed graph that tempts reading causal relationships into the links (de Weck, Ross & Rhodes, 2012) but the existence of "co-occurrence" cannot describe the nature of the link. As a complementary approach to discovering relationships amongst ilities, prescriptive assertions can be based upon theory or experience, making conceptual leaps in proposing how ilities should relate to one another.

Building upon the insights from looking at various approaches for describing ilities, and drawing inspiration from the linear algebra concept of a basis as a spanning set that defines a space, the research has proposed a prescriptive semantic basis for consistently representing change-type ilities within a particular semantic field (Ross & Rhodes, 2015). The semantic basis was investigated and evolved throughout this research program using new knowledge, test cases to access usability, critical review by peer researchers, and review/feedback from practitioners. Ilities metrics were investigated through the basis. The conceptual design of an ilities semantic translation layer for ease of use was generated and tested through demonstration cases. Several application areas were explored. (Ross & Rhodes, 2019). Collaboration with other SERC universities enabled the research to explore synergies with related work.

EVOLUTION OF THE SEMANTIC BASIS

MIT's prior work on a semantic basis for ilities (Ross, et al., 2011) has served as the foundation for this research activity. The initial prescriptive semantic basis had ten categories, and was subsequently expanded to fourteen categories. Together, these categories that form the semantic basis are intended to collectively define a change in a system, thereby creating a consistent basis for specifying change-type ilities in formal statements. A system can be verified to display the quality described in the statement and therefore be traceable to a desired higher order system property.

The SERC research provided the opportunity for feedback and collaborating with other universities (Dou et al., 2015). Based on a critique of the semantic basis provided by UVA, MIT convened an internal working group and provided further clarifications and reformulations to address the feedback. The interim version at end of the second phase of this research project addressed a number of critiques on the 14-category semantic basis, as well as accumulated MIT-internal critiques. The resulting refined semantic basis is illustrated in Figure 5. The fourteen categories are: cause, context, phase, agent, impetus, impetus nature, impetus parameter,

impetus destination state size, impetus aspect, outcome effect, outcome parameter, outcome destination state size, outcome aspect, level of abstraction, and value qualities of the change. Unique choices for each of these categories, when applied to a particular system parameter will formulate the change-type ility statement. The semantic basis aims to capture the essential differences among change-type ilities through specification of the following general change statement with regard to a particular system parameter:

In response to "cause" in "context", desire "agent" to make some "impetus parameter change" in "system" resulting in "outcome parameter change" that is "valuable."

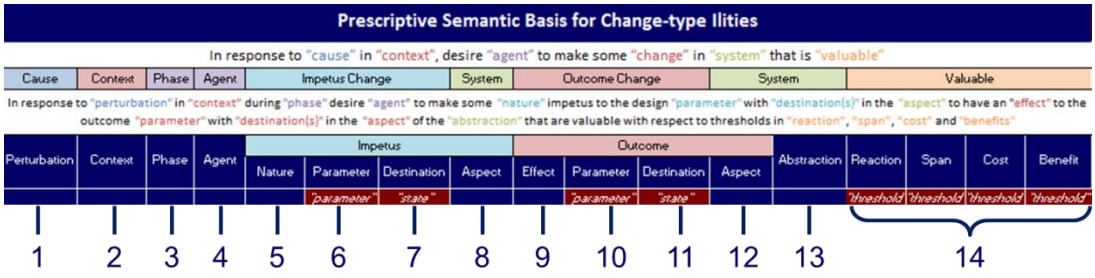


Figure 5. Change-type prescriptive semantic basis in 14 categories. (Ross, Beesemyer, Rhodes 2011)

Application of the semantic basis begins with a user generating a change statement. The change statement is refined and assigned categorical choices within the basis, with the intention that the applicable ilities will emerge from the specified change statement. In this way, a user does not need to use a particular ility label a priori, thereby avoiding the semantic ambiguity in the terms. If the basis accurately and completely describes the underlying categories for change-type changes, then a user should be able to describe any change-type ility through the basis.

With continued research, the semantic basis was expanded to the full version comprised of twenty categories (Fig. 1) believed to span the change-type ility semantic field. The Phase 2 report has additional detail and information on how the semantic basis evolved.

Prescriptive Semantic Basis for Change-type Ilities																		
In response to "perturbation" in "context", desire "agent" to make some "change" in "system" that is "valuable"																		
Perturbation	Context	Phase	Agent	Impetus Change					Mech	Outcome Change					System	Valuable* (this category is not complete)		
In response to "perturbation" in "context" during "phase" desire "agent" to make some "nature" impetus to the system "parameter" from "origin(s)" to "destination(s)" in the "aspect" using "mechanism" in order to have an "effect" to the outcome "parameter" from "origin(s)" to "destination(s)" in the "aspect" of the "abstraction" that are valuable with respect to thresholds in "reaction", "span", "cost" and "benefits"																		
				Impetus* (optional)					Mech	Outcome								
Perturbation	Context	Phase	Agent	Nature	Parameter	Origin	Destination	Aspect	Mechanism	Effect	Parameter	Origin	Destination	Aspect	Abstraction	Reaction	Span	Cost	Benefit
optional	circumstantial: required; general: optional	null	optional	null	required	optional	optional	null* (this is implied by "parameter")	Optional	null	required	optional	optional	null*	optional	required	required	required	required
"name"	*"name(s)"*		*"name(s)"*		*"parameter"*	*"state(s)"*	*"state(s)"*		*"name"*		*"parameter"*	*"state(s)"*	*"state(s)"*		*"name"*	*"threshold w/units"*	*"threshold w/units"*	*"threshold w/units"*	*"threshold w/units"*
none	circumstantial	pre-ops	none	decrease	level	one	one	form		decrease	level	one	one	form	architecture	sooner	shorter	less	more
disturbance	general	ops	internal	same	set	few	few	function		same	set	few	few	function	design	later	longer	same	same
shift	<empty>	inter-LC	external	increase	<empty>	many	many	operations		increase	<empty>	many	many	operations	system	always	same	more	less
<empty>		<empty>	either	not-same		<empty>	<empty>	<empty>		not-same		<empty>	<empty>	<empty>	<empty>	<empty>	<empty>	<empty>	<empty>
			<empty>	<empty>						<empty>									

Figure 6 Twenty-category semantic basis fields

While a subset of the basis can be used to generate simpler statements, using the twenty categories provides a richer statement (see Figure 7). Applied to the example of changeability of a stereo system, the resulting statement is shown below in blue text. Note: The grey italicized text is included here to show the mapping to categories from the basis.

In response to a loud noise (*perturbation*) late at night (*context*), during operations (*phase*) of system, desire owner (*agent*) to be able to *impetus* {increase (*nature*) the knob angle level (*parameter*) from one state (*origin*) to many states (*destination*) in the system form (*aspect*)} through turning the knob (*mechanism*) that results in the *outcome*{increasing (*effect*) the volume level (*parameter*) from one state (*origin*) to many states (*destination*) in the system function (*aspect*)}in the owner's stereo system (*abstraction*) that takes less than 1 second (*valuable*).

Figure 7 Constructed Ilities Statement

Refinement of the basis occurred with iterative application tests. The team applied the basis to a number of constructed cases. This supported refinement of the basis, as well as the development of useful illustrative examples. One emergent result of the basis is that it can serve to reveal "new" ilities yet to be labeled (e.g., "flexibility").

ILITIES METRICS AND THE SEMANTIC BASIS

One beneficial outcome of using the ilities semantic basis as a common representation of ilities is that it can indicate a set of distinct, but related, metrics for measuring three aspects of a given ility: whether it is present, the degree to which it is displayed, and the value of that ility. Since the ility term labels map to particular (potentially overlapping) basis choices, the metrics most cleanly map to the basis choices, and then can be post facto assigned to ility terms. For example, scalability (ability to change the level of a system aspect from one state to another) can be measured in terms of number of states accessible, either as a set, or on a per parameter basis. In fact, the basis can be partitioned into three types of defined factors that impact potential metrics: 1) antecedent descriptions (5 categories), 2) state counting (11 categories), and 3) path valuation (4 categories).

Figure 11 shows the three partitions across the basis.

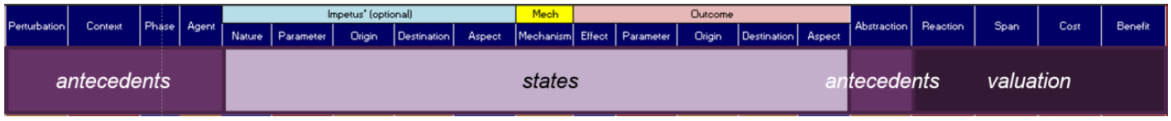
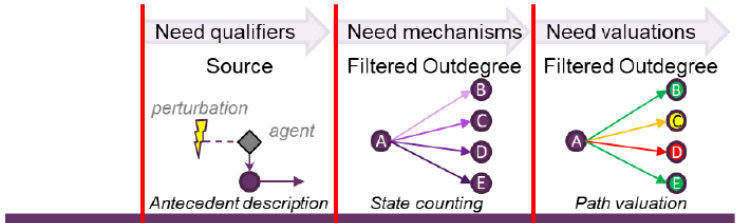


Figure 8 Semantic basis as supply information for antecedent description, state counting and path valuation

Using this perspective, appropriate metrics (existence, degree, or value) can be derived for a given ility that is described by a particular ilities statement using the basis. Figure 9 for example, illustrates several ility labels and whether their specification is sufficient for existence, degree, or value metrics. In this example, flexibility and adaptability relate to antecedent categories (i.e., agent) and therefore are a description of the existence. Their presence in an ility statement is sufficient for existence, but not sufficient to describe degree. For that, we also need to be able to count states (e.g. if we had change mechanisms that describe reachable states). If state counting is supplied, then scalability can be measured. In this example, agility, affordability, and reactivity are defined relative to acceptance thresholds and therefore additional information is needed to measure these (i.e., valuations: how much execution time, cost, and activation time is required for a given change to be acceptable).



Ility Label	Existence	Degree of	Value of
Flexibility	✓	✓	✓
Adaptability	✓	✓	✓
Scalability		✓	✓
Modifiability		✓	✓
Agility			✓
Affordability			✓
Reactivity			✓

Figure 9 Example of ility label relationship to existence, degree and value of state changes

Additional discussion can be found in Ross & Rhodes (2019) and prior phase research reports.

The semantic basis table format is effective for descriptive purposes; however it is not designed for ease of use. Various possibilities have been explored to find a way to "translate" the basis information into natural language and provide a guided experience. The research generated a concept for a software-based implementation of an evolved semantic basis that would be designed for usability by practitioners.

A conceptual design for a proof of concept translation layer was developed, referred to as the "ilities Semantic Translation Layer Assistant." An end goal (beyond the scope of the research investigation) would be to produce operational software that is customizable to the domain and specific language of user organization. When using the envisioned translation layer assistant, as a first step the user chooses an ilities dictionary to use. Presently, there are a number of different definition sets for ilities, and the prototype allows the user to make a choice consistent with their own organization. The various elements used in the Ilities Semantic Translation Layer Assistant prototype include ility specificity diagram, queries with response choices, and ilities advisor sidebar. Core supporting constructs are shown in Figure 10.

Following the design of the translation layer assistant, this conceptual approach was tested through several sessions with researchers and graduate students. This investigation confirmed potential usefulness of such a translation layer, and adjustments were made based on these sessions. The scope of the current research precluded more formal testing with a broader stakeholder community, which would be a necessary next step prior to developing a software prototype. (Ross & Rhodes, 2019)

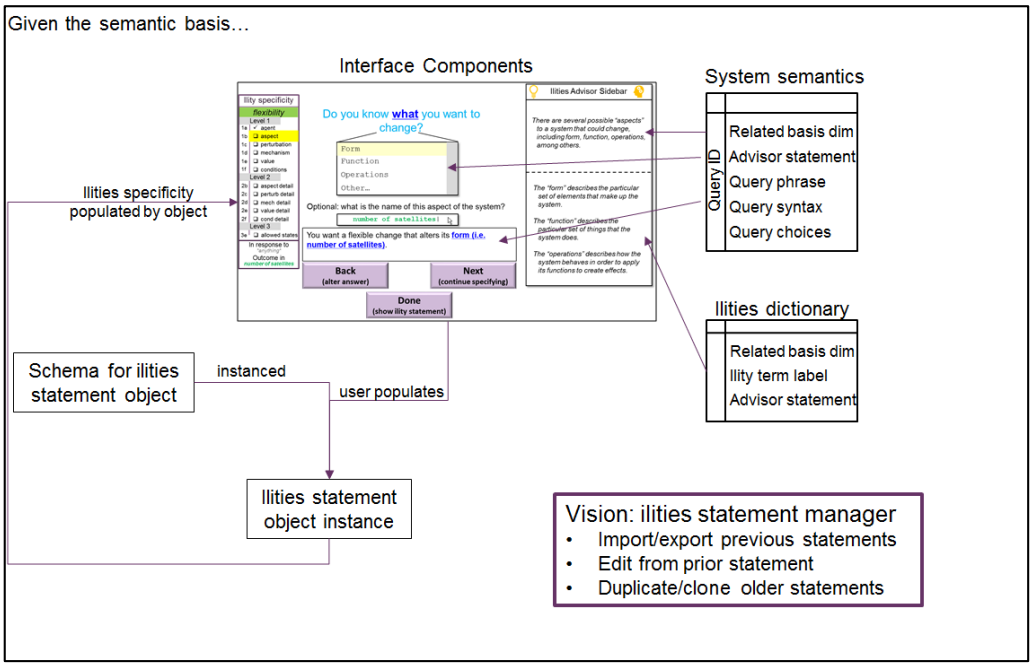


Figure 10 Core Supporting Constructs for Translation Layer Assistant

POTENTIAL FUTURE APPLICATIONS OF PRESCRIPTIVE SEMANTIC BASIS

Several potential application areas have been investigated during the research, including:

1. **Constructing requirements statements for system qualities (ilities)**. Well-written requirements need to be complete, consistent and concise, in context of the specific usage (e.g., RFP, systems spec, detailed design document, etc.). Given the intended use of the requirements statement, the minimum necessary semantic basis fields can be used to compose well-specified requirements. (Ross & Rhodes, 2015)

2. **Educating students and practitioners.** Given the lack of accepted common definition of ilities, or system qualities, engineers learn the definition that is used by the educating organization or instructor. Additionally, definitions are learned as written text, inhibiting enriched understanding of the ilities term. Our research suggests the semantic basis could assist in developing a more precise understanding and appreciation of what is required in an ilities statement. Use of various versions of the basis (e.g., 10-category basis, 14-category basis, 20-category basis) and classroom use of a translation layer assistant may provide a means to show how simple statements can be logically elaborated as more information is gained. It is hypothesized that use of the

semantic basis in educating students about ilities will enable more critical thinking than simply teaching simple definitions of ilities. (Ross & Rhodes, 2019)

3. **Automated technical document comprehension of system qualities (ilities).** The nature of modern defense systems necessitates being able to translate ilities information across an increasing number of domains and disciplines (e.g. computer science, biotechnology, politic science, cognitive science), driving the need for a 'Rosetta Stone." The prescriptive semantic basis appears to be a useful structure for supporting automated extraction of a comprehensible ilities statement from technical documents. Further, research in computer science and data science offer new possibilities for automatic extraction and methods to structure and analyze a comprehensible ilities statement. Such a proposed application of the semantic basis has several potential impacts (1) automated extraction of comprehensible ilities statements during tasks such as capability requirements assessment can reveal and enable comparison of ilities (e.g., flexibility, agility) found in documentation for existing and planned systems, thereby informing decisions; (2) ability to rapidly generate design concepts in support of repurposing system capabilities could be facilitated by generating options through automated composability of ilities semantic fields (akin to a recommender system for the systems designer); and (3) human decision-making could be augmented with curated libraries of ilities statements that can be queried with questions normally taking significant time to investigate. For instance: What are all the facets of how my current system supports agile operations? What architecture ilities (e.g., modularity) have been used in space communications systems to enable resilience? (Ross & Rhodes, 2019)

EXPLORATION OF SYNERGIES WITH RELATED RESEARCH

The research project has provided opportunities for exploring synergies of the research across several different universities. Boehm & Kukreha (2017) presents an initial ontology for system qualities developed at USC that was compared with the MIT ilities and semantic basis research. The purpose was to identify synergies and conflicts between these. Leveraging early work by USC to map their system qualities list to the semantic basis,

Figure 11 compares USC ility term mapping to the MIT semantic category choices. This comparison highlights a few points under consideration in the current work: some basis categories have different choices, there is potentially different interpretation of the meaning of "disturbance," the set of system quality terms are not the same, and potential interpreted differences arise from the perspectives

taken. For example, MIT semantic basis researchers take a technology system-centric perspective and USC researchers take a software-centric perspective, perhaps resulting in the different choices for aspect (e.g., “form”, “function”, and “operations” for MIT, and “form” and “scope” for USC).

Reconciling and harmonizing these perspectives has promise for further maturing the underlying basis and its applicability across a broad range of system types. In particular, this approach can be used to force clarification discussions on distinctions among the basis choices (either implied, or expressly considered), potentially resulting in new ilities of relevance. For example, the USC ility term “contractability” appears to be similar to “scalability-down.” The result of this comparison is that the choices for the basis dimensions may be dependent on domain of interest (e.g., hardware versus software), and similar ility term labels can clearly map to different choices in the basis, highlighting opportunities for further technical clarifications.

Perturbation	Agent	Impetus		Outcome		USC Ility Label
		Nature	Aspect	Effect	Aspect	
disturbance	internal	increase	form	same	form	
defect	external	decrease	scope	not-same	scope	
opportunity	<empty>	re-host	<empty>	reduced	<empty>	
<empty>		<empty>		increased		
				<empty>		
				not-same		Changeability
disturbance	internal			same		Robustness
disturb, opp'y	internal			not-same		Adaptability
disturb, opp'y	external			not-same		Modifiability
			form	not-same	form	Reconfigurability
disturb, opp'y		increase	scope	not-same	scope	Extensibility
disturb, opp'y		decrease	scope	not-same	scope	Contractability
disturb, opp'y			scope	same	scope	Versatility
						Survivability
						Evolvability
						Flexibility
						Agility
						Reactivity
disturbance		increase		same, increased		Resilience
defect	internal			same		Fault-Tolerance
defect	internal			not-same		Self-repairability
defect	external					Repairability
	external					Maintainability
	internal			reduced		Graceful Degredation
disturb, opp'y		re-host	form	same	form	Portability
						Exchangeability

USC-MIT Differences
Match
Shift vs. Disturbance
Match
Just External (internal = Adaptability)
Similar
Same - Also Scalability-Up
Not included - Also Scalability-Down
Similar
Robustness + Adaptability?
Same as Changeability?
External Adaptability?
Focus on fix speed
Focus on duration of fix needs
Not specified
Not specified
Not specified
Not specified
Not specified
Not specified
Not specified
Like Modifiability?

MIT Ility Label	Perturbation	Agent	Impetus		Outcome	
			Nature	Aspect	Effect	Aspect
	none	none	decrease	form	decrease	form
	disturbance	internal	same	function	same	function
	shift	external	increase	operations	increase	operations
	<empty>	either	not-same	<empty>	not-same	<empty>
		<empty>	<empty>		<empty>	
Changeability		either	not-same		not-same	
Robustness	shift				same	
Adaptability		internal	not-same		not-same	
Modifiability			not-same		not-same	
Form Reconfigurability			same	form	not-same	form
Extensibility		either	not-same		increase	
Scalability			not-same		not-same	
Functional Versatility			same	form/ops	not-same	function
Survivability	disturbance				same	
Evolvability	shift		not-same		not-same	
Flexibility		external	not-same		not-same	
Agility			not-same		not-same	
Reactivity			not-same		not-same	
Operational Reconfigurability			same	operations	not-same	operations
Operational Versatility			same	form/fnct	not-same	operations
Substitutability			same	fnct/ops	not-same	form
Value Robustness	shift				same	
Value Survivability	disturbance				same	
Active Robustness	shift		not-same		same	
Passive Robustness	shift		same		same	
Classical Passive Robustness	shift	none	same		same	form

Figure 11 Comparison of USC System Qualities and MIT Semantic Categories

Discussion and Future Research Directions

Several future areas of research for the ilities semantic basis include extension of the semantic basis approach for other ilities types, cross-comparison of the semantic basis approach with other emerging approaches, and application of the semantic basis in additional use cases. Validation of the usefulness of the semantic basis is needed, including case studies across different types of systems, pilot project use on one or more existing systems programs, and experimenting with applying the basis in classroom education. The terminology and approach should be compared against

current standards and formal guidance documents.

The ilities semantic basis research aims to stimulate broad discussions and research around developing a basis (or bases) as prescriptive instrument(s) for spanning semantic fields whose union consistently encompasses sets of "ilities." The envisioned end result is that practitioners can have a consistent and (potentially) more complete list of possible ilities to consider for their systems. From a practice standpoint, the semantic basis research is focused on finding a means to assist in generating verifiable requirements and system specifications. Significant further investigation is needed to mature this work, including further comparison, alignment and harmonization with other ilities work in the systems community. In parallel, research is needed to refine and implement a "translation layer" for more natural interaction with the basis, as discussed in section 3.

Continuing research is needed to further understand the various emerging ilities classifications and frameworks. The DoD SERC has provided the opportunity for many researchers to share and collaborate on ilities research. Dou et al. (2015) brought SERC researchers together to investigate embedded theory-systems (ETS) approach, which included applying ETS to the semantic approach. According to these authors, the goal of the work is "to enable system developers to think and to communicate more effectively than they can today about system properties, underlying design decisions, and stakeholder value." The formalism of the ETS approach and its application to the ilities semantic basis as a case study provided many new insights and additional research questions. It validates the value of collaborative efforts to build understanding and harmonize approaches.

There are additional promising opportunities to compare and bring together research on ilities, beyond the work of the SERC researchers. For example, recent work by Colombo et al. (2016) proposes a unified vocabulary based on an extensive literature survey and provides analysis of nine change-related ilities frameworks. The relationship of this work to the ilities semantic basis approach has yet to be explored. Gasper et al. (2015) has adapted the semantic basis construct in their investigation of the lifecycle property of stability of ships and ocean vessels.

Future research is needed on the area of ilities quantification and metrics. Some limited investigation has been performed (e.g., Fitzgerald & Ross, 2012; Sinha & de Weck, 2016), but much of recent focus on ilities has been on characterization rather than measuring. And, the ilities semantic basis research has included initial exploration of various metrics for changeability-type ilities. Turner et al. (2017), in their recent systematic review of the literature, found that "quantitative measures of

the ilities are uncommon." These authors identify and summarize various approaches to quantifying ilities, in particular for robustness, interoperability and agility. They found that robustness has a variety of quantifications, but recommend further efforts for interoperability and agility (Turner et al., 2017). To date, the systems community has had limited collaborative engagement on ilities quantification.

Our research indicates that at least three semantic fields may exist for the general set of system "ilities" including change-type ("flexibility", "agility"), architecture-type ("modularity", "interoperability") and new ability-type (e.g., "auditability"). Identifying and classifying the in- use ility terms into appropriate semantic fields serves to eliminate ambiguity in meaning, usage, and application, as well as allow for the explicit consideration of trade-offs within the semantic field. A consistent basis within a field can allow for direct comparison of its members. Revisiting the concept of relationships amongst the ilities, the basis can provide a first order approximation to clarify semantic differences amongst ilities within a particular semantic field. For example, we differentiate "flexibility" and "adaptability" in terms of whether the change agent is external or internal to the system's boundary, respectively. The basis can also show how a given change statement can display multiple ilities simultaneously. For example, agility relates to how quickly a change can be executed, so one could desire an agile, scalable change to describe a quick and level-increasing system parameter change. A working hypothesis is that that "architecture-type" ilities are enablers for "change-type" ilities (Lockett, Swan & Unai, 2017). This is a significant topic for future research.

Research on the characterization of ilities (system qualities) has been ongoing for more than a decade, but has yet to reach a saturation point. Many literature survey investigations have been conducted that inform current investigation. As a community, research progress has been made toward increase structure and formalization of ilities terms, and the semantic basis approach is one contribution to this goal. There are many promising research directions related to ilities, which grow increasingly important given the technical and non-technical challenges faced in the engineering of systems for the future.

CITED REFERENCES

Boehm, B., & Kukreja, N. (2017). An Initial Ontology for System Qualities. INSIGHT, 20(3), 18-28.

Daclin, N., Moradi, B., & Chapurlat, V. (2018).Analyzing Interoperability in a Non-functional Requirements Ecosystem to Support Crisis Management Response. Enterprise Interoperability: Smart Services and Business Impact of Enterprise Interoperability, 429-434.

de Weck, O.L., Ross, A.M., and Rhodes, D.H. (2012). "Investigating Relationships and Semantic Sets amongst System Lifecycle Properties (Ilities)". *3rd International Engineering Systems Symposium*. CESUN 2012. TU Delft. 18-20 June 2012

Dou, K., Wang, X., Tang, C., Ross, A.M., and Sullivan, K., "An Evolutionary Theory-Systems Approach to a Science of the Ilities," 13th Conference on Systems Engineering Research, Hoboken, NJ, Mar. 2015

Gaspar, H. M., Brett, P. O., Ebrahimi, A., & Keane, A. (2015). Lifecycle Properties of Stability–beyond Pure Technical Thinking. Proceedings of the 12th International Conference on the Stability of Ships and Ocean Vehicles, Glasgow, UK, 575.

Lee, J.Y. & Collins, G.J. (2017). On Using ilities of Non-functional Properties for Subsystems and Components". Systems, 5(3), 47

Lockett, J., Swan, M., Unai, K., (2017). The Agile Systems Framework: Enterprise Content Management Case. 15th Annual Conf on Sys Engineering Research. Redondo Beach, CA.

Lowe, Donald. Exploring system ilities and their relationship. In: Systems Evaluation Test and Evaluation Conference 2018: Unlocking the Future Through Systems Engineering: SETE 2018. Melbourne: Engineers Australia, 2018: 352-365.

Ross, A.M., Beesemyer, J.C., and Rhodes, D.H., (2011) "A Prescriptive Semantic Basis for System Lifecycle Properties", SEAri Working Paper WP-2011-2-1. MIT. Cambridge, MA.

Ross, A.M., Rhodes, D.H., and Hastings, D.E. (2008), "Defining Changeability: Reconciling Flexibility. Adaptability, Scalability, Modifiability, and Robustness for Maintaining Lifecycle Value," *Systems Engineering*. Vol. 11. No. 3. pp. 246-262

Ross, A.M., and Rhodes, D.H. (2015). "Towards a Prescriptive Semantic Basis for Change-type Ilities." 13th Conference on Systems Engineering Research. Hoboken, NJ. March 2015

Ross, A.M., and Rhodes, D.H. (2019). "Ilities Semantic Basis: Research Progress and Future Directions". 17th Conference on Systems Engineering Research. Washington, DC.

Turner A.J., Monahan W., Cotter M. (2018) Quantifying the Ilities: A Literature Review of Robustness, Interoperability, and Agility. In: Madni A., Boehm B., Ghanem R., Erwin D., Wheaton M. (eds) Disciplinary Convergence in Systems Engineering Research. Springer.

PUBLICATIONS

Ross, A.M., and Rhodes, D.H. (2015). "Towards a Prescriptive Semantic Basis for Change-type Ilities." 13th Conference on Systems Engineering Research. Hoboken, NJ. March 2015

Ross, A.M., and Rhodes, D.H. (2019). "Ilities Semantic Basis: Research Progress and Future Directions". 17th Conf on Systems Engineering Research. Washington, DC. April 2019

TRANSITION

Ilities Semantic Basis. The semantic basis research generated foundational knowledge regarding system qualities/ilities. Research focused on developing a generalizable basis with which one could capture the essence of a large number of ilities (related within that semantic field, such as "change-related"). The intended early users for the semantic basis have been academic researchers, for the purpose of uncovering whether such a basis might exist and how it might be expressed (e.g., Dou, et al., 2015; Gasper et al., 2015). MITRE Corporation incorporated elements of the research into a research projects on agile systems framework (Lockett et al., 2017). The research has been presented in several systems engineering expert forums in government and industry, and has been incorporated into graduate lecture material on system properties. Two publications resulted from this research, which were presented at the Conference of Systems Engineering Research in 2015 and 2019. Additionally, a joint paper with UVA was published.

9 EXECUTIVE SUMMARY

NPS developed and demonstrated affordability tradeoff analysis methods employing parametric cost models with multiple case studies. A focus was on the interface of cost and architecture modeling. This includes the automatic extraction of system size attributes from architecture models at the levels of software, system and product line. Costs can then be associated with architecture variants to assess tradeoffs between affordability and other ilities.

Value for DoD was created in parametric cost models and tools made available; allied new methods for tradespace analysis; associated DoD case studies with lessons learned; and DoD education and personnel competency by supporting NPS and AFIT graduate student research. NPS Master's and Ph.D. theses for seven students in Systems Engineering were completed leveraging and/or furthering this research.

Cost models were extended for integration with Model-Based Systems Engineering (MBSE) methods including SysML, Orthogonal Variability Modeling (OVM), discrete event simulation, and system behavioral modeling. Some of the research was conducted jointly with AFIT on relevant DoD case studies.

Existing models were applied in some cases to demonstrate tradeoff analyses. New models and methods were also created to better represent actual conditions empirically observed in practice. This report will highlight the new unique aspects to the work, showing capabilities and model extensions developed and introduced during this research. Table 1 summarizes the primary affordability applications that were conducted.

Table 2 Primary Affordability Research Applications

DoD Programs and Applications

Models and Methods	NAVAIR FACE Initiative	UAS Anti-Terrorist Missions (notional)	UAS Remote Targeting System (notional)	Aegis Ship Software Product Line	ASW Cross-domain Product Line (notional)	Cruise Missile Defense Product Line (notional)	General Application
COSYSMO		architecture costing	architecture costing				COSYSMO 3.0 updating support
COCOMO							COCOMO III updating support
Basic COPLIMO	avionics software product line architecture and cost savings			product line ROI and cost savings (retrospective and prospective)	ASW product line architecture software development ROI and cost savings		
System COPLIMO					ASW product line architecture system ROI and cost savings	product line architecture system ROI and cost savings	
SysML		architecture modeling	architecture modeling				SysML integration with system cost model
Orthogonal Variability Modeling					product line architecture modeling	product line architecture modeling	OVM integration with product line cost model
Discrete Event Simulation							MP integration with software cost model

10 BACKGROUND

The primary models and methods used throughout this research are introduced next. Cost models provide an easy-to-use framework for performing broader ility and affordability analyses when tied to architecture models. The cost models have several variants for different contexts, and their respective applications are described.

11 Parametric Cost Modeling

Parametric cost models use cost estimating relationships as mathematical algorithms relating cost factors. The parametric models use numeric inputs for explanatory variables reflecting system characteristics to compute cost. A parametric cost model is defined for a specific aspect of a project, product or process. The models are developed using regression analysis and other methods with empirical data from completed projects.

This research leverages parametric cost models in the public domain, with the formulas open and available for use. They are more valuable this way when all DoD stakeholders can ascertain why the models produce the results they do and be better informed for making decisions. The parametric models used are called "constructive" for these reasons.

The Constructive Systems Engineering Cost Model (COSYSMO) [12] estimates the labor cost of performing systems engineering. It is used throughout this research to support affordability tradeoffs. An allied model is the Constructive Cost Model (COCOMO) for estimating software development costs supporting tradeoff analyses [8].

The Constructive Product Line Investment Model (COPLIMO) is used to assess the costs, savings, and return on investment (ROI) associated with developing and reusing software product line assets across families of similar applications [4]. Detailed COPLIMO for software [20] and System COPLIMO as a system level extension were both created during this research.

Chronological Phase Summary

NPS supported the SERC Phase 1 analysis of tradespace capabilities, gaps and limitations with DoD stakeholders [1]. Demonstrations of initial toolsets were shown to elicit feedback. The NPS Phase 2 activities improved and piloted several existing tradespace analysis toolsets based on the results of Phase 1. The focus for tool extensions and applications was in the Ships and Aircraft domains.

The tools were tailored for software product line cost modeling, and total ownership cost for integrated engineering activities. The early adopters represented NAVAIR and NAVSEA. An array of improvements for our models and tools were identified for going forward in Phase 3 for ility tradeoffs.

We supported outreach meetings to summarize and demonstrate tradespace analysis capabilities to potential early-adopter organizations. These included visits to the Army Engineer Research and Development Center (ERDC), and NAVSEA CREATE-Ships personnel in Carderock associated with DoD Engineered Resilient Systems (ERS).

We also engaged in new community-building activities with NAVAIR stakeholders. NPS and USC began collaboration with the NAVAIR avionics software product line FACE program. We supported their surveys with recommendations, data collection, interpreting software lifecycle cost models and calibrations of the COPLIMO product line cost model. This application is a highly relevant example of modeling product line benefits for the DoD.

A previous shortfall of our NPS Total Ownership Cost (TOC) toolset [12] was lacking the capability to estimate operations and maintenance. We added parametric maintenance models into our system cost model suite for systems engineering, software engineering, hardware development and production.

Cost uncertainty modeling was also extended via improvements in Monte Carlo analysis. Additional size inputs were made available for probabilistic distributions, as well as a wider array of distribution types. This feature works in tandem with the added lifecycle extensions for maintenance.

We began a ship case study for design and cost tradeoffs with military students at NAVSEA. The group designed a new carrier and integrated cost models into a Model-Based Systems Engineering (MBSE) dashboard for Total Ship Systems Engineering (TSSE). This allows comparison and refinement of potential ship cost models for affordability tradeoffs in the MBSE framework.

Initial comparisons of MIL-STD 881 Work Breakdown Structures (WBS) were performed to find commonalities and variabilities across DoD domains, and identify suggested improvements. This analysis informed us how to best structure canonical TOC tools to address multiple DoD domains efficiently. Additionally, a detailed review and critique of the recent MIL-STD 881 UAV WBS was done and deficiencies noted for autonomy trends which are of increasing importance.

During Phase 3, NPS reached out to DoD organizations and sought methods/processes/tools piloting opportunities. We added incremental capability enhancements to our suite of system cost models and tools. Phase 3 included the successful adoption of the COPLIMO at NAVAIR resulting in follow-on research funding to extend the software cost model.

In Phase 4 we built on prior work to explore the use of executable architectures with MBSE tools to guide early design decisions given requirements changes and architectural variations in collaboration with AFIT. Operational and system architectures of military scenarios were developed and represented in an MBSE environment. The models were used to support evaluation of system performance, higher-level qualities, and software and system cost. Translation rules and constructs between MBSE methods, performance analysis and cost model inputs were developed and demonstrations of tool interoperability and tailorability to DoD domains were conducted.

The Phase 4 joint AFIT-NPS research defined scenarios to include heterogeneous teams of UASs performing ISR missions in increasingly complex environments. Measures of Effectiveness (MOEs) were established, uncertainty associated with the adversary and the environment characterized and modeled, and system parameters chosen to demonstrate the ability to perform meaningful tradespace analysis that considers affordability.

In Phase 5, NPS and AFIT continued developing and demonstrating methods for integrating MBSE approaches for early architectural definition, effectiveness analysis, and cost estimation using shared case studies and models for ISR missions of increasing complexity with multi-tiered collections of heterogeneous UAS. Architectures were defined using SysML compliant modeling packages for direct simulation and system evaluation, and the population of early cost estimation tools to provide relative cost estimates associated with variations of the architectures.

NPS demonstrated the viability of using the SysML models for direct inputs to cost models [15][16][17]. A case study with AFIT then demonstrated the method applied to a multi-vehicle architecture using small UAS to locate, confirm, track and engage widely dispersed targets. Architectural variations included numbers of vehicles, sensor quality, and C2 variations for operator-in-the-loop vs. full autonomous operation. The effectiveness analysis revealed significant differences in overall performance, and that cost variations could be analyzed together.

At the end of Phase 5, we developed a more complex architecture based on small UAS providing remote targeting support for larger, standoff vehicles [16][17]. Architectural views included requirements diagrams, functional decomposition (hierarchical), activity diagrams, block definition diagrams, and interface definition both across system elements and at the subsystem level within a system element. This level of definition included operational threads, requirements, and interfaces at the appropriate decomposition level for direct input to parametric cost models. The thorough and detailed SysML model comprising size inputs enabled a system engineering cost estimate and extrapolated full lifecycle cost, amenable to later architectural variations. Completed student theses in support of the Phase 5 research include [10][13][14][15].

In Phase 6 research, NPS continued refining and demonstrating integrated modeling methods with AFIT for common ISR pilot applications with UAS and satellite architectures. We furthered MBSE approaches for early architectural definition, effectiveness analysis, and cost estimation with new ISR applications and modeling tools. New applications were for a Remote Targeting System (RTS) and a space-based regional ISR application using satellites.

SysML-based cost analyses for the satellite variants were conducted from reference architectures defined by AFIT. Preliminary cost estimates of CubeSat vs. traditional satellites were based on COSYSMO size drivers, yet other sources of cost variance for hardware were not captured. This will be addressed to better reflect expected cost differences (see Future Work).

NPS supported refinement of definitions for COSYSMO 3.0 data collection, including Rosetta Stone parameter mappings from previous COSYSMO versions to COSYSMO 3.0. A COCOMO III extension for estimation of secure software development was initially defined.

NPS also worked with the Naval Center for Cost Analysis (NCCA) in developing and calibrating domain-specific early phase software cost models. The corresponding paper "Early Phase Cost Models for Agile Software Processes in the US DoD" won best paper award at the IEEE Empirical Software Engineering and Measurement (ESEM) [23].

We also supported research for the new COCOMO III and COSYSMO 3.0 cost model developments [8][12]. The system cost modeling activities with USC engaged domain experts for Delphi estimates, evolved baseline detailed definitions of the cost driver parameters and rating scales for use in data collection, and gathered initial data.

In Phase 7, our affordability case study on the Aegis ship software product line economics resulted in an enhanced cost model [20]. To better reflect differences across products, the Basic COPLIMO model for software was refactored for detailed per-product estimates. The homogeneous inputs for mission-unique, adapted and reused system portions across products can now be assessed for individual builds for better planning and precise estimates.

We performed a detailed return-on-investment analysis of the Aegis ship software product line. Empirical measurements on size, reuse factors, and effort were collected from multiple sources (program office, contractor, SRDRs) and analyzed in order to populate and tailor COPLIMO for both retrospective and prospective estimation of cost savings. In validation testing the results compared favorably with actual reported savings [20].

Cross-domain product line architecture and cost modeling was performed for ASW system types. Using OVM models in conjunction with COPLIMO, both system and software affordability was analyzed.

Selected Research Highlights

Some highlights of the new methods are presented next. These illustrate first-time research results to go forward for further elaboration in practice, and lessons learned from case studies.

MBSE Model Translations
This research necessitated translations between models/tools in MBSE, specifically mapping architectural elements into behavior/performance analysis and cost model inputs. System size is the largest source of cost variance in all parametric cost models. It is the foremost cost factor to capture and was assessed for correspondence in SysML, OVM modeling, and simulation using MP.

Assessments of sizing correlation strengths with the respective MBSE methods was undertaken [16] [17]. These correlations answer the question "To what degree are these factors captured in the method (as-is)?" They also show how available the parameters are for automated extraction. To a large degree, we found the existing MBSE methods and notations can be leveraged as-is to populate cost models but are not yet automated. These sizing aspects for system, software and product lines are overviewed next.

SysML and System Cost Modeling

A mapping between SysML entities and COSYSMO size entities was required to translate between models. Systems size inputs for COSYSMO include include Requirements, Interfaces, Algorithms and Scenarios. The correlations with SysML modeling entities was developed and used throughout the case studies.

The count of Requirements can be ascertained from a Package Diagram for each type of requirement of concern (e.g. mission, reliability, safety, etc.) and a summation of the Requirements Diagrams for each package. The Interfaces can be counted from either a high-level Block Definition Diagram or an Internal Block Diagram as the number of ports. A Parametric Diagram contains equations as constraints representing Algorithms to count. The number of Operational Scenarios can be counted as Use Cases on Use Case Diagrams (which must be represented at the corresponding level of implementation detail).

These are the primary standard mappings available, but various modeling styles may afford additional mappings. For example, countable requirements could be represented in other

diagram types or list notations. Further there could be non-standard augmentations or extensions for MBSE tools incorporating SysML not reflected here.

The integrated approach with SysML and COSYSMO is summarized as:

- Develop operational and system architectures to capture sets of scenarios
- Transition the architectures to MBSE environments.

SysML diagrams and executable activity models

- Extract cost model elements from components of the architectures in order to evaluate cost effectiveness
- Design and demonstrate tradespace including cost in integrated MBSE environment with executable models of architectures

Simulation for Architecture Behavioral Analysis and Software Cost Modeling

MP was assessed for the availability of early lifecycle software size in function points. Behavioral modeling with MP offers a way to assess architectural design decisions and their impacts, but not a way to estimate the cost impacts of the decisions.

This research integrated function point analysis, software cost modeling, and executable behavioral modeling of system and software architecture specifications with MP. It demonstrated a new methodology to extract Unadjusted Function Point (UFP) counts from MP models for use in COCOMO [14].

The function point measures of size to extract from model entities are: External Inputs (EI), External Outputs (EO), External Inquiries (EQ), Internal Logical Files (ILF), and External Interface Files (EIF). The mapping is described below:

<u>Interactions for data functions (ILF, EIF)</u>

- Count number of SHARE ALLs, for each data function
- Count number of COORDINATEs and number of ADDs if detailed source information is available, for each data function

<u>Interactions between processes (EI, EO, EQ)</u>

- Count number of COORDINATEs for each transactional function
- Count number of COORDINATEs and number of ADDs if detailed source information is available, for each transactional function

Complexity weights can be determined by the number of ADDs counted in COORDINATE composition operation in an MP model.

The integrated tradespace methodology with MP and COCOMO can be summarized as:

- Identify the tradespace problem statement.
- Describe the behaviors of the system and environment in natural language

- Unambiguously represent behaviors using MP, extract Use Cases/initial views from MP model
- Perform Architecture verification and validation, and assertion checking
- Un-ambiguously relate system and environment behaviors to Function Point behaviors
- Extract coefficients that inform complexity and scale
- Determine Unadjusted Function Point (UFP) count for COCOMO II/III
- Assess assumptions of complexity in cost estimates
- Visualize tradespace results in views specific to stakeholders

DoD Product Line Architecture and Cost Modeling

OVM modeling of architectures was the basis for product line cost modeling input for portions of mission-unique, adapted, and reused size. This modeling approach was applied to combat system case studies. In DoD systems there are a variety of configurations that include sensors, weapons, and hardware/software integrations to accomplish similar goals. These common hardware and software elements with their interfaces can be modeled as flexible product lines, which are then enumerated for product line cost and investment analysis. This integrated method was the research basis for [19] [20] [21].

The system architecture starts with the Hatley-Pirbhai notation and associated architecture template. An enhanced data flow diagram and architectural flow diagram (AFD) describe the functional and physical behavior of the combat system. Each system architecture diagram utilizes the detect, control, engage paradigm as the central premise of the combat system architecture, both functional and physical.

The AFD provides the structure for variation point identification necessary for OVM modeling in the product line construct. Variations points are identified for sensors, HSI / consoles, weapons, and data links. The variation points and associated variants are presented as OVMs and consolidated into a product line OVM with packaged variants and constraint dependencies. The constraint dependencies demonstrate feasible combinations of packaged variants, variation points, and variants for the combat system product line. OVMs are then used to quantify variation points for COPLIMO product line percentages for mission-unique, reused and adapted portions [19] [22].

The overall technical approach integrating the OVM method and COPLIMO applied to combat systems is summarized below [22]:

- Describe a general domain model of the given system with common elements
    - Generic kill chain architectures including sensors, weapons, and hardware/software are formally modeled to identify common functions and variations.
- Develop a reference product architecture with variation points

    - Variation points are identified for sensors, HSI / consoles, weapons, and data links with choices for a combat system product line. These also serve as cost model inputs.
- Map existing systems to the reference architecture
- Collect empirical costs and map them to system elements from above
    - Empirical cost data from DoD systems programs is allocated to the system functions in the architecture models to calibrate and populate cost model for specific system configurations.
    - Alternatively use detailed parametric cost models instead of empirical averages when data is available
- Tailor the System COPLIMO framework for the reference architecture or develop new cost models for each application, as necessary.
- Use the cost model to assess product line economic tradeoff decisions for the given system.
    - The value of investing in product-line flexibility is quantified using ROI and TOC vs. traditional one-off designs for specific systems and their constituent elements.

Case Study: UAS Anti-Terrorist Mission Architecture Cost Comparisons
Mission architecture variations were defined by AFIT for six anti-terrorist missions employing UAS's. The detailed use cases were the basis for translation into system size for the COSYSMO model. The most immediate size measure available from modeled mission descriptions is the number of operational scenarios that a system must satisfy. Such scenarios include both the nominal stimulus-response thread plus off-nominal threads resulting from bad or missing data, unavailable processes, network connections, or other exception-handling cases.

Enumeration of operational scenarios from the architecture model required a parsing of the detailed descriptions of the scenarios for a nominal costing. This method was used for affordability tradespace analysis across the mission architectural variants by assigning costs to the variants.

The preliminary results were nominal for two reasons. The off-nominal threads aren't yet included for exception-handling cases. Further they aren't yet assessed for relative complexity levels and are tagged as nominal complexity for a starting default. The scenarios can then be assessed for complexity levels and weights adjusted accordingly (see Future Work).

Case Study: Remote Targeting System (RTS) Architecture Cost Comparison
This military case study with AFIT incorporates several new methods developed in this research. It improves on the partial sizing approach on the anti-terrorist mission analysis. It compares architecture variants of an RTS performed by semi-autonomous aerial

vehicles. Though intended for unmanned aerial vehicles, the RTS system model can be easily adapted to other domains. The case study demonstrates a general method applicable to any DoD system that is modeled similarly with SysML.

The RTS baseline variant can have multiple vehicles, but uses human-in-the-loop to declare targets. It requires a data link back to operator for each vehicle. The Autonomous Target Recognition (ATR) variant has heterogeneous sensors, and can use multiple vehicles to auto confirm target declarations without requiring a human.

Several representative architectural variations of the RTS with associated cost comparisons for tradespace analysis were developed. These were represented in SysML requirements diagrams, functional decomposition (hierarchical), activity diagrams, block definition diagrams, and interface definitions. A full elaboration of the requirements, interfaces and operational scenarios were modeled using the CORE tool and used as cost model inputs.

The SysML to COSYSMO mappings developed in this research [15] [17] were key to the integrated tradespace analysis. Use cases were counted as operational scenarios (threads) for COSYSMO input. Each represents a thread of interaction with the system that requires elaboration. Requirements in the system model were enumerated for the Ground Station and Air Vehicle. All of these are counted as COSYSMO size inputs. Interfaces in the system model were the count of bidirectional ports (shown with double arrows) at all system levels. All of these are counted as COSYSMO size inputs.

The cumulative counts of requirements, interfaces and use cases (threads) in the SysML models were used as inputs to COSYSMO. Requirements are in an enumerated list, interfaces are the connector ports on block diagrams, and use cases are ovals in the use case diagram. For simplicity the size entities were all rated as Nominal complexity, and similarly were the other cost factor ratings. Future models will handle different levels of complexity.

Both the baseline and ATR cases were subjected to Monte-Carlo simulation producing probability distributions for comparison of the options. The ATR architecture was expected to cost more, but there was fair amount of overlap in the distributions for systems engineering costs.

Finally, a full lifecycle cost was extrapolated from the COSYSMO estimate for TOC. The extrapolated RTS total program cost distributions using SysML-derived inputs enabled a final comparison of the total costs for the architectures. This full example is contained in [18].

Case Study: AEGIS Ship Combat System Product Line Affordability Analysis

This case study leveraged COPLIMO to estimate product line savings and ROI for the AEGIS combat system software. The AEGIS common source library is a proven standard for an evolving product line architecture to meet Navy requirements and has proven cost savings since its inception.

The case study required an extension to COPLIMO to compensate for limitations in the default Basic COPLIMO. The extension allows for varying sized products vs. an assumed homogeneous size, and it models different relative portions for each individual baseline for unique, adapted and reused code.
Five consecutive Aegis baselines were modeled retrospectively and one future baseline to estimate product line effort, savings, per product cost savings, per product cost avoidance and cumulative ROI. For the Aegis baseline of 1.8 million software lines of code, the model indicates a potential ROI of 3.88 after the seventh product is delivered.

The Detailed COPLIMO results compared favorably to the cost avoidance metrics Lockheed Martin has provided from the years 2011 through 2014. Detailed COPLIMO provided similar cost avoidance varying from 21-31% after the delivery of the first product, and estimated ROI of 3.54 for the fifth delivered Aegis baseline in the product line and 5.40 for a future Aegis baseline.

Case Study: ASW Combat System Product Line Architecture Affordability Analysis
This study addressed cross-domain applicability of an Anti-Submarine Warfare (ASW) combat system product line for air, surface, and subsurface applications (LAMPS MK III (SH-60 Helicopters), AN/BYG-1 (Virginia Class), SQQ-89 (FFG 7, DDG 51, and CG 47 class). By defining the ASW domain in product models, the commonality was assessed of cross-domain systems to determine if an overarching product line approach can help reduce cost, increase mission effectiveness, and enable rapidly deployment.

Reference architectures were developed to identify variation points for enumerating variability across the operational domains. The domain model subsystems include Signal Processing, Fire Control, and Weapon Control, SONAR and the weapons. The reference architecture is derived from the domain model and consists of a block diagram of components utilized in a "kill-chain" and a functional block diagram for the functions detect, plan, launch, and communicate. Each individual system fulfills the functions but use varying components to perform the kill-chain.

The architecture models provide structure for defining variation points for the orthogonal variability modeling. Five variation points were identified as Sensors/Arrays, Weapons, Tactical Control, Data Link, and HSI. The OVM variabilities for LAMPS MK III, AN/BYG-1 and SQQ-89 were mapped to portions of unique, adapted and reused for product line costing.

The COPLIMO framework was used with the combat system variations for both system and software. Actual DoD system costs from were mapped to the architecture subsystems. Both models show an initial increase in development cost with a decline in subsequent product line cost, which yields a high return-on-investment.

High ROI were yielded for both system and software COPLIMO using a triangular distribution for pessimistic, most likely and best case scenarios for the relative cost factors in COPLIMO. Overall results indicate high ROI for the Navy to invest in a generic ASW combat system product line.

Conclusions and Recommendations

Cost models provide an easy-to-use framework for performing broader ility and affordability analyses when tied to architecture models. This research developed approaches for integrated systems and software cost models with product-oriented models for tradespace analysis.

Cost models were integrated in different ways with MBSE architectural modeling approaches. By developing mapping rules between the tools of MBSE, we demonstrated integrated approaches for architectural analysis, behavior/performance analysis, and cost estimation. This furthered the tradespace interoperability of cost models and tools applied to real DoD mission types.

System size inputs for costing can be derived from countable SysML entities. We found a strong correspondence between SysML constructs and size measures of requirements, interfaces, algorithms, and operational scenarios. It was demonstrated these entities can be automatically extracted and used as COSYSMO inputs to estimate systems engineering effort and extrapolate full lifecycle costs. Software size also needs to be captured for TOC. The MP simulation language encapsulates function point measures for software size. Product line architecture variabilities can be quantified with OVM modeling for software and hardware subsystems and used for costing.

Results from the model-centric architecture analyses with AFIT UAS models were promising. It demonstrated architectural tradespace analysis with simple UAS swarm models. The techniques can highlight significant differences between architectural variations being considered early in the acquisition before major financial commitments must be made. It is recommended to further the tool integration to be more robust for increasingly complex scenarios with provisions for extracting size complexity attributes.

For tool interoperability we integrated cost models with MBSE approaches and as web services. We enhanced service-based cost model tools for additional factors, and

demonstrated a COCOMO tool that reads MP input files for integrated costing [14] that should be further developed.

Product Lines

DoD systems within and across domains exhibit much functional commonality but are largely acquired independently leading to suboptimal designs and unnecessary costs. A product line approach can reduce costs, increase mission effectiveness, and accelerate deployment. Product line investment returns accrue from reusing common pieces in different systems that share features. An associated product line architecture and cost modeling framework was created to support related acquisition decisions.

The case studies demonstrated high ROI within and across domains for product line architectures. But not all attempts at product line reuse will generate large savings. Domain engineering needs to be done well to identify product line portions most likely to be mission-specific, fully reusable, or reusable with adaptation. Product line architecting needs to be done well to encapsulate the sources of product line variation effectively.

Cost models help evaluate the tradeoffs of different architectural options and determine when product line approaches are justified. The added granularity of Detailed COPLIMO covers more realistic situations. It offers increased value for DoD decision makers for tradeoff analysis supporting individual projects up through the product line level.

Future Work

Collaboration with AFIT on SQOTA laid groundwork for a sabbatical at AFIT to learn UAS development and flight testing to extend our research. This will complement and strengthen our joint capabilities for a more complete end-to-end MBSE approach through CONOPS definition, UAS design, construction and flight testing.

Plans are to continue expanding the COSYSMO framework for architectural tradeoffs with better fidelity for hardware aspects. The CubeSat architectures being developed at AFIT are another planned case study. Collaboration will also involve the design and evaluation of resilient systems with application to the small UAS domain.

Bibliography

1. Tradespace and Affordability – Phase 1 A013 - Final Technical Report SERC-2013-TR-039-1, Systems Engineering Research Center, July 2013

2. Tradespace and Affordability – Phase 2, A013 - Final Technical Report SERC-2013-TR-039-2, Systems Engineering Research Center, December 2013

3. -ilities Tradespace and Affordability – Phase 3, Technical Report SERC-2014-TR-039-3. Systems Engineering Research Center, December 2014, http://www.sercuarc.org/wp-content/uploads/2014/09/RT-113-Phase-3-Final-Technical-Report-2014-TR-39-3-20141231.pdf

4. Boehm, B., A. W. Brown, R. Madachy, and Y. Yang, "A Software Product Line Life Cycle Cost Estimation Model," Proceedings of the 2004 International Symposium on Empirical Software Engineering, 2004.

5. R. Madachy, Heuristic Risk Assessment Using Cost Factors, IEEE Software, May 1997

6. K. Giammarco and M. Auguston, Well, You didn't Say not to! A Formal Systems Engineering Approach to Teaching an Unruly Architecture Good Behavior, Complex Adaptive Systems Conference, 2013

7. M. Auguston and C. Whitcomb, "Behavior Models and Composition for Software and Systems Architecture", ICSSEA 2012, 24th International Conference on Software & Systems Engineering and their Applications, 2012

8. Boehm, B., C. Abts, A.W. Brown, S. Chulani, B. Clark, E. Horowitz, R. Madachy, D. Reifer, & B. Steece (2000). Software Cost Estimation with COCOMO II, Prentice Hall.

9. Department of Defense, MIL-STD-881C, Work Breakdown Structures for Defense Materiel Items (WBS), October 3, 2011

10. Maj. Ryan Pospisal (DTRA/A9, Kirtland AFB), "Application of Executable Architectures in Early Concept Evaluation", M.S. thesis, AFIT, December 2015

11. D. Jacques and R. Madachy, "Model-Centric UAV ISR Analysis," presented at Systems Engineering Research Center, 7th Annual SERC Sponsor Research Review, Washington, DC, December 3, 2015

12. R. Madachy, *Systems Engineering Cost Estimation Workbook*, Naval Postgraduate School, May 2017

13. Maj. Zhongwang Chua (Singapore AF), "Application of Executable Architecture in Early Concept Evaluation using DoDAF", M.S. Thesis, AFIT, September 2016

14. Monica Farah-Stapleton, “Resource Analysis Based On System Architecture Behavior”, Ph.D. thesis, NPS, September 2016

15. CPT Dennis Edwards (USArmy), “Exploring the integration of COSYSMO with a model based system engineering methodology in early trade space analytics and decisions”, M.S. thesis, NPS, June 2016

16. D. Jacques and R. Madachy, “Model-Driven UAS ISR Tradespace Analysis,” presented at Systems Engineering Research Center, 8th Annual SERC Sponsor Research Review, Washington, DC, November 17, 2016

17. R. Madachy, “System Cost Modeling and SysML Integration in Model-Based Systems Engineering,”, INCOSE San Diego Chapter Meeting, San Diego, CA, January 18, 2017

18. D. Jacques and R. Madachy, “Model-Based Systems Engineering Tradespace Analysis with SysML and COSYSMO,” 9th Annual SERC Sponsor Research Review, Washington, DC, November 2017.

19. Hall, R. (LT), “Utilizing a Model Based Systems Engineering Approach to Develop a Combat System Product Line,” MS Thesis, Naval Postgraduate School, June 2018

20. K. Chance, “Naval Combat Systems Product Line Economics: Extending the Constructive Product Line Investment Model for the Aegis Combat System”, MS Thesis, Naval Postgraduate School, June 2019

21. V. Manfredo, T. Jackson-Henderson, N. Fraine, "Naval ASW Combat System Product Line Architecture Economics," MS Capstone, Naval Postgraduate School, June 2019

22. R. Madachy, J. Green, , "Naval Combat System Product Line Economics," Proceedings of the 16th Annual Acquisition Research Symposium, Monterey, CA, May 2019

23. W. Rosa, R. Madachy, B. Clark, and B. Boehm. “Early phase cost models for agile software processes in the US DoD”. In Proceedings of the 2017 ACM / IEEE International Symposium on Empirical Software Engineering and Measurement, Toronto, Canada, November 2017. Best Paper Award.

PSU Final Report

PSU’s early SERC contributions included research on Product Architectures, Design, & Manufacturing for Operational Responsiveness, particularly in the area of 3D bringing of unmanned aerospace systems (UAS). This included participation in the "SERC Workshop:

Managing Acquisition and Program Risk" held on December 13, 2017. Role was two-fold in this workshop: (1) attend, discuss, and engage with members of the community having a shared interest in risk (e.g., strategies, approaches), and (2) to report on PSU research in the area, which included having designed, built, and flown 3D printed UAS, including instrumentation the design process, gathering metrics on the design evolution to include total man-hours, models used, and design iterations. PSU also prepared and submitted for publication an invited article to the Naval Engineers Journal, “Set-Based Design Model-Based Systems Engineering and Sequential Decision Processes.” provided below.

USC Final Report

The primary outcomes of the USC deep dive on improving system Maintainability are summarized in the Wiley Systems Engineering article below. Besides Maintainability, USC addressed next-generation cost estimation models, including completion of the COSYSMO 3.0 synthesis of the previous versions of the Constructive Systems Engineering cost models (Alstad, 2018); a cost model for agile DoD projects (Rosa-Madachy-Clark-Boehm, 2017), also provided below, and a detailed survey of approaches and models for estimating the cost of developing secure software systems (Venson et al., 2019)

Alstad, J., Ph.D. Dissertation, "The COSYSMO 3.0 Cost Model," USC Dissertation archive

Rosa, W., Madachy, R. Clark, B., and Boehm, B., "Early Phase Cost Models for Agile Software Processes in the US DoD," Proceedings, 2017 Empirical Software Estimation and Measurement Conference (Best Paper Award).

Venson, E., Alfayez, R., Gomes, M., Figueiredo, R., and Boehm, B., "The Impact of Security Practices on Development Effort: An Initial Survey," Proceedings, ESEM 2019.

Anticipatory Development Processes for Reducing Total Ownership Costs and Schedules

Barry Boehm[1] | **Pooyan Behnamghader**[1]

[1]Computer Science Deparment, University of Southern California, Los Angeles, California, 90007, USA

Correspondence Barry Boehm, Computer Science Department, University of Southern California, Los Angeles, California, 90007, USA
Email: boehm@usc.edu

Present address
† Department of Computer Science, University of Southern California, Los Angeles, California, 90007, USA

Funding information This material is based upon work supported in part by the U.S. Department of Defense through the Systems Engineering Research Center (SERC) under Contract H98230-08-D-0171. SERC is a federally funded University Affiliated Research Center managed by Stevens Institute of Technology. It was also supported by the National Science Foundation grant CMMI-1408909, Developing a Constructive Logic-Based Theory of Value-Based Systems Engineering.

Many systems and software processes overfocus on get- ting a project and product from an initial set of require- ments to an Initial Operational Capability (IOC). Examples are most waterfall and V models. Projects following such processes may pass acceptance tests for functionality and performance, but may leave the product with serious main- tainability shortfalls. Many agile processes focus on users' initial usage priorities, but often make development commit- ments for earlier needs that are incompatible with achieving later critical needs (e.g., security, safety). Incremental de- velopment process models can do better, but often later increments may find that the earlier increments have not prepared them for ease of modification and repair. Besides increasing Total Ownership Costs (TOCs), long mean times to repair result in long downtimes, which can be critical to an organization's income and reputation. Further, many of these shortfalls take the form of Technical Debt (TD), in that the later they are fixed, the more slow and expensive will be the fixes.

This paper summarizes three process frameworks and tools providing more anticipatory ways to improve systems and software maintainability and life-cycle cost-effectiveness. The first framework is an Opportunity Tree for identifying and anticipating such ways. The second framework (SQUAAD) is a toolset for tracking a software project's incremental code commits, and analyzing and visualizing each commit's

incremental and cumulative TD. The third framework is a Software/Systems Maintenance Readiness Framework (SMRF), that identifies needed maintenance readiness levels at de- velopment decision reviews, similar to the Technology Readi- ness Levels framework.

KEYWORDS

Software Engineering, Software Maintainability,

Introduction

The inexorable increase in software demand and the pace of changes in software technology, competition, interde- pendencies, and sources of vulnerability, will seriously strain the capacity of the available software maintenance labor force.

We have been involved in several aspects of analyzing and addressing the root causes of expensive software maintenance. These include serving on assessments of exemplary and problem software projects in government and industry; evolving and calibrating models for estimating software development and maintenance costs; leading a multi-year, multi-university US Department of Defense Systems Engineering Research Center to research and develop Software/Systems Qualities Ontology, Tradespace, and Affordability (SQOTA) capabilities; and researching, developing, and evaluating promising methods, processes, and tools (MPTs) for improving software life cycle productivity and qualities.

We try to quantify such relationships where possible. For example, the 2015 US General Accountability Office report (Dodaro, 2015) identified annual US Government Information Technology (IT) expenditures of $79 billion, of which $58 billion were in operations and maintenance (O&M). Table 1 provides more detail on fractions of hardware and software costs from the 2008 Redman (Redman, 2008) and 2009 Koskinen (Koskinen, 2009) studies.

TABLE 1 Percentage of Post-Deployment Life Cycle Cost

Hardware (Redman, 2008)	Software (Koskinen, 2009)
60% – Ships (average)	50-80% – Complex cyber-physical systems
12% – Missiles (average)	75-90% – Business, Command-Control
78% – Aircraft (F-16)	10-30% – Simple embedded software
84% – Ground vehicles (Bradley)	

The SQOTA ontology identified Maintainability as not only supporting Affordability in terms of total ownership costs, but also supporting Changeability and Dependability (see Table 2). Maintainability supports Changeability in terms of rapid adaptability to new opportunities and threats, and also supports Dependability in terms of Availability, in that reducing Mean Time to Repair (MTTR) for a system with a given Reliability in terms of Mean Time Between Failures

(MTBF) improves Availability via the equation Availability = MTBF / (MTBF+MTTR) (Boehm et al., 2016).

Note also that the SQOTA ontology's definition of the key quality of Resilience or the combination of Dependability and Changeability is consistent with the INCOSE Systems Engineering Handbook's definition of Resilience or "the ability to prepare and plan for, absorb or mitigate, recover from, or more successfully adapt to actual or potential adverse events" (INCOSE, 2015; Haimes, 2012).

Many system development projects strongly focus on creating and evaluation the systems' Initial Operational Capability (IOC). In doing so, they often miss opportunities to make the system more cost-effectively maintainable. More recently, both commercial and defense organizations have found it competitively critical to perform continuous development & delivery, or DevOps, involving significant changes in preparing for post-IOC evolution, as recommended in the 2018 Design and Acquisition of Software for Defense Systems report (Defense Science Board, 2018).

TABLE 2 Upper Levels of SERC Stakeholder Value-Based System Quality (SQ) Means-Ends Hierarchy (Boehm et al., 2016)

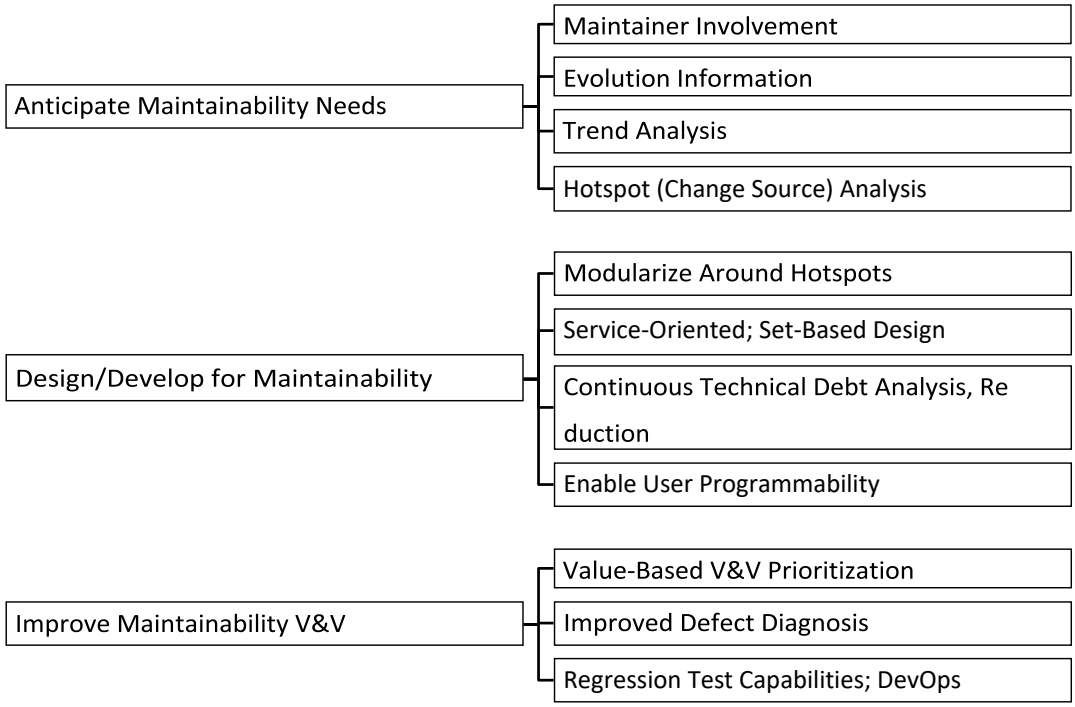
STAKEHOLDER	Contributing SQ Means
Value-Based SQEnds	
	Stakeholders-Satisfactory Balance of Physical Capability, Cyber Capability, Human
Mission Effectiveness	Usability, Speed, Endurability, Maneuverability, Accuracy, Impact, Scalability,
Life Cycle Efficiency	Development and **Maintenance Cost**, Duration, Key Personnel, Other Scarce Versatility, Interoperability, Domain-Specific Objectives Resources; Manufacturability, Sustainability
Dependability	Reliability, **Maintainability**, Availability, Survivability, Robustness, Graceful Degradation, Security, Safety
Changeability	**Maintainability**, Modifiability, Repairability, Adaptability

Composite Quality Attributes (QAs)	
Affordability	Mission Effectiveness, Life Cycle Efficiency
Resilience	Dependability, Changeability

Section 2 summarizes the Opportunity Tree of ways to improve systems and software maintainability and life- cycle cost-effectiveness. Section 3 summarizes the Software Quality Understanding by Analysis of Abundant Data (SQUAAD) toolset for tracking a software project's incremental code commits, and analyzing and visualizing each commit's incremental and cumulative Technical Debt (TD). Section 4 summarizes the Systems and Software Maintenance Readiness Framework (SMRF), that identifies needed maintenance readiness levels, and reflects that many sources of TD are non-technical. Section 5 summarizes the resulting conclusions.

12 | MAINTAINABILITY OPPORTUNITY TREE

Figure 1 provides an opportunity tree for Maintainability. As with our other opportunity trees for reducing Cost, reducing Schedule, and improving Dependability aspects, it provides a checklist for systems engineers and developers to use in improving a system's maintainability.



F I G U R E 1 Maintainability Opportunity Tree

Anticipate Maintainability Needs: Maintainer Involvement. Lack of maintainer involvement in system and soft- ware development often results in key maintainability enablers being either neglected or implemented in incompatible ways. It may take some effort to get the maintainers involved, as often they are stuck in a vicious circle and are too busy overcoming the maintainability shortfalls caused by previous projects lacking maintainer involvement in their definition and development.

Evolution Information and Trend Analysis. Often, the requirements for a system acquisition are determined by prioritizing the capabilities, and using a Cost as Independent Variable (CAIV) analysis to determine which capabilities fit within the available budget and are to be included in the Request for Proposal (RFP). This unfortunately throws away valuable below-the-line information on the most likely

directions of system evolution, often leading to a brittle point-solution architecture as the chosen solution. Including the below-the-line capabilities in the RFP as candidate Evolution Capabilities Information, and indicating that it should be considered in preparing the system's life cycle architecture, is more likely to result in reducing operations and maintenance costs. Trend Analysis is another way to identify likely directions of system evolution.

Hotspot (Change Source) Analysis. Common sources of change include user interface changes, device driver changes, external interface changes, and changes in Non-Developmental Items (NDIs). Many Commercial-Off-The-Shelf

(COTS) NDI products have new releases every 8-12 months, and continue support for only the latest three releases. In gathering data for a COTS-integration version of the COCOMO cost model, we found one large project entering maintenance with 120 COTS products, 55 of which were no longer supported (Abts et al., 2000). Other NDIs are more volatile: for example, Amazon's cloud services are updated every 11 seconds. Open-source NDIs vary widely on their change frequency. Systems employing numerous COTS products, and NDIs may therefore have numerous changes, often becoming too expensive to maintain.

Design/Develop for Maintainability: Modularization Around Hotspots. The 1979 Parnas paper, "Designing Soft- ware for Ease of Extension and Contraction," (Parnas, 1979) and earlier Parnas papers identify this maintainability strategy. Common sources of change that have been encapsulated in modules will contain the change effects within the module rather than having them ripple across the other parts of the system. Using this and related TRW data-driven strategies enabled large projects to perform maintenance changes at lower costs than development changes (Royce, 1998).

Service Orientation; Set-Based Design. Service-oriented loose coupling is a design principle that is applied to the services in order to ensure that the service contract is independent of the underlying service logic and implementation (Sundbo and Gallouj, 2000). Basically, the concept of loose coupling provided by a Service-Oriented Architecture (SOA) component is that it publishes a contract indicating that if it is furnished with a specified set of inputs, it will produce a specified set of outputs, without otherwise interacting with the user's environment (Boehm and Bhuta, 2008; Borges et al., 2004).

If a project has a range of choices among services, algorithms, COTS products, etc., it is often better to identify the strongest sets of choices of each and carry them along in a Set-Based Design (SBD) and to converge on the best choices as sufficient information becomes available (Bernstein, 1998; Reinertsten, 2009). Set-based design is also valuable if a system is part of one or more systems of independently-evolving systems.

Continuous Technical Debt Analysis and Reduction. Technical Debt (TD) refers to delayed technical work or rework that is incurred when shortcuts are taken. The shortcuts may have good rationales such as the need to meet a market window or

to field defenses against cyber or physical attacks. Or they may result from poor project coordination or sloppy work habits. Examples are duplicated code or files; unused code; uninitialized variables; and unguarded exceptions. Either way, the later the debt is paid, the more it will cost, corresponding to interest on a financial debt.

Finding such sources of TD via code inspections can be labor-intensive if done for each commit, or delayed if done for each release, although human-assessed indicators of software understandability have been shown to be helpful in estimating maintenance costs (Chen et al., 2016). Fortunately, tools are becoming available for assessing software TD, such as SonarQube, CAST, PMD, and FindBugs. Section 3 will summarize the Software Quality Understanding by Analysis of Abundant Data (SQUAAD) toolset for tracking a software project's incremental code commits, and for analyzing and visualizing each commit's incremental and cumulative TD.

Enable User Programmability. In many circumstances, users have found it easier to develop spreadsheet applica- tions for specific needs rather than trying to master and tailor complex general applications to fit their needs. Many special devices with numerous options (e.g., medical infusion pumps; educational robots) have simple special-purpose languages for specifying the options. User Programmability may have problems: an IBM study found that 44% of a large sample of spreadsheet programs had defects that would if exercised have had major negative financial outcomes for the organization.

Improve Maintainability Verification and Validation (V&V): Value-Based V&V Prioritization. Most V&V aids, such as automated test case generation, assume that every test case and defect is equally important. However, in practice, projects find that the value of running the test cases follows a Pareto distribution, in which 20% of the test cases produce 80% of the business value. For example, one company found that one of its 15 customer sets accounted

for 50% of the business value of prompt billing, and 3 of its customer sets accounted for 80% of the business value (Bullock, 2000). A further industrial application increased the business value of testing their annual product release features from 58% to 91% (Li, 2012).

Regression Test Capabilities; DevOps. Amazon's DevOps ability to reliably upgrade its huge variety of services every 11 seconds implies a remarkable ability to perform regression testing of each release's changes. Providing similarly reliable, rapid large-scale DevOps upgrades will require similar rapid regression testing upgrades. Software maintenance organizations expected to carry on similar DevOps capabilities would need similar rapid regression testing capabilities, along with further Testability enablers of having and evolving cost-effective test drivers, test oracles, test data management capabilities, and diagnostic capabilities.

13 | SOFTWARE QUALITY UNDERSTANDING BY ANALYSIS OF ABUNDANT DATA (SQUAAD) TOOLSET

Software developers can prevent failures and disasters and reduce total ownership costs by putting more emphasis on improving software maintainability in their software development process. One way to improve software maintainability is to produce clean code while changing the software and to continuously assess and monitor code quality while the software is evolving (Mexim and Kessentini, 2015).

Prior research has been focusing on the analysis of official releases of software to understand how its code quality evolves (Pinto et al., 2015; Godfrey and Tu, 2000; Ganpati et al., 2012; D'Ambros et al., 2008; Le et al., 2015). This approach gives an insight on change in code quality over the major milestones of software development, rather than how code quality evolves during software development process. For example, a developer may unknowingly increase the amount of Technical Debt (TD) over a few commits. If that debt is not addressed quickly, it can impose extra cost. It gets even worse, if they leave the team without paying that debt. In another example, a developer may simply commit broken code to the repository. This will break the code for other contributors and slows down the development. It also causes the unavailability of byte-code for that commit in a post-development analysis. Since

software developers do not ship uncompilable code in official releases, this detail may not be revealed in that coarse-grained analysis.

Our approach to understand software quality evolution is not limited to study the official releases but to take a step further by analyzing the state of the software after each commit. For example, our study on the evolution history of 68 open-source software systems shows from one release to the next one, software contains fewer lines of code, classes, code smells, and security vulnerabilities in 8%, 4%, 14%, and 6% of times. However, from one revision (produced by a commit) to the next one, these ratios are 18%(↑), 3%(↓), 17%(↑), and 2%(↓). Analyzing the impact of each commit on software quality can reveal a wealth of information because commits carry fine-grained data on every stage of software evolution, such as the author, the time, and the intent (i.e., commit message) of change.

Over the past couple of years, multiple tools and techniques are introduced to study software evolution by commit- level (Tufano et al., 2017; Dyer et al., 2015; Diamantopoulos et al., 2016; Tiwari et al., 2017; Pinto et al., 2015; Gousios et al., 2014; Rozenberg et al., 2016; Kaur and Chopra, 2018; Sokol et al., 2013; Trautsch et al., 2017, 2016). Some of them do not run analysis on source code (Gousios et al., 2014; Rozenberg et al., 2016). Some of them are designed and implemented to run source code analysis sequentially (Tufano et al., 2017; Sokol et al., 2013). Consequently, the execution of the study requires strong on-premise resource and takes multiple weeks (Tufano et al., 2017). There are some mining techniques designed to scale by running static analysis on different revisions of each file in parallel (Dyer et al., 2015; Alexandru et al., 2017; Trautsch et al., 2017). These techniques are extremely efficient in generating an Abstract Syntax Tree (AST) (Alexandru et al., 2017; Dyer et al., 2015) and calculating file-based quality metrics, such as

size, cohesion, and complexity (Trautsch et al., 2017). They can also aggregate the result of individual files analysis to evaluate the quality of a project. However, this approach for achieving scale and avoiding redundancy is not suitable to understand software quality evolution by program analysis techniques that

- analyze a **module** considering all of its source code entities and their **relationship**. For example, architecture recovery techniques generate clusters of source code entities by analyzing semantic and/or structure relationships between them (Garcia et al., 2013; Tzerpos and Holt, 2000). It does not suffice to separately analyze the changed files and aggregate the results to recover the architecture of a new revision.
- require **bytecode**. Some static and dynamic program analysis techniques depend on the availability of the compiled version (e.g., FindBugs (Ayewah et al., 2008) and test-coverage (Malaiya et al., 2002)). A commit may change the version of a dependency. The new dependency is available and the build configuration (e.g., *build.gradle*) is syntactically correct. However, the new revisions do not compile as the dependency is not backward compatible (Behnamghader et al., 2017a). A recent study (Alexandru et al., 2017) declares the unavailability of the compiled versions as the main unresolved source for the manual effort in software evolution analysis.

In addition, some tools rely on a complex **environment** to run. This includes dynamic techniques that need an execution environment and static techniques with specific requirements. For example, SonarQube (Campbell, 2015) requires deploying its own analysis server, generating a configuration file for each revision, executing the analysis, and fetching the results using SonarQube Api.

We took steps toward addressing that scarcity by developing Software Quality Understanding by Analysis of Abundant Data (SQUAAD) (Behnamghader and Boehm, 2018), a comprehensive framework including a cloud-based automated infrastructure accompanied by a data analytics toolset and web interfaces. Our multi-perspective software quality evolution approach assesses different quality attributes such as software size, code quality, and security. It utilizes complex program analysis techniques (e.g., byte-code analysis using FindBugs and architecture recovery using ACDC), COTS tools with complex environments (e.g., SonarQube), and dynamic analysis (e.g., unit-test pass-rate). It enables analysis of the conflicts and synergies between different quality attributes and the difference between developers in terms of their impact on software quality. Our integrated tool-based approach has been documented in multiple research publications (Behnamghader and Boehm, 2018; Behnamghader et al., 2018, 2017a,b; Alfayez et al., 2017, 2018) empowering their empirical studies and is used by a major governmental entity.

SQUAAD is designed to target a module, compile its distinct revisions, and run static/dynamic analysis on it. Before conducting a large-scale analysis, SQUAAD runs a light-weight mining task on the software's Git repository to determine which commit changes a module (impactful commits (Behnamghader et al., 2017a)) and the evolutionary relationship between those commits (Behnamghader et al., 2018). Then it automatically 1) distributes hundreds of revisions on multiple cloud instances, 2) compiles each revision using default and/or user defined configurations, 3) provisions the environment and runs static/dynamic programming analysis techniques on each revision, 4) collects the generated artifacts, 5) either parses them to extract quality attributes or compares them to each other to calculate the difference (e.g., by architectural distance metrics), and 6) runs various statistical analysis on software quality evolution.

The entire analysis workflow is automated. As soon as the framework is configured for a subject system, we can run the represented analysis on that system and study its evolution. This full automation makes analysis conducted by SQUAAD replicable and addresses a major threat (i.e., not being replicable) to the external validity of repository mining studies (Trautsch et al., 2016). We have also developed web interfaces to illustrate the evolution of different quality attributes and the impact of each developer on software quality.

For example, one of our recent maintainability trends analysis (Behnamghader et al., 2017a) involves a total of

19,580 examined revisions from 38 Apache-family systems across a timespan from January 2002 through March 2017, comprising 586 MSLOC. In this analysis, to obtain software quality, we used three widely-used open-source static analysis tools: FindBugs, PMD, and SonarQube. We selected a subset of quality attributes related to size (basic), code quality, and security. We found that on average, 2% of impactful commits break the compilation. We qualitatively investigated when, how, and why developers break the compilation and introduced a guideline for preventing developers from committing uncompilable code. We found that different quality attributes may change even if the size (i.e., code count) of the software does not change. We calculated the probability for a metric to change while another one is constant based on the collected data. Our result showed that employing multiple security metrics together can reveal points where security problems are introduced.

In a follow-up study (Behnamghader et al., 2018), we extended our original dataset to include 30 new subject systems from Google and Netflix, as well as revisions committed in 2017 and 2018 of 38 Apache systems of the original dataset. The extended dataset comprises more than 37k analyzed distinct software revisions and more than 1.5 billion lines of code. We replicated the analysis mentioned above on the new dataset and extended it to reach the maximum commit compilability (97.7% for Apache, 99.0% for Google, and 93.9% for Netflix) and identify all uncompilable commits that are caused by a developer's fault. We quantitatively analyzed the reasons for introducing compile errors and proposed a model to detect the uncompilable commits in a post-development analysis based on the meta-date of the commits (i.e., time, message, and committer) and without considering code artifacts. Then we analyzed all compilable commits to understand the difference between affiliated and external developers of each organization in terms of impacting different quality attributes. For example, our analysis shows that although there is no difference between affiliated and external developers in terms of changing the size of the software, external developers of Netflix and affiliated developers of Google have higher ratio of non-compilable commits.

Tools such as SQUAAD can enable organizations to continuously monitor their sources of TD and remove them quickly during development, rather than pay for them

with interest during maintenance. We are creating versions for some government support organizations to aid in their monitoring and prioritization of TD and its removal.

14 | ADDRESSING NON-TECHNICAL SOURCES OF TECHNICAL DEBT

Our multi-year, multi-university US DoD Systems Engineering Research Center (SERC) Software/Systems Qualities Ontology, Tradespace, and Affordability project has held and participated in several industry-government workshops to clarify the relations among their various systems and software quality attributes. These have led to the stakeholder value-based, means-ends ontology structure shown in Table 2, which highlights Maintainability as a key Means to three of the four stakeholder-value Ends categories [35]. The workshops also confirmed the key contribution of Technical Debt (TD) to systems and software maintenance costs, and identified that there were a number of non-technical sources of TD, to be summarized in section 4.1, and addressed via the Systems and Software Maintainability Readiness Framework (SMRF) presented in section 4.2.

14.1 | TOP-10 LIST OF MAJOR NON-TECHNICAL SOURCES OF TECHNICAL DEBT

The inexorable increase in software demand and the pace of changes in software technology, competition, interdepen- dencies, and sources of vulnerability, will seriously strain the capacity of the available software maintenance labor force. Recently, a US industry-government workshop was held to address this challenge and what to do about it. A good deal of the discussion was focused on the high cost of software TD (Kruchten et al., 2012), and on ways to identify it and

reduce it more quickly. During the discussions, several observations were made that a good many sources of TD are non-technical, and that addressing these would likely be cost-effective. In response, a portion of the workshop was devoted to identifying and prioritizing these non-technical sources of TD.

The working group addressing the non-technical sources of TD identified 17 non-trivial, relatively non-overlapping sources. The 12 group members were then given 20 points each to distribute across the 17 sources, and the points added up to determine the order of the Top-10. Most of the participant distributions of points were relatively flat, but some gave 5-7 points for sources they felt were particularly critical.

Here is the resulting Top-10 list of the primary sources of software process foresight shortfalls causing significant levels of TD, along with their point scores.

1. Separate organizations and budgets for software acquisition and maintenance (34 points). The acquisition organization will tend to over-optimize on its primary responsibility for acquisition cost-effectiveness, leaving the maintenance organization unprepared for cost-effective maintenance.
2. Overconcern with the Voice of the Customer, as in Quality Function Deployment (31) (Akao, 1994). Delighting customers with attractive features often leads to commitments to incompatible and hard-to-maintain capabilities, which could be detected by listening to the Voice of the Maintainer. A good example was the Bank of America's Master Net trust management system. It proposed to excel by including the union of its trust management system customers' wish lists, ending up with 3.5 million lines of code worth of promised capabilities, clearly far beyond what they could produce with their $22M budget and 9 month schedule. In searching for options, they found Premier Systems, which had produced successful trust management systems for several small banks. Not only was Premier unable to scale its software up to BofA's promises, but also its software ran on Prime computers, which were unacceptable to BofA's software maintenance organization, which only operated IBM mainframe computers. The project was cancelled after 4 years and an $88 million expenditure (Glass, 1997).
3. The Conspiracy of Optimism (28). The project sponsors are competing for resources with sponsors of other projects, and will tend to be optimistic about what the project will deliver and how much it will cost. They will often try to get well by outsourcing the development to the lowest cost, technically acceptable bidder. Contractors bidding to perform the project will also tend to be optimistic about what the project will deliver and how much it will cost. A good example is the US Air Force F-22 aircraft. It proposed to deliver 750 aircraft for $26.2 billion, and when cancelled had delivered 187 aircraft for $79 billion (Haffa and Datla, 2016).
4. Inadequate system engineering resources (21). The first organization to be impacted by inadequate budgets and schedules will be system engineering. The result will be exponentially-large amounts of TD due to poorly-defined interfaces, unaddressed rainy-day use cases and risks, and premature commitments to hopefully-compatible but actually-incompatible COTS products, cloud services, open-source capabilities, and hopefully-reusable components. The inadequate resources provide no opportunity to develop and review evidence of the feasibility (scalability, compatibility, performance, dependability, maintainability, etc.) of the commitments. A good example identified in the workshop was a space system optimistically costed at $2 billion. The project budgeted 30% of the cost or $600 million to ensure a thorough job of systems engineering. However, the actual cost of the system was $8 billion, and the $600 million was only 7.5% of the cost. This led to incomplete specifications and prototypes, weak evidence of feasibility, undefined interfaces, vague plans, etc., accounting for much of the cost growth.

5. Hasty contracting that focuses on fixed operational requirements (21). If budgets and schedules are tight, and the contract does not require delivery of test and debugging support, architectural descriptions, development support and configuration management capabilities, and latest release COTS products, these will not be made available to the maintainers. Even if contracted-for, these may be dropped or minimized as lower-priority needs as compared to

operational capabilities. Having a fixed-requirements contract is a source of significant delays, particularly as the pace of change in software-intensive systems continues to accelerate. Another pair of large projects undergoing rapid change identified in the workshop required averages of 27 workdays to close simple one-company change requests; 48 workdays for 2-3 company change requests; and 141 workdays for change requests requiring contract modifications.

6. CAIV-limited system requirements (20). Often, customers' desired capabilities exceed the available conspiracy-of-optimism budgets, and a Cost As Independent Variable (CAIV) exercise is performed to prioritize the capabilities, and to include only the above-the-line capabilities in the Request for Proposals or Statement of Work. This throws away valuable information on the likely sources of future maintenance activity.
7. Brittle, point-solution architectures (18). The lowest-cost, technically-acceptable winning bidder will generally commit to a brittle, point-solution architecture addressing only the capabilities in the Statement of Work, minimizing development costs but again escalating maintenance costs.
8. The Vicious Circle (15). Even when acquisition organizations wish to include the Voice of the Maintainer and invite them to participate in defining a new system, they will often be met with apologies that the maintainers are too busy compensating for the maintainability shortfalls in their current systems caused by their inability to participate in the current-systems' definition.
9. Stovepipe systems (12). Having different organizations implement increasingly-interdependent systems leads to numerous clashes in coordinating changes across systems with incompatible internal assumptions, infrastructure commitments, user interfaces, and data structures. Regional and national healthcare systems are just one of many examples.
10. Over-extreme forms of agile development (10), such as rejecting architectural descriptions as Big Design Up Front (BDUF) and saying You Aren't Going to Need It (YAGNI). This may be true on small projects where the developers continue into maintenance, but will be disastrous if provided to a different maintenance organization, or when the small project grows into a 50-person team coping with evolving a 500K source lines of code (KSLOC) project (Elssamadisy and Schalliol, 2002). More recently, however, organizations are mastering disciplined forms of agile development and continuous delivery, such as Amazon with its new release every 11 seconds or methods such as the Scaled Agile Framework (Leffingwell, 2007), Kanban (Anderson, 2010), DevOps (Davis and Daniels, 2016), and the Jan Bosch incremental Speed, Data, and Ecosystems approach (Bosch, 2017).

The 7 additional non-technical sources of TD received relatively small numbers of votes, but each received at least 3 votes:

- Choosing lowest-cost, technically acceptable maintenance contractor (6);
- Delivering systems with no-longer-supported COTS products (5);
- Easiest-first development problem-report closure (4);
- High development personnel turnover (4);
- High requirements volatility (3);
- Neglecting interoperability challenges (3);

- Over-optimizing performance via tightly-coupled, speed-optimized components (3).

TABLE 3 Software/Systems Maintenance Readiness Framework (SMRF)

SMR Level	OpCon, Contracting: Missions, Scenarios, Resources, Incentives	Personnel Capabilities and Participation	Enabling Methods, Processes, and Tools (MPTs)
9	5 years of successful maintenance operations, including outcome based incentives, adaptation to new technologies, missions, and stakeholders.	In addition, creating incentives for continuing effective maintainability. Performance on long-duration projects.	Evidence of improvements in innovative O&M MPTs based on ongoing O&M experience.
8	One year of successful maintenance operations, including outcome based incentives, refinements of OpCon. Initial insights from maintenance data collection and analysis (DC&A).	Stimulating and applying People CMM level 5 maintainability practices in continuous improvement and innovation such as smart systems, use of multicore processors, and 3-D printing.	Evidence of MPT improvements based on maintenance DC&A based ongoing refinement, and extensions of ongoing evaluation, initial O&M MPTs.
7	System passes Maintainability Readiness Review with evidence of viable OpCon, Contracting, Logistics, Resources, Incentives, personnel capabilities, enabling MPTs, outcome-based incentives.	Achieving advanced People CMM level 4 maintainability capabilities such as empowered work groups, mentoring, quantitative performance management and competency based assets.	Advanced, integrated, tested, and exercised full-LC MBS&SE MPTs and Maintainability-other-SQ tradespace analyses.
6	Mostly-elaborated maintainability OpCon, with roles, responsibilities, workflows, logistics management plans with budgets, schedules, resources, staffing, infrastructure and enabling MPT choices, V&V and review procedures.	Achieving basic People Capability Maturity Model (CMM) levels 2 and 3 maintainability practices such as maintainability work environment, competency and career development, and performance management especially in such key areas such as V&V, identification & reduction of Technical Debt.	Advanced, integrated, tested full-LC Model-Based Software & Systems (MBS&SE) MPTs and Maintainability-other-SQ tradespace analysis tools identified for use, and being individually used and integrated.
5	Convergence, involvement of main maintainability success-critical stakeholders. Some maintainability use cases defined. Rough maintainability OpCon, other SCSHs, staffing, resource estimates. Preferred maintenance organization option, incentive structures determined.	In addition, independent maintainability experts participate in project evidence-based decision reviews, identify potential maintainability conflicts with other SQs. Selected developers and maintainers work out skills mixes, collaboration options.	Advanced full-lifecycle (full-LC) O&M MPTs and SW/SE MPTs identified for use. Basic MPTs for tradespace analysis among maintainability & other SQs, including TOC being used.
4	Artifacts focused on missions. Primary maintenance options determined, Early involvement of maintainability SCSHs in elaborating and evaluating maintenance-organization options.	Critical mass of maintainability SysEs with mission SysE capability, coverage of full maintainability SysE skills areas, representation of maintainability SCSH organizations.	Advanced O&M MPT capabilities identified for use: Model-Based SW/SE, TOC analysis support. Basic O&M MPT capabilities for modification, repair and V&V: some initial use.
3	Elaboration of mission Operational Concept (OpCon), Architectural views, life-cycle cost estimation. Key mission, O&M, success-critical stakeholders (SCSHs) identified, some maintainability options explored.	O&M success-critical stakeholders provide critical mass of maintainability-capable SysEs. Identification of additional Maintainability-critical stakeholders.	Basic O&M MPT capabilities identified for use, particularly for OpCon, Architecture, and Total Ownership Cost (TOC) analysis. Some exploratory initial use.
2	Mission evolution directions and maintainability implications explored. Some mission use cases defined, some O&M options explored.	Highly maintainability-capable Systems Engineers (SysEs) included in Early SysE team.	Initial exploration of O&M MPT options.
1	Focus on mission opportunities, needs. Maintainability not yet considered.	Awareness of needs for early expertise for maintainability. Concurrent engineering, O&M integration, Life Cycle cost estimation.	Focus on O&M MPT options considered.

Several classes of organizations generally do not have serious problems with high maintenance cost for their more diverse and dynamic software-intensive systems. Some have developers who continue with the project through its life cycle. Some whose business or mission depends critically on high levels of service employ and support highly-capable in-house maintenance organizations.

Classes of organizations most needing to reduce high maintenance costs for their more diverse and dynamic software-intensive systems are those in which research and development (R&D) and operations and maintenance (O&M) are separately funded and managed; organizations which outsource software maintenance to external companies; and organizations with in-house software maintenance centers that receive and maintain software developed either elsewhere in the organization or externally. For such organizations, three of the primary symptoms of high maintenance costs identified in the workshop above were (1) life cycle management shortfalls; (2) maintenance personnel shortfalls; and (3) maintenance methods, processes and tools (MPTs) shortfalls.

The concepts of Technology Readiness Levels (TRLs) (DoD, 2011), Manufacturing Readiness Levels (MRLs) (Cundiff, 2003), and System Readiness Levels (SRLs) (Sauser et al., 2006; Sauser, 2007), have been highly useful in improving the readiness of systems to be fielded and operated. Given the discussions above on the non-technical sources of Technical Debt, it appears worthwhile to develop and use a similar Software Maintainability Readiness Framework (SMRF) to improve future systems' continuing operational readiness and total ownership costs. Most likely, its content would also help on hardware-intensive systems or cyber-physical-human systems.

Table 3 provides our current SMRF. Its columns are organized around the three major maintainability readiness shortfall categories of Life Cycle Management, Maintenance Personnel, and Maintenance MPTs. In general, one would expect a major defense acquisition project to be at SMRF 4 at its Materiel Development Decision milestone; at SMRF 5 at its Milestone A, also called its Architecture Alternatives Analysis Review; SMRF 6 at its Milestone B, also called its Preliminary Design Review; and SMRF 7 at its completion of its Operational Readiness Review. Smaller

less-critical systems would be expected to be at least at SMRF 3 at its Materiel Development Decision milestone and at SMRF 4 at its Milestone A. Note that the SMRF framework emphasizes outcome-based maintenance incentives such as with Performance-Based Logistics or Vested Outsourcing (Vitasek and Ledyard, 2013) at SMRF 7, and maintainability data collection and analysis (DC&A) at SMRF 8.

14.3 | EARLY EVALUATION RESULTS

The SMRF has been used on over 10 milestone reviews, generally resulting in improvements in maintainability planning, maintainer participation in project activities and reviews, and identification of methods, processes, and tools needed by maintainers such as for requirements traceability, architecture definition and evolution, configuration management, problem diagnostics, TD analysis, and regression testing. At this point, a major company organization is preparing to apply it to its projects. As one example, one fairly large project doing a Milestone B Preliminary Design review (level 6), was found to be at Level 4 in Maintainability Planning and Maintainability Personnel Capabilities, and at Level 5 in Enabling Methods, Processes, and Tools. This led to a number of corrective actions to become close to Level 7 by the time of transition to its maintenance organization.

There are some process standards that address these concerns, such as ISO/IEC 24748-1, "Systems and Software Engineering – Life Cycle Management." Its sections 4.5, Utilization stage, and 4.6, Support stage, both say "It is presumed that the (supporting) organization has available the facilities, equipment, tools, processes, procedures, trained personnel and instruction manuals." Its section 4.3.3 (e) identifies evidence of the systems supportability as a delivery outcome.

However, these objectives are often minimized, a situation not despised by its Section 3.3.1. b), stating that the various processes should be "Loosely coupled, meaning that the number of interfaces among the processes is kept to a minimum." (Software & Systems Engineering Standards Committee, 2011)

Other evaluation results have included the evaluation of projects having high TD in both development and mainte- nance, and development of parametric models that relate the sources of TD to their ultimate magnitude. These include calibration of a model to evaluate the return on investments in maintainability based on data from two TRW projects that did not make the investments and one that did: CCPDS-R, described in (Royce, 1998). Another corroborative result is the analysis of exponential growth of TD due to systems engineering underinvestment experienced across the 161 projects involved in the calibration of the COCOMO II model's Architecture and Risk Resolution parameter (Boehm et al., 2000). The Vicious Circle phenomenon was exhibited in major architecture reviews of two large government projects. One project fortunately had two people with maintenance experience on the review team, who were able to provide maintainability recommendations that helped the project avoid significant maintenance costs. The other maintenance project did not have such people, and its maintenance organization experienced extensive workload growth and an inability to quickly and cost-effectively respond to needed changes.

15 | CONCLUSIONS

The increasing complexity of software-intensive systems and the rapid pace of change in technology are driving organizations' software total ownership costs more and more toward software maintenance. Examples are more, larger, and more complex software systems such as Internet (or Internets) of Things and self-driving vehicles; increasing needs for software dependability and interoperability; increasing software autonomy; increasing data capture and data analytics; increasing legacy software; and mounting Technical Debt (TD).

Fortunately, increasing data capture and data analytics capabilities are providing organizations with stronger ways to analyze and reduce their software's TD. Commercial capabilities such as CAST, and TD analysis tools such as those

summarized in Section 3 of this paper are just the beginning of the capabilities possible in this area.

However, much greater savings can be achieved by addressing three non-technical sources of TD due to overem- phasis on initial acquisition cost-effectiveness. These include shortfalls in Life Cycle Management aspects; Personnel capabilities and participation aspects; and Maintenance methods, processes, and tools aspects. Drawing on the success- ful use of the Technology Readiness Level, Manufacturing Readiness Level, and System Readiness Level frameworks, this paper provides a similar Software/Systems Maintenance Readiness Framework (SMRF), based primarily on cumulative improvement of the three acquisition shortfalls that result in increased maintenance costs, that can enable development project management to anticipate and prepare for much more cost-effective software maintenance.

REFERENCES

Abts, C., Boehm, B. W. and Clark, E. B. (2000) Cocots: a cots software integration cost model - model overview and preliminary data findings. In *Proceedings ESCOM-SCOPE 2000 Conference*, 325–333.

Akao, Y. (1994) Development history of quality function deployment. *The Customer Driven Approach to Quality Planning and Deployment*, **339**, 90.

Alexandru, C. V., Panichella, S. and Gall, H. C. (2017) Reducing redundancies in multi-revision code analysis. In *2017 IEEE 24th International Conference on Software Analysis, Evolution and Reengineering (SANER)*, 148–159.

Alfayez, R., Behnamghader, P., Srisopha, K. and Boehm, B. (2017) How does contributors involvement influence open source systems. In *2017 IEEE 28th Annual Software Technology Conference (STC)*, 1–8.

— (2018) An exploratory study on the influence of developers in technical debt. In *Proceedings of the International Conference on Technical Debt*.

Anderson, D. J. (2010) *Kanban: successful evolutionary change for your technology business*. Blue Hole Press.

Ayewah, N., Hovemeyer, D., Morgenthaler, J. D., Penix, J. and Pugh, W. (2008) Using static analysis to find bugs. *IEEE Software*, **25**, 22–29.

Behnamghader, P., Alfayez, R., Srisopha, K. and Boehm, B. (2017a) Towards better understanding of software quality evolution through commit-impact analysis. In *2017 IEEE International Conference on Software Quality, Reliability and Security (QRS)*, 251–262.

Behnamghader, P. and Boehm, B. (2018) Towards better understanding of software maintainability evolution. In *2018 Confer- ence on Systems Engineering Research (CSER 2018)*. Charlottesville, USA.

Behnamghader, P., Le, D. M., Garcia, J., Link, D., Shahbazian, A. and Medvidovic, N. (2017b) A large-scale study of architectural evolution in open-source software systems. *Empirical Software Engineering*, **22**, 1146–1193. URL: https://doi.org/10.1007/s10664-016-9466-0.

Behnamghader, P., Meemeng, P., Fostiropoulos, I., Huang, D., Srisopha, K. and Boehm, B. (2018) A scalable and efficient ap- proach for compiling and analyzing commit history. In *Proceedings of the 12th ACM/IEEE International Symposium on Empiri- cal Software Engineering and Measurement*, ESEM '18, 27:1–27:10. New York, NY, USA: ACM. URL: http://doi.acm.org/10.1145/3239235.3239237.

Bernstein, J. I. (1998) *Design methods in the aerospace industry: looking for evidence of set-based practices*. Ph.D. thesis, Massachusetts Institute of Technology.

Boehm, B. and Bhuta, J. (2008) Balancing opportunities and risks in component-based software development. *IEEE software*, **25**.

Boehm, B., Chen, C., Srisopha, K. and Shi, L. (2016) The key roles of maintainability in an ontology for system qualities. *INCOSE International Symposium*, **26**, 2026–2040. URL: http://dx.doi.org/10.1002/j.2334-5837.2016.00278.x.

Boehm, B. W., Madachy, R., Steece, B. et al. (2000) *Software cost estimation with Cocomo II with Cdrom*. Prentice Hall PTR. Borges,

B., Holley, K. and Arsanjani, A. (2004) Delving into service-oriented architecture. *Online: http://www.developer.com/java/data/article.php/3409221/Delving-into-Service-Oriented-Architecture.htm*, 14.

Bosch, J. (2017) *Speed, data, and ecosystems: Excelling in a software-driven world*. CRC Press.

Bullock, J. (2000) Calculating the value of testing from an executive's perspective, software testing is not a capital investment in the physical plant, an acquisition, or another readily accepted business expense. a quality assurance manager describes how to present testing as a business-process investment. *Software Testing and Quality Engineering*, **2**, 56–63.

Campbell, A. (2015) Sonarqube: Open source quality management. *Website: tiny. cc/2q4z9x*.

Chen, C., Alfayez, R., Srisopha, K., Shi, L. and Boehm, B. (2016) Evaluating human-assessed software maintainability metrics. In *Software Engineering and Methodology for Emerging Domains*, 120–132. Springer. Cundiff,

D. (2003) Manufacturing readiness levels (mrl). *Unpublished white paper*.

D'Ambros, M., Gall, H., Lanza, M. and Pinzger, M. (2008) Analysing software repositories to understand software evolution. In *Software evolution*, 37–67. Springer.

Davis, J. and Daniels, R. (2016) *Effective DevOps: building a culture of collaboration, affinity, and tooling at scale*. " O'Reilly Media, Inc.".

Defense Science Board, D. o. D. (2018) Design and acquisition of software for defense systems. *Tech. rep.*, Under Secretary of Defense for Research and Engineering (USD(R&E)). URL: https://apps.dtic.mil/dtic/tr/fulltext/u2/1048883.pdf.

Diamantopoulos, T., Thomopoulos, K. and Symeonidis, A. (2016) Qualboa: reusability-aware recommendations of source code components. In *Proceedings of the 13th International Conference on Mining Software Repositories*, 488–491. ACM.

DoD, U. (2011) Technology readiness assessment (tra) guidance. *Revision posted*, **13**.

Dodaro, G. L. (2015) Government efficiency and effectiveness: Opportunities to reduce fragmentation, overlap, and duplica- tion and achieve other financial benefits. *Tech. rep.*, GOVERNMENT ACCOUNTABILITY OFFICE WASHINGTON DC.

Dyer, R., Nguyen, H. A., Rajan, H. and Nguyen, T. N. (2015) Boa: Ultra-large-scale software repository and source-code mining. *ACM Trans. Softw. Eng. Methodol.*, **25**, 7:1–7:34. URL: http://doi.acm.org/10.1145/2803171.

Elssamadisy, A. and Schalliol, G. (2002) Recognizing and responding to "bad smells" in extreme programming. In *Proceedings of the 24th International Conference on Software Engineering. ICSE 2002*, 617–622.

Ganpati, A., Kalia, A. and Singh, H. (2012) A comparative study of maintainability index of open source software. *Int. J. Emerg. Technol. Adv. Eng*, **2**, 228–230.

Garcia, J., Ivkovic, I. and Medvidovic, N. (2013) A comparative analysis of software architecture recovery techniques. In *Auto- mated Software Engineering (ASE), 2013 IEEE/ACM 28th International Conference on*, 486–496. IEEE.

Glass, R. L. (1997) *Software runaways: monumental software disasters*. Prentice Hall Englewood Cliffs, NJ. Godfrey, M. W.

and Tu, Q. (2000) Evolution in open source software: A case study. In *Software Maintenance, 2000. Proceedings. International Conference on*, 131–142. IEEE.

Gousios, G., Vasilescu, B., Serebrenik, A. and Zaidman, A. (2014) Lean ghtorrent: Github data on demand. In *Proceedings of the 11th Working Conference on Mining Software Repositories*, 384–387. ACM.

Haffa, R. and Datla, A. (2016) Learning from acquisition history. *AEROSPACE AMERICA*, **54**, 30–33.

Haimes, Y. Y. (2012) Systems-based guiding principles for risk modeling, planning, assessment, management, and communica- tion. *Risk Analysis: An International Journal*, **32**, 1451–1467.

INCOSE, D. D. W. (2015) Systems engineering handbook: A guide for system life cycle processes and activities. *Fourth Edition*. Kaur, A. and

Chopra, D. (2018) *GCC-Git Change Classifier for Extraction and Classification of Changes in Software Systems*, 259–267. Singapore: Springer Singapore. URL: https://doi.org/10.1007/978-981-10-5523-2_24.

Koskinen, J. (2009) Software maintenance fundamentals. *Encyclopedia of Software Engineering, P. Laplante, Ed., Taylor & Francis Group*.

Kruchten, P., Nord, R. L. and Ozkaya, I. (2012) Technical debt: From metaphor to theory and practice. *IEEE Software*, **29**, 18–21.

Le, D. M., Behnamghader, P., Garcia, J., Link, D., Shahbazian, A. and Medvidovic, N. (2015) An empirical study of architectural change in open-source software systems. In *Proceedings of the 12th Working Conference on Mining Software Repositories*, 235–245. IEEE Press.

Leffingwell, D. (2007) *Scaling software agility: best practices for large enterprises*. Pearson Education. Li, Q.

(2012) *Value-based, dependency-aware inspection and test prioritization*. University of Southern California.

Malaiya, Y. K., Li, M. N., Bieman, J. M. and Karcich, R. (2002) Software reliability growth with test coverage. *IEEE Transactions on Reliability*, **51**, 420–426.

Mexim, B. and Kessentini, M. (2015) An introduction to modern software quality assurance. *Software quality assurance: in large scale and complex software-intensive systems. Morgan Kaufmann, Waltham*, 19–46.

Parnas, D. L. (1979) Designing software for ease of extension and contraction. *IEEE transactions on software engineering*, 128– 138.

Pinto, G., Torres, W., Fernandes, B., Castor, F. and Barros, R. S. (2015) A large-scale study on the usage of java's concurrent programming constructs. *Journal of Systems and Software*, **106**, 59 – 81. URL: http://www.sciencedirect.com/science/ article/pii/S0164121215000849.

Redman, Q. (2008) Weapon system design using life cycle costs. *Raytheon Presenatation*.

Reinertsten, D. G. (2009) *The principles of product development flow: second generation lean product development*. Celeritas. Royce, W. (1998) Software

project management-a unified approach.

Rozenberg, D., Beschastnikh, I., Kosmale, F., Poser, V., Becker, H., Palyart, M. and Murphy, G. C. (2016) Comparing repositories visually with repograms. In *2016 IEEE/ACM 13th Working Conference on Mining Software Repositories (MSR)*, 109–120.

Sauser, B., Verma, D., Ramirez-Marquez, J. and Gove, R. (2006) From trl to srl: The concept of systems readiness levels. In *Conference on Systems Engineering Research, Los Angeles, CA*, 1–10.

Sauser, B. J. (2007) System maturity metrics for decision support in defense acquisition users guide: Version 1.0. Software & Systems Engineering

Standards Committee, I. C. S. (2011) Iso/iec tr 24748-1: Systems and software engineering– life cycle management–part 1: Guide for life cycle management. *Tech. rep.*, ISO. URL: https://standards.ieee.org/ standard/24748-1-2011.html.

Sokol, F. Z., Aniche, M. F. and Gerosa, M. A. (2013) Metricminer: Supporting researchers in mining software repositories. In *2013 IEEE 13th International Working Conference on Source Code Analysis and Manipulation (SCAM)*, 142–146.

Sundbo, J. and Gallouj, F. (2000) Innovation as a loosely coupled system in services. *International Journal of Services Technology and Management*, **1**,15–36.

Tiwari, N. M., Upadhyaya, G., Nguyen, H. A. and Rajan, H. (2017) Candoia: A platform for building and sharing mining software repositories tools as apps. In *2017 IEEE/ACM 14th International Conference on Mining Software Repositories (MSR)*, 53–63.

Trautsch, F., Herbold, S., Makedonski, P. and Grabowski, J. (2016) Adressing problems with external validity of repository min- ing studies through a smart data platform. In *Proceedings of the 13th International Conference on Mining Software Repositories*, MSR '16, 97–108. New York, NY, USA: ACM. URL: http://doi.acm.org/10.1145/2901739.2901753.

— (2017) Addressing problems with replicability and validity of repository mining studies through a smart data platform. *Em- pirical Software Engineering*. URL: https://doi.org/10.1007/s10664-017-9537-x.

Tufano, M., Palomba, F., Bavota, G., Oliveto, R., Penta, M. D., Lucia, A. D. and Poshyvanyk, D. (2017) When and why your code starts to smell bad (and whether the smells go away). *IEEE Transactions on Software Engineering*, **43**, 1063–1088.

Tzerpos, V. and Holt, R. C. (2000) Acdc: An algorithm for comprehension-driven clustering. In *Proceedings of the Seventh Working Conference on Reverse Engineering (WCRE'00)*, WCRE '00, 258–. Washington, DC, USA: IEEE Computer Society. URL: http: //dl.acm.org/citation.cfm?id=832307.837118.

Vitasek, K. and Ledyard, M. (2013) *Vested outsourcing: five rules that will transform outsourcing*. Springer.

U. Virginia Final Report

PSU's early SERC contributions included research on Product Architectures, Design, & Manufacturing for Operational Responsiveness, particularly in the area of 3D bringing of unmanned aerospace systems (UAS). This included participation in the "SERC Workshop: Managing Acquisition and Program Risk" held on December 13, 2017. Role was two-fold in this workshop: (1) attend, discuss, and engage with members of the community having a shared interest in risk (e.g., strategies, approaches), and (2) to report on PSU research in the area, which included having designed, built, and flown 3D printed UAS, including instrumentation the design process, gathering metrics on the design evolution to include total man-hours, models used, and design iterations. PSU also prepared and submitted for publication an invited

Wayne State Final Report

System Engineering (SE) for Acquisition Qualities and Tradeoffs in Autonomy Enabled Military Systems (AEMS) with Machine Learning (ML) Applications for Manned-Unmanned Teaming (MUM-T)

Gary Witus, Wayne State University, Detroit, Michigan

16 DEDICATION

This report is dedicated to Gen. William E. DePuy (ret., 1919-1992). He worked with me once a week for three years. He said "Your project is stupid. I am working with you to teach you to think like a commander." He gave four priorities

1. Enhance the ability of the Commander to impose his will on the battlefield – limit the threat options and force them to reveal their intentions, know where your forces are and their status
2. Protect and enable the troops
3. Train and select leaders
4. Don't waste the taxpayers' money

General DePuy was the toughest, scrawniest, and smartest person I ever knew. At age 23 he led a landing craft onto the beach at Normandy. He fought through all sorts of combat and leadership changes across Europe. He led the Allied Advance Guard Regiment into Germany (held up only by political agreements with the Soviet Union). After WWII, he worked Soviet and China Intelligence desks. He took over the 1st Infantry Division in Vietnam – then led reforming the Army after the Vietnam war. He formulated the Active Defense doctrine for the strategic defense of the Fulda Gap, which evolved into the Air-Land Battle doctrine under his protégé, General Starry. He was the first Commander of the Army training and doctrine command (TRADOC). He sponsored advanced military Operations Research at the University of Michigan, spun off as a company private company, Vector Research, where I reported to General DePuy. This report is an attempt to apply what I learned from him through Systems Engineering for Autonomy-Enabled Military Systems for Manned-Unmanned Teaming.

Introduction

This report addresses research, development and application of Systems Engineering methods, procedures and tools for Autonomy Enabled Military Systems (AEMS), with particular emphasis on development of integrated Machine Learning (ML) applications, and use of AEMS in Manned-Unmanned Teaming (MUM-T). The report addresses AEMS and ML acquisition qualities and tradeoff decisions. It is intended to provide practical and relevant insights for the development and acquisition of timely, safe, secure and effective AEMS, and ML for AEMS, that can be integrated effectively into the Command and Control (C2) of military operations.

This report is an attempt to provide practical and relevant insights for SE for acquisition of quality AEMS, and SE for developing ML application for AEMS. Two key issues addressed are the functional organization of ML applications, and the role of SE in developing training programs for ML. This report articulates acquisition qualities for AEMS and ML, and provides a functional decomposition into independently trainable and composable ML modules. A key insight is the tradeoff between the time and cost of obtaining training data versus the

assurance of safety and effectiveness over a wide and robust Operational Domain. This report articulates the importance of training data over the Operational Domain, and a strategy for SE combining incremental training and promotion in ML application development – a standard strategy for training animals and humans. A third key insight is the need for accelerated SE in the arms race with near-peer threats.

Machine Learning (ML)

ML is a branch of Artificial Intelligence (AI) that uses training cases to generate executable code. ML is proving to be an effective approach for real-world problems for which there are no known algorithmic solutions, or where algorithmic solutions work only under idealized theoretical conditions. Autonomous driving applications are preeminent examples.

There are three ML approaches: supervised learning, unsupervised learning, and reinforcement learning. In supervised learning, ML applications are created by training an ML algorithm with labeled training data. Supervised learning is used when safety and reliable behavior are needed. Unsupervised learning and reinforcement learning can have unpredictable results and emergent behaviors.

ML for AEMS should rely on supervised learning with labeled training data for safety and reliability.

Systems Engineering (SE) in Development: Capability versus Acquisition & Fielding Time and Cost

This report addresses SE in the development of ML for AEMS in military operations. It does not address acquisition strategy or the technologies for ML. It focuses on two core SE cost-benefit tradeoffs. One is the cost of obtaining adequately robust training data versus the range of conditions of the Operational Domain (OD) and the Operational Tasks/Functions for the AEMS. The second is the time to field and integrate AEMS into unit operations.

The AEMS Arms Race and Tradeoffs

The U.S. military is engaged in an arms race with near-peer threats to acquire and field affordable, safe, effective and secure AEMS, in greater numbers, over a broader Operational Domain (OD), with greater capability and competency. Near-peer adversaries with less concern for safety and less concern for advanced capabilities will have an advantage is rapidly fielding AEMS. There is a tradeoff between the pace of development and fielding versus safety assurance and advanced capabilities.

Near-peer threats are fielding AEMS. System acquisition has two key considerations: (1) creating a combat advantage, and (2) sustaining the combat advantage. In the domain of AEMS, near-peer threats put different value on equipment item *safety assurance vs combat impact*, and time-to-field versus robustness from intelligent behaviors.

The US competes against near-peer threats with different acquisition and fielding strategies - “fast and loose” versus “slow and cautious”, fielding primitive systems with limited capability quickly versus developing advanced and sustainable systems. Near-peer threats may value inexpensive, rapidly field-able lethality, mass and disposability for “simple” tasks/functions.

The difference between automatic and intelligent systems is that automatic systems execute actions, while intelligent systems accomplish objectives. This is a subtle, but important difference, from the perspective of AEMS design. AEMS that execute actions mechanically execute automatic behaviors. What they will do is known in advance. What they will accomplish is uncertain. Intelligent systems are given objectives, constraints, and capabilities.

Exactly what they will do to accomplish the objectives as they adapt prior behaviors and plans as the situation unfolds is unknown, and the results are uncertain.
Intelligent systems might or might not be more effective or more robust than automatic autonomous systems.

AEMS Development Strategies

At the present time, AEMS research and development consists of DARPA-type demonstration projects and Science and Technology (S&T) demonstrators and demonstrations. Demonstrators and demonstrations are valuable to test technologies, explore integration issues and approaches, and provide evidence of some degree of safety, operability, and effectiveness. *Demonstrators and demonstrations are not a substitute for Systems Engineering*. An AEMS should be safe and secure (with some degree of assurance) over a broad and relevant Operational Domain (OD) with competency and capability to perform valuable roles, tasks, and functions. The job of SE is to translate these into system concepts and requirements for competitive acquisition. Demonstrators and demonstrations promote technology development and integration, but are not a substitute for Systems Engineering.
For traditional systems, such as a tank or tactical truck, in which the role, tasks, and functions and the OD are well understood, and for which the military has standard models for performance capabilities and requirements, *competitive prototyping* is a practical strategy compare design concepts and refine requirements. This is not the case for AEMS.

SE for ML Applications

SE for ML applications is terra nova. The technology to train multi-layer neural networks ("Deep Learning", or DL) is relatively new, and remains an active area of research. Systems Engineering research is needed to develop methods, procedures and tools to *apply* ML in general, and DL in particular, for use in Defense systems.

Autonomy versus Direct Control. Currently drones are used in MUM-T with helicopters for surveillance and reconnaissance. However the drones are not making independent decisions or taking independent actions. Similarly, uninhabited ground vehicles have been used under remote control or teleoperation. (Remote control refers to direct control from an overwatch position. Teleoperation refers to control from a video feed).
Direct control systems can be acquired and fielded quickly without the need to develop and assure safety of machine intelligence. There are multiple drawbacks and limitations.

- It requires a dedicated operator, thus preventing the force multiplier effect of one-directs-many
- Direct control has been difficult and clumsy (at least for ground vehicles), largely because the hardware did not provide the operator with perception capabilities that are needed for off-road driving in adverse terrain. The systems were cobbled-together technologies without adequate SE for the MUM-T interface.
- Range and physical conditions are limiting
- When radio communication is used there is the risk of jamming, spoofing, and intercept
- Anticipating secure, unjammable, uninterruptable communications technology adds delay and risk from dependency on other R&D programs

SE Challenges of AEMS. AEMS perform roles, tasks and functions that were traditionally performed by selected and trained warfighters, without real-time direct control, but the command of a superior element commander and in coordination with adjacent elements that may be manned or unmanned, using military command and control practices and procedures. This raises several SE challenges:

- Task/function decomposition and allocation to (a) autonomous operation, (b) interaction between man and machine, and (c) human
- Specifying the Machine Learning (ML) applications, and organizing them into independently trainable, composable ML modules.

The challenge of ML applications is the high cost of obtaining sufficient quantity of diverse and realistic training data over the span of the Operational Domain. To reduce the quantity of training data needed, and to assure capability of elements of autonomy, this report proposes a SE approach for ML applications in AEMS and organizing them into independently trainable, composable ML modules. It has three elements

- A taxonomy of independently trainable functional elements of autonomy
- An approach to decomposing the Operational Domain into independently trainable regions
- An incremental training workflow for robust composable capabilities and parsimonious data needs

Acceptance of AEMS. AEMS must be accepted by the operational forces in order to make a contribution. Historical examples illustrate some of the acceptance issues and tradeoffs

- *Automatic transmission*. Today automatic transmission is widely accepted, except for extreme performance driving. The benefits include reduced driver training time, reduced transmission damage and maintenance costs, and reduced fuel consumption. The tradeoff is a limitation in ability to execute extreme maneuvers.
- *Anti-Lock Brakes*. Today it is accepted, and insurance companies give a discount for it. An automatic system takes of control from the driver to provide maximum braking without skid (steering is uncontrollable in a skid). It took years to be accepted, as it had to be proven out over a wide range of roadway, weather and traffic conditions (the Operational Domain). Even then it had an unexpected consequence: drivers became more aggressive.
- *GPS Navigation*. GPS navigation relieves cognitive burden: the driver does not need to know, just where they want to be, and it tells them what to do. Of course sometimes it is wrong, or unaware of immediate local conditions. Then the driver is lost.
- *Search Engines*. Search engines are an example of cyber MUM-T. User types in a few words, and presto, some AI algorithm turns up all sorts of relevant sites. It would even build a model of the interests of individual users and the relationship of their interests to those of others. It has been so useful it was quickly accepted. The

system sold data to advertisers in order to provide free service, allowing advertisers to inject ads and tracking. The tradeoff is effectiveness versus security.

Value Propositions for AEMS. Some of major value propositions for AEMS include

- *Force protection* – using uninhabited systems in place manned systems in high-risk situations
- *Force multiplication* – swarming large numbers of individual systems jointly executing a mission
- *Self-sacrifice* – AEMS can be commanded to fight to extinction, beyond what troops could be expected to do
- *Endurance, diligence and responsiveness* beyond human capabilities
- *Mobility* to go when manned systems cannot
- *Reduced crew workload* by off-loading tasks and functions

Mission Engineering versus Systems Engineering for AEMS. To deliver combat value AEMS need to be competent and capable of performing relevant operational roles, tasks and functions over a broad and relevant Operational Domain (OD). It is the job of Mission Engineering to select the operational roles, tasks and functions and the OD. It is the job of Systems Engineering to decompose the roles, tasks and functions, and the OD into independently trainable and composable modules.

Operational Models for AEMS.

- Crew augmentation
- Swarms of drones and/or autonomous ground vehicles
- Individual autonomous vehicles working paired with a manned element (a "scout dog")
- Virtual crew for optionally manned vehicles

AEMS can be developed with systematic SE, or with an ad hoc, opportunistic build-and-try approach, or a hybrid of the two. The challenge for SE in the arms race is accelerate, not retard, development and fielding. Threats that place a higher value on impact versus safety than the U.S. can field systems faster. Threats that adopt the deploy-and-replace strategy versus the deploy-and-upgrade strategy may or may not be able to field effective systems faster.

This report is oriented towards Army operations, aviation and ground units. AEMS are likely to have roles in cyber, satellite, missile, and air domains. While the application case was Army operations, the findings and results related to SE in defining the extent of autonomy and system tradeoffs apply in other domain. The findings and results related to SE in the development of Machine Learning (ML) applications for AEMS are broadly applicable.

Challenges in AEMS for MUM-T. Challenges include

- Task/function allocation between autonomous operation of the AEMS and its human commander or handler

- The interaction model with adjacent elements (no interaction, pre-mission coordination measures, mutual adjustment without explicit communication, adjustment with explicit communication)
- The interaction model with the remote handler
- Risks from dependence on non-existing developmental technologies (unjammable GPS-like position location; communications with high-bandwidth, unjammable, not susceptible to threat Direction Finding, intercept or spoofing, and not blocked in urban terrain, tunnels, etc.; low-probability of detection radar and lidar; etc.)

SE for ML Applications in AEMS

- How to formulate the role of ML in AEMS
- How to decompose complex ML applications into independently *trainable, and composable* ML modules (MLM) that minimize the time and cost of acquiring the training data
- How to organize the training program for a MLM to maximize robustness, extensibility, transferability, adaptability, etc. while minimizing training data acquisition time and cost
- How to obtain sufficient, organized training data

Field Fast versus Field Safe

In the arms race, there is a contention between "dumb is good enough" and "smarter is better". There is a contention between "field fast" and "field safe". There is a contention in how safety is considered: (1) safety is in execution of the mission, at whatever cost to the unit or civilians, because accomplishing the mission protects other units and enables them; (2) safety means not taking an action that might injure an adjacent element or neutral. These strategies are completely opposite. The solution principle of "execute the action" is a dynamic programming formulation. The solution principle of "do no harm but be effective" is a local optimization formulation. One is about executing actions, the other is about achieving goals within constraints. In reality, things are blended and adjusted. But it also changes the military value of quantity vs capability. It is also related to the C2 approach, devolution of command authority. It is the difference between "follow orders" and "exercise initiative." This is an element of the strategic thinking difference between U.S. and potential threats. AEMS technologies and concepts bump up against this tradeoff.

The Curse of Dimensionality in ML

The curse of dimensionality is that more variable conditions and complex responses require *exponentially* more training data for an ML application. Decomposing the role of ML into incrementally and independently trainable and composable modules addresses the curse of dimensionality. This is the place of SE in ML development for AEMS.

Collecting training data for ML is *expensive*. Live, virtual, and constructive simulations are expensive, each with increasingly questionable realism. The variety of potential situations vastly exceeds the ability to generate training data, labeled with correct response.

Decomposition of the ML applications into elementary tasks/functions, and first training for benign conditions, then expanding the range of conditions and composition of tasks/functions is and has been the procedure for training animals and students.
For humans and animals, training has traditionally been coupled with selection. Some dog's don't hunt. Training is a non-deterministic process. Training is not programming. Many students are trained. The best are selected and promoted. This is a new evolution of the model for ML in AEMS. Current ML practice is to train a single model, versus the Set-Based Design approach of training many alternative models, rejecting the worst and promoting the best.
The science of SE for ML is primitive. The science of decomposing complex cognitive functions into independently trainable and composable modules is primitive.

Value Propositions and Tradeoffs for AEMS

It is simple: protect the force, enhance their capability, put more "eyes, teeth and bodies on legs or wings" onto the battlefield. AEMS possess diligence, endurance, responsiveness, and willingness for self-sacrifice. They can go where crewed systems cannot, and can potentially as a swarm with mass effects. Their minds do not wander, they do not tire, they are prompt to react. Uninhabited systems can be sacrificed, because their cost is measured in dollars not lives. The value of executing the mission versus self-preservation is a Command and Control (C2) parameter.
Military units are expected to be able to recognize and react to unanticipated contingencies and opportunities. *From an SE perspective, two ML models are needed for AEMS: the first to select and specify the best response, and the second to determine whether or not to trust the first.* This is outside the common scope of ML applications
ML models are trained to answer a question. SE for ML applications need to address how the problem is structured, how the ML models were organized, trained and composed into action-worthy AEMS.
An ML application that acts with certainty despite uncertainty is a risk. ML models produce an answer to act on. The actually produce values over all possible answers, and chooses the answer with the highest value. In principle, the AEMS could determine that a situation is ambiguous because conflicting answers have near equal value. Then, if communications is available, the AEMS could ask the human handler to resolve the conflict. This is a referred to as mixed-initiative supervisory control. The tradeoff is risk of reliance on communications and burden on the handler versus appropriate action in an ambiguous situation.
Some of the AEMS value proposition tradeoffs include

- Force protection (removing troops from harms way) vs mission capability of the AEMS
- Autonomous swarms released under prior mission command vs individual items of equipment responding to and closely teamed with an individual handler
- Autonomous behaviors versus supervised action dependent on communications and the attention of the handler, i.e., the force multiplier ratio of uninhabited systems to handlers
- Time-to-field vs capability and growth potential

- Extent of the Operational Domain (OD) and robustness of the system versus time and cost of obtaining training data for adequate assurance of safety and effectiveness

In Combat, Nothing Is Safe and Nothing Is Certain

AEMS must balance safety versus effectiveness. It is a classic multi-scale perspective problem with no known solution. Suppose we arm and enable swarms of autonomous agents (they could be crewed systems or automatic). Sometimes they will "make mistakes" and hit friendlies or civilians. This has always happened in combat. There is collateral damage. Reluctance to fire may cause greater harm. To allow a threat to survive may cause greater harm. These are non-linear propositions. If the threat is allowed to take the high ground, U.S. forces will suffer losses that may be too much to move forward, or will march assets into a killing zone.

Individual safety and group safety are different considerations. Accomplishing the immediate task, and retaining assets and capabilities for a next task are different problems.

Commanders have dealt with this in the past: Hold at all costs, breakoff if xyz, *"Cry 'Havoc' and Let Slip the Dogs of War"* (from Shakespeare, but the command was earlier). This is the tradeoff that the military has always faced, but the U.S. system has not acknowledged or exploited. It is a leverage point.

"*Havoc!*" has been a standard military command since 1300. It means that elements are released to do what they have been trained to do, adapting to the situation, without any further guidance or control, assuming each will watch each others' back as we wish them to watch ours.

Orchestration of team plays is good. But when the conditions are uncertain, when the threat is adaptive, when things are happening too fast – providing intel and waiting for instructions means waiting while bullets are coming – that is the purpose of training: so that we will all do the best thing as part of a unit without explicit coordination, because we count on each other.

Beware of Simulations

Live, virtual and constructive (LVC) simulations have a role in generating training case data. They also have systemic risks. Are the simulations sufficiently realistic and robust, or are we "drinking our own bath water"? If LVC simulations are to be the source of training data, SE for the target system should encompass the simulations used to produce the training data.

Live simulations are field trials. Virtual simulations are man-in-the-loop trials. Constructive simulations are computer algorithm trials. The cost and realism both decrease at teach stage.

SE parameters of the simulation address

- The range of conditions that can be controlled and simulated
- Realism of the dynamics of modeled processes and factors
- Realism of the variability of uncontrolled factors in the real world not explicitly modeled in the simulation
- Unstated or hidden assumptions

All models are wrong. Some are useful. But what does it take to be useful to generate training data for operational systems?

The parameters of the simulation limit what can be learned. We can only learn about the factors built into the simulation model.

Historically, these models and exercises have been built to evaluate systems. Using LVC simulations to generate training cases is a different use. The Army Materiel Systems Analysis Agency (AMSAA) validates models and simulations *for a particular use*. Validating simulations to generate training data for operational systems is a novel use.

Computer game programming technologies are good at making players feel they are in a real world. Creating physically realistic input for sensor processing to detect targets discriminated from clutter (especially in an adversarial world of camouflage, concealment and deception – CCD – is another matter entirely.).

Systems Engineering to develop requirements for LVC simulations to produce training data for AEMS ML applications is an open question.

Types of AEMS

AEMS can exist in many forms, from embedded intelligence for crew augmentation to robotic scout dogs and drones to swarming uninhabited systems to virtual crew operating uninhabited vehicles.

While swarming mobile physical systems may be an enticing vision, crew augmentation, specifically autonomy enabled intelligence analysis of data feeds from remote sensing, may be the higher near-term priority. Currently, the approximate rule-of-thumb is that it takes three remote crew per day to pilot a drone, and 20 or more per day to review the data feeds. Of course there are many other types of surveillance and reconnaissance systems – satellites, unattended ground sensors, sonobouys, etc.

Ethics of the Kill Chain for AEMS

Fire and maneuver are the two legs of ground combat - fire to be able to maneuver, maneuver to be able to fire.

Indirect fire delivers area effects munitions, though the munitions themselves may be terminally guided or sensor fused. Direct fire aims precision fire at specific targets, but direct fire is also used for suppressive fire – to shoot at any possible threat or hiding place without having an identified target.

The Forward Observer (FO) that calls in a target location for indirect fire is part of the kill chain. The Automatic Target Cuing system that locates a signature of a shooter that seems to be firing on friendly forces is part of the kill chain. The Automatic Target Cuing system that locates possible military vehicles or positions within a free fire zone is part of the kill chain. Terminally guided and sensor fused munitions, e.g., Family of Scatterable Mines and SKEET munitions, are part of the kill chain.

In all such cases, the actions are taken within the parameters of the rules of engagement and fire coordination measures established by the superior unit commander. A human may or may not be involved in an individual trigger-pull or target acquisition. Nonetheless, there is human command and control over weapon release.

The ethics question for AEMS is whether there are adequate assurances that AEMS will perform its operations within standard rules of engagement and fire coordination measures. The tradeoff is between high assurance and delayed action versus responsiveness for pre-emptive action that protects friendly forces against threats.

Military operations cannot wait for higher command to approve each trigger pull. The U.S. military has established C2 practices for rules of engagement, free-fire zones, and target Identification Friend of Foe (IFF). As long as an AEMS conforms to established practices and procedures, there is no ethical dilemma or difference between instructions given to PFC Smith or an AEMS.

Integration of AEMS and MUM-T into Operations

AEMS and MUM-T require design to integrate into the command and control organization and operation of military units. Soviet-era tactics and doctrine emphasized assigning *actions* to units, replacing them with a new unit when exhausted, and reattaching any remaining elements to other units. U.S. training and doctrine emphasizes a pre-coordinated Course of Action with *objectives* and coordination measures allowing freedom to exploit opportunities and respond to initiatives while retaining the effects of coordination with minimal explicit real-time coordination and control.

Actions are things to do. Objectives are outcomes to achieve. This is a different combat operation decomposition strategy. "Actions to execute" may be more compatible with AEV and MUM-T than "objectives to achieve." It is easier to design systems to "do things" than to "achieve results." Achieving results requires greater degree of autonomous intelligence versus automatic action.

Instructional Materials vs Training Cases

U.S. training materials characterize general cases and responses. Developing systematic training cases needs to also know the diversity in the specifics. Generating training cases needs to sample the diversity of specifics. Training materials lack systematic specification of the dimensions of diversity, the ranges of values, and correlations. That is exactly what is needed to generate a set of diverse training cases to train MLM.

Decomposition of ML Applications into Independently Trainable and Composable ML Modules

Decomposing the broad ML application into independently trainable and composable ML modules

- Vastly decreases the total cost of training data – training many narrowly focused functions in coherent regions of the Operational Domain will take less data than developing broader ML applications, and can be pursued incrementally
- Provides module-level assurance of robustness of individual functions over the Operational Domain

The SE challenge is decomposition into independently trainable and composable modules. Many small ML modules for simple tasks are more trainable and diagnosable than a few ML modules for complex tasks. Three elements are needed

- A functional decomposition of the elements of autonomy
- A principled decomposition of the Operational Domain into regions for localized training
- A training program that begins with training basic tasks/functions in benign/simple conditions and incrementally expands to more ambiguous conditions and complex tasks

ML Problem Formulation: Discrete Decision versus Continuous Control

ML applications are commonly divided into classification problems (discrete decisions) and regression problems (continuous control). In reality, there is a continuum. A classification formulation can solve a regression problem by interpolation. A regression formulation can solve a classification problem by binning.

In general, ML formulations produce a distribution of weights over set of discrete classifications.

Composition is an Unresolved Challenge

The elements of autonomy (addressed later) are sequential, but the output of one module must match the input to the next.

Decomposing the Operational Domain and composing modules or training cases across regions is multi-resolution and ambiguous. Adjacent regions of the Operational Domain may have similar correct behaviors and ML models – similar, but not identical. At some point, further subdividing the Operational Domain increases the cost of training data because it does not exploit cross-training. But failure to subdivide – excessive aggregation – creates confusion that vastly increases the training data requirements.

In ML training, as in any human or animal training, early lessons are the most pervasive and persistent. Later lessons build on earlier lessons. Start training elementary tasks/functions under standard/benign conditions, then incrementally expand the domain of conditions and complexity of the task involving combinations of elementary tasks/functions. This is standard training practice for humans and animals, but has not been pursued for ML. For ML, a lesson is the set of training cases, not necessarily the model produced by training.

Requirements for Training Data

Collecting labeled training data is expensive. The training data needs to adequately cover the Operational Domain and range of decisions for which the ML model is responsible. Some of the factors affecting the training data requirements include

- The number of weights in the model to be computed from training data
- The range of possible correct answers over the Operational Domain – cases are needed that adequately sample all the correct solutions
- The extent of the Operational Domain – cases are needed that adequately sample all the regions of the Operational Domain in order to assure, with some confidence, correct response over the Operational Domain
- The granularity or resolution of the output states, i.e., the precision required
- Uncertainty (inaccuracy) in the input data
- Ambiguity in the training data, when cases with the same inputs have different correct output labels in different cases, i.e., when the input data does not represent all the relevant real-world factors
- Errors in labeling the training data, i.e., when the label (the "correct" response) for a case is incorrect
- Correlations between training cases, i.e., when the training cases oversample some region of the Operational Domain

There are no generally accepted models to generate a training data collection plan, calculate how much training data is needed, and tradeoff the cost of the training data versus the quality of the model.

Systems Engineering to the Rescue – ML Application Organization to Minimize Training Data Requirements

This is the crux of the research findings – how organize the ML for AEMS application to minimize the time and cost of obtaining adequate training case data for safe and effective operation over the Operational Domain. The claims or hypotheses are

- Organizing the ML application into independently trainable and composable modules with result in vastly less training data needs
- Organizing the Operational Domain into regions with consistent response and training ML modules for each region
    - Regions are different if the correct response is different
    - Regions are not different if the differences are not controllable for data collection and labeling
    - Regions are ambiguous if they cannot be discriminated by sensory input, and require probing or experimentation
- Organizing the training program – sequential training – first for basic tasks/functions and benign/simple regions of the Operational Domain, then incrementally increasing ambiguity
- Many ML modules for simple decisions, e.g., binary or locally linear, then composing the decisions is a more robust, effective and parsimonious strategy than integral large problems. Furthermore, is allows localization of errors.

Work Breakdown Structure for AEMS

DoD acquisition is based on a Work Breakdown Structure (WBS). The WBS is how tasks are justified and bid. Different types of systems (e.g., ground vehicles versus satellites) have different WBS. The current WBS model was not designed to acquire AEMS that can transfer human tasks/functions to machines.

SE is needed to generate a WBS for the machine intelligence in AEMS. The following functional elements of autonomy breakdown is an initial step.

Functional Elements of AEMS

Humans and animals begin with some basic sensory and actuation faculties "programmed" through evolution, including the ability to learn certain tasks and functions. SE is needed in determining the role of ML in light of the engineered AESM features. An AEMS may include the following functions:

- **Sensing and signal processing**. This involves sensory hardware, firmware and software to capture and process raw signals. It includes sensors related to the external world, to the internal state of the AEMS, and to other entities in the team or opposition. Commonly, signal processing is explicitly programmed, but it is possible that isolated ML as a non-linear multi-dimensional regression model can be involved in signal conditioning and sensor fusion. This use of ML is more the domain

of the functional specialist than the AEMS systems engineer. Sensing and signal processing feeds perception.

- **Actuation and control**. This refers to the hardware, firmware and software to execute physical actions. Actuation can, and often does, involve local control loops. Isolated ML, as a regression model, is commonly embedded in actuator control loops. Anti-skid brake systems and gun-turret control are examples. This use of ML is more the domain of the functional specialist than the AEMS systems engineer.
- **Perception**. This refers to the classification and measurement of entities, events, and relationships. *ML is emerging as the central technology for perception*. The SE challenge is to identify what needs to be entities, events, and relationships need to be detected/classified, and what about them needs to be measured. This imposes requirements on sensing and signal processing, and is limited by sensing and signal processing. A challenge is to develop *transferable* ML when new sensing and signal processing is employed. Perception is a real-time process which should include a quantitative assessment in the confidence of the classification and accuracy of measurements. Perception feeds situational awareness.
- **Situational awareness**. This refers to the model of the world, as related to the model of the mission and Course of Action. It is a distillation of the history of perception into a consolidated picture of the current situation. Ideally it captures both the "best guess" as to the situation and competing interpretations, as well as projected contingencies and opportunities. The specifics of the situational awareness model are developed in the mission planning process.
- **Planning**. In traditional AI "planning" refers to formulating a sequence of tasks, with possible branch points, to achieve an objective or end state. In the traditional model, planning means decomposing a task into composable subtasks. Planning is part of mission rehearsal.
- **Behavior patterns**. Behavior patterns are the things the AEMS knows how to do without further instruction, including how to adapt them to the immediate situation. In ground combat these are called battle drills and crew drills.
- **Decision making**. Decision making is required when there are discrete mutually-exclusive alternatives, commonly as a result of (a) conflicting objectives, and (b) incomplete information.
- **Explicit interaction**. Explicit interaction and communication with superior and adjacent elements is needed to receive instructions, report and receive information, request guidance, etc.
- **Experimentation**. Experimentation refers to actions to obtain information to resolve ambiguity and reduce uncertainty in the situation awareness world model. Experimentation actions do not directly contribute to accomplishing physical goals.

- **Learning**. Learning refers to creating new behavior patterns and adapting generic behavior patterns, just as units tailor Standard Operating Procedures (SOP) to create the Tactics, Techniques and Procedures (TTP) for the theater of operations.

Functional Elements of AEMS

Humans and animals begin with some basic sensory and actuation faculties "programmed" through evolution, including the ability to learn certain tasks and functions. SE is needed in determining the role of ML in light of the engineered AESM features. An AEMS may include the following functions:

- **Sensing and signal processing**. This involves sensory hardware, firmware and software to capture and process raw signals. It includes sensors related to the external world, to the internal state of the AEMS, and to other entities in the team or opposition. Commonly, signal processing is explicitly programmed, but it is possible that isolated ML as a non-linear multi-dimensional regression model can be involved in signal conditioning and sensor fusion. This use of ML is more the domain of the functional specialist than the AEMS systems engineer. Sensing and signal processing feeds perception.
- **Actuation and control**. This refers to the hardware, firmware and software to execute physical actions. Actuation can, and often does, involve local control loops. Isolated ML, as a regression model, is commonly embedded in actuator control loops. Anti-skid brake systems and gun-turret control are examples. This use of ML is more the domain of the functional specialist than the AEMS systems engineer.
- **Perception**. This refers to the classification and measurement of entities, events, and relationships. *ML is emerging as the central technology for perception*. The SE challenge is to identify what needs to be entities, events, and relationships need to be detected/classified, and what about them needs to be measured. This imposes requirements on sensing and signal processing, and is limited by sensing and signal processing. A challenge is to develop *transferable* ML when new sensing and signal processing is employed. Perception is a real-time process which should include a quantitative assessment in the confidence of the classification and accuracy of measurements. Perception feeds situational awareness.
- **Situational awareness**. This refers to the model of the world, as related to the model of the mission and Course of Action. It is a distillation of the history of perception into a consolidated picture of the current situation. Ideally it captures both the "best guess" as to the situation and competing interpretations, as well as projected contingencies and opportunities. The specifics of the situational awareness model are developed in the mission planning process.
- **Planning**. In traditional AI "planning" refers to formulating a sequence of tasks, with possible branch points, to achieve an objective or end state. In the traditional

model, planning means decomposing a task into composable subtasks. Planning is part of mission rehersal.

- **Behavior patterns**. Behavior patterns are the things the AEMS knows how to do without further instruction, including how to adapt them to the immediate situation. In ground combat these are called battle drills and crew drills.
- **Decision making**. Decision making is required when there are discrete mutually-exclusive alternatives, commonly as a result of (a) conflicting objectives, and (b) incomplete information.
- **Explicit interaction**. Explicit interaction and communication with superior and adjacent elements is needed to receive instructions, report and receive information, request guidance, etc.
- **Experimentation**. Experimentation refers to actions to obtain information to resolve ambiguity and reduce uncertainty in the situation awareness world model. Experimentation actions do not directly contribute to accomplishing physical goals.
- **Learning**. Learning refers to creating new behavior patterns and adapting generic behavior patterns, just as units tailor Standard Operating Procedures (SOP) to create the Tactics, Techniques and Procedures (TTP) for the theater of operations.

Insufficient Quantity And Diversity Of Training Data

Computer Scientists commonly speak of "overfitting" and "underfitting" as the error mechanisms. Overfitting refers to overuse of the same data leading to learning spurious and irrelevant relationships. Underfitting refers to underuse of the data leading to fail to learn significant and relevant relationships.

For some assurance of correctness in AEMS applications over an Operational Domain, the data collection plan needs to cover the Operational Domain. In survey research, this is called "stratified sampling." It means that there are adequate samples over all distinctive regions of the Operational Domain. Some regions may be unlikely, but with severe consequences for incorrect response.

Stratified sampling is needed to assure safety and effectiveness over the Operational Domain.

Decomposing the Operational Domain

- Simpler situations – a narrower scope of the OD for training – means that less training data is needed. There are currently no algorithms to calculate the amount of training data needed as a function of the scope of the OD, the network complexity in the model, and the range of the output space. This is an important area of research.
- Simpler situations – narrower scope of the OD for training – means that it is easier to verify and validate the model, incrementally, to assure performance within a narrower domain, and to have structure to understand regions of the OD where the model has been verified and validated
- Acquiring labeled data, or acquiring data then labeling it is expensive. Acquiring real-world test data under controlled or even just known conditions is expensive.

Requirements for live, virtual or constructive simulations that have sufficient fidelity for reliable training for "wild type" situations is expensive. Experimental design to generate test data requires a model of the OD for stratified sampling ("stratified sampling" is a term from survey research that refers to oversampling cases from edge conditions, vice randomly sampling over the probability distribution of conditions.)

- Decomposing the complex autonomous behavior into composable elements is a challenge. An important contribution of this research is to composable "Elements of Autonomy" – i.e., task elements that can be defined and trained separately and composed
- We also need to decompose the OD into regions for incremental training – starting with basic behaviors in benign conditions. The claim is that this incremental approach will require less training data.
- We need to decompose the OD into regions for separate training where a correct behavior or answer in one region is substantially different than another

SE for ML Should Learn From Training Animals

Animals – dogs, horses, birds – are all trainable to work with humans, to work with each other, and to work with other animals. Animals are trained not taught. They do not learn from books. They are trained by instruction with operant conditioning. Humans, with ability for assimilating and applying abstractions, still are trained by example.

Animals are a non-human intelligence that can be trained for teamwork with humans and are a good study for training non-human intelligences. Engineers who model swarming behavior should study animals and animal training. A scout dog or hunting dog has speed and sense beyond human, learns a job, and learns when to look back for direction. The alpha leader of a wolf pack leads from the rear. The young dogs go forward and spread out, but do not go too far afield. The pack follows ahead, but the pack leader leads from behind. When one yellow-jacket is crushed, it releases a pheromone that tells all the other yellow-jackets "attack here." A football team can study plays, but learning how to execute them requires practice against adaptive adversaries.

Operant Conditioning – Positive and Negative Reinforcement

Operant conditioning is a training model for animals. It combines positive and negative reinforcement. Incremental positive rewards are given for desired response. Positive rewards can have some delay. Some practitioners find that constant encouragement is the best way to train complex behaviors. Negative rewards – punishment – must be delivered immediately at that onset of the undesired behavior so that the inappropriate behavior is clearly recognized.

Training strategies for ML applications are more simplistic. ML training only uses positive reward, or no reward. Operant conditioning with positive and negative reinforcement teaches what to do and what not to do. It teaches how to behave with positive reinforcement, and how not to behave for safety with negative reinforcement.

This strategy for an "inner loop" promoting desired behavior, and an outer loop preventing unsafe or unacceptable behavior is outside the current ML model.

Lessons for Developing ML Applications from Training Animals and People. The ML research community has focused on training ML models, but has not incorporated competition among models in the application development process. Developing scout dogs and leaders combine training with selection, and promotion for further training. Some dogs hunt, some don't. Those that are able to perform and adapt are selected and promoted for further training. 50-percent of scout dog candidates wash out. 50-percent of troops wash out at each level of promotion. Creating variation and subsequent selection complements training.

Training and selection are used in combination for developing non-deterministic systems. ML are non-deterministic. What an ML learns and its future adaptability depends on the order of presentation of training cases. Early cases are more influential. The ML research community recognizes this and has some techniques to mitigate variation.

They have not recognized that variation can be desirable. An alternative to trying to build the best model (a point solution), a *Set-Based Design* strategy would be to generate multiple models, discard poor performers, and continue to develop the remainder. This is a more effective SE strategy to develop effective systems with training limited data.

Incremental Training

- Train decomposed MLM for elementary tasks/functions in nominal/benign situations
- Select those with competency, train in more adverse/uncertain situations
- Train an higher-level module to compose MLM for more complex tasks/functions

The Mission Planning Process Model

In Army mission planning, the superior unit commander assigns tasks and objectives to subordinate elements, provides an overall Course of Action (CoA) with coordination measures (phase lines, communications guidelines, etc.), and may cross-attach elements. The subordinate commands mentally wargame how they will accomplish their tasks, including possible contingencies and opportunities, then replay as a group in mission rehearsal. This pre-coordination is so that adjacent elements need only minimal explicit coordination during the operation.

AEMS SE Challenges

- Formulating the AEMS/MUM-T interaction models integrating AES with O&O practice and doctrine; how AEMS could lead to new Tactics, Techniques and Procedures (TTP)
- The multiple objectives and quality attributes and tradeoffs in AEMS concept development in an arms race
- The workflow process in developing (training) non-deterministic ML application models
    - Decomposing the overall ML application into functional ML Modules (MLM) that can be trained independently, then composed into the ML application (exponentially reducing the training data requirements)
    - Formulating training programs for the MLM using incremental and extensible training from simple situations and basic skills (to reduce the

training data requirements, to assure core competencies, to expand from benign to adverse conditions – basic training program practices for students and animals)

- The high cost of obtaining adequate, robust, and realistic labeled training data over the OD
    - Obtaining training data is expensive and time-consuming
    - Cost-vs-realism tradeoffs in training data from Live, Virtual and Constructive (LVC) simulations, vs real but opportunistic data from field operations – SE for LVC simulations
- Tradeoffs between safety and effectiveness for the *unit in its mission* versus safety and correctness of MLM actions

Machine Learning (ML)

ML applications are "programmed" by training with cases, not by formulating theoretical algorithms. This presents quality tradeoffs for ML that are inseparable from (a) the training program, and (b) the systems and methods to acquire the training data.

ML in the form of multi-layer Deep Learning (DL) is emerging as the pre-eminent technology to automate complex "cognitive" tasks for which no robust and reliable algorithm is known. ML, being trained with extensive data covering a range of conditions and multiple sensor inputs holds promise effective and robust for such difficult problems.

An ML application example relevant to AEMS of current interest is autonomous driving in traffic with lane change decisions, varying roadway conditions, and uncertain behaviors of other driver. Explicit algorithms have been developed to solve bits and pieces of the problem, under idealized benign situations and assuming the model's parameters are known and constant. This results in "brittle" algorithms.

Weapon Release

There are historical practices and existing policies for free-fire zones, and for area effects munitions. There is *always* command authority for weapon release. When a commander designates a free-fire zone, anything that could resemble a target is a target. When area-effects munitions (artillery) or suppressive fire are employed, anything that area at that time is part of the target. When a minefield is emplaced (manually or by FASCAM) or swarms of Sensor Fused Munitions are flown, anything they sense over the designated area and time, are legitimate targets.

There is a fuzzy boundary decision space, whether the shooter is human or automated. Indirect fire and suppressive fire is simple: cover some area at some time with fire, per instructions based on input from a Forward Observer (FO). But is the FO human, automated, or AEMS? Is the allocation of fire human, automated, or AEMS? Except for finding cover, moving to a firing position, and moving to a new covered position, field artillery can be highly automated. Indirect fire is told when and where to cover some area with rounds.

Direct fire (excluding area-effects suppressive fire) requires the shooter (or hunter-killer pair) to decide that an entity is a target or location of a target. In the space and time of a free fire zone, anything that remotely resembles a hostile (e.g., a truck moving), or area that gave evidence of being a hostile (e.g., firing) is a target for direct fire. No higher authority is needed.

They just waited on the nod that was release authority. Since the 1300's "Havoc" was as military command. It meant "Go do what you have been trained to do. We trust your training."

SE Workflow Model for AEMS and ML Applications

ML for complex applications using rich model structures requires tremendous amount of training data. Obtaining training data is expensive. To assure capabilities over the Operational Domain (OD), training data is needed over the OD. This report presents an approach to minimize the data cost while producing ML applications with assured behaviors, and the ability to diagnose problems in composition.

- Decomposing the AEMS into independently trainable and composable functional elements of autonomy
- Decomposing the OD into incrementally trainable and composable regions
- Incremental training to build and assure behaviors and capabilities
- Combining training with down-selection and promotion for further training

ML Application Qualities

ML application qualities are directly related to the cost of the training data, which in turn depends on the training program. *SE for the training program is key* to quality ML applications. Types of qualities and tradeoffs: qualities of the integrated ML application, qualities of individual ML modules, qualities of the formulation of the ML modules, qualities of the training data acquisition.

- **Extensibility** – the ability to be extended to a broader Operating Domain and task/function difficulty without starting from scratch
- **Composability** – the ability to combine two MLM into a more capable model, including series composability and parallel composability, extending the OD and combining elementary tasks/functions into more complex behavior
- **Independence** – ML modules can be constructed and trained independently
- **Trainability** – labeled training data are sufficiently unambiguous for a robust and reliable ML module
- **Adaptability** – the ability to perform in situations outside the OD of the training data
- **Decomposability** – the property of the overall ML application
- **Retrainability** – the ability to be trained to unlearn behaviors and learn new behaviors
- **Robustness** – the breadth or scope of the OD over which the ML module is acceptably reliable
- **Reliability** – acceptable performance against the performance measure used in training and evaluating the model, over some sample of cases not used in training.
- **Transparency** – a property of the entire ML application of knowing what the components are doing

- **Simplicity** - the quantity of independent training cases needed to build the mapping from input space to output space

Artificial Intelligence (AI), Machine Learning (ML) and Deep Learning (DL)

Artificial Intelligence (AI) is a general term encompassing many programming technologies to produce answers to questions that were traditionally addressed by trained and selected humans (or animals). AI includes ML, and other approaches – from genetic algorithms to clever feature extraction and process control. AI is about ability to execute tasks that humans evolved, were trained, and selected to do. AI is not a technology.

Machine Learning (ML) is a subset of AI to produce computer programs with "cognitive-like" capabilities without being explicitly programmed algorithms, but are "trained" from cases. There are various ML algorithms. Artificial Neural Networks (ANN) is an ML technique widely embedded in control systems to model relationships for which training data are available, with non-linear relationships for which no algorithmic solution is known. Deep Learning (DL) is a form of ML using multi-layer ANN. Only recently, in the past 5-10 years, did computer scientist figure out algorithms to train multi-layer ANN. There has since been a proliferation of techniques.

ML applications are commonly considered as being of two types: classification problems and regression problems. Regression applications predict the correct value of a parameter, e.g., a control parameter. Classification applications predict the discrete state of a system, as in perception of a type of thing at a time and place, or a decision to turn right or left, stop or go.

Set-Based Design in Action

In common practice, ML applications are designed to provide the "maximum likelihood" prediction. A single "best" point solution is selected, and propagated in the chain of ML modules.

In both classification and regression problems, ML can provide the probability distribution of alternative results. In systems of cascading or composed ML modules, assessment of the distribution of answers may need to be propagated. Some answers may have low probability, but high cost of consequences. In real-time applications, at some point in time an option must be chosen or rejected because it will be too late to execute the action, a fork in the road.

Consider lane choice in expressway driving. The driver enters at some on-ramp intending to exit on some off-ramp, as safely as possible within some time and risk constraints. The driver can choose which lane to drive in. The driver can change lanes to pass a vehicle, allow a vehicle to pass, etc. The driver anticipates upstream roadway and traffic conditions, but conditions are uncertain. A driver might change lanes to see beyond a truck to observe the traffic conditions. Every lane change has a cost/risk, which depends on immediate neighbor conditions and their uncertain responses. Frequent lane changes – thrashing – amplifies risk. At some point, approaching the exit, the driver must be in the pre-exit lane. The driver must balance risk of missing the exit with time to reach the exit with risk of collision.

When one action is chosen, alternatives are eliminated. When time passes and it is too late for an action, that alternative is eliminated. All interpretations are maintained as long as they are possible. At any point in time, the is a best choice or maximum likelihood answer. If external events or new information require immediate action, this will be the choice. Otherwise, the alternative, conflicting, plans or interpretations can be maintained in parallel for *multiple hypothesis tracking* and *delayed differentiation*. This is Set-Based Design in action.

("Multiple hypothesis tracking" and "delayed differentiation" and "Set-Based Design" are essentially synonyms from different academic domains.)
The challenge in applying SBD is determining when to eliminate alternative interpretations. AEMS can be set up to mechanically execute the actions of a plan. These are simplistic systems. In military applications there are significant unknowns and uncertainties. The AEMS has the alternatives of

- Committing to a course of action based on the maximum likelihood interpretation, more-or-less automatic, mechanical execution of a prior plan with local adaptation to the situation
- Taking actions to obtain information to resolve ambiguity & uncertainty before committing to a course of action (i.e., experimentation or probing)
- Taking the immediate action or interpretation that delays commitment and maximizes the range of future choices
- Deciding that the AEMS is unable to resolve the ambiguity, and asks the handler/commander to decide

These are the problems for the executive decision-making function, requiring input from the planning function.

Summary

This report was an attempt to provide practical and relevant insights for SE for acquisition of quality AEMS, and SE for developing ML application for AEMS. Two key issues addressed were the functional organization of ML applications, and the role of SE in developing training programs for ML. This report articulated acquisition qualities for AEMS and ML, and provided a functional decomposition into independently trainable and composable ML modules. A key insight was the tradeoff between the time and cost of obtaining training data versus the assurance of safety and effectiveness over a wide and robust Operational Domain. This report articulated the importance of training data over the Operational Domain, and a strategy for SE combining incremental training and promotion in ML application development – a standard strategy for training animals and humans. A third key insight was the need for accelerated SE in the arms race with near-peer threats.