Impacts and Opportunities for Engineering in the Era of Cloud Computing Systems

Technical Report SERC-2012-TR-023-1

January 31, 2012

Principal Investigator: Dr. Dr. Christopher Ackermann,

Fraunhofer Center for Experimental Software Engineering

Team Members:

Fraunhofer Center for Experimental Software Engineering:

Dr. Madeline Diep, Dr. Forrest Shull

Southern Methodist University: Dr. LiGuo Huang, Xu Bai, Yingmao Li

University of Virginia: Dr. Marty Humphrey, Dr. Kevin Sullivan

Contract Number: H98230-08-D-0171 Task Order: 0002, DO 001, RT 039

Report No. SERC-2012-TR-023-1

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 H98230-08-D-0171.

Contents

Executive Summary

This report discusses the impact of cloud computing and the broader *revolution in computing* on systems, on the disciplines of systems engineering that have evolved over the last half century, and on new opportunities for these disciplines. This report sees the need for a new synthesis of traditional and computer-science-based variants of systems engineering. It recommends that the Department of Defense (DoD), perhaps in collaboration with other agencies, invest in new research and educational initiatives to develop and implement a new function, and perhaps a new hybrid discipline, of ***computational systems engineering.***

As software and computing have moved to the very center of system design, it is no longer feasible to treat these issues as mere component-level concerns. Rather, they must now be addressed at the highest level of system definition and development. That said, the traditional systems engineering issues of integrating across many other areas of expertise have not gone away. To succeed now requires an integration of expertise from traditional systems engineering disciplines *and* from the de facto systems engineering disciplines that have evolved in parallel in the computer sciences, notably but not only software engineering and cyber-physical systems. Neither discipline is configured to succeed on its own. A new synthesis is required.

This report focuses on impacts that this revolution in computing, and that cloud computing, in particular, are likely to have on systems engineering and the disciplines of systems engineering. This report uses the term *systems engineering* to include several fields that address such issues as system requirements, architecture, allocation of development responsibilities to sub-areas, lifecycle process, systems test, evaluation, deployment, and evolution, economics, and human-system integration. We use the term *traditional systems engineering* to refer to the professional discipline of systems engineering as embodied in institutions such as academic departments of systems and industrial engineering and related professional societies.

This report findings and recommendations for systems and systems engineering in this regard. Appendices A-C then (1) identify opportunities for short-term research with the Systems Engineering Research Center and its contributing member institutions, including members of the group that has produced this report, and (2) provide contact information for the author team.

The Origins of This Report

This brief report is the result of a rapid-turnaround study conducted by researchers at three University-based institutions affiliated with the Systems Engineering Research Center (SERC), a University-Affiliated Research Center (UARC) at Stevens Institute of Technology (SIT). The

study was funded by and conducted for the U.S. Department of Defense (DoD). The study period began on November 16, 2011 and ended on January 31, 2012.

The Study Group

This study was conducted by a collaborative team of seven researchers from three member organizations of the SERC UARC: the Fraunhofer Center for Experimental Software Engineering at the University of Maryland (FCMD), the Department of Computer Science at the University of Virginia (UVa), and the Department of Computer Science and Engineering at Southern Methodist University (SMU). The group members were Dr. Forrest Shull, Dr. Madeline Diep and Dr. Christopher Ackermann (FCMD), Dr. Marty Humphrey and Dr. Kevin Sullivan (UVa) and Dr. LiGuo Huang, Xu Bai and Yingmao Li (SMU).

Scope of this Work

This work focused on three questions of primary interest to the sponsor:

- What will be the impact of cloud computing on the *systems* of the future?
- What will be the impact of cloud computing on the discipline of *systems engineering*?
- How can systems engineering use cloud computing to advance systems engineering?

The study group addressed these questions by taking into account the future integration of systems engineering with de facto systems engineering sub-disciplines of computer science. This report focuses on these specific questions, within a context of our broader perspective on the future evolution of both fields.

Because the sponsor asked that this work not focus on a particular cloud model, this report provides an overview of topics and their relation to the general cloud computing paradigm. This work takes a broad view of cloud computing as including the networked provisioning of both commodity computing products and of specialized and sophisticated computing services. The latter are sometimes recognized as service-oriented architectures (SOA: an architectural model for building software and system using reusable and interoperable services, where a service is a software component implementing a specific business capability). When findings are relevant for only a subset of the distinct cloud models, this has been indicated appropriately in the text. These models include:

- Infrastructure as a Service (IaaS): Delivers to the customers, as capability, a computing infrastructure, including processing, storage, networks, and other fundamental computing resources [NIST 11b].
- Platform as a Service (PaaS): Delivers to the customers, as capability, the ability to deploy consumer-created or acquired applications onto the cloud by providing programming languages, libraries, services, and tools supports [NIST 11b].
- Software as a Service (SaaS): Delivers to the customers, as capability, software solutions running on a cloud infrastructure [NIST 11b].

Sources of Information for this Report

The findings and recommendations included in this report were drawn from a number of sources to ensure that the issues elucidated were both rigorous (i.e., drawn from peer reviewed scientific literature) and up to date (i.e., drawn from organizations and environments doing significant practical work in cloud-based systems). Those sources included:

1) *Personal expertise of study authors.* Members of the team work with commercial and scientific cloud systems that exemplify challenges for system engineering. They also conduct research and provide expertise to industry in such areas as standards and cloud-relevant national systems in healthcare informatics. Empirical data garnered from these experiences has been used to make sure that recommendations to the sponsor are well-grounded and specific, and to provide corroboration and detailed examples of the phenomena discussed in our report. The environments to which our team has access cover key issues related to cloud, such as:

- *Design and implementation of a large-scale cloud-based system:* A team member has co-led the creation of *MODISAzure* to analyze the MODIS satellite data at scale. MODISAzure is one of the first large-scale systems to use the Microsoft Windows Azure cloud. Recent efforts on MODISAzure include studying the costs and benefits of placing the functionality entirely within the cloud vs. spanning the application across the enterprise and the cloud.
- *Multi-institutional data integration:* A team member has provided invited input to the Institute of Medicine and the U.S. Department of Health and Human Services to understand possible architectures for future national health information networks, to support data availability at points of need (care), public health and bio-surveillance, and research. Key issues include integration of heterogeneous data sources across multiple institutions; divergent syntax and semantics of structured data, and the prevalence (and our increasing abilities to deal with) unstructured data; incremental and evolution development of healthcare systems of systems at a national scale; and architectural support for security, privacy and survivability.
- *Opportunistic system development*: Team members have been involved in the development of several cloud systems in which multiple data streams are being processed. The usage of these applications is often unclear during development time and unexpected usage patterns often emerge after deployment. Furthermore, these applications must offer at least a reduced set of functions if limited or no network connection is available.
- *Cloud dependability and quality assessment*: Members of our team have been involved with building an environment to assess the quality and dependability of cloud computing systems. A key differentiating feature of this environment is the ability to measure cloud application dependability in the target or intended usage environment using real-world operational conditions.
- *Cloud-based modeling:* Several team members have been working to apply a cloud-based systems modeling and analysis tool to map and improve the performance of technical and social networks in forming Army contingency bases.

- *Funded research:* Several members of the study team have past and current research projects in cloud computing as well as in related areas funded by agencies including the National Science Foundation.

2) *Literature search.* In previous and ongoing research efforts, our team has conducted extensive literature research in areas closely related to the focus of this research. In preparing this report, we conducted additional reviews of the scientific literature with the goal of summarizing the research that focuses on the opportunities and risks to cloud computing systems particularly from their quality attributes, architecture and data management aspects.

3) *Interviews with key figures in industry*. We use our contacts with key practitioners and researchers in this area to ensure that our results are up-to-date and reflect current practice. For example, Dr. Shull conducted an interview with Dr. James Whittaker, an engineering director at Google who has overseen that organization's cloud testing activities. Much of this conversation focused on the skillset needed by effective testers when working in the cloud paradigm.

4) *Expert workshop.* A one-day expert workshop, held at the Fraunhofer Center on December 16, 2011, provided a forum for a technical interchange among invited experts and the sponsor. The result was a documented list of concerns to be further investigated by the study group. The workshop started with invited talks on implications of cloud computing for *security* and on cloud computing *standards*, by Barry Horowitz, an invited expert from the University of Virginia (chair of UVa's Systems and Information Engineering Department and a former CEO of the Mitre Corporation), and Marty Humphrey, respectively. We recognize and thank Dr. Horowitz for his contributions of information for this report, particular in the areas of security and analytics.

The Structure of this Report

This report presents findings and recommendations organized around the questions in the charge to the committee. Each section summarizes technical topics that are enabling new innovations in cloud-based systems or generating challenges for system engineers developing those systems – and often both simultaneously. Where possible, concrete recommendations have been extracted for those topics which are clearly focused for key stakeholders:

- **DoD and other organizations** with significant needs for high-quality systems that are enabled by cloud technologies, for which recommendations are often related to funding research, or funding development of educational programs and curricular modules for full-time students and for continuing education of the systems engineering workforce;
- **Researchers, educators and universities** who are further developing the disciplines related to systems engineering, for which recommendations include building new multi-disciplinary programs (e.g., involving traditional systems engineering and computer science), or to develop research programs in cloud-intensive systems;
- **Practicing systems, software and computer engineers** who are working to develop cloud-based systems today, for whom recommendations often include pointers to technical

approaches, checklists of concerns that need to be addressed during the engineering of these systems, or strategic approaches to cloud-based system design.

The final section of this report, Section 3, *The New Era of Computational Systems,* discusses how the overall revolution in computing demands new thinking, and highly novel approaches to overall systems engineering in an era in which software and computing have moved to the very center of system design in essentially all major domains of interest to the DoD (and beyond).

Appendix A contains short descriptions of research opportunities identified by the study group. Appendix B describes a research proposal on a social systems engineering modeling and trade-space analysis environment. Appendix C presents biographical and contact information for members of the study group.

1. Introduction

Cloud computing, and the ongoing revolution in computing more broadly, is profoundly disrupting both for technological systems and the organization and methods of the disciplines that produce them. These disruptions are creating both deep problems and remarkable new opportunities for all disciplines, including notably for systems engineering.

The Apple iPhone provides a case in point. In an instant it propelled a leading computer and software company to the forefront of personal mobile communications, and sent traditional cell phone firms, organized around expertise in radio frequency circuitry (RF), into disarray. Nor can the iPhone be seen merely as either an RF system with an embedded software module, or as a small computer with RF peripherals. Rather, each phone is a sophisticated software and computing node in a rich and rapidly evolving software and cloud computing ecosystem. Its software platform design has created an astonishingly vibrant industry in high quality, inexpensive, complementary (third-party) software products. And its integration with a massive cloud-based infrastructure (particularly around Siri) promises revolutionary changes in human computer (spoken-language) interaction. Radio frequency communication circuitry remains a vital issue, as evinced by problems with the iPhone 4 antennas, but RF electrical engineering is no longer the central organizing discipline for this industry. Computing is now the central force and integrating function in product design, from concept to manufacturing.

Such paradigm-altering changes have implications on many levels: On the types of functionality, which suddenly become possible to put into the hands of users; on the expectations of the users and sponsors of new systems; on the quality attributes (e.g. reliability, robustness) that become the new norm for systems engineering; and on the relevant skillsets for the system engineers themselves.

This report summarizes our findings on the key technical dimensions that are influencing such issues related to the increasing pervasiveness of the cloud computing paradigm. Our goal has been to highlight the important technical topics that system engineers working in this new paradigm need to be aware of, as well as to highlight specific recommendations where possible that reflect actionable findings regarding areas of risk or promising methods, processes, or tools.

2. Finding and Recommendations

In this section, we discuss the findings of our study and give recommendations for actions to improve systems engineering in the cloud computing era. Our findings and recommendations are based on research literature, discussions with cloud computing experts, and the expert judgments of the study team. This section, indeed this report, is not intended to be exhaustive. There are important impacts and opportunities that this report does not discuss. The study team identified areas of particular importance for systems engineering. A more fully comprehensive treatment of the deep questions asked of the study group would benefit from an in-depth study, perhaps of the kind conducted by the National Academy of Engineering.

2.1 How the Cloud Will Impact Systems

Radical new system capabilities are already being enabled by cloud computing. System users are developing new expectations regarding system functionality and quality – in areas such as robustness, availability, rapid evolution, and performance . This section addresses actual and projected impacts of cloud computing on technology systems.

2.1.1 Deep and Integrated Pervasive Computing

One of the main contributions of cloud computing is to decouple the availability of substantial computational power from the need for the physical co-location of computing machinery. Datacenter-scale computing functions can be delivered even to small, mobile, low-power devices (a good example is the iPhone) contingent on availability of sufficient network capacity. The opportunities afforded by this change in the design space will create enormous pressure to exploit significant computing capabilities in complex systems.

Indeed, future systems and components at all scales, even tiny, will increasingly be designed to exploit substantial back-end computing functionality provided by cloud and related systems. The sensor and actuator components must be proximal but the provision of computing power can be remote. Such a future is likely to profoundly transform most major technological systems. Not only whole systems but recursively the individual components can be deeply computational. Computational behavior and its underlying software representations will emerge as fundamental concerns at all levels of system specification, development, deployment and evolution. Systems structured as societies of computational systems elements will increasingly exhibit and/or exploit complex emergent behaviors. The computing challenges that will be presented by such systems are unsolved today, and will require substantial research in computational systems engineering to resolve.

Recommendations

The DoD should establish research programs on the nature and development of *computational systems*. An experimental approach involving representative systems is suggested. A central research question is how to integrate traditional systems engineering with software design and other related fields of computer science and engineering into a new, overall *computational systems engineering* (CSE) function, and perhaps even a hybrid discipline. Research on the nature of such systems should address such issues as (1) secure infrastructure for collecting, sharing and analysis of vast data sets collected from across computational systems; (2) the specification, validation and composition of computationally rich components and systems; (3) new protocols and tools for large-scale system mapping and understanding, health monitoring, repair and evolution; (4) lifecycle models for computational systems development and evolution.

Academic researchers and funding agencies should establish collaborations between traditional systems engineering and the relevant areas of computer science to conduct the research and to train a research and a practitioner workforce with knowledge and skills matched to the needs of computational systems.

Practitioners of traditional systems engineering should work with managers to bring computer science based approaches to systems development to the *systems engineering table*. At the same time, systems engineering practitioners should participate in continuing education work in such areas as distributed systems, big data, software requirements and specification, and the software lifecycle models (particularly iterative and evolutionary lifecycles). At the same time, students and practitioners of software, cyber-physical, and other related computing disciplines should identify the major areas of systems engineering in which they are not well trained, and should gain *familiarity,* and even basic *competency,* if not *mastery* in these areas, so that they are equipped properly to collaborate with traditional systems engineering in determining major system requirements, lifecycle models, and so forth. Managers should consider new models in which experts trained in the software/computing disciplines play major roles in overall systems engineering activities.

2.1.2 Human Factors

Apple Computer has emerged as one of the most successful company in history. Its success is due in large part to (1) its understanding of computational platforms and ecosystems; (2) its commitment to human factors in design; and (3) its success in creating an efficient, flexible supply chain. People who for decades were afraid of computers are suddenly delighted and empowered by the function, elegance, and usability of Apple's products. Apple understood that people are not just the users of their devices, but crucial and complex elements in the design of their overall industrial and computational ecosystem. The traditional discipline of systems engineering has historically included significant attention to human factors, as has computer science, within its sub-disciplines of human-computer interaction (HCI) and social computing. However, neither discipline is yet adequately configured to training people who are capable of

combining the linear problem solving skills of traditional engineers and computer scientists with the creative and humanistic insights needed to build systems that really work for people.

Recommendations

The DoD should require that some of the research programs it funds in computational systems engineering include major experimental systems efforts where the systems include substantial human-facing interfaces, and where these interfaces raise key human factors and social issues, including such issues as cognitive ergonomics, social computing, and information privacy. *Design thinking* involving rigorous forms of reasoning that are not typically well developed in either traditional or computer science variants of systems engineering should be emphasized. Inputs from social science and even the fine arts and humanities should be considered.

Departments of computer science, schools of engineering, and other academic disciplines and departments should undertake efforts to establish research and education programs in the design of human-intensive computational systems. A key question concerns individual training and the structure of teams needed to produce cloud-based computational systems with human interfaces with the quality, usability and functionality of the best available commercial products.

Practitioners in traditional systems engineering and computer science variants should avail themselves of opportunities to understand the major issues in such areas as interaction design, cognitive ergonomics, privacy, and online interaction---critical to human-intensive computational systems. Managers should strongly emphasize the inclusion of human-aware expertise at the highest levels of system definition, development and evaluation.

2.1.3 Centralized Data Analytics for Future Systems

The ability of cloud computing to support centralized data analytics for large distributed systems will create enormous new opportunities in many areas, including but not limited to cyber security. For example, centralized analytics of cloud-enabled applications can enable

- collecting and assessing information on military equipment in support of readiness enhancement, real-time logistics, and intrusion detection;
- collecting and assessing information on military processes in support of doctrine enhancement;
- system of system restorations with centralized resilience management and authorities for reconfiguration;
- centralized security and privacy monitoring and control including insider threat monitoring and rapid forensic assessments, controlled defensive deception operations, state estimation based evaluation for detection of manipulated operator displays for physical systems, etc.

Exploiting centralized data analytics for large-scale distributed systems poses many challenges. Limited network bandwidth to the edges of systems will continue to be an issue, especially when

networks are wireless. The looming exhaustion of frequency spectrum, at least in the domestic market, will create difficulties for users of wireless networking. Wireless connectivity creates new vulnerabilities, e.g., to jamming. Networking also remains a significant issue within the data center. Today limitations on bandwidth and protocols makes it impossible to maintain replicated copies of changing data at large scale for purposes of availability. This issue leads to a hard-to-avoid tradeoff among consistency and availability in the face of partitions, as expressed in the so-called "CAP theorem". This tradeoff has led most designers of cloud-based systems to avoid application strong requirements for data consistency, in favor of availability. Systems engineers will have to be aware of major constraints on system functionality produced by the current state of the art in data center networking.

Recommendations

The DoD in collaboration with other funding agencies should continue to invest in fundamental research on system and network architectures for big data analytics. The study team is aware that this area is already a high priority for Federal research investments. Research should be pursued, in particular, in which data center analytics are just a part of larger-scale systems. The needs of larger systems will help to reveal unmet requirements imposed on data center designs. This style of research should be pursued by multi-disciplinary teams of computer scientists and systems engineering researchers.

To exploit the centralized large-scale data analytics enabled by cloud computing effectively, practicing systems engineers must become familiar with current trade-spaces in such areas as data consistency and availability in the presence of network partitions (which do occur in large data centers). Assumptions that highly available, assured-consist data can be produced by large-scale, cloud-based analytics systems, for example, could lead to serious problems in system development.

Educational offerings in the area of cloud systems capabilities and *limitations* should be developed. More broadly, best practices and lessons learned in such areas as data storage [Colarelli 02] [Farber 11], virtualization management, licensing evaluation and management, resources allocation, etc. should be made available to practitioners to hep them to understand such issues as how to balance the performance of data analytics and resource utilization in cloud-computing systems [DePompa 11].

Engineers should also be aware of such distinctions as those between "data clouds" and "utility clouds" [Farber 11]. *Utility clouds* focus offer infrastructure, platforms, and software as services that many users consume. These basic building blocks of cloud computing are essential to achieving real solutions at scale. *Data clouds* leverage those utility building blocks to provide data analytics, structured data storage, databases, and parallel computation, which provide analysts with unprecedented access to mission data and shared analysis algorithms.

System engineering practitioners engaged in cloud system design, development and configuration should also be educated in optimization of cloud resources [Spang 11]. Checklists of design patterns or other technical considerations based on organizational experience could be maintained for use in such assessments. An initial (and by no means complete) set of such concerns could include:

- **Centralized data storage.** Consolidation and central coordination minimizes the total number of hard disks used, while greatly decreasing the overall network bandwidth occupation. For example, files that are not accessed regularly are stored in a different set of capacity optimized hard disks. These hard disks enter a sleep mode when not in use and consume negligible bandwidth [Colarelli 02].
- **Analytics close to the storage.** System engineers need to consider the network transportation within the cloud, that is, shifting massive data collections to banks of processors seems take more network bandwidth than moving specific computing job close to data storage [Farber 11].
- **Appropriate monitoring, measuring and understanding of current system performance.** Automated tools can track key metrics such as server utilization, available storage capacity and other important elements. It's impossible to optimize data center resources in the most effective way without a clear understanding of how current systems perform as a baseline.
- **Appropriate software licensing models.** Virtualization can cause duplicative copies of operating systems and applications even though usage is the same. Many software manufacturers are altering their licensing terms to account for virtualization. This can help government organizations reduce expenses in software licensing costs.

2.1.4 Interface Specification, Validation and Documentation

Virtually all systems are composed of smaller components that are often dispersed and interact via networks. Thus, specifying, implementing and testing interfaces and their protocols is a well-known and common systems engineering task. However, the importance of interfaces is significantly changing in the context of SOA and cloud computing, affecting how they are designed and documented. Many if not most designed interfaces, both internal and outward-facing, will now include complex computational behavior, including concurrency and distributed systems behaviors. Designing and validating such interfaces and testing for compliance with them remains a challenge at the forefront of both computer science and systems engineering and will require tight collaboration among experts in these areas. In areas where the external interfaces are to people, human factors experts will also have to participate centrally in system design.

In order to guarantee that a service is accessible by consumers, detailed and intuitive documentation of its interfaces and protocols is crucial. Integration problems in many of today's systems can be attributed to insufficient interface documentation. Details are often lacking and documentation is difficult to comprehend. The documentation is then interpreted differently by

the development teams that use the protocols, resulting in implementations that deviate from the intended protocol behavior and components that do not interact properly. Extensive post-implementation failure analysis is necessary to resolve such integration problems.

While insufficient documentation poses a risk to reliability and performance of traditional systems, it also has a direct impact on business goals for cloud services. In a cloud environment, the consumer can presumably choose from a variety of services that provide similar functionality. One of the factors that can influence the decision for or against a service is the ease of use. Thorough interface documentation can help consumers to quickly integrate the service in their system. Examples of comprehensible interface specifications can be found in the Google Search API [Goo 12] or the Twitter API [Twi 12]. The documentation for these services is structured as a tutorial with examples that make it easy for others to use.

Since achieving business goals can be directly related to the quality of the interface documentation, we believe that the quality of protocol designs and documentation will increase. Furthermore, even non-cloud-based systems might benefit from this trend.

Recommendations

The DoD should invest in research and advanced education at the intersection of traditional systems engineering, software engineering, and cyber-physical systems with an emphasis on rigorous interface specification, validation, and verification of implementations. The emphasis should be on precise semantic specification, refinement, testing verification of cyber-*physical* phenomena. Such research should be conducted by multi-disciplinary teams from traditional systems engineering, software engineering (including formal methods), and cyber-physical systems.

Since interface documentation often suffers from being out-of-date and hence untrustworthy, automatic detection of interfaces and their also parameters remains a potentially important area. Researchers and educators should investigate methods and tools to address this problem. Some solutions do exist for automatic analysis of codebases to generate documentation on code (e.g. Doxygen, JavaDocs) or architecture (e.g. SAVE [Lindvall 08]) as-needed. However, these solutions produce documents that are merely technical descriptions of individual functions or components. They lack guidance on how to actually integrate the service, similar to the way user documentation guides users through interacting with a system to achieve certain goals. Thus, in addition to the description of functions, processes for how to integrate the service for different purposes must be outlined perhaps annotated with examples.

Practitioners and managers should treat interface documentation as a first-class deliverable, recognizing that a minimal set of interfaces and minimal level of detail is necessary to support the development, validation, compliance testing with respect to and evolution of such interfaces. Managers should develop models for allocating sufficient time and resources to not only create such documentation but also for thorough validation. This recommendation is not opposed to

agility; agile development does not imply absence of documentation, just that teams identify documentation-creation activities and balance them in importance against other possible tasks such as further systems development. No matter the context, analysis is required to identify the key interfaces and the sufficient level of detail for documentation. The examples from Google and Twitter, provide good examples of a useful level of detail while demonstrating that API documentation can be done in an agile context. Amazon's practices with respect to design of externalizable interfaces provides a positive case study in the discipline of interface design and documentation and in enforcement of the use of such interfaces without exception.

2.1.5 System Health Monitoring and Repair

Cloud-based systems will have more stringent availability requirements and expectations (i.e., the degree to which the system is expected to be available) than non-cloud systems and will consequently have greater need for health monitoring and rapid repair mechanisms. A health monitoring mechanism provides an early indicator of potential problems so that avoidance and mitigation steps can be taken to prevent systems becoming unavailable. Meanwhile, when systems do become unavailable, the rapid repair mechanism aims to provide rapid recovery from the problem. Neither mechanism is a novel concept: Systems with client-server architecture, for example, already rely on health monitoring of server components; meanwhile fault recovery mechanisms range from employing various types of redundancies or relying on better problem reporting, both manual and automatic, and triage to resolve problems as quickly as possible.

Existing health monitoring and repair mechanisms, however, have not fully accounted for the characteristics of cloud technology and may fail to be effective when employed for cloud-based systems. Cloud-based systems have more “components” to be monitored than non-cloud systems, counting the hardware infrastructure (servers, storages, etc.) and the third-party software services that make up and interact with the systems (assuming the use of SaaS model), which further escalates existing issues found in the area of health monitoring such as the synchronization of observations. Moreover, the focus or priority of what should (and could) be monitored also shifts. For example, if the system utilizes the IaaS model, the cloud service provider is usually responsible for the “health” of the infrastructure - they perform monitoring and warn their customers of potential problems. The cloud customers have less visibility and control of the systems’ infrastructures, and have to rely on the monitoring capabilities provided by the cloud provider. Instead, more importance should be placed on the monitoring of the services that compose the cloud system, specifically to detect when they have become unavailable, and more importantly, to understand their impact to the system functionality.

Recommendations

The research community should look into methods to expand the current mechanisms of performing health monitoring and repair of systems by specifically utilizing the cloud technology to provide more rapid and ubiquitous feedback, with information integral with the state of the

cloud infrastructures that the systems inhabit. Cloud and Web technologies have enabled the use of direct feedback from software/system users, most notably through automatic error reporting mechanisms. In such reporting mechanisms, data regarding software usage and state are continuously tracked, and when system failure or crash occurs, the information is then automatically reported (with users' agreement) to assist in the repair process. Similar mechanisms can be employed for cloud-based systems, although they must be further adapted. Cloud technology alleviates some of the issues faced in automatic error reporting, such as challenges related to storing and processing the captured data, but also introduce additional research challenges, such as to identify what type of usage and infrastructure-related information would be needed to aid the repair process for the cloud-based systems, and how to obtain them in an effective manner.

Practitioners/system engineers should evaluate systems being developed regarding their utilization of cloud technology in providing redundancies as part of failure recovery mechanisms. Cloud technology offers the capability for diverse redundancies that are also cost-effective. For example, hardware redundancies can be easily achieved due to the elastic nature of the cloud-based infrastructure. The SaaS model also potentially enables the existence of a large pool of applications, which would make building software service redundancies feasible. Practitioners/system engineers should look into defining system architecture and infrastructure that enables such redundancies as well as a process for identifying and selecting two or more compatible services that could be used as redundancy.

2.1.6 Standards

Efforts to develop standards and to establish best practices related to cloud computing have aimed to address issues regarding security, data and application interoperability and portability, governance and management, and monitoring and metering. Table 1 describes several of these efforts. The table provides: the organization or group name that is developing the standard/best practices and the name of the standard (when applicable); a brief description of the technical focus area of the standard/best practice; a categorization of the effort with respect to the area of concern that the organization or the standard is attempting to address; examples of known supporters or adopters of each effort; and the current status of the effort, including materials that have been released (and when they were published). The table demonstrates that many of the efforts are still in early stages.

Table 1: Cloud Standardization Efforts

Organization – Standard Name	Description	Category	Supporters	Status
DMTF – Open Virtualization Format (OVF)	Provides format for the packaging and distribution of software to be run in	IaaS Interoperability	XenSource, IBM, VMWare, Microsoft	Standard, 1/20/2010

	virtual machine			
IEEE - P2302: Intercloud Interoperability and Federation	Defines topology, functions, and governance for cloud-to-cloud interoperability and federation	Interoperability		Working Group approved, Jan 2011
OGF – Open Cloud Computing Interface	Defines a boundary protocol and API that acts as a service front-end to a provider's internal management framework	IaaS, PaaS, SaaS, Manageability, Monitoring, Data transfer	OpenNebula, OpenStack, Rackspace, Oracle, Platform Computing	OCCI Core Model and Infrastructure Documents - Under review
SNIA - Cloud Data Management Interface (CDMI)	Provides functional interface for applications to create, retrieve, update and delete data elements from the cloud	Data storage and portability		Technical position, v1.0.1, Sept 15, 2011
Open Data Center Alliance	An independent organization whose goal is to define vision for cloud (provider) requirements focusing on secure federation, automation, data management and Policy, and Transparency.	Security, interoperability	BMW, Deutsche Bank, JP Morgan Chase, Lockheed Martin, Marriott, terremark, UBS, etc.	Usage models, best-practices paper
Cloud Security Alliance	A non-profit organization focusing on promoting best practices for providing security assurance within cloud computing (through training and certification)	Security	Accenture, AT&T, HP, Athena health, Dell, eBay, Google, Hitachi, Microsoft, etc.	

Despite these efforts, it is generally recognized that there do not yet exist meaningful standards for cloud computing today. This is partly because we are still at the early state of the paradigm (i.e., cloud computing is just starting to blossom) – and also because, at this moment, there is little to no motivation among the big players of the cloud vendors (Amazon, Google, Microsoft, Salesforce, etc.) to agree upon or to comply with existing and upcoming standards, specifically standards that encourage interoperability. The lack of support on standards could dampen the adoption of cloud computing, specifically for medium or small businesses that have fewer resources to "navigate" the cloud and that have less leverage against the big players of the cloud vendors in avoiding vendor lock-in.

The impact of the lack of standards might not be as significant as in other discipline areas, because the overall number of cloud providers will be relatively small for the near future. Furthermore, there is still “portability” in certain types of clouds, even without standards. For instance, IaaS clouds provide the OS layer, which could be the same across multiple clouds.

Recommendations

While it appears unlikely that meaningful cloud standards will appear in the near future, DoD and system engineers should closely follow the NIST effort to encourage the formulation of cloud Standards, such as through its Standards Acceleration to Jumpstart Adoption of Cloud Computing (SAJACC) initiative and the establishment of various standard working groups, to assess the status of cloud standards broadly.

DoD should leverage NIST’s Cloud Computing Technology Roadmap [NIST 11c], which specifies high-priority strategic and tactical requirements related to security, interoperability, and portability requirements to further cloud adoption for US agencies. This technology roadmap is a useful reference for understanding what standards need to have in place in order to meet these requirements, and should be used by the DOD when developing its own standards or adopting standards developed by other agencies.

In lieu of cloud standards, systems engineers should become familiar with current best practices. For example, in the case of IaaS public cloud, Amazon AWS is widely popular and their practices could be adopted by others. Furthermore, if standards are an important consideration of the particular system being designed, IaaS should strongly be considered, as the OS layer inherently provides a mechanism for portability between the system component and the cloud infrastructure.

2.1.7 Engineering of Computational Behavior

The computational nature of systems now demands vastly increased attention to modeling, validation, specification, verification and evolution of computational behaviors. Computational behaviors are extraordinarily varied and complex and are enormous sources of both risk and capability. Significant complexities arise in such areas as high accuracy time synchronization; communication network performance, concurrency and synchronization, data consistency in very large scale data-intensive systems, and evolving standards for software interoperability. Computing clearly must be exploited as a central force in system development and operation, but if systems and projects are to succeed, the engineering of their computational behaviors must be based on scientific knowledge, particularly computer science. Crucial semantic issues, e.g., involving currency and synchronization, are simply too difficult to be addressed by ad hoc or informal means. While anyone can be a programmer, the engineering of complex behaviors requires highly specialized knowledge. Nor can such issues continue to be isolated in specialist sub-modules: they are now pervasive concerns at the highest levels of systems design.

Recommendations

The systems engineering function has to be re-conceived as requiring and integrally involving expertise from computer science at the highest levels of system definition, architecture, and lifecycle management. The fields of traditional systems engineering and computer science, and particularly such areas as software engineering and cyber-physical systems, will have to find new ways to collaborate to define this new synthesis. The appropriate partitioning of expertise among these disciplines remains unclear, but what is clear is that no one of them alone will be able to address the top-level needs of the coming generation of computational systems. A new hybrid form of computational systems engineering is needed, drawing on knowledge that is now dispersed across several professions.

2.2 How the Cloud Will Impact Systems Engineering

The creation of systems that utilize the cloud in some form creates new opportunities and challenges for systems engineers. The following section presents the study group's findings and recommendations for how systems engineering can best leverage the cloud.

2.2.1 Lifecycle Models and Development Processes

The computationally-intensive nature of future systems and the nature of software will combine to demand an accelerating shift to evolutionary system life-cycle models. This shift will be away from elaborate, up-front definition of *ostensibly* stable requirements to an emphasis on evolution and the kinds of life-cycle models pioneered in software engineering, including prototyping, early and frequent deployment, and incremental enhancement based on feedback from use. Reliance on extensively engineered tests will increasingly be augmented by user testing of early releases and subsequent system evolution: not only to fix non-compliance with requirements but in some cases to learn what the requirements are. Current systems engineering processes, particularly those that assume that it is profitable to lock down requirements early, are not well-matched to the needs of an increasing variety of systems in the cloud computing era. We now separately consider requirements, design, testing, and maintenance.

2.2.1.1. Requirements

Cloud computing technology changes the ability to meet certain classes of requirements. For example, it will be easier to achieve scalability using cloud technology. At the same time, such technologies will drive a continuing transition to computational systems, with significant impacts for requirements engineering. The best processes for computational systems requirements engineering are often different than those for more traditional systems. First, computation introduces behavioral complexities that are hard to design and validate. We are still in the early stages of developing strong engineering foundations for design and validation of computational behaviors and their underlying software representations. These weaknesses in engineering foundations present particular challenges for specifying, realizing and evolving non-functional requirements: in such areas as information security, and system safety, survivability and reliability. Formulating, realizing, validating and evolving non-functional properties in systems that use cloud-based computing systems will be a significant challenge.

Second, the unprecedented nature of many behavioral requirements in computational systems of the future means that systems design will occur under high uncertainty: uncertainty that can be resolved only by building and assessing prototype or early-version systems. Requirements in these cases simply *cannot* be locked down at an early stage of system development, as is the norm for DoD and for traditional systems engineering processes. Rather, requirements will have to be *learned by doing* to a significant degree over given systems lifetimes. Third, software, though profoundly hard to change *reliably* (particular if it is poorly designed of designed in ways

that did not anticipate the required changes) clearly does still provide unprecedented flexibility to make late changes in system function. Lifecycle models that are evolutionary and adaptive *even at the level of overall system requirements* thus become not only possible but necessary.

Additionally, cloud platforms provide the capability for systems to be composed from separately evolving applications and services. The current process of selecting cloud services, from a pool of available services, often done in an ad-hoc manner, could benefit from structured analysis. Cloud-based system development should consider a requirement engineering process when adopting cloud services. Researchers have started to define such processes [Zardari 11], though additional studies to evaluate their effectiveness and applicability are still needed.

Recommendations

The DoD should invest in research on evolutionary development of computationally complex systems, and on associated *procurement* methods: particularly on methods that accommodate contracting and progress on system development in the *absence of stabilized requirements*. Detailed requirements might still be necessary for costing and contracting, but new mechanisms are needed to clarify where there are significant uncertainties in, and thus needs for and options providing technical and managerial flexibility to evolve system *requirements and specifications.*

The cloud's rapid deployment and user feedback capabilities provides and reason for systems engineering functions to develop evolutionary methods, e.g. based on agile software methods. Research and training are both still needed to reconcile unresolved tensions between agile approaches and traditional systems engineering, particularly hardware, lifecycle models.

System engineering researchers and educators should focus on development processes that reconcile waterfall-style lifecycle processes with software-originated agile methods. As software emerges as a dominant concern, it will no longer be adequate to accommodate agile software development within an overall waterfall-based systems process. Rather, systems development will have to become agile in the large. How to achieve this state when systems have substantial hardware and manufacturing elements remains an important question.

Enabling technologies for more agile systems processes include metrics-based approaches and automated code and architecture analysis techniques that provide detailed system models and analysis results on an as-needed basis, rather than though a reliance on manually created documentation, which often gets out of date.

Additional complexities in requirements arise when people are significant elements of a system. Human cognitive, physical, social and *intellectual* (information processing) properties greatly complicate system analysis. A new systems engineering function should involve experts in both human factors and computing when defining system requirements and throughout the lifecycle.

When a SaaS model is used, system engineers should clearly define boundary needs and constraints of the system. Possible questions in verification and validation include: Do we have a clear view of the boundary of the system? Is a static boundary needed or should changing boundaries be allowed (e.g., to take advantage of emerging services)? System boundaries can be modeled by system-context diagrams. System engineers should consider using such notations in requirement. Additional notations may be need to represent “soft” boundaries.

2.2.1.2. Service Composition in System Design

The design of systems in a cloud environment will be heavily influenced by the services already available. The complexities of designing computing hardware and software platforms will recede as cloud vendors handle these back-end issues. Rather than choosing hardware, system engineers will focus on selection of service and infrastructure vendors who satisfy their technical and business needs. Choices of vendors will often entail significant design commitments, as it is currently hard to migrate data and services among vendors. A lack of widely adopted standards militates against modularity and substitutability. While efforts are underway to standardize cloud infrastructures, we do not expect significant standardization in the near future.

Service Discovery. Cloud services are already available today and it is expected that the number of services will dramatically increase in coming years. The availability of functionality in the form of services presents an opportunity for systems engineering, as the focus can shift to adding new or enhanced services as opposed to re-implementing functionality that might exist in other systems. However, effectively discovering services when they exist is still a challenge; and even when services are available, engineers might not be able to take advantage of them due to lack of awareness. The Web Service Description Language (WSDL) provides a mechanism for describing web services, making them discoverable and to some extent imposing consistency among web service descriptions. The Universal Description Discovery and Integration (UDDI) offers a platform for registering web services described using WSDL: a first step towards a service discovery platform. However, UDDI has not been successful and several platforms have been closed. Creating a successful service discovery platform requires not only technical components such as WSDL and UDDI but also a community that utilizes these technologies.

Recommendations

Practitioners should develop platforms that not only centralize service availability but that also have features that lead to the formation of a strong communities around them. Markets for mobile applications have shown one model for how service discovery can be approached.

Focus on Interface Design. Interface design is hardly a new idea for systems engineers. However, the importance of computational interfaces will increase dramatically as systems come to be composed of cloud services and resources. The ease with which services can be used is largely determined by their interfaces. The interfaces of a service must be designed and documented in a way that allows potential consumers to integrate it into their systems with minimal effort and with high assurance that subtle interface errors will not compromise system

integrity. Furthermore, in the past interface protocols have evolved and changed, forcing cloud services to adapt. We can expect such an evolution of protocols in the future.

Recommendations

In order to avoid re-implementing large parts of the application logic, practitioners should employ techniques to separate the application logic from the interface and, thus, reduce the impact a change to the interface can have on the rest of the application. The goal is to be able to adopt new protocols, and make the service available to a large audience, with minimal change effort.

Impact on Policies. The ability to ignore how and where data is stored is considered one of the key advantages of cloud computing. It frees the system engineer from managing infrastructure issues. However, in some cases, policies might impose restrictions on physical locations where data are stored. One of the examples of such a policy has resulted from the U.S. Patriot Act, which gives the U.S. government access to electronic information. This has triggered foreign governments and organizations to restrict the flow and storage of information in the United States. For instance, “Bill No. 16 - Entitled an Act to Protect the Personal Information of Nova Scotians from Disclosure Outside Canada” [NSC 06] restricts the storage of data in the U.S. In order to accommodate such policies, cloud infrastructure providers must be aware of the data location and be able to guarantee that data will remain within a certain physical area.

Recommendations

Systems engineers should carefully consider data location requirements and should investigate (where appropriate) commercial cloud providers’ ability to meet those requirements (such as the Amazon GovCloud). Researchers should develop methods to keep track of the location of data sets as well as to ensure that certain physical boundaries are not breached.

2.2.1.3. Testing

Cloud technologies can reduce the need for some performance-related testing, such as volume and stress testing. These are tests to understand and assure system behavior when presented with a larger-than-anticipated volume of transactions and data processing. On the other hand, testing for other non-functional requirements, such as security and privacy, needs to be more rigorous, and can be greatly complicated when software services that are integral parts of a given system are operated outside of the control of the systems engineering and integration function. (High-coverage white box testing becomes difficult or impossible for example.) New challenges are also introduced in regression testing as providers update component services, sometimes without actions on the part of the service consumer.

On the other hand, cloud computing also promises new opportunities for system testing. We are already seeing new methods of testing, such as outsourcing functional testing as a crowdsource activity. It should be recognized that such an approach is not suitable for all cloud-based system components, nor to eliminate in-house verification and validation. For example, the correctness

of the infrastructure supporting cloud-based services is vital to assure before deployment (or re-deployment), since any failure occurring on these component would have a significant impact to the entire system. Special care needs to be given to identifying outsourcing opportunities and restrictions.

Recommendations

System engineers should recognize both the new challenges and new opportunities in testing and validation created by cloud computing technology. Conducting testing in the cloud requires testing artifacts and frameworks to be migrated into the cloud, an effort that requires an upfront cost. System engineers, prior to migrating testing resources to the cloud, should first understand the characteristics of the system to be tested and the types of testing to be performed and then make the decision, based on this investigation, as to when and which testing activities should be migrated to the cloud. A recent report discussing the issues of migrating test artifacts to cloud as well as a preliminary evaluation of the suitability of various types of testing (e.g., unit testing, high volume automated testing, performance testing) to be done in cloud can be used as guidance for performing this activity [Parveen 10]

2.2.1.4. Maintenance

System maintenance traditionally entails the modification of systems to e.g., resolve bugs, to improve maintainability, and to implement new features. System maintenance processes and tools must be adopted to accommodate the trend of developing smaller applications more rapidly, with increased focus on user feedback, as demonstrated by some of the key technical topics below.

Benefitting from Modularization. The modifiability of a system depends on a variety of factors. Some of the most important factors are the complexity of the components to be changed as well as the degree to which they are coupled. Cloud-based systems are often composed of smaller, loosely coupled services. Loose coupling reduced the extent to which changes can propagate to other systems and, thus, reduces the change impact on services that are only indirectly affected by the change. Change impact and maintenance cost are only reduced for services whose interfaces are properly designed: to encapsulate likely changes and to remain stable as such changes occur.

Recommendations

System engineers should learn how to design and evaluate interfaces and protocols employing principles of information hiding modularity, as this concept is understood in the software design community.

Processing User Feedback. Some cloud computing service providers will rely on user feedback for defining the features of their products and for detecting defects. Conceptually, this approach has a significant impact on the meaning or definition of software maintenance: When a system is initially deployed, it will not be considered complete, but rather ready to receive

feedback. Thus the maintenance process, which typically starts after the system has been deployed, plays a much more active role in shaping the system early in its lifetime. Systems must be designed to include mechanisms to facilitate quick and easy user feedback and the respective infrastructure to respond to that feedback.

Recommendations

System engineers should develop approaches to building in mechanisms for user to provide feedback immediately and with minimal effort. Such mechanisms might include the automatic capture of machine configuration and state, so that cross-platform variations in functionality and performance can be assessed automatically. The ability for the user to provide feedback can be treated as a crosscutting concern during system design, implementation and verification.

Change Management. A particularly challenging aspect in maintaining and modifying cloud-based services can be releasing new versions of a service. While the technical effort involved in making a new version of a service available to the user is small in any Web-based system, releasing a new version might disrupt operation of the entire system formed around that service. Service consumers must be able to rely on the availability of a service. At the same time, service providers need to be able to modify services to release new features and bug fixes. Cloud computing leads to new challenges as service providers can have little knowledge about who uses the service and service consumers have little or no control over its evolution.

Recommendations

DoD should support researchers into mechanisms for managing service evolution. Mechanisms could include interface immutability and versioning, contracts and service agreements governing services changes, mechanisms for notifying service consumers about changes, and methods for consumers of services to detect changes, to support consumers adaptation to evolution of the surrounding service environment. DoD should support research into the nature and design of complex systems that rely on complex, asynchronously evolving service environments. This issue is a system-of-systems issue of a particularly software-intensive nature.

System Architecture Discovery. As more cloud services are deployed, we expect systems will form more rapidly and evolve at a faster pace than traditional systems. Given this rapid system evolution and the general lack of transparency in the cloud, it will be hard to know what services and resources a given system is composed of at any given time. System design in a sense can evolve beyond the full control of the systems engineer. At the same time, there will be increased need maintain intellectual control over complex systems: to comprehend both the structural and behavioral aspects of a system, to ensure the quality and conduct maintenance tasks.

The complex, evolving nature of systems will in some cases require a shift from up-front documentation of system architectures to system understanding based on the discovery of structure and runtime behavior during system operation. Systems will need to be monitored to

collect runtime information, and abstraction techniques will have to be employed to produce high-level models that emphasize relevant aspects.

Researchers and practitioners are developing methods and tools [Hassan 01][Baresi 04][Harman 07] to address well-known challenges such as lack of or inaccurate documentation. We believe system engineers can benefit greatly from that work in gaining insight into cloud-based system designs. However, the highly dynamic nature of cloud systems also presents new challenges for architecture recovery. Many of the techniques assume at least some knowledge about the composition of a system in order to place probes for collecting runtime information in the appropriate places. Such constraints are not feasible in cloud computing environments where little is known about system composition. New strategies will have to be developed.

Recommendations

Researchers should develop new methods for the mapping and automated documentation of system architectures, leveraging methods and tools already available, enhanced for monitoring of systems whose detailed component-and-dependency structures are partially unknown.

2.2.2 Increasing Need for Non-Expert Programming

In order to take advantage of the computing power for computationally intensive tasks, programmers are often required to be familiar with parallel programming techniques. While this is not a new concept, we expect that parallel programming will become much more widespread and in much greater demand as service providers try to take advantage of the cloud. In the past, setting up a parallel computing environment was a non-trivial and not inexpensive task. As a result, parallel programming was limited to problems that could not be solved efficiently on a single processor. With the availability of cloud processors on demand, parallel computing environments become much more accessible. In turn, parallel programming skills will be demanded not only from expert programmers.

Frameworks have already been developed and widely adopted to make parallel computing more accessible to non-experts. The Apache Hadoop framework implements the MapReduce mechanism in which a problem is divided into sub-tasks, executed in parallel, then aggregated by the reduce function. Hadoop handles much of the management effort in terms of to how many computers a computing is distributed and how to aggregate the results. However, the task of dividing a computing task into parts that can be executed in parallel is largely a manual one. It requires understanding of parallel algorithms as well as insight into the performance of source code constructs. For instance, system engineers must reason about what parts of the application have potential to benefit from parallelization. They must also know about how to reduce the number of interactions among concurrently executing threads. Currently, such skills

are taught only to a limited number of system engineers operating in specialized high performance computing environments.

With the increased availability of parallel computing and storage resource, we expect that the demand for the respective skill sets will be higher as well. This means that system engineers in general must have a basic knowledge of these topics. Also, the number of experts focusing on parallel processing and storage can be expected to increase dramatically.

Recommendations

Systems engineers who will rely on the cloud for processing should learn MapReduce and related data processing paradigms. Systems engineers should see those educational opportunities that utilize cloud resources for student projects in order to not only gain access to a parallel computing environment but also to become familiar with cloud computing in general.

2.2.3 Modeling Notations and Analysis Techniques

Current system modeling languages, such as SysML, are inadequate to support scientifically meaningful and effective modeling and analysis of the computational aspects of computational systems. They do not provide clear semantics for such fundamental constructs as data flow across channels. Edward Lee at the University of California Berkeley [Lee 10], has provided a cogent analysis of this issue. Substantial research and development efforts will be needed to produce new modeling notations, tools, and analysis techniques for effective modeling and analysis of computational systems from a systems engineering perspective. This effort will require deep involvement of computer science researchers, working with systems engineeris, in such areas as applied formal methods, software languages, formal verification, software static and dynamic analysis, and human factors issues.

Recommendations

DoD should invest in substantial multi-disciplinary research to develop semantically precise modeling notations and analysis methods for the systems engineering modeling and analysis of computational systems. Software modeling languages, such as UML, are not enough because they do not adequately model non-software aspects of complex systems (and they, too, lack design simplicity, straightforwardness and semantic precision). Required research in this area includes substantial *fundamental* (6.1) research. Such research should be grounded in real or realistic complex systems, and should be conducted by integrated research teams of systems engineers and computer scientists.

2.2.4 Business Case, Cost and Schedule

Cloud computing will have great impact on cost, schedule and business case issues. For example, while the business case for using cloud technology will be unique to each organization, some considerations will be consistent across the Federal government [Pizetre 10]. Private clouds can offer significant cost savings through reductions in hardware and

associated expenses. By sharing infrastructure through a foundational layer of virtualization software, utilization of individual physical components can be increased, which decreases the total number of processors needed, reducing capital expense and ongoing operating costs. Savings accrue in the acquisition and maintenance of hardware and usage of electricity, building space, and HVAC. In addition, because the cloud migration may require parallel IT operations, the shorter the migration schedule can be, the greater the economic benefits will be generated (measured in benefit-to-cost ratios, BCR) [Alford 09].

Recommendations

We recommend that DoD invest in exploring the new research and educational opportunities to educate and train system engineering researchers and practitioners to perform business case, cost and schedule analysis on the cloud computing systems in collaboration with other funding agencies such as the National Science Foundation (NSF). System engineering research needs to produce components and attributes in analyzing business cases, cost and schedule for cloud-based system development, maintenance and evolution. We recommend practitioners/system engineers take the following significant considerations into account for business cases including: *Costs* including acquisition costs, e.g., fixed vs. variable Cost [Aembrust 09]; *Porting, integration, and testing*; *Data migration* [Aembrust 09]; *Cloud features/requirements*; *Timing of decision*; and *Financial risks*. The following list provides a few concrete examples of recommended areas to be considered in facilitating a private cloud business case analysis [Pizetre 10].

- Identify the reductions in physical servers and associated costs (e.g., electricity, HVAC, hardware maintenance, data center labor). An analysis should be conducted to determine the number of virtual servers that can run on each physical server and the applications that can be hosted in the virtual environments. This will form the basis of the anticipated costs savings [Aembrust 09]
- Identify the new cost savings and increases. For example, identify the increase in costs for virtualization software and security products [Neamtiu 11], porting, integration and testing.
- Identify the value of new features (e.g., IT agility, location independent access for users, COOP) [Zardari 11] [Kherahani11]

Furthermore, we recommend DoD invest in research areas such as

- investigating the impacts of cloud-computing on the business cases, cost and schedule in rapid system (system of systems) development which enables the swift adaptation to system changes;
- integrating various success-critical stakeholders' perspectives into business case analysis as well as performing tradeoff analysis.

2.2.5 System Dependability and Certification

Emerging cloud applications, such as future combat, national security, health-care and transportation applications, are safety-critical and must meet stringent dependability requirements, including high availability, reliability, performance, resilience, safety, and security. Research is needed on tools and techniques for building and certifying cloud-based systems under demanding dependability requirements. A major impediment to dependability and quality assurance for engineering, maintaining and evolving cloud systems is the difficulty of conducting rigorous experiments to evaluate the resulting systems. Assessment, evaluation, and testing of research ideas aimed at enhancing the performance and dependability of cloud systems, including cloud platforms and services, are difficult to conduct due to the difficulty of monitoring the behavior of these widely distributed systems.

Recommendations

DoD should invest in exploring the new research and educational opportunities to identify major dependability issues for engineering cloud–based systems and to develop tools, analysis methods and techniques for building and certifying cloud-based systems that must meet stringent dependability requirements. Dependability encompasses system reliability, availability, maintainability, safety, security, etc. [Avizienis 04] [Basili 04] [Laprie 92] System availability and reliability, performance, security and accountability assurance are important for cloud-based platforms. System safety and survivability are also important for cloud-based platforms since catastrophic environmental events can potentially lead to correlated loss of the entire cloud platform.

Different applications can have different dependability concerns [Huang 06] [Kallepalli 01] and most of these are usage-sensitive [Basili 04]. For example, reliability is not only related to the number of internal faults, but also the usage scenarios that trigger observed external failures [Musa 93]. While industrial Service Level Agreement (SLA) standards provide guidelines for gauging the quality of cloud system offerings, we recommend that DoD invest in research on approaches to assessing and certifying dependability of cloud computing systems under various operational scenarios. For instance, instruments are needed to measure the cloud SLAs under different operational conditions. The instrumentation platform for dependability assessment on cloud-computing systems might consist of: 1) **Sensors** that monitor several run-time operational parameters that can be processed to accurately measure various dependability, performance, and power usage parameters. 2) **Actuators** that provide capabilities for injecting multiple dependability related scenarios at run-time, including hardware and software failures, security attacks, overload conditions, etc. 3) **Analyzers** that process data from appropriate sensors and provide integrated modeling and analysis instruments to facilitate evaluation of specific dependability attributes of cloud applications. 4) **Coordinators** that activate appropriate actuators to create specific operational scenarios, including extreme and/or hostile environments, catastrophic platform failures, etc. 5) **Benchmark applications** to enable

researchers and practitioners to rapidly conduct experiments to evaluate their cloud dependability enhancement techniques using real-world applications.

2.2.6 Security and Privacy

Confidential data leakage and loss of security in the cloud becomes a barrier to the adoption of cloud services for mission-critical defense systems. New sensor technology, centralized data collection, expert-developed analysis tools and cross-hierarchy decision-making enable a number of military system enhancements including cyber security. A key shortage is the lack of skilled people for identifying opportunities, and conducting the analyses needed for decision support. Cross-hierarchy decision-making can be constrained by existing organizational chains of command. Cloud computing can play a key role as an enabler and, when necessary, a consolidator for control for this class of enhancements. Examples below demonstrate the potential operational values for cloud computing in military system security enhancements:

- Insider threat monitoring and rapid forensic assessments with centralized analysis;
- Centrally controlled defensive deception operations;
- State estimation based evaluation for detection of manipulated operator displays for physical systems;
- For latency tolerant system functions, configuration hopping between local system and cloud computing based replica to handle moving target security.

However, traditional systems engineering is not well configured to assess and assure data security and privacy in cloud systems. Security and privacy risk assessments are considered a best practice for evaluating a system or application for potential risks and exposures. Traditional risk assessment approaches were designed for systems with static and human process-oriented nature. However, cloud computing introduces several characteristics (on-demand, automated, and multi-tenant nature) which challenge the effectiveness of current assessment approaches. The characteristics that make cloud computing attractive also tend to make it hard to assess. The five cloud characteristics articulated in NIST's definitions [NIST 11b] complicate the assessment of security and privacy of a system deployed into a cloud computing environment, which make current approaches ineffective in cloud security and privacy risk assessments.

Recommendations

The DoD should continue to support research on privacy and security, particularly for cloud-reliant complex systems. Potential examples include system architectures for security and privacy, impact assessment frameworks to assess security risks in reliance on cloud computing, security architectures to accommodate various levels of privacy concerns by users, and new database management schemes for storing and managing sensitive, dynamic, distributed data.

System engineers should consider security, privacy and data protection issues from the outset so as to avoid problems associated with security as an afterthought. System-aware security design patterns (e.g., *Reconfigurable Diverse Redundancy*, *Physical/Virtual Configuration Hopping*, *Data Consistency Checking*, *Defensive Deception*, *Rapid Forensics*, *Physical*

Confirmation of Data) should be evaluated and deployed in cloud-based system architecting and design with the corresponding implementation issues being taken into account. More details on these design patterns can be found in the open literature [Jones 11a][Jones 11b].

In order to develop the effective security and privacy risk assessment approaches for cloud-based systems, DoD should invest in exploring the new research and educational opportunities to explore the synergies between cloud computing and systems engineering. Some example research areas might include investigation of dynamic assessment methods based on periodically iterated static assessments with infrequent changes; and exploration of the "on-demand" assessment approaches compatible with the cloud computing.

2.2.7 Role of System Engineers

In the 1990's, Sheard [Sheard 96] described twelve system engineering roles, which have been occasionally or frequently assumed to constitute the practice of systems engineering, based on her literature review. However, the current systems engineering processes and role assignments are not ideally matched to future cloud computing needs. Systems engineers and systems engineering are at increasing risk of not being able to keep up pace because the field is currently not well enough configured to connect with other disciplines including computer science and software engineering. System engineers need a deeper understanding of advanced computer science technologies in the cloud era (e.g., the subtle but crucial difficulty involved in data consistency and availability at cloud scales, or the deep semantic models of concurrent systems). System engineer roles in the cloud era are becoming multi-dimensional in terms of their new relationships or new ways of working with other disciplines (including computer science and software engineering), for example:

- **System engineers as composers**: taking a higher level view of functionality that can be provided by leveraging the applications being opportunistically developed;
- **System engineers as design space architects**: providing the space in which opportunistic development can work effectively, with minimal aggravations due to communication problems, and leveraging other opportunistic work;
- **System engineers as coaches:** providing an up-to-date understanding of development status and risks for developers;
- **System engineers as quality domain specialists:** evaluating and assessing the system performance, availability, security, safety, privacy etc.;
- **System engineers as the "hardeners":** improving the quality and enforce clean design, once opportunistic development turns out to be valuable;
- **System engineers as educators**: helping to shape the skill sets for the next generation of system engineers.

Recommendations

DoD should invest in conducting a study to reconsider systems engineering roles and processes by mapping the "Transformed Roles of System Engineers in Cloud" (including Composer, Design Space Architect, Coach, Quality Domain Specialist, "Hardener" and Educator identified in this report) to the twelve roles identified by Sheard almost two decades ago. An initial version of such a mapping analysis as shown in Table 2 suggests missing skill sets for current SE roles [Sheard 96] [Boehm94] [Fisher 92] [Rechtin 91].

Table 2: Mapping Transformed Roles of System Engineering in the Cloud to Current System Engineering Roles

	Composer	***Design Space Architect***	***Coach***	***Quality Domain Specialist***	***Hardener***	***Educator***
Requirement Owner	Partial		Partial	Partial		Partial
System Designer	Yes	Yes	Partial	Yes	Yes	Partial
System Analyst	Yes	Yes	Partial	Yes	Yes	Partial
V&V Engineer		Partial	Partial	Yes	Yes	Partial
Logistics/Ops Engineer				Yes		
Glue Among Subsystems	Yes	Yes	Partial			Partial
Customer Interface				Partial		
Technical Manager		Yes	Partial	Partial	Yes	Partial
Information Manager	Partial		Partial	Partial		Partial
Process Engineer	Yes	Yes	Yes	Yes	Partial	Yes
Coordinator		Yes	Yes	Partial		Yes
Classified Ads SE						

Yes: The transformed role can be taken by the existing SE role;
Partial: The transformed role may be partially taken by the existing SE role.

***Twelve systems engineering roles,* SA Sheard - Proceedings of INCOSE, 1996**

Secondly, based on the mapping analysis results in Table 2, we further recommend that systems engineers, and system engineering educators, undertake to augment the existing training with the following new skill sets:

- Understanding the concepts and technical properties of cloud infrastructure and cloud based components;
- Developing processes, methodologies and strategies to configure and integrate cloud infrastructure and cloud based components into a system;
- Developing metrics and measurement methodologies for assessing the quality of cloud-based systems;
- Exploiting cloud infrastructure and cloud based components in effective and efficient resource allocation, cost saving and information management.
- Meeting the challenges of data integrity, security, privacy and other issues emerged in cloud-based systems.

Specific skill sets for each system engineering role are elaborated in Table 3.

Table 3: Recommended New Skill Sets for System Engineering (SE) Roles

SE Roles	Potential Missing Skill Sets
Requirement Owner	● Understand and specify the associated attributes of cloud components, e.g., can the overall goals be achieved with selected cloud components? are there any negative effects if cloud architecture is deployed? (as composer)) [Kherahani11] [Zardari 11] ● Develop, teach and apply the methodologies to define and specify quality requirements for cloud-based components and systems. (as coach, quality domain specialist or educator) [NIST 11a]
System Designer	● Understand cloud architecture and related quality attributes, e.g., scalability, elasticity, etc. (as composer, design space architect, coach, quality domain specialist or educator) [Kherahani11] ● Develop and apply methodologies in cloud based system design; (as composer, design space architect, coach, quality domain specialist or educator) [Kherahani11] ● Maintain clean design without specific “niches” or transient techniques; (as hardener) [Neamtiu 11]
System Analyst	● Develop metrics, measurement methodologies and strategies based on cloud related quality attributes; (as composer, coach, quality domain specialist or educator) ● Develop system feasibility analysis methodologies for the selection and integration of cloud components; (as design space architect)
V&V Engineer	● Develop methodologies and strategies in system V&V with cloud components; (as design space architect, coach or educator) [NIST 11a] ● Perform effective and efficient system V&V and system assurance; (as quality domain specialist) [Neamtiu 11] [Gandea 10] ● Addressing transient issues with cloud components; (as “hardener”) [Neamtiu 11]
Logistics/Ops Engineer	● Understand the impacts of exploiting cloud components in the system, e.g., do they incur or save resources? (as quality domain specialist)
Glue Among	● Develop methodologies for the integration of cloud components in

Subsystems	systems; (as composer, coach, educator) ● Select cloud component vendors and integrate the cloud components with the rest of system components; (as composer, design space architect)[Head 09]
Customer Interface	● Understand the impact of cloud components to system user interface and customer communications, e.g., how to improve system UI to aid in distributed data sharing and to protect data privacy? (as quality domain specialist)
Technical Manager	● Understand cloud related techniques; (as composer, design space architects, coach, educator)[Kherahani11] ● Decision support for orchestration with other IT components; e.g., does cloud incur/save technical efforts in the long-term and short term? (as composer, design space architects, coach, educator and quality domain specialist)[NIST 11a] ● Manage transient issues in cloud techniques; (as "hardener")[Neamtiu 11]
Information Manager	● Understand the impact of cloud to information security, privacy, etc.; (as composer,)[NIST 11a] ● Understand the impact of cloud to distributed information management; (as design space architect, coach, and educator).[Abadi 09] [Agrawal 11]
Process Engineer	● Understand and engineer the integration and certification processes for developing and maintaining cloud computing systems; (as composer, coach, quality domain specialist and educator) [Head 09] ● Optimize resource allocation in cloud based system engineering; (as design space architect) ● Orchestra the integration of cloud components (as "hardener")
Coordinator	● Optimize resource allocation in cloud based system engineering; (as design space architect) ● Policy enforcement (as quality domain specialist)
Classified Ads SE	N/A

2.2.8 Education

The field of systems engineering as defined today needs to adapt to a future of computationally intensive systems. System engineers must be educated in architecting, developing, integrating, assuring, and maintaining systems that are largely defined by their computational behaviors. To handle the shift in the system focus and to facilitate transition to the new system engineering roles, as described in Section 2.2.7, systems engineers must become more versed in the complex computational behavior and software issues faced by the modern computational systems, including formal specification of interfaces and verification of compliance, big data analytics, big data management (including synchronization and retention), concurrency and parallelization of distributed infrastructure, foundational security skills, agile development, etc.

Recommendations

Educators, with collaboration or support from the academic community, should determine how best to restructure the current training/education curriculum for system engineers. In general, new curriculum should include basic concepts and technical properties of cloud infrastructure

and cloud-based components, policy and compliance issues for the cloud, and quality metrics for cloud-based systems, as well as deeper training in fundamentals of computer science and software engineering. Institutions should consider developing *hybrid* cross-disciplinary degree programs to train students at the intersection of traditional systems engineering and computer science.

Throughout this report, we have also identified several recommendations that would impact the restructuring of the System Engineering curriculum:

- In Section 2.1.3., “Data Analytics”, we recommended that a module about how to develop and integrate data analytics capabilities (e.g., data mining, data integrity, security and privacy, etc.) into the cyber-physical systems to be added into the system engineering curriculum.
- In Section 2.2.1, “Lifecycle models and development process”, we recommended that a new process for system development, that merges the traditional waterfall model and the agile models, should be considered. If this recommendation is accepted, the overall curriculum should be centered on the new development process. We also discussed that new strategies, processes, and methodologies for composing cloud-based components into a system and for performing trade-off studies of design spaces and service vendors should be developed. The Architecture/Design development module/field should teach these techniques.
- In Section 2.2.3, “Modeling Notations”, we recommend a new system engineering modeling language (to replace SysML). The curriculum should include the teaching of the new modeling language.
- In Section 2.2.7., “The Role of System Engineers”, we recommended a set of new skillsets needed by the new System Engineers Roles.

2.3 How Systems Engineering Can Exploit the Cloud

Cloud computing and related computing technologies provide tremendous opportunities to advance and augment systems engineering. We have already address many ways in which cloud computing opens up new possibilities for system functioning. We now end with a brief discussion of ways in which cloud computing systems, and particularly platforms that support social networking and crowdsourcing technologies, might transform and improve complex systems engineering.

Current web advancement and cloud technologies have produced major innovations in modern social applications that support group interactions: social networking systems, collaborative environments, etc. New paradigms for crowdsourced problem solving are also emerging (a paradigm that leverages a large group of people to solve problems require large amounts of effort when using traditional methods).

The myriad of modern social applications, in turn, have made the collaborative environment more accessible, which consequently promotes the adoption of social software engineering, defined as community-driven creation, management, deployment, and use of software in online environments [Hammouda 08]. Meanwhile, crowdsourcing aims at tasks that humans are good at solving, but not computers, such as performing transcriptions, and can yield results that are superior to the ones produced by (complex) systems developed to do such tasks at a lower cost [Callison 09].

We now recognize that software development, and system development more generally, is to a significant degree a *social decision making* activity – involving collaborations of large teams with frequent exchange of knowledge among members (e.g. ideas, technical information, work status and progress). Begel et. al. at Microsoft Research have reported on the different ways software development efforts can benefit by automatic construction of social networks to ensure that the right developers are coordinating around shared and interdependent technical issues [Begel 10]. We believe that there are now significant opportunities to bring such ideas to systems engineering.

Recommendations

DoD should expand research on social technologies for systems engineering. The paper by Begel et. al., and current SERC research by UVa and Fraunhofer Center, could be a starting point for work on social network technologies. Research on crowdsourcing of such activities as verification and validation should also continue to be pursued, and not just for software but for systems. This challenge has been recognized by DoD: in late 2011, DARPA funded a project called Crowdsourced Formal Verification which aims to make formal program verification more cost-effective by transforming the verification activity into game instances, where each game instance is then made accessible to the “crowd” via the web, and the solutions of the game are

then merged together to reason about a specific program property. Understanding how such approaches can benefit systems engineering is an interesting and important problem.

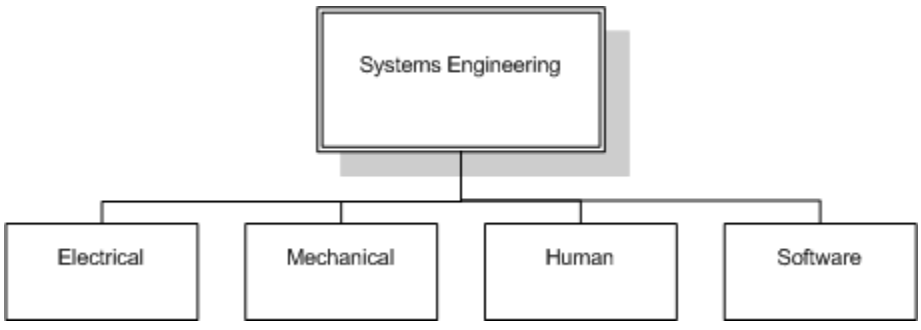
Research issues include the quality of crowdsourced tasks, and of the work products produced. How does one determine that the results of a crowdsourced task are trustworthy, and that they are free from any tampering schemes? Amazon's Mechanical Turks (MT) is a marketplace for crowdsourced tasks, in which quality assurance of tasks is mainly achieved by first specifying the required qualification of the "crowd." This qualification tends to revolve around rating the participants, usually based on their number of tasks completed. However, tasks offered in MT tend not to be of a high critical nature. Thus it remains as a research challenge to investigate other techniques or approaches for assuring the quality of crowdsourced tasks that are part of the development of highly critical systems.

3. The Era of Computational Systems

Most of this report has focused on a set of key technical topics which are most affected by, or most affecting, system engineering in a world increasingly built on cloud technologies. The pace of change in these areas is deeply supportive of our claim in Section 1 regarding the disruptive effects of not only cloud computing, but the *ongoing revolution in computing* more broadly. In this section, we focus not on individual technical topics but on the overall effect to the discipline of systems engineering.

In Section 1, we briefly discussed the Apple iPhone as an example of an industry-wide sea change brought on by effective exploitation of cloud computing and other revolutionary computing technologies. Such changes are hardly confined to telephony. They are the same for everything from fighter jets to intelligence systems. We have entered an era of *computational systems:* systems with complex physical, and often cognitive and social elements, the principal functions of which are substantially defined by software and computing hardware elements. Computing has so changing the nature of technology that essentially all major systems are now, or will have to become, *computational systems*. The engineering methods by which systems are developed, and the roles and organization of traditional engineering disciplines in design, development and production are also having to change accordingly. These changes are producing real disruptions, but they also create stunning new opportunities. These changes will be felt particularly in systems engineering and in the related sub-disciplines of computer science.

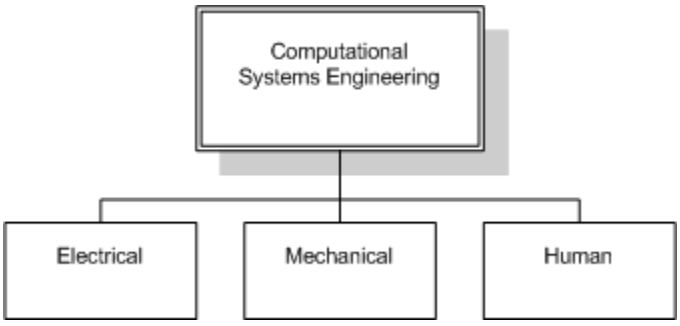
Systems engineering will be particularly affected because its fundamental model of system partitioning, and the role that it has traditionally played based on that model, are no longer well matched to the needs of computational systems. The traditional systems engineering view is that, like other specialized engineering issues (electrical, mechanical, etc.), computing can be modularized, with software and computing partitioned into system sub-modules, the interfaces of which are specified by systems engineers and implementation of which are to be developed mainly by software engineers. The following Figure illustrates this schema.



This model is no longer operative for many systems. The shift to computation as the *principal* enabling and integrating aspect of systems means that formulating requirements, designing, realizing and evolving *computational behavior*, is now such a dominant issue that it must be addressed at the overall systems level. It can no longer be relegated to consideration only within a "software" sub-module of a traditional system or traditional systems engineering process.

To continue to operate on the basis of the traditional model, with software/computing as but a sub-module, would be to make the same mistake that companies made that continued to treat cellular telephony as primarily an RF issue and computing as a mere module in devices and development processes. Some manufacturers of film cameras in the consumer marketplace similarly failed to adapt. Having an embedded microprocessor and software-based features was no longer enough when *the camera became a computer.* The real opportunity, as Apple saw, was to conceive and realize cell phone telephony as a truly massive computational ecosystem leveraging major advances in computer hardware, software platform design, human computer interaction, and cloud computing infrastructure. Systems no longer simply have computers. In many cases they *are* computers, with highly sophisticated and extensive peripheral devices

What is now needed is a new understanding of what it will take to define and develop major *computational* systems, with corresponding changes both within, and in the relations among, traditional engineering disciplines. *Comprehensive consideration of computing and software must now move to the top level of concern for major systems across all engineering domains.* This change will require significant adjustments in several disciplines, including both systems engineering and computer science, particularly software engineering, but also in such areas as cyber-physical, and cyber-human systems. The following figure illustrates the change. It is significant enough that it requires a new function--and perhaps even eventually a new hybrid discipline--of what this report calls *computational systems engineering (CSE).*



3.1 Computational Systems Engineering (CSE)

The overall challenge for systems engineering and computer science is to develop a new approach to systems in which consideration of the whole *system,* its driving *computational* behavior, and the *software* and hardware that express it are integrated into a computational

systems engineering function. To see how this might happen, one must look beyond systems engineering, per se, to computer science and its own “systems engineering” sub-disciplines.

The field of computer science largely created the revolution in computing, through advances in computational theory, computing machinery, systems, languages, compilers, algorithms and protocols, data structures, integration of computing and physics through sensors and actuators, and human-computer interaction and social computing. Several areas of computer science will have roles in a discipline of computational systems engineering: cloud computing, cyber-physical systems, machine learning, social computing, software engineering, etc.

Software engineering deserves a particular mention, because of its status as a *systems engineering* sub-discipline of *computer science.* It addresses the systems engineering of computational behaviors and their expression and representation in the form of software. It is not a complete instantiation of the discipline of systems engineering, but it has developed deep knowledge and a practice of systems engineering for computational behaviors and software, in particular, apart from the discipline commonly called systems engineering.

Even a cursory comparison of definitions of software engineering and systems engineering shows that their fundamental concerns are aligned. They both address systems analysis, requirements, specification, architecture, integration, cost, schedule, human factors, test, evaluation, evolution, etc. What distinguishes them has been their attention to fundamentally different classes of systems. Software engineering and related areas in computer science focus on *computational behaviors*, either in software products or in software components of larger non-software systems. Systems engineering focuses on the broader class of *general* systems, assuming that computing and software can be compartmentalized and treated as sub-component specialty concerns in systems that need them.

The outcome of these historical developments is that we find ourselves with two systems engineering communities *neither* of which is fully adequate to the challenges we now face. Practitioners of traditional systems engineering have a broad inclusive understanding of and ability to deal with a wide range of systems issues, but generally lack sufficient expertise in computing to deal directly with the complex computational behavior and software issues at the very heart of modern computational systems. Practitioners in software and related areas of computer science understand the systems engineering of computation but lack knowledge in many other critical areas in which trained systems engineers are highly competent. Parallel development and the co-existence of these respectively broad and deep communities has persisted for many years. Now, however, with the emergence of *computational systems* as a dominant and revolutionary form of systems in the 21st century a new synthesis is needed.

It is first vitally important to understand that neither of these systems engineering communities is going away. Both are needed. Software engineering has been an enduring sub-discipline of computer science for almost half a century, ever since the 1968 NATO Software Engineering

Conference. In terms of employment, it is now larger than any other field of engineering and it rivals all other fields combined. It is also growing far faster than any other field of engineering except for biomedical engineering, which it dwarfs in absolute size. Systems engineering has a similarly distinguished past and a bright future.

Rather, the complementary bodies of knowledge of these two fields, tied together by common interest in, and understanding of, complex synthetic systems, their properties, and the means by which people can develop them effectively, must now be integrated much more effectively. There is a powerful case for investments in initiatives, rooted in a recognition of the co-equal statures and complementary roles of the two traditions, to produce the novel approaches that are needed to deal with the complex *computational systems* of the future.

3.2 CSE Findings and Recommendations

In light of these observations, we have formulated a final set of findings and a recommendation specific to the development of a hybrid discipline of Computational Systems Engineering (CSE).

CSE Finding #1: We have entered a new era of *computational systems.* These are systems in which software and computation are *principal* design issues and source of value and risk, and in which computing is pervasively and deeply integrated with the physical, cognitive, social and environmental aspects of complex systems.

CSE Finding #2: No established systems engineering discipline, whether the traditional system engineering field, nor the current systems engineering sub-disciplines of computer science (and particularly software engineering but also including cyber-physical systems), is fully prepared to deal with the complexities of modern computational systems.

CSE Recommendation #1: The US Department of Defense, alone or in collaboration with other agencies, should invest in research on a hybrid *computational systems engineering* function for systems of the future. This function would recognize computation as a pervasive, crosscutting, dominant concern at the highest levels of system definition and development, and would involve professionals from both traditional systems engineering, computer science (particularly software engineering and cyber-physical systems), and cognitive science, human factors and economics. Among other things an initiative in this area would:

- conduct fundamental research to produce new knowledge of computational systems
- develop a comprehensive body of knowledge in computational systems engineering
- disseminate this knowledge through both professional and academic channels
- engage real computational systems in key domains, such as healthcare and defense

Key **jump-start activities** for such an initiative could include the funding of university-based research centers to conduct fundamental research and to develop curricular materials in this

area; and the funding of international workshops and conferences that would bring together constructive individuals from the constituent communities to develop and articulate ideas and plans for such a new discipline as a novel synthesis drawing from the contributing disciplines.

One expected **outcome** would be, not the elimination or disadvantaging of any participating community, but rather new directions for reconfiguring them around their areas of strength, in light of the realities of the rapid dawning era of *computational systems,* as a way to preserve and enhance their relevance and value to our society going forward.

For systems engineers and for the field of systems engineering, in particular, such initiatives would breathe significant new life into the field by connecting it more tightly with the main driver of innovation, need, value and risk in the economy today -- with advanced computing systems such as cloud computing, and with the historically separated but deep and important systems engineering sub-disciplines of computer science and engineering.

Bibliography

[Abadi 09] Abadi, D.J., Data Management in the Cloud : Limitations and Opportunities. *Data Management*. IEEE Data Engineering Bulletin, 32(1), March 2009. p.1-10. Available at: http://sites.computer.org/debull/A09mar/A09MAR-CD.pdf#page=5.

[Armbrust 09] Michael Armbrust, Armando Fox, Rean Griffith, Anthony D. Joseph, Randy H. Katz, Andrew Konwinski, Gunho Lee, David A. Patterson, Ariel Rabkin, Ion Stoica and Matei Zaharia, Above the Clouds: A Berkeley View of Cloud Computing, EECS Department, University of California, Berkeley, Technical Report No. UCB/EECS-2009-28, February 10, 2009.

[Agrawal 11] Divyakant Agrawal, Sudipto Das and Amr El Abbadi. Big Data and Cloud Computing: Current State and Future Opportunities, *EDBT 2011*, March 22–24, 2011, Uppsala, Sweden.

[Alford 09] Ted Alford and Gwen Morton, Booz Allen Hamilton. The Economics of Cloud Computing: Addressing the Benefits of Infrastructure in the Cloud, 2009.

[Avizienis 04] A.A. Aviˇzienis, J.-C. Laprie, B. Randell, and C. Landwehr. Basic concepts and taxonomy of dependable and secure computing. IEEE Trans. on Dependable and Secure Computing, 1(1):11–33, Jan. 2004.

[Avritzer 95] A. Avritzer and E.J.Weyuker. The automatic generation of load test suites and the assessment of the resulting software. IEEE Trans. on Software Engineering, 21(9):705–716, Sept. 1995.

[Basili 04] V.R. Basili, P. Donzelli, and S. Asgari. A unified model of dependability: Capturing dependability in context. IEEE Software, 21(6):19–25, Nov. 2004.

[Baresi 04] Luciano Baresi, Carlo Ghezzi, and Sam Guinea. 2004. Smart monitors for composed services. In Proceedings of the 2nd international conference on Service oriented computing (ICSOC '04). ACM, New York, NY, USA, 193-202.

[Begel 10] Andrew Begel, Robert DeLine, Thomas Zimmermann, Social Media for Software Engineering, FoSER, 2010.

[Boehm 94] Boehm, Barry. Integrating Software Engineering and Systems Engineering. *Systems Engineering*, Vol. 1, No. 1, 1994.

[Callison 09] C. Callison-Burch. Fast, cheap, and creative: Evaluating translation quality using amazon's mechanical turk. In Proceedings of EMNLP, pages 286–295, 2009.

[Chen 95] M.H. Chen, P. Garg, A.P. Mathur, and V.J. Rego. Investigating coverage-reliability relationship and sensitivity of reliability estimates to errors in the operational profile. Computer Science and Informatics Journal – Special Issue on Software Engineering, 25(1):4–16, Sept. 1995.

[Colarelli 02] D. Colarelli and D. Grunwald, BMassive arrays of idle disks for storage archives, in Proc.ACM/IEEE Conf. Supercomput., Los Alamitos, CA, 2002, DOI: 10.1109/SC.2002.10058.

[DePompa 11] Barbara DePompa, Data Center Optimization, 1105 Government Information Group, 2011.

[Donzelli 05] P. Donzelli, M.V. Zelkowitz, V.R. Basili, D. Allard, and K.N. Meyer. Evaluating COTS component dependability in context. IEEE Software, 24(4):46–53, July 2005.

[Fisher 92] Fisher, Jack. Systems Engineering for Commercial Space Programs*. Proceedings of INCOSE*, 1992.

[Gandea 10] Candea, G., Bucur, S., and Zamfir, C.. Automated software testing as a service. *Proceedings of the 1st ACM symposium on Cloud computing*, 2010.

[Goo 12] "Google JSON/Atom Custom Search API," http://code.google.com/apis/customsearch/v1/overview.html, 2012.

[Harman 07] Gerardo Canfora Harman and Massimiliano Di Penta, New Frontiers of Reverse Engineering. In 2007 Future of Software Engineering (FOSE '07). IEEE Computer Society, Washington, DC, USA, 326-341, 2007

[Hammouda 08] Imed Hammouda, Jan Bosch, Mehdi Jazayeri, Tommi Mikkonen: First International Workshop on Social Software Engineering and Applications (SoSEA 2008), In: Proceedings of the 23rd IEEE/ACM International Conference on Automated Software Engineering (ASE 2008), 2008, pp. 531-532

[Hassan 01] Hassan, A.E.; Holt, R.C.; , "Towards a better understanding of Web applications," Web Site Evolution, 2001. Proceedings. 3rd International Workshop on , vol., no., pp. 112- 116, 10 Nov. 2001

[Head 09] Michael R. Head, Anca Sailer, Hidayatullah Shaikh, Mahesh Viswanathan, Taking IT Management Services to a Cloud. *Proceedings IEEE International Conference on Cloud Computing*, 2009.

[Huang 06] L. Huang. Software Quality Analysis: A Value-Based Approach. PhD dissertation, University of Southern California, May 2006.

[Huebscher 08] Huebscher, M. C., and McCann, J. A. A survey of autonomic computing – degrees, models, and applications. ACM Computer Survey, (2008), 40(3), pp. 7-28.

[Jones 11a] Jones, R.A.; Nguyen, T.V.; Horowitz, B.M., "System-Aware Cyber Security, International Conference on Information Technology: New Generations (ITNG), April 2011, pp. 914-917.

[Jones 11b] Jones, R.A.; Nguyen, T.V.; Horowitz, B.M., "System-Aware Security for Nuclear Power Systems", IEEE International Conference on Technologies for Homeland Security (HST), Nov 2011, pp. 224-229.

[Kallepalli 01] C. Kallepalli and J. Tian. Measuring and modeling usage and reliability for statistical web testing. IEEE Trans. on Software Engineering, 27(11):1023–1036, Nov. 2001.

[Kaliski 10] Kaliski, Burton S., and Pauley, Wayne.Toward. risk assessment as a service in cloud environments. Proceedings of the 2nd USENIX conference on Hot topics in cloud computing HotCloud'10, 2010.

[Kherahani 11] Monika Kherajani and Ajit Shrivastava, *Impact of Cloud Computing Platform Based on Several Software Engineering Paradigm*, International Journal of Advanced Computer Research, Volume 1, Number 1, September, 2011.

[Kiran 11] Kiran, Mariam, Jiang, Ming, Armstrong, Django J., Djemame, Karim. Towards a Service Lifecycle Based Methodology for Risk Assessment in Cloud Computing. IEEE Ninth International Conference on Dependable, Autonomic and Secure Computing (DASC), December 2011, pp.449 – 456.

[Laprie 92] J.-C. Laprie, Editor. Dependability: Basic Concepts and Terminology, Dependable Computing and Fault Tolerance. Springer-Verlag, New York, 1992.

[Lee 10] Edward A. Lee, "Disciplined Heterogeneous Modeling," Invited Keynote Talk, Models 2010, Oslo, Norway, October 6-8, 2010. Slides available at http://models2010.ifi.uio.no/material/Heterogeneous_Models.pdf.

[Lindvall 08] Lindvall, M.; Muthig, D., "Bridging the Software Architecture Gap," Computer , vol.41, no.6, pp.98-101, June 2008

[Lyu 92] M.R. Lyu and A.A. Avi˘zienis. Assuring design diversity in N-version software: A design paradigm for N-version programming. In J. F. Meyer and R. D. Schlichting, editors, Dependable Computing for Critical Applications 2. Springer-Verlag, New York, 1992.

[Lyu 95] M.R. Lyu, Editor. Software Fault Tolerance. John Wiley & Sons, Inc., New York, 1995.

[Matthew 10] Matthew Wheeland, Cloud Computing Is Efficient, But It's Not Green – Yet. GreenBiz . March 30, 2010 . http://www.greenbiz.com/blog/2010/03/30/cloud-computing-efficient-but-not-green-yet?page=0%2C

[Farber 11] Michael Farber, Mike Cameron, Christopher Ellis,Josh Sullivan, Massive Data Analytics and the Cloud A Revolution in Intelligence Analysis, Booz Allen Hamilton, 2011.

[Musa 93] J.D. Musa. Operational profiles in software reliability engineering. IEEE Software, 10(2):14–32, Mar. 1993.

[Neamtiu 11] Lulian Neamtiu and Tudor Dumitras, *Cloud Software Upgrades: Challenges and Opportunities*, IEEE International Workshop on the Maintenance and Evolution of Service-Oriented and Cloud-Based Systems, 2011.

[NIST 11a] NIST Cloud Computing Standards Roadmap Working Group, *NIST Cloud Computing Standards Roadmap*, July 2011.

[NIST 11b] NIST, The NIST Definition of Cloud Computing, September 2011.

[NIST 11c] NIST, US Government Cloud Computing Technology Roadmap Volume 1 Release 1.0 (Draft), November 2011

[NSC 06] Nova Scotia Canada, Department of Justice, New Legislation to Protect Privacy. May 5, 2006. http://www.gov.ns.ca/news/details.asp?id=20060505006

[OCTAVE 10] OCTAVE: Operationally Critical Threat, Asset, and Vulnerability Evaluation. Accessed March 2010. http://www.cert.org/octave/ .

[Parveen 10] T. Parveen and S. Tilley, When to Migrate Software Testing to the Cloud? International Conference on Software Testing, Verification, and Validation Workshops, pp. 242-427, May 2010.

[Pizette 10] Lawrence Pizette and Jennifer Manring, Cost and business case considerations, The MITRE Corporation, 2010.

[Rechtin 91] Rechtin, Eberhardt. *Systems Architecting, Creating and Building Complex Systems*, Prentice-Hall, Inc., New Jersey, 1991.

[Sahai 02] Sahai, A., Machiraju, V., Sayal, M., van Moorsel, A, and Casati, F. Automated SLA monitoring for web services. In M. Feridun et al. (eds.), Lecture Notes in Computer Science, 2506, Springer (2002), pp. 28-41.

[She 10a] W. She, I-L. Yen, B. Thuraisingham, and E. Bertino. Policy-driven service composition with information flow control. In International Conference on Web Services, pages 50–57, Florida, July 2010.

[She 10b] W. She, I-L. Yen, and B. Thuraisingham. Enhancing security modeling for Web services using delegation and pass-on. In International Journal of Web Service Research, pages 1-21, Vol. 7, No. 1, 2010.

[Sheard 96] Sheard, Sarah A. The Value of Twelve Systems Engineering Roles. *Proceedings of INCOSE*, 1996.

[Spang 11] Bernie Spang, Workload Optimized Systems – Don't Waste Resources on Inefficient Systems. The Smarter Computing Blog, December 16, 2011. http://www.smartercomputingblog.com/2011/12/16/workload-optimized-systems-dontwaste resources-on-inefficient-systems/

[Twi 12] "Twitter Developers," https://dev.twitter.com, 2012.

[Vorster 05] Vorster, A., and Labuschagne, L. A framework for comparing different information security risk analysis methodologies. In Proceedings of the South African Institute for Computer Scientists and Information Technologists, (2005), pp. 95-103.

[Zardari 11] Shehnila Zardari and Rami Bahsoon, *Cloud Adoption: A Goal-Oriented Requirements Engineering Approach*, Workshop on Software Engineering for Cloud Computing, SECLOUD 2011.

Appendix A: Research Opportunities

Prior sections of this report have identified a number of gaps in current capabilities related to Cloud Computing, some of which will require focused research in order to develop the methods, processes, and tools to address the needs of contemporary and future cloud-based systems. This appendix summarizes some of the key areas where research is needed and highlights initial work.

A.1 Security and Privacy

To implement dynamic security and privacy risk assessment approaches [Kaliski 10] [Kiran 11], important systems engineering research directions include:

- Autonomic systems, which have the ability to measure their environment and then adjust their behavior based on goals and the current context, have to be both reactive and proactive [Huebscher 08]. Enabling such functionality requires the investigation of sensors and an autonomic manager that analyzes risks and implements changes. Automated measurement and analysis is the basis for delivering risk assessment as a service; automated adjustment is a further (and much more complex) extension. For risk assessment, research is needed on the sensors that would collect relevant data in real time in a cloud environment. Research is also required on how to define measurements to support the viewpoints of multiple tenants and for service providers, different than previous approaches focusing on a single stakeholder. As a starting point in this direction, adjustments could be made in the conventional way based on risk reports: risk reports (or risk analysis assessment) usually define what warning signs to look for as well as the prevention measures (e.g. mitigation plans) that defines what adjustments to make.
- Automated Service Level Agreements (SLAs) require a dictionary, an SLA specification language, and a correlation engine [Sahai 02]. Risk assessment as a service could apply the same principles where the dictionary holds the risk assessment rules and asset valuations based on data entered by the tenant. This would provide the basis for a weighted scoring method such as those that are in OCTAVE [OCTAVE 10] [Vorster 05].
- CloudAudit has begun defining a directory/namespace for security audits and assessments that includes frameworks such as PCI DSS, HIPAA, COBIT, ISO 27002, and NIST SP800-53. Such a directory/namespace offers a common language that both tenants and service providers can use to collect information in support of continuous assessment. Further research in computational system engineering may be needed on how to express the rules in a cloud setting.

A.2 Big Data: Underlying Technology

- **Data security in virtualization.** The increasing use of virtualization in cloud computing environments, has been driven by improved server utilization rates, increased operational efficiency, and the ability to leverage desktop virtualization to centrally control operating systems and meet security requirements. However, despite the many benefits of virtualization as a tool to help optimize data centers, it has some negative security implications government agencies must address.
- **Seeking a balance between fault tolerance and performance.** Maximizing fault tolerance typically means carefully checkpointing intermediate results, but this usually comes at a performance cost in cloud [Abadi 09].
- **Think green.** There is a huge demand for cutting the growing dependence of data centers on energy consumption. Data centers developed with large greenhouse gas reduction solutions, can have a dramatic impact in reducing warm houses gas emissions, and become a necessary basis for the future Data Cloud [Matthew 10].

A.3 Dependability

- **End-to-end Monitoring of Cloud-based Systems.** The Fraunhofer Center is conducting research to enable a more holistic view of cloud-based systems' behavior, which is to be obtained through an end-to-end monitoring scheme. Leveraging such information has the potential for significantly improving the support for the maintenance and operation of Cloud-based systems, including providing better health monitoring and repair mechanisms, as well as the ability to capture key metrics (e.g., usage, utilization) to enable better resource optimization and cost estimation. Achieving end-to-end monitoring is not trivial – it involves observing/capturing the interaction between user and the cloud system as well as what is going on within the cloud system itself (e.g. what services are used, how they are used, the frequency in which they are used, etc.), and it has to deal with issues such as merging and synchronizing the multiple logs/traces, identifying and removing noise, etc. Fraunhofer currently has a toolset to monitor software and web applications, and plans to work on extensions in scope and scalability for cloud systems.
- Investigate the ability to measure cloud application dependability in the target or intended usage environment [Avritzer 95] [Chen 95] [Donzelli 05] using real-world operational conditions (e.g., through instrumentation) aimed at developing continuous monitoring and proactive recovery strategies for enhancing the dependability and survivability of emerging safety- and/or mission-critical cloud-based applications. For example, automatically re-calibrate sensors and actuators adaptive to system context.
- Develop methods of enhancing the dependability of cloud-based systems by moving beyond product and process diversity [Lyu 92] [Lyu 95], examining service diversity as measured by their respective quality attributes for each environment, and selecting the set of services with the maximal diversity.

- The web service environment is a typical open system across multiple organizational boundaries and, hence, security management of web services is very challenging. Various models have been proposed to address security issues in such an environment. However, most of these focus on individual web services, and do not consider service composition. In a composite service, sensitive information is passed from one service to another through the service chain, which may cause information leakage. Investigating an innovative access control model to empower the services in a service chain to effectively control the flow of their sensitive information [She 10a] [She 10b].

A.4 Social Systems Engineering

Kevin Sullivan's group at the University of Virginia is developing a cloud-based software service that infers and supports the social networks needed for effective collaborative decision making and trade-space analysis for complex systems. The integral treatment of modularity in design structures *and* in the social networks around them promises to identify significant opportunities for improved process and product flexibility, cost and schedule, and operational performance. This work is currently being developed in part in a SERC project for the U.S. Army (RT-33, with Army contingency bases as the target systems), in collaboration with the Fraunhofer Center, and in a collaboration between University of Virginia, Carnegie Mellon University, the University of British Columbia and eventually a major DoD aerospace contractor. The systems considered so far include software elements for the Apache Helicopter and Army contingency bases. Kevin Sullivan proposes research to extend this work into a general social systems engineering design and trade analysis environment. Additional details are provided as Appendix B.

A.5 Metrics for Cloud-Based Systems

This report has called attention to the need for new metrics and measurements that are tailored for cloud-based system and its unique characteristics (e.g., multi-tenancy, dynamic allocations of hardware and software resources). In addition, these new metrics and measurements need to be meaningful for the various cloud system stakeholders, including: cloud providers (both management and technical staff), cloud customers, and cloud brokers (i.e., middleman to cloud customers and providers) – each stakeholder would respond differently to a different set of metrics and measures. Fraunhofer Center is in the process of collaborating with a major cloud service provider to better define availability and throughput metrics, so as to reflect more accurately the state of the system (which is of interest to their technical staff) and the satisfaction of their customer (which is of interest to management). This process is supported by the application of Fraunhofer's GQM+Strategies methodology, a measurement planning and analysis approach, developed by Fraunhofer CESE and IESE. GQM+Strategies provides a framework for aligning the goals and strategies of different stakeholders, and for deriving measurements sufficient and meaningful to each, so as to allow an organization to assess both technical and business progress.

Appendix B: Research Proposal – Social Decision Network Analysis for System Engineering

Ge Gao & Kevin Sullivan

Department of Computer Science
University of Virginia
Charlottesville, VA 22904 USA
{gg5j,sullivan} @virginia.edu

Abstract—Systems engineering outcomes depend on how well networks of people collaborate to formulate and resolve evolving networks of decisions. Teams underperform today due to a lack of support for collaborative, networked decision making. We propose an approach to improving the performance of such teams with technology to support social construction of and coordination through social decision networks (SDNs). An SDN models a set of decisions linked by relationships, the association of individuals with decisions, and social links necessary for effectively making shared and coupled decisions. Our results and contributions include a precise model of SDNs, a Web 2.0 system enabling groups of people to develop and use SDN models, tools for inferring required social networks from decision networks and participant associations, and tools for network analysis, e.g., of modularity in decision and inferred social networks. We are conducting studies with external partners to develop and evaluate this approach.

Keywords-Decision networks, Social networks, Design

I. PROBLEM AND MOTIVATION

In software and software-intensive systems engineering, networks of people coordinate to formulate and resolve networks of decisions, in dynamic environments, to achieve goals. Examples of such decisions include what architectures to use, features to provide, languages to use, bugs to fix. Systems are products of collaborative decision making by teams of people. Problems occur when teams do not clearly identify important decisions and relationships among them, and when they do not coordinate adequately to ensure that shared and related decisions are made well [1]–[3].

Our aim is to improve system quality and productivity by improving the performance of social networks in systems engineering decision tasks. To this end, we have developed a new design space modeling and social networking technology to improve networked decision making in systems engineering. This technology is based on a new model: the social decision network (SDN). An SDN connects the members of a social network to decisions in a decision network then expresses the social links that are necessary for coordination around shared and coupled decisions.

II. BACKGROUND AND RELATED WORK

Our work builds on diverse foundations. The first is work on decision-oriented abstraction. Examples include work on design rationale capture [4] [5], architecture [6]–[8], and on software modularity and evolvability [9]. The second is work based on Conway's Law [10]. Examples include Cataldo and Herbsleb's use of semantics dependencies between files and knowledge of who works on each file as a basis for inferring social coordination requirements [11]; work by Begel et al. on computing social links to improve coordination in changing related source code artifacts [12]; and work by Baldwin and Clark linking technical dependencies in design to task and industrial interaction dependencies in computer design [13].

III. APPROACH AND UNIQUENESS

Prior work on decision abstraction has not emphasized social network implications. Prior work on computation of social networks from technical dependencies and participant associations has not adopted decision networks as the underlying technical network model. The uniqueness and the power of our approach come from combining these ideas.

At the heart of our approach is the social decision network (SDN) model. An SDN models a design space, selected points in this space, and the structure of a team that will make a set of decisions to solve some problem. The key components of an SDN model are a social network—in which people (or roles) are connected by links representing communication and coordination needs; a decision network; and a participant map associating people with decisions. A decision network, in turn, comprises a set of variables and a set of n-ary relationships among variables. Variables are designated as decision, environment or outcome variables. A decision variable models a choice that decision makers control. Assigning a value to a variable models a choice. The value of an environment variable represents a condition that the decision makers do not control (e.g., requirements). The value of an outcome variable is determined by values of other variables and often represents a performance measure (e.g., achieved quality attribute) implied by prior choices. Relationships express how values of variables should relate to each other. For example, a relationship might indicate that an affirmative decision to perform logging will reduce performance. A full or partial valuation of the variables in an SDN represents a point or subspace of a modeled design space.

A key idea in our work is that, from a combination of a decision network and participant map, we can compute required social network links. If people share a decision, they must coordinate and so should be linked. Coordination requirements also arise when people are involved in different but related decisions. The main algorithm in our work is one that computes a labeled social network as a hyper-graph over participants, where the labels on the hyper-edges indicate the decisions and dependencies that require the links.

To help us evaluate these ideas, we have developed a prototype tool, The Decider, providing (1) a web service and client application for social development and evolution of decision networks and participant maps; (2) the ability to compute and support social networks for effective coordination; (3) tools for visualizing and analyzing decision and social networks; and (4) means for connecting SDNs to external data and processes (in progress, e.g., to compare required versus actual social networks). The Decider uses Web 2.0 technology (REST, Ajax, Comet, etc.) for multiparty, online evolution and use of SDNs. Updates to shared elements are immediately visible to all participants.

We comment on two additional characteristics that distinguish our work. First, the choice of decision-based abstraction is meant both to provide a natural language for expressing a diverse range of system concerns and relationships, and to support the participation of a far broader range of stakeholders in software development process. Decisions enable natural representation of diverse concerns, and are understandable by both technical and non-technical personnel. Second, in comparison with the earlier formal work of Cai and Sullivan on design spaces [9], in which relationships were represented as logical constraints over finite domain variables, we link relationships to the variables they influence, while using natural language to describe their semantics. We thus trade precision and sound and complete computation of all inter-variable dependencies for far greater expressiveness, ease of use and fluid development, and computational scalability.

IV. RESULTS, CONTRIBUTIONS AND RESEARCH PLANS

Our preliminary work has produced several contributions and results to date. We developed the SDN model. We have implemented and deployed a RESTful web service and Web 2.0 tool that enables groups of people to develop, analyze and use SDNs. Figure 1 presents a screen shot of a small part of The Decider interface for an example SDN that we developed to model how breakdowns in communication led to conflicting design decisions and failure of the Mars Climate Orbiter Mission. In this case, unrecognized inconsistency in decisions about what units to use for physical quantities led to computational errors that resulted in the loss of the spacecraft. Our tool includes algorithms for computing social networks and analyzing social and decision networks. It supports novel visualizations of network analysis results, with a new form of interactive design structure matrix (DSM) and a related social structure matrix (SSM). Figure 1 presents a small set of variables, two participants mapped to decisions, a relationship among decisions, and the resulting, computed SSM. These visualizations are interactive in the sense

that clicking on dependence marks in them reveals the underlying relationships that account for them.

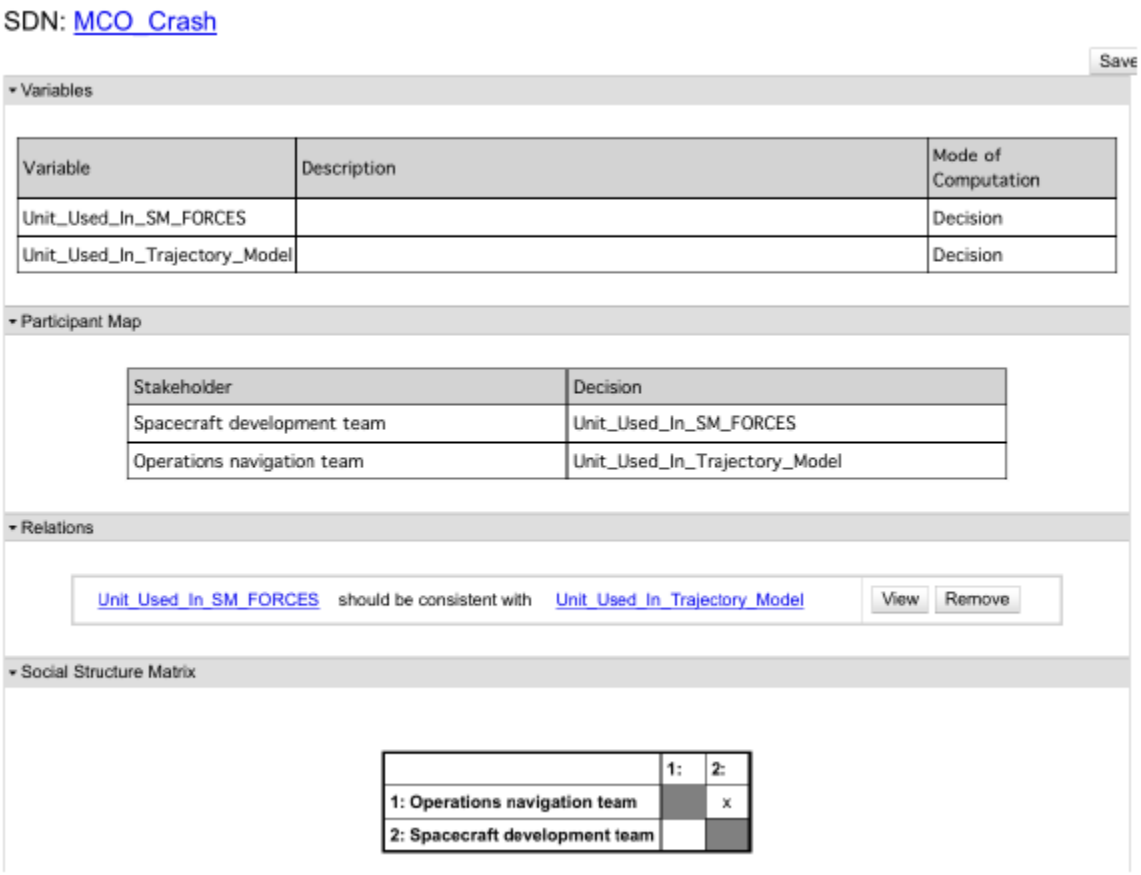


Figure 1. Screen shot of a simple SDN for the failed Mars Climate Orbiter

We are conducting early formative evaluation of our work by building and analyzing models for our own purposes. For more objective, summative evaluation, we have engaged two external groups in projects using this technology. First, we are working (with colleagues at the Fraunhofer Center for Experimental Software Engineering, under a contract from the Systems Engineering Research Center) as part of a team funded by U.S. Army whose goal is to improve how decentralized units of the Army work together to develop temporary Army bases. In the second project, joint with the Carnegie Mellon University Software Engineering Institute and the University of British Columbia, we are working to understand how models such as ours can improve the outcomes of complex architecture projects for cyber-physical systems in the

aerospace domain. These projects are also providing an empirical basis of understanding where the model and tools fall short and can be improved.

Key areas of interest for future work include the following: First, we need to leverage type theory to develop a type system for SDN variables. A type system will improve expressiveness, composability, scalability and abstraction, and composition of complex SDN models. Second, we need ways to enrich SDN models with behaviors, e.g., to obtain data updates from external sources. Third, we need to extend SDN technology to connect with existing system engineering tools, artifacts and workflows (email feeds, artifacts, etc). Third, we plan to pursue the analysis of mismatches between required and actual social networks, as well as identification of counterproductive network patterns. Version control is also needed to support large-scale concurrent model development. Finally, significant experimental validation is needed.

Representing complex systems as sets of system parameters and sets of relationships is an idea at the foundation of the field of systems engineering. Linking social networks to underlying decision networks is a promising next step in the evolution of the field. In particular, it is vitally important to enabling systematic expression and exploration of complex trade spaces. This work we expect to have both fundamental and short-term applied value.

REFERENCES

[1] J. Bosch, “Software architecture: The next step,” in Software Architecture, ser. Lecture Notes in Computer Science, F. Oquendo, B. Warboys, and R. Morrison, Eds. Springer Berlin / Heidelberg, 2004, vol. 3047, pp. 194–199.

[2] A. Begel, “Effecting change: coordination in large-scale software development,” in Proceedings of the 2008 international workshop on Cooperative and human aspects of software engineering, ser. CHASE ’08. New York, NY, USA: ACM, 2008, pp. 17–20.

[3] A. Tang, M. A. Babar, I. Gorton, and J. Han, “A survey of architecture design rationale,” J. Syst. Softw., vol. 79, pp. 1792–1804, December 2006.

[4] J. Lee and K.-Y. Lai, “What’s in design rationale?” Hum.-Comput. Interact., vol. 6, pp. 251–280, September 1991.

[5] C. Potts and G. Bruns, “Recording the reasons for design decisions,” in Proceedings of the 10th international conference on Software engineering, ser. ICSE ’88. Los Alamitos, CA, USA: IEEE Computer Society Press, 1988, pp. 418–427.

[6] J. Tyree and A. Akerman, “Architecture decisions: Demystifying architecture,” IEEE Softw., vol. 22, pp. 19–27, March 2005.

[7] A. Jansen and J. Bosch, “Software architecture as a set of architectural design decisions,” in Proceedings of the 5th Working IEEE/IFIP Conference on Software Architecture. Washington, DC, USA: IEEE Computer Society, 2005, pp.109–120.

[8] P. Kruchten, R. Capilla, and J. C. Due˜nas, “The decision view’s role in software architecture practice,” IEEE Softw., vol. 26, pp. 36–42, March 2009.

[9] Y. Cai and K. J. Sullivan, “Modularity analysis of logical design models,” in Proceedings of the 21st IEEE/ACM International Conference on Automated Software Engineering. Washington, DC, USA: IEEE Computer Society, 2006, pp. 91–102.

[10] M. Conway, “How do committees invent?” Datamation, vol. 14, no. 4, pp. 28–31, 1968.

[11] M. Cataldo, J. D. Herbsleb, and K. M. Carley, “Sociotechnical congruence: a framework for assessing the impact of technical and work dependencies on software development productivity,” in Proceedings of the Second ACM-IEEE international symposium on Empirical software engineering and measurement, ser. ESEM ’08. New York, NY, USA: ACM, 2008, pp. 2–11.

[12] A. Begel, Y. P. Khoo, and T. Zimmermann, “Codebook: discovering and exploiting relationships in software repositories,” in Proceedings of the 32nd ACM/IEEE International Conference on Software Engineering -Volume 1, ser. ICSE ’10. New York, NY, USA: ACM, 2010, pp. 125–134.

[13] C. Baldwin and K. Clark, Design rules: The power of modularity. The MIT Press, 2000, vol. 1.

Appendix C: Collaborators

Dr. Christopher Ackermann is a research scientist at the Fraunhofer Center for Experimental Software Engineering in College Park, MD. He has been active in areas including software reliability analysis, architecture design and software verification applied to various types of systems including distributed space mission systems, commercial cloud systems, and embedded automotive control software. Dr. Ackermann has been involved in projects for NASA's Office of Safety and Mission Assurance, the NASA IV&V Center, NASA's Jet Propulsion Laboratory, the Department of Defense and the National Science Foundation. Email: CAckermann@fc-md.umd.edu

Dr. Madeline Diep is a research scientist at the Fraunhofer Center for Experimental Software Engineering in Maryland (FC-MD) in the Measurement and Knowledge Management Division. At FC-MD, she participates on projects for NASA's Office of Safety and Mission Assurance, NASA GSFC, and companies such as MITRE and ESR. Her research interest is in software and system assurance, verification and validation, monitoring, analysis, and measurement. She is an associate adjunct professor at the University of Maryland College Park. Email: MDiep@fc-md.umd.edu

Dr. LiGuo Huang is an Assistant Professor in the Dept. of Computer Science and Engineering at the Southern Methodist University (SMU). Her research centers around software-intensive system dependability analysis, software process improvement via mining software repository, software and system assurance applied to various types of systems including space mission systems in NASA's Jet Propulsion Laboratory and cloud systems. Her current research is funded by the DoD and NSF.

Dr. Marty Humphrey is an Associate Professor in the Department of Computer Science at the University of Virginia. For the last three years, his primary research focus has been Cloud computing (the prior 8+ years focused on Grid computing). He is the Program Chair for the "Grids and Clouds" area of IEEE/ACM Supercomputing 2012 (and member of the Program Committee for 15 other cloud-related conferences in the past 3 years). He just completed a term as the sole academic member of the Microsoft Windows Azure Customer Advisory Board (CAB). Two of his PhD students have recently joined Eucalyptus, Inc., a major open-source cloud vendor.

Dr. Forrest Shull is a senior scientist at the Fraunhofer Center for Experimental Software Engineering in Maryland (FC-MD), a nonprofit research and tech transfer organization, where he leads the Measurement and Knowledge Management Division. At FC-MD, he has been a lead researcher on projects for NASA's Office of Safety and Mission Assurance, the NASA Safety Center, the U.S. Department of Defense, the National Science Foundation, the Defense Advanced Research Projects Agency (DARPA), and companies such as Motorola and Fujitsu

Labs of America. He is an associate adjunct professor at the University of Maryland College Park, and Editor-in-Chief of *IEEE Software.* Email: fshull@fc-md.umd.edu

Dr. Kevin Sullivan is an Associate Professor in the Department of Computer Science at the University of Virginia. For the last several years he has been trying to understand what changes are needed in traditional engineering approaches in the era of ultra-complex systems, with particular emphasis on socio-technical and modularity aspects of such systems. He is current program chair for Aspect-Oriented Software Development: Modularity Visions. He co-authored the report, *Ultra-Large-Scale Systems: The Software Challenge of the Future*. He was a member of the National Research Council Committee on Software Producibility. His most recent research efforts focus on (1) formally model-based synthesis of software architectures and application frameworks and (2) cyber-social systems, an emerging notion in which networks of people are viewed as computational entities engaged in complex problem solving activities. The questions include how to understand what networks and communications are needed based on dependencies in technical decision and trade spaces.