2016 NDIA GROUND VEHICLE SYSTEMS ENGINEERING AND TECHNOLOGY SYMPOSIUM
SYSTEMS ENGINEERING (SE) TECHNICAL SESSION
AUGUST 2-4, 2016 – NOVI, MICHIGAN

BRIDGING RELIABILITY ENGINEERING AND SYSTEMS ENGINEERING

Venkatesh Agaram
Quality & Reliability Engineering Practice
CIMdata Inc, Ann Arbor, MI

ABSTRACT

The increasing application of sensors, actuators, and complex algorithms for delivering artificial intelligence and connectivity in products and product-systems will drive an unprecedented growth in design complexity and software content, making it increasingly more difficult to ensure dependability in an economical manner. Much learning about the dependability of such new and innovative products is likely to happen as they are conceived and designed. Consequently, accelerated verification and validation iterations supported by easy and rapid storage and retrieval of failure knowledge must be enabled. No single software solutions provider effectively covers all three critical areas required for developing and delivering dependable smart connected products, namely, reliability engineering, systems engineering, and failure knowledge management. This paper mainly presents a potential map of the commonly used reliability engineering tools overlaid on the systems engineering technical processes. The paper recommends including a formal knowledge storage and retrieval system in the closed-loop between systems engineering and reliability engineering so that the details observed in past failures are not missed in future design iterations.

INTRODUCTION

Connectivity and artificial intelligence are major features of many upcoming products that are increasingly likely to be systems-of-systems. A large set of complex algorithms is needed for estimating accurately the instantaneous states of these systems and their operational environments and for exercising robust control over the systems to deliver the benefits desired by the end users. Intricate algorithms are used for sensor data fusion, remote diagnostics, remote repair, autonomous control, timely hand-off to humans, and many other functions. The increasing application of sensors, actuators, and the associated algorithms in products and product-systems will drive an unprecedented growth in design complexity and software content, making it increasingly more difficult to ensure dependability in an economical manner.

Even without connectivity and artificial intelligence, the launch delays and recalls associated with today's less sophisticated electronically controlled mechanical systems, due to performance issues, can very often be traced to design complexity and software-content. In case of connected, automated or autonomous systems, the problem can be expected to worsen.

A US FDA study [1] has determined that software-related recalls measured as a percentage of all recalls in the medical devices industry have gone up from 14% in 2005 to 25% in 2011. This percentage has been trending upwards since 1983, with software-related recalls as a percentage of overall recalls averaging 6% between 1983 and 1991, 8% between 1992 and 1998, 11% between 1999 and 2004, and 19% between 2005 and 2011.

The relationship between the number of recall events in a period of time and the number of units impacted by the recalls can be vastly different between industries [2]. For example, 150 recall events in the medical device industry may amount to 300,000 affected units but in the automotive industry, 30 recall events could impact 2 million vehicles.

A financial advisory blog [3] mentions that there has been a substantial increase in software-related recalls in the automotive industry since 2012. The authors cite 32 software-related recalls that affected 3.6 million light vehicles between 2005 and 2012. However, they mention 6.4 million additional vehicles affected by 63 additional software-related recalls between 2013 and 2015. The blog also mentions going from 0.3% of recalls being software-related in 2005 to 4.3% of recalls being software-related within the first 6 months of 2015, and this trend the authors state, is showing no signs of reversing. The authors of the blog also report a similar trend seen in NHTSA's complaint data. Over the period covering 2005 to 2009, 55 software-

related complaints were logged with NHTSA, whereas, over the period 2010 to 2014, 197 complaints contained the same reference to software related-issues, highlighting the increased role of software in automotive safety.

Notable software-related issues in recent times, from the aerospace industry have been in connection with the Boeing 787 and the F-35 Joint Strike Fighter. A software bug in the Boeing 787 was found to be capable of shutting down the plane's electric generators every 248 days because a software counter, internal to the generator control units (GCUs) could overflow after 248 days of continuous power. This could cause the GCU to go into failsafe mode, resulting in a loss of all electrical power regardless of the flight phase. The F-35 Joint Strike Fighter is expected to be further behind in its combat-readiness due to issues with its RADAR software and vulnerability to cyber-attacks, and these require the system to be rebooted every four hours of flight time while the desired reboot interval of the F-35 is eight to ten hours of flight time.

The failure modes of software-intensive, control systems driven products are difficult to guess a priori due to the complexity of their functions and information flow, and consequently, crucial failure modes can easily be missed. Much learning about the dependability of such new and innovative products is likely to happen as they are conceived and designed. Consequently, accelerated verification and validation iterations supported by easy and rapid storage and retrieval of failure knowledge must be enabled.

Today, enterprise level reliability engineering tools and systems engineering are not well-connected, preventing many lessons learned in reliability engineering from helping drive robust designs via systems engineering. This situation is further aggravated when we consider the silos of expertise and data among mechanical, electrical, and software disciplines. The product development tools used in these disciplines' silos are very different and most often not connected with each other, rendering the ability to carry out systems engineering very difficult. A major challenge arises due to improper channeling of prior experience and knowledge about reliability into design, leading to repeated dependability issues of complex products.

No single software solutions provider effectively covers all three critical areas required for developing and delivering dependable smart connected products, namely, reliability engineering, systems engineering, and knowledge management. Further, given the complexity of the development of the large number of specialty tools required to do this, it is perhaps unrealistic to expect that a completely integrated suite of solutions will be available from a single software solutions provider.

The most practical solution for developing dependable, connected, and intelligent products could come through the application of specialty software tools that conform to applicable interoperability standards so that the enterprise level system integrators can seamlessly connect the tools needed for reliability engineering, systems engineering, and knowledge management.

This paper mainly presents a potential map of the commonly used reliability engineering tools overlaid on the systems engineering technical processes. The paper adheres to the technical process of systems engineering described by INCOSE [4] and the commonly known tools and processes used in design-for-six-sigma and reliability engineering [5, 6]. The paper recommends including a formal knowledge storage and retrieval system in the closed-loop between systems engineering and reliability engineering so that the details observed in past failures are not missed in future design iterations.

RELIABILITY ENGINEERING MEETS SYSTEMS ENGINEERING

To enable fast learning cycles that will help identify potential failure modes of complex systems, a seamless enterprise level connection between the systems engineering technical processes, the reliability engineering tools, and a knowledge management system is needed, so that efficient storage and retrieval of failure modes information is possible.

To accomplish the development of the enterprise level connection mentioned above, the activities related to the thirteen technical processes of systems engineering [4] have been considered in this work as a higher-level product lifecycle structure. Those thirteen systems engineering technical processes are:

- Stakeholders' Requirements Identification
- System Requirements Definition
- System Architectural Design
- System Elements Definition
- System Analysis
- System Elements Realization
- System Elements Integration
- System Design Verification
- Verified System Transition
- System Performance Validation
- System Operation
- System Maintenance
- System Disposal

In the following section the diverse tools used in design-for-six-sigma and reliability engineering that should support the systems engineering technical processes are briefly described.

Beyond the next section, each activity associated with the thirteen technical processes of systems engineering are associated with a set of design-for-six-sigma and reliability engineering tools, wherever those tools are likely to have a beneficial impact. The purpose is to present a connection

between the tools used in reliability engineering and design-for-six-sigma, and the main technical process activities of systems engineering that need to be accomplished by enterprise level software systems integration so as to adequately deal with the complexity of designing and delivering smart connected products.

Reliability Engineering Tools

A wide variety of tools are used in reliability engineering and design-for-six-sigma. These help prevent failures and increase the operational life of systems, subsystems, and components.

1. Affinity Diagrams (KJ Analysis)—for clustering similar items to get a higher level view of a large number of entities being analyzed, e.g., VOC.
2. Quality Function Deployment (QFD) or House of Quality (HOQ)—meant to develop the interpretation matrix for creating functional requirements that will satisfy the stakeholder requirements—serves as a traceability matrix.
3. Kano Analysis—way of classifying requirements into "delighters," "performance needs," and "basic needs" for rank ordering the stakeholder requirements based on their importance.
4. Functional Flow Block Diagram (FFBD)—multi-tier, time-sequenced, step-by-step flow diagram of a system's functional flow.
5. SysML Diagrams—representation of different aspects of systems through diagrams, namely, requirements, activity, block definition, internal block definition, parametric representation, use cases, system state, and sequence.
6. Integrated Definition for Functional Modeling Diagrams (IDEF0)—designed to model the decisions, actions, and activities of a system.
7. N2 Charts—matrix, representing functional or physical interfaces between system elements, also applicable to hardware and/or software interfaces.
8. Specification Tree—specifications of a technical system under development in a hierarchical order going from system requirements, to system design specifications, to subsystem specifications, to assembly specifications, and to component specifications.
9. Failure Modes Effects & Criticality Analysis (FMECA)—bottom-up, inductive analysis performed at the functional or component level, enhanced by the relationship between the probabilities of occurrence of failure modes against the severity of their consequences.
10. TRIZ: Theory of Inventive Problem Solving—problem solving, analysis and forecasting derived from the study of patterns of invention in the global patent literature.
11. Physics of Failure Analysis—modeling and simulation based understanding of processes and mechanisms that lead to failure, to predict reliability and improve product performance.
12. Robust Optimization—optimization of system performance while increasing the stability of the system against product, process, and usage variability.
13. Design of Experiments (DOE)—appropriate choice of discrete values of independent variables for finding the corresponding values of the chosen dependent variables to understand the system response.
14. Response Surface Analysis—expressing the response of complex systems as a continuous function of chosen independent variables to cost-effectively generate insights about systems' behavior.
15. Monte Carlo Simulations—random sampling of independent variables and the generation of the corresponding independent variable values to understand the behavior of the dependent variables.
16. Conjoint Analysis—ranking of stakeholders' requirements based on combinations of attributes and weights representing the relative importance of the attributes.
17. Kepner-Tregoe Analysis (KTA)—systematic way of decision making based on wants and must haves, possible alternative solutions, probability of adverse effects, and significance.
18. Analytic Hierarchy Process models (AHP)—modeling the problem as a hierarchy containing the decision goal, the alternatives for reaching it, and the criteria for evaluating the alternatives.
19. Multi-Attribute Utility Analysis (MAUA)—a scalar utility function over the domain of the attribute values to assess tradeoffs between different attributes.
20. Fault Tree Analysis (FTA)—top down, deductive failure analysis in which system failure is analyzed using Boolean logic to combine a set of lower-level events.
21. Event Tree Analysis (ETA)—bottom up modeling technique for success and failure which explores responses through a single initiating event and helps assess overall probabilistic system response.
22. Reliability Block Diagrams (RBD—for showing how components and subsystems contribute to the success or failure of a complex system.
23. Failure Reporting Analysis & Corrective Action System (FRACAS)—for reporting, classifying, and analyzing failures, and planning corrective actions in response to failures.
24. Corrective Action & Preventive Action (CAPA)—systematic investigation of the root causes of system problems to correct the situation or prevent it from occurring.
25. Markov Analysis—breaking the final (or failed) system state into a number of intermediate states, connected

with each other by transition matrices, under the assumption that each state is memory-less.

26. Weibull Analysis—makes predictions about the life of products by fitting statistical distributions to product life data (performance over product lifetime).
27. System Maintainability Analysis—determines the ease and speed with which a system can be restored to operational status after a failure occurs. It is calculated based on time-to-repair as the random variable instead of time-to-failure for estimating system reliability.
28. System Availability Analysis—probability of a unit being available in a fully functional state, calculated based on mission duration and observed or simulated mission downtime.
29. Asset Performance Management (APM)—condition monitoring, predictive forecasting, and reliability-centered maintenance of systems based on data capture and analytics.
30. Accelerated Life Testing (ALT)—subject systems to stress, strain, temperatures, voltage, vibration rate, pressure, etc., in excess of their service levels for quickly uncovering potential modes of failure.

In the Table I, each of the thirty tools described above have been mapped to the thirteen technical processes of systems engineering where they are likely to be most beneficial. The purpose is to present a connection between the tools used in reliability engineering and design-for-six-sigma, and the main technical processes of systems engineering that needs to be accomplished through enterprise level software systems integration.

FAILURE KNOWLEDGE CAPTURE AND REUSE

Systems engineering through its technical processes has traditionally been applied for realizing highly complex systems where chances for unreliability of designs abound. Systems engineering begins with the discovery of the real problems that need to be solved and identification of highest impact failures that can occur, and then, through an interdisciplinary approach to engineering, attempts to find solutions to those problems and potential failures. However, many of the systems dependability issues that are likely to arise have their roots at the intersection of different disciplines of engineering and at the interfaces between different subsystems where engineering intuition tends to be low and rapid learning is imperative.

The rapid learning, which needs to happen over verification and validation cycles of systems engineering or over newer instances of products as they evolve from past designs and configurations, deals with implicit knowledge which is not immediately accessible; and in particular cannot be acquired from conventional databases. Further, the useful knowledge is obfuscated by different subject matter experts using different terms when addressing the same issue. For instance, implicit knowledge about failure modes exists in natural language, and consequently, the meaning depends on interpretations by the team which is reusing that knowledge.

The problem of reusing pre-existing knowledge about failure modes could be solved effectively through the definition of an ontology, which enables a common understanding of the domain specific concepts without need for interpretation, while making the ontology-held knowledge explicit and machine-readable. An ontology helps to integrate the elements of task-relevant knowledge by uniformly structuring the domain knowledge.

An ontology, which consists of definitions of concepts, relationships, and rules, is used in knowledge-based systems where formalized knowledge is represented in a language that supports reasoning and inference. The past knowledge about system failures, which is implicitly contained in documents, can be made explicit for use in information systems by an inference engine. Using non-deductive inference rules, the scope of making implicit knowledge explicit, can be expanded significantly. By using an ontology-based information system, as part of the closed-loop between system failure and performance issue occurrences, and the upstream design activity, new and fast learning can be enabled to ensure that the risk of overlooking failure modes is mitigated.

Although an ontology, as a way of converting implicit system failure knowledge into machine-readable explicit knowledge for reuse has often been mentioned in technical literature, it is not currently offered commercially by software providers either as part of systems engineering or reliability engineering tool suites. In view of the increasingly complex smart, connected products that are emerging, the lack of understanding about their failure modes will only increase. In addition, the market pressures on time-to-launch and product costs will require much faster learning cycles than ever before regarding product performance and product failure. Consequently, ontology-based or similar knowledge reuse tools are needed at the enterprise level, in earnest, to deliver complex products that are dependable, affordable, and available on time.

LEARNING SYSTEMS BASED DESIGN-FOR-RELIABILITY

The technical processes of systems engineering ensure robust capture of stakeholders' requirements in the beginning, followed by a hierarchical flow of those requirements into system, subsystem, and component design. Verification and validation activities occur at all three levels of the system development hierarchy, and they offer learning opportunities about the system's performance at different levels.

Systems Engineering Technical Processes	Reliability Engineering Tools	Affinity Diagrams	QFD (HOQ)	Kano Analysis	Functional Flow Block Diagram	SysML Diagrams	IDEF	N^2 Charts	Specification Tree	FMECA	TRIZ	Physics of Failure Analysis	Robust Optimization	Design of Experiments (DOE)	Response Surface Analysis	Monte Carlo Simulations	Conjoint Analysis	Kepner-Tregoe Analysis	Analytic Hierarchy Process (AHP) Models	Multi-Attribute Untility Analysis (MAUA)	Fault Tree Analysis (FTA)	Event Tree Analysis (ETA)	Reliability Block Diagrams (RBD)	FRACAS	CAPA	Markov Analysis	Weibull Analysis	System Maintainability Analysis	System Availability Analysis	Asset Performance Management	Accelerated Life Testing
		1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30
Stakeholders' Requirements Definition	1	■		■	■	■			■					■			■							■	■						
System Requirements Definition	2		■		■	■			■	■	■	■												■	■			■	■	■	
System Architectural Design	3				■	■			■	■	■		■	■	■	■					■		■	■	■	■	■				
System Elements Definition	4		■		■	■	■		■	■	■	■	■	■	■	■					■	■	■			■	■	■	■		
System Analysis	5		■		■	■	■		■	■	■	■	■	■	■	■		■	■	■											
System Elements Realization	6									■	■		■	■	■	■								■	■						
System Elements Integration	7				■	■	■		■	■	■		■	■	■	■					■	■		■	■			■	■	■	■
System Design Verification	8				■	■	■	■	■	■		■	■	■	■	■					■	■		■	■			■	■		
Verified System Transition	9				■	■	■	■	■	■	■		■	■	■	■					■	■	■	■	■				■		
System Performance Validation	10				■	■	■		■	■	■	■	■	■	■	■					■	■	■	■	■						
System Operation	11				■	■	■		■	■	■	■	■	■	■	■					■	■	■	■	■	■		■	■		
System Maintenance	12				■	■	■		■	■	■	■	■	■	■	■	■							■	■		■	■	■	■	
System Disposal	13				■	■	■			■	■																				

Table I Relating Reliability Engineering Tools with Systems Engineering Technical Processes

The reliability engineering tools depend on the technical expertise of subject matter experts to develop robust and optimum designs that meet the stakeholders' requirements and do not run the risk of customer annoyance or unsafe outcomes. To design against potential performance issues and failure modes, the subject matter experts need to be able to identify those failure modes and be knowledgeable about the potential for their occurrence.

Given the runaway complexity of products today which often straddle two or more traditional areas such as automotive, entertainment, information, banking, etc., major problems can occur when the intuition of the subject matter experts at the intersections of these domains is lacking. In fact, we can say that the systems could become so complex and interrelated that the so-called subject matter experts are not really subject matter experts any more. The speed of evolution of products demands a "learning system based design for reliability" capability in which systems engineering, as we know it, is seamlessly coupled with reliability engineering and design-for-six-sigma, supported by an ontology-based or similar knowledge management system that can manage implicit knowledge by converting it into machine-readable explicit knowledge.

Software solutions providers today offer enterprise solutions to enable large parts of systems engineering as described by INCOSE [4]. However, those systems engineering solutions do not communicate well with reliability engineering and design-for-six-sigma tools. Too much manual transfer of information is needed between the systems engineering and the reliability engineering solutions, leaving the door open for many quality and reliability problems to be missed, allowing them to reemerge during service and operations. In many situations, product developers and manufacturers have had to custom-develop the connection between reliability engineering and systems engineering.

In order to compensate for the limited intuition of subject matter experts regarding potential failure modes of complex, software-intensive products, while remaining competitive in cost and time-to-market, we need a seamless coupling between reliability engineering tools and systems engineering processes that leverages ontology-based or similar failure knowledge capture and reuse in order to realize rapid, enterprise level learning.

CONCLUSION

The need for seamlessly connecting systems engineering solutions and reliability engineering solutions through a knowledge management system has received very little attention. With the growing complexity of software-intensive products, whose failure modes are difficult to guess a priori, it is imperative to establish such a connection in order to realize the "learning system based design-for-reliability" capability. Given the expansiveness of the solution that could satisfy such a need, it appears that the global information technology system integrators could consider this as a strategic business opportunity. They could drive the interoperability standards between diverse systems engineering, reliability engineering, and knowledge management tools and develop integration offerings that can be tailored to the specific needs of diverse industry verticals and businesses of different sizes within those verticals.

One opportunity for driving the interoperability standards between different systems engineering, reliability engineering, and knowledge management tools could be through a user group under the Open Services for Lifecycle Collaboration (OSLC) [7]. Currently a user group for Quality Management [8] is working under OSLC towards defining a common set of resources, formats and RESTful services for quality management tools to interact with other application lifecycle management (ALM) tools.

No user groups for reliability engineering and/or knowledge management exist under OSLC today. However, that can be remedied through CIMdata®'s leadership, once experts from industry, software solutions providers, and system integrators agree with CIMdata that the "learning system based design-for-reliability" capability must be developed in the near future.

For socializing the contents of this whitepaper with the experts from industry, software solutions providers, and systems integrators, CIMdata will organize a series of webinars in 2016 which will go into the details of several aspects of this topic. CIMdata also plans to organize a workshop in 2017 where different viewpoints will be presented on this subject by the experts from industry, software solutions providers, and system integrators, culminating in the strategic direction of proceeding further within the framework of OSLC or some other framework.

REFERENCES

[1] Simone, Lisa. K., "Software-Related Recalls: An Analysis of Records", Biomedical Instrumentation & Technology, pp. 514-522, November/December 2013.

[2] Edwards, Steve, "The Tech Effect: Examining the Trend of Software-Related Recalls", Stericycles Thought Leadership Blog."

http://www.stericycleexpertsolutions.com/the-tech-effect-examining-the-trend-of-software-related-recalls/

[3] Reed, Jake, "The "Softer" Side of Things – The Impact of the Increasing Software and Complexity on Vehicle Defects", SRR Automotive Warranty & Recall Blog, July 29, 2015.

http://blog.srr.com/automotive-warranty-and-recall/the-softer-side-of-things-the-impact-of-the-increasing-software-and-complexity-on-vehicle-defects/

[4] INCOSE, "Systems Engineering Handbook – A Guide for System Life Cycle Processes and Activities", 4th Ed., Wiley, INCOSE-TP-2003-002-04, 2015.

[5] Maass, McNair and Patricia, "Applying Design for Six Sigma to Software and Hardware Systems", Prentice Hall, 1st Ed., 2009.

[6] Creveling, C. M., Slutsky, J. L., and Antis, Jr. "Design for Six Sigma in Technology and Product Development", Prentice Hall, 2003.

[7] Open Services for Lifecycle Collaboration (OSLC)

http://open-services.net/

[8] Quality Management User Group at OSLC

http://open-services.net/workgroups/quality-management/