Four Pillars for Improving the Quality of Safety-Critical Software-Reliant Systems

Peter Feiler
John Goodenough
Arie Gurfinkel
Charles Weinstock
Lutz Wrage

April 2013

Current build-then-test practice for software-reliant systems has become increasingly unaffordable. An improvement strategy has four pillars of an integrate-then-build practice that lead to improved quality through early defect discovery and incremental end-to-end validation and verification.

Studies of safety-critical software-reliant systems developed using the current practices of build-then-test show that requirements and architecture design defects make up approximately 70% of all defects, many system level related to operational quality attributes, and 80% of these defects are discovered late in the development life cycle [Redman 2010]. Exponential growth in software size and complexity has pushed the cost for the current generation of aircraft to the limit of affordability.

We present four pillars of an improvement strategy for an integrate-then-build practice that result in early defect discovery and increased confidence through incremental end-to-end system validation and verification throughout the life cycle (Figure 1).

- Capture of mission and safety-criticality requirements in analyzable form;
- Virtual integration of the physical system, hardware platform, and software architectures through consistent analyzable architecture models;
- Static analysis techniques applied to the models and actual system implementation to complement testing; and
- Incremental assurance of justified confidence through consistent end-to-end evidence throughout the development life cycle.

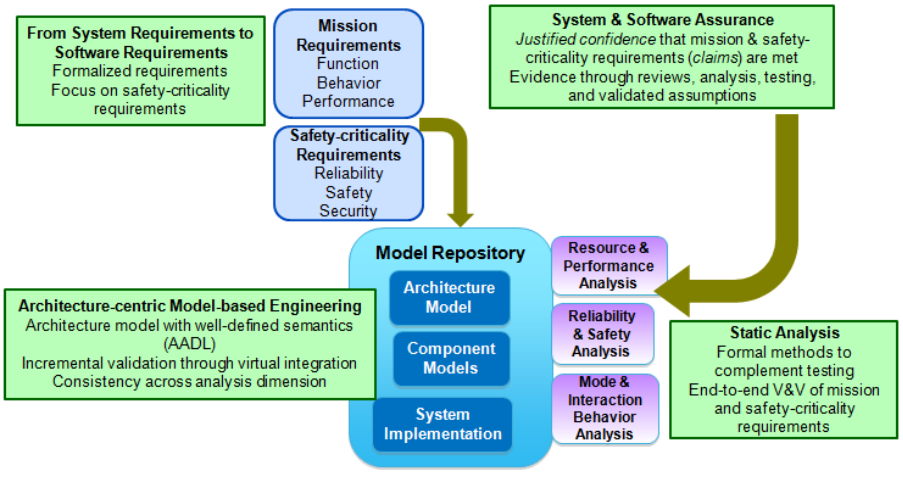

Figure 1: Four Pillars of Reliability Improvement

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

Phone: 412-268-5800
Toll-free: 1-888-201-4479

www.sei.cmu.edu

SHORTCOMINGS IN AND PROCESS-RELIANT TESTING-BASED QUALIFICATION

Current build-then-test practice uses reliability metrics and qualification test criteria to assess the quality of safety-critical software. Despite best practices the aircraft industry struggles with exponential growth in complexity and cost.

Engineers have developed safety-critical systems by relying on conservative best practices and a a culture where safety considerations are integral to all aspects of an organization. These practices reflect the use of ISO 9001/CMMI®,[1] the suite of ISO-IEC SC 7 process standards, and standards and practices specific to the certification of safety-critical software systems such as DO-178B and C, SAE ARP 4754, and SAE ARP 4761 [RTCA 1992/2011; SAE 2010, 1996].[2] The on-board software of aircraft has experienced exponential growth in size and complexity (Figure 2). Under current *build-then-test* practices, the industry cost for the software of current-generation aircraft has reached an unaffordable $8 billion [Redman 2010]. Similarly, the U.S. Army has recognized that qualifying the airworthiness of rotorcraft has increasingly become infeasible with current software test practices trying to achieve full code coverage due to increased software size and interaction complexity [Boydston 2009].

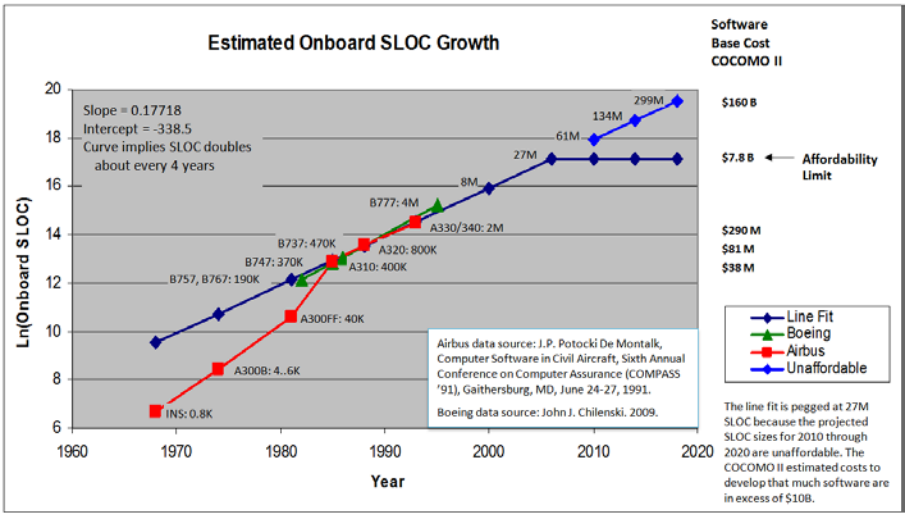

Figure 2: Exponential Growth in Software Size and Complexity Makes Systems Unaffordable

[1] ®CMMI, the Software Engineering Institute Capability Maturity Model Integration framework, is registered in the U.S. Patent and Trademark Office by Carnegie Mellon University.

[2] ISO, International Organization for Standardization; ISO-IEC SC 7, ISO-International Electro technical Commission Software and Systems Engineering Subcommittee; SAE ARP, SAE International Aerospace Recommended Practices.

In current build-then-test practice, two approaches determine system quality. The first approach focuses on defining qualification criteria that if satisfied are sufficient to demonstrate the safe operation of a system with acceptable risk. The second approach focuses on process metrics, using statistical techniques to predict residual faults in software.

Certification of safety-critical systems is driven by qualification criteria with increasing stringency according to criticality levels that primarily address process compliance and code coverage.

The DO-178B standard provides a set of qualification criteria that if satisfied are considered to be sufficient evidence that the system is safe to operate with acceptable risk [RTCA 1992]. System engineers assign criticality levels to different subsystems in a system during safety assessment early in the life cycle. Criti- Criticality levels reflect the severity of impact of a subsystem failure on the safe operation of the system. While many criteria are process oriented, such as traceability to requirements, some criteria focus on coverage of application logic to ensure that the system handles nominal and anomalous operational scenarios. For example, DO-178B Level B requires decision coverage, while Level A requires modified condition/decision coverage [FAA 2002].

Reliability metrics are rooted in hardware and focus on assessing sufficient quality in terms of residual fault density. Engineers rely on testing to eliminate faults with little investment in software reliability improvement programs.

Reliability engineering, as practiced, has its roots in the use of statistical techniques to assess the hardware reliability of a slowly evolving system design and an operational system affected by wear and aging over time. Engineers often use two common metrics, fault density and reliability growth, as predictive measures for residual faults to decide when the hardware has reached sufficient quality.

The failure distribution curve for hardware does not hold for software because continuous changes in software designs and the operational environment introduce new faults and activate latent faults, and the independent failure assumption of hardware fault tolerance has been shown to be invalid for software [Butler 1993]. This has led to the common practice of engineers making reliability predictions for a system, often assuming that software is perfectible and deterministic. The methods are shown to lead to exorbitant amounts of testing.

While the U.S. Army Materiel Systems Analysis Activity (AMSAA) provides funding for hardware reliability-improvement programs that use modeling, analysis, and simulation to identify and reduce design defects before the system is built. No such reliability improvement programs exist for software. Instead AMSAA funding for software focuses on finding and removing code faults through code inspection and testing [Goodenough 2010].

SOFTWARE INDUCED FAULTS IN SAFETY-CRITICAL SYSTEMS

Despite best build-then-test practice, 70% of faults are introduced during requirement and architecture design and 80% are discovered at system integration or later, with very high rework cost.

Despite best build-then-test practices, system-level faults due to software have increasingly dominated the rework effort for faults discovered during system integration and acceptance testing. Several studies of safety-critical systems show that 70% of errors in embedded safety-critical software are introduced in the requirements (35%) and architecture design phases (35%) (Figure 3) [Boehm 1981, NIST 2002, Dabney 2003, Galin 2004]. At the same time, 80% of all errors are not discovered until system integration or later. The rework effort to correct a problem in later phases can be as high as 300-1000 times the cost of in-phase correction. Therefore, it is desirable to discover such problems earlier in the life cycle and increase the chance of tests passing the first time around.

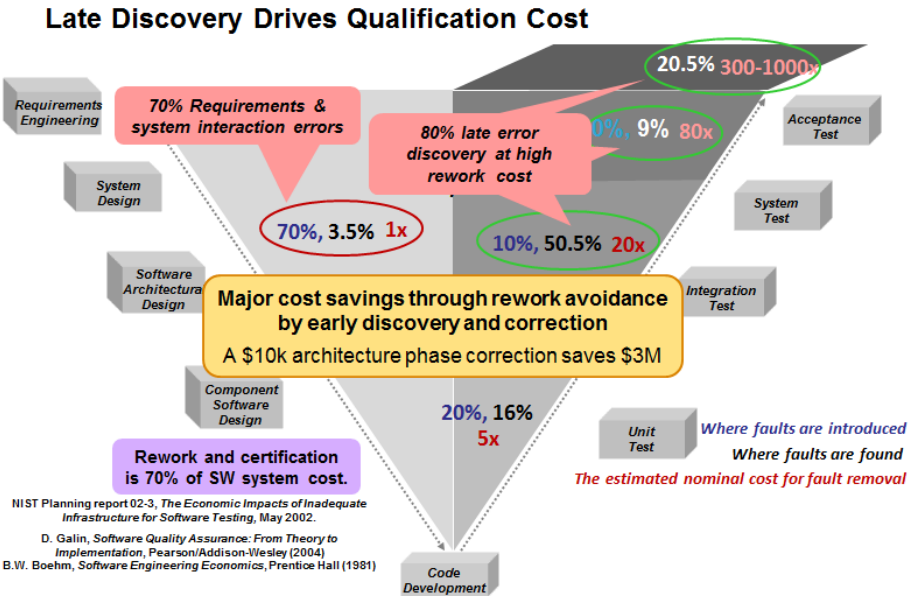

Figure 3: Late Discovery of System-Level Problems

Many nonfunctional system requirements do not translate into derived requirements on software systems.

Software testing is often performed software testing against incomplete and inconsistent requirements.

Testing validates the system implementation against requirements. A NASA study showed that this is due to missing requirements (33% of all requirements errors), incorrect requirements (24%), incomplete requirements (21%), and ambiguous (6%), inconsistent (5%), or overspecified requirements (6%) [Hayes 2003]. Boehm observed that current system engineering practice focuses on architecting the physical system to address functional and nonfunctional requirements such as safety before developing requirement specifications for software. As a result software requirement specifications focus primarily on functionality [Boehm 2006].

Engineers often do not fully understand the impact of design decisions in the runtime architecture of embedded software on functional and nonfunctional properties of the system.

System-level functional and failures are often triggered by unexpected and complex interactions among the physical system, the embedded software, and the operational environment whose results can be catastrophic, such as the near loss of aircraft [ATSB 2008]. Figure 4 shows while system engineering addresses interactions between a system under control and a control system and its environment including the operator, the realization of the system capability in software introduces new fault hazards due to additional interaction complexity and mismatched assumption that are typically not addressed during safety analysis. System functions implemented in software may be faulty because a 16 bit variable cannot handle the range of values in the physical domain. The application software operates as concurrently executing interacting tasks with operational modes resulting in unexpected race conditions and mode interactions, especially under fault conditions. Deployment of the task architecture on the computer platform utilizes virtualization concepts. Relocation of logical channels and virtual machines to balance workload may result in loss of physical redundancy, invalidating reliability predictions early in the system engineering process. Use of partitions virtualizes sampling time, resulting in frame-level sampling jitter and potential loss of data leading to potential control-system instability and inconsistent system states [Feiler 2009].

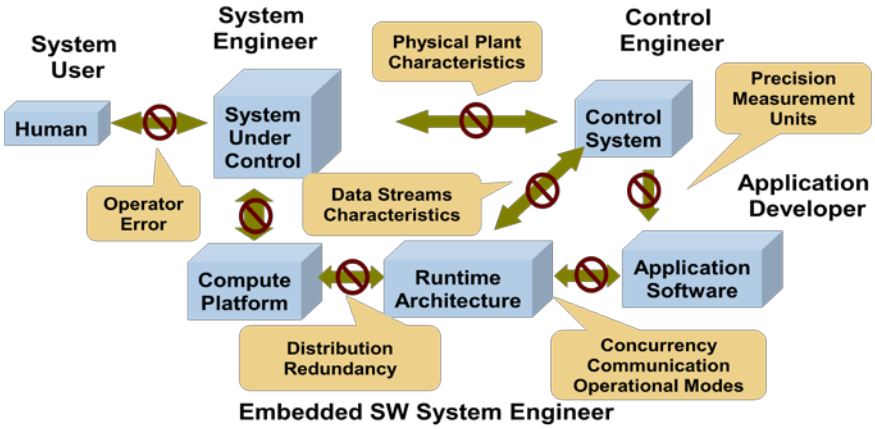

Figure 4: New Levels of System Interaction Complexity and Mismatched Assumptions

A recent study by the National Research Council, *Software for Dependable Systems: Sufficient Evidence?*, stated that testing and good development processes, while indispensible, cannot by themselves ensure high levels of dependability [Jackson 2007]. Experts consider *formal analysis* of *mission and safety-criticality requirements* and *architecture design* to provide key *assurance evidence* for improving dependability of software-reliant systems, which leads us to the four pillars of reliability improvement.

ARCHITECTURE-CENTRIC MODEL-BASED VIRTUAL INTEGRATION

The aircraft industry views an architecture-centric, analytical virtual-integration approach as a necessary change in practice to address the exponential growth in system interaction complexity and in development and qualification costs.

The aircraft industry has recognized that software-reliant system development must take an architecture-centric, model-based analytical approach to overcome the limitations of current recommended practice.

> *The System Architecture Virtual Integration (SAVI) initiative is a program being executed within the Aerospace Vehicle System Institute (AVSI) that aims to address issues stemming from the lack of an integrated systems modeling environment. The SAVI approach of "Integrate, then Build" acknowledges the modern distributed development environment while seeking to arrest the propagation of requirements errors through the development life cycle, primarily by capturing design assumptions and shared properties of the system design in an authoritative, annotated architectural model. Such a reference model provides a common, analyzable model to confirm that the definition of the system (requirements through design) remains complete, consistent, and correct at all levels of system decomposition. [Redman 2010, p. 1]*

Industry experience has shown that the value of model-based engineering greatly diminishes if engineers do not address the "multiple-truth" problem (Figure 5) [Redman 2010].

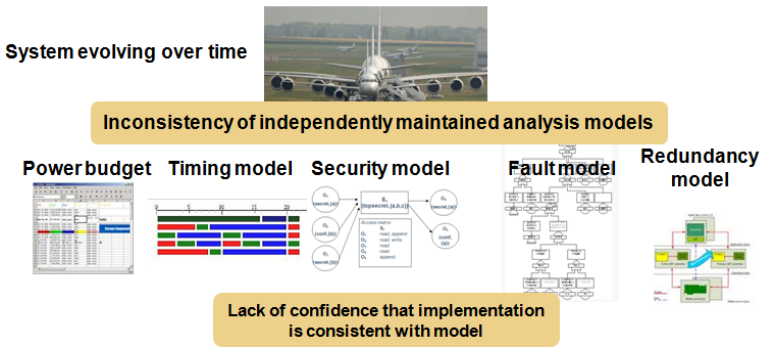

Figure 5: Multiple-Truth Issues in Model-Based Development

Model-based engineering has some pitfalls, but a single-source virtual-integration approach addresses them. Using architectural analysis to discover problems early leads to reduced fault leakage and fewer failed tests.

The virtual integration concept has been demonstrated to address this multiple truth issue through annotations to the architecture model for multiple quality attribute dimensions and automatic generation of analyzable models as well as executable code (Figure 6). A key technology is the SAE International Architecture Analysis & Design Language (AADL) industry standard, a notation with well-defined semantics for representing embedded software system architectures through concepts that capture the semantics of software runtime architectures deployed on computer platforms and interacting with physical systems [SAE 2012].

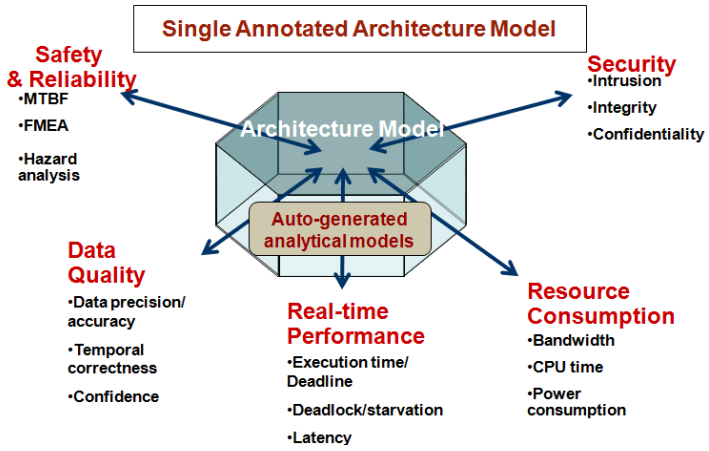

Figure 6: Multiple Analysis Dimensions Based on Architecture Reference Model

This has led to a multi-notation model repository approach based on standardized model interchange formats supporting SAE AADL, OMG SysML, and other modeling notation while maintaining cross-model consistency [Redman 2010].

A SAVI proof-of-concept demonstration illustrates the incremental end-to-end validation and verification of a multi-tier system architecture that includes an Integrated Modular Avionics (IMA) system and mechanical system models and demonstrates the value of architecture models in subcontracting (Figure 7). A companion Return-On-Investment study showed major cost savings due to rework cost avoidance [Feiler 2010].

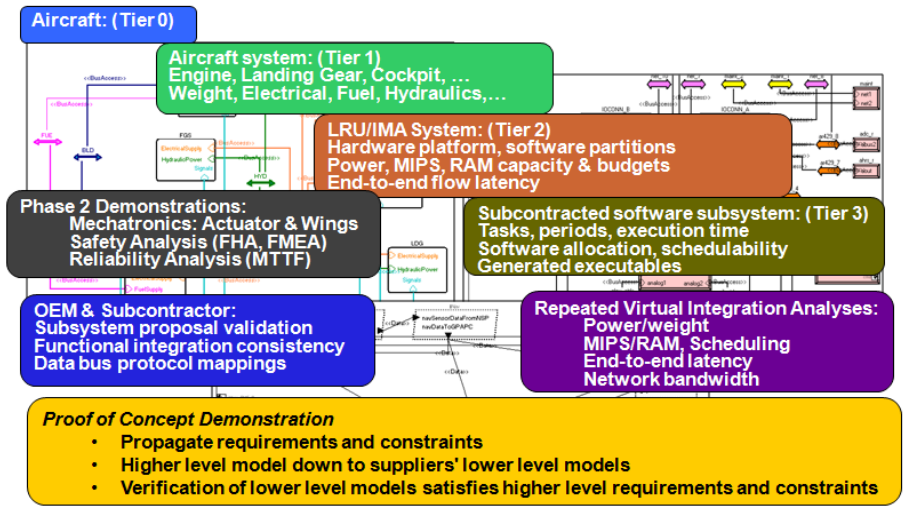

Figure 7: Early Problem Discovery through Multi-tier Virtual System Integration

ANALYZABLE MISSION AND SAFETY-CRITICALITY REQUIREMENTS

Capture of analyzable mission and safety-criticality requirements can become a reality. Ambiguity and inconsistency is reduced through tool-based validation resulting in high payoff. An industry standard architecture modeling notation provides the linkage between requirements and architecture design.

The challenge in requirements engineering is to offer a systematic way of capturing analyzable requriements without overbrudening the engineer with formalisms. A recent study of industrial requirments engineering practice has shown that textual *shall* statements, tables and diagrams still dominate the practice with some behavioral specification expressed in state machine notations [FAA 2009b]. A Requirement Engineering Management Handbook recent developed for NASA provides a practical 11 steps process to more formally capture requirements in a contextual system architecture description to facilitate valdiation by analysis and verification of design specification against requriemetns [FAA 2009a]. A Requirements Definition and Analysis Annex to the AADL standard is currently in preparation for ballot, which supports goal-oriented requirement capture and verification and has been demonstrated to support the process outlined in the Requirement Engineering Management Handbook.

Safety engineering of software-reliant systems continues to be a challenge [Leveson 2004, Dvorak 2009]. Traditionally, safety analysis consists of functional hazard assessment (FHA), Common Cause Analysis (CCA), and Safety Assessment (SSA). System hazard-assessment methods, such as the hazard and operability study with roots in the chemical industry, provide a predefined vocabulary for use with identification and coverage of potential faults. This vocabulary has been adapted for software hazard assessment, has led to a fault classification and a formalization of failure-assumption coverage [Powell 1992], and has been associated with architecture patterns such as feedback loops [Leveson 2009]. Such a fault classification is currently getting incorporated into the revision of the Error Model Annex standard of the SAE AADL [SAE 2006].

Safety analysis involves failure mode and effects analysis (FMEA) and Fault Tree Analysis (FTA). These analytical models are often created manually and due to their labor-intensive nature are usually performed once in the life cycle. System architecture models expressed in AADL and annotated with fault behavior specifications have been used to automatically generate FMEA and FTA models for analysis and also predict system reliability and availability [Hecht 2010].

VALIDATION AND VERIFICATION THROUGH STATIC ANALYSIS

Formal analysis frameworks ranging from timing analysis to model checking have become scalable solutions for incremental validation and verification throughout the life cycle.

Static analysis, simulation, and rapid prototyping through generation provide a means to discover inconsistencies in requirements, architecture, and design specifications early in the process.

Scheduling analysis techniques such as Rate Monotonic Analysis [Klein 1993] have facilitated the migration from legacy systems implemented by cyclic executive to preemptively scheduled systems in a partitioned architecture. Such scheduling and resource allocation techniques are currently being extended to support predicable use of multi-level caches and multi-core processors predictably for mixed criticality applications [DeNiz 2011].

Model checking technology has matured to become a practical technology for industrial scale systems interfacing formal validation and verification frameworks to established modeling tools such as Mathworks Simulink [Miller 2010] and applied to code to formally verify the implementation [Gurfinkel 2008].

Partitions are a fault containment concept supporting space partitioning through runtime enforced protected address spaces and time partitioning through runtime enforcement of resource budgets [Rushby 1999]. Partitions are the key concept in the ARINC 653 standard [Prisaznuk 2008] in support of DO-178B/C Level A/B certification. This concept is explicitly supported by the SAE AADL standard allowing for architecture fault analysis and for end-to-end latency analysis to take into account the impact of partition scheduling on latency and latency jitter [Feiler 2008].

The verification of safety requirements that were derived from safety analysis through model checking has been demosntrated on Avionics systems [Miller 2005]. A system-software co-engineering approach focusing on a coherent set of specification and analysis techniques for evaluation of system-level correctness, safety, dependability and performability of on-board computer-based aerospace systems has been demonstrated in the space domain with SAE AADL and the Error Model Annex [Katoen 2011].

This analytical virtual-integration approach is not intended to replace testing, but complement it by earlier discovery of testable and difficult to test for system level defects.

JUSTIFIED CONFIDENCE IN ASSURANCE EVIDENCE

The goal-structured assurance case [Kelly 2004] has been used extensively outside of the United States for a number of years to assure the safety[3] of nuclear reactors, railroad signaling systems, avionics systems, and so forth. Assurance cases are now being developed for other attributes (e.g., security of a software supply chain [Ellison 2008]). International Standards Organization (ISO) standard (15026-2) for assurance cases is now under development. The U.S. Food and Drug Administration (FDA) recently began to suggest their inclusion in regulatory submissions [FDA 2010].

In the best practice, an engineering organization will develop an assurance case in parallel with the system it assures. The case's structure will be used to influence assurance-centered actions throughout the life cycle. The assurance case guides what evidence is most needed to support claims, and serves as documentation for design decisions and their assumption, and estimating the impact of design and requirements changes by showing which portions of the case may be affected. The SEI is currently developing a measure of confidence for different forms of evidence against certain types of claims based on elimination of defeater [Weinstock 2013].

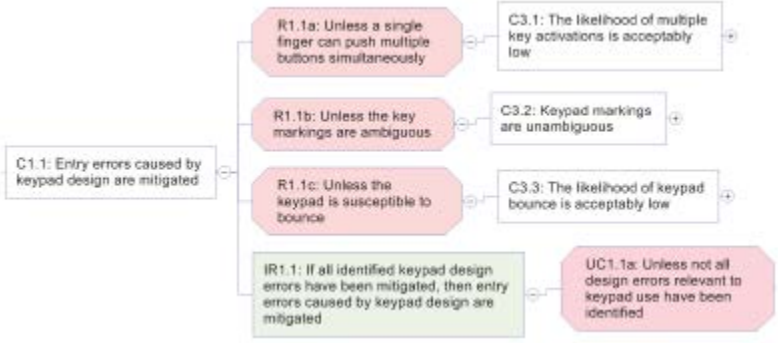

Figure 8: Eliminative Induction of Defeaters as Basis for Confidence

[3] When used to show safety, an assurance case is called a *safety case*.

CONCLUSION

Current build-then-test practice has limitations that lead to exponential cost explosion for large-scale software-reliant systems. Statistics show that there is a high percentage of fault leakage from requirements and architecture design into system integration and acceptance testing. Several root cause areas of software induced faults have been identified and can be investigated through architecture-centric analysis. A strategy to improve the quality of software-reliant systems through early discovery of system-level errors consists of four pillars:

- Capture of mission and safety-criticality requirements in analyzable form;
- Virtual integration of the physical system, hardware platform, and software architectures through consistent analyzable architecture models;
- Static analysis techniques applied to the models and actual system implementation to complement testing; and
- Incremental assurance of justified confidence through consistent end-to-end evidence throughout the development life cycle.

This will reduce the amount of rework at high cost late in the life cycle and increase the number of tests that pass without rework and retest. These methods are an integral part of a comprehensive architecture-focused model-based engineering practice based on incremental end-to-end validation conducted throughout the life cycle in parallel with system construction (Figure 8) [Feiler 2012].

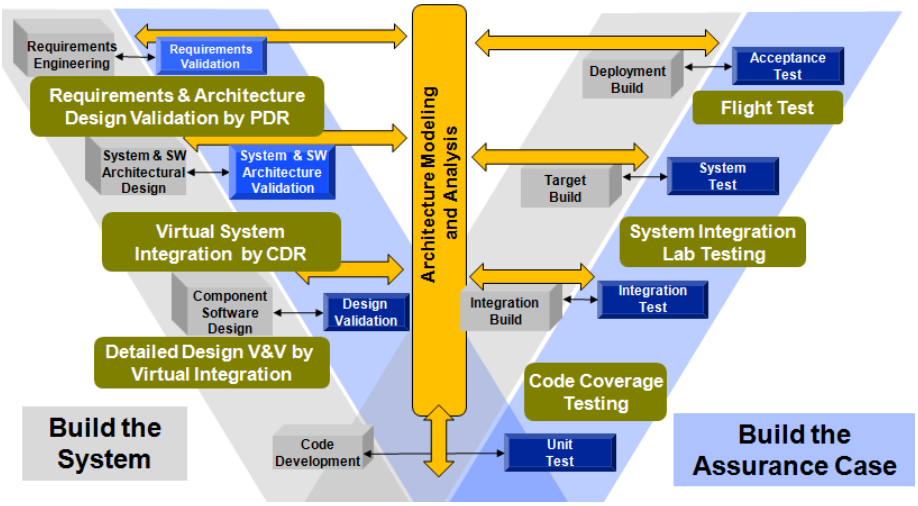

Figure 9: Incremental, End-to-End System Validation & Verification

REFERENCES

[AMSAA 2000]

Army Materiel Systems Analysis Activity. *AMSAA Reliability Growth Guide* (Technical Report No. TR-652). Department of Defense, 2000.

[ATSB 2008]

Australian Transport Safety Bureau. "Qantas Airbus A330 Accident Media Conference" (press release). ATSB, Oct. 2008. http://www.atsb.gov.au/newsroom/2008/release/2008_43.aspx

[Boehm 1981]

Boehm, B.W. *Software Engineering Economics*. Prentice Hall, 1981.

[Boehm 2006]

Boehm, B. "Some Future Trends and Implications for Systems and Software Engineering Processes." *Systems Engineering 9*, 1 (January 2006): 1–19.

[Boydston 2009]

Boydston, A. & Lewis, W. "Qualification and Reliability of Complex Electronic Rotorcraft Systems." *American Helicopter Society (AHS) Technical Specialists Meeting on Systems Engineering 2009*. Hartford, CT, Oct. 2009.

[Butler 1993]

Butler R., Finelli, G., *The Infeasibility of Quantifying the Reliability of Life-Critical Real-Time Software*. IEEE Transacitons on Software Engineering, Vol. 19, No.1, Jan 1993.

[Dabney 2003]

Dabney, J. B. *Return on Investment of Independent Verification and Validation Study Preliminary Phase 2B Report*. NASA, 2003.

[DeNiz 2011]

Dionisio de Niz, Raj Rajkumar, and Gabriel Moreno. "Overload Provisioning in Mixed-Criticality Cyber-Physical Systems" To Appear on: ACM Transactions on Embedded Computing Systems. 2011.

[Dvorak 2009]

Dvorak, Daniel L., ed. *NASA Study on Flight Software Complexity* (NASA/CR-2005-213912). Office of Chief Engineer Technical Excellence Program, NASA, 2009.

[Ellison 2008]

Ellison, R., Goodenough, J., Weinstock, C., & Woody, C. *Survivability Assurance for System of Systems* (CMU/SEI-2008-TR-008). Software Engineering Institute, Carnegie Mellon University, May 2008. http://www.sei.cmu.edu/reports/08tr008.pdf

[FAA 2002]

FAA Certification Authorities Software Team (CAST).*What Is a "Decision" in Application of Modified Condition/Decision Coverage (MC/DC) and Decision Coverage (DC)?* (Position Paper CAST-10). CAST, 2002.
http://www.faa.gov/aircraft/air_cert/design_approvals/air_software/cast/cast_papers/media/cast-10.pdf

[FAA 2009a]

Federal Aviation Administration. *Requirements Engineering Management Handbook DOT/FAA/AR-08/32*. 2008. http://www.faa.gov/aircraft/air_cert/design_approvals/air_software/media/AR-08-32.pdf

[FAA 2009b]

Federal Aviation Administration. *Requirements Engineering Management Findings Report DOT/FAA/AR-08/34*. 2008.
http://www.faa.gov/aircraft/air_cert/design_approvals/air_software/media/AR-08-34.pdf.

[FDA 2010]

U.S. Food and Drug Administration. *Guidance for Industry and FDA Staff – Total Life Cycle: Infusion Pump – Premarket Notification [510(k)] Submissions*.
http://www.fda.gov/MedicalDevices/DeviceRegulationandGuidance/GuidanceDocuments/ucm206153.htm

[Feiler 2008]

Feiler, Peter H. & Hansson, Jörgen. "Impact of Runtime Architectures on Control System Stability," R-2008-01-4C02. *Proceedings of the 4th International Congress on Embedded Real-Time Systems*. Toulouse, France, Jan. 2008. Societe des Ingenieurs de l'Automobile, 2008.

[Feiler 2009]

Feiler, Peter H. "Challenges in Validating Safety-Critical Embedded Systems," 09ATC-0271. *Proceedings of SAE International AeroTech Congress*. Seattle, WA, Nov. 2009. SAE International, 2009. https://www.sae.org/technical/papers/2009-01-3284

[Feiler 2010]

Feiler, P., Wrage, L., and Hansson, J. "System Architecture Virtual Integration: A Case Study." Embedded Real-time Software and Systems Conference (ERTS 2010). May 2010.

[Feiler 2012]

Feiler, P. H., Goodenough, J. B., Gurfinkel, A., Weinstock, C. B., & Wrage, L. *Reliability Validation and Improvement Framework*. Technical Report CMU/SEI-2012-SR-013, Software Engineering Institute, Carnegie Mellon University, 2012.

[Galin 2004]

Galin, D. *Software Quality Assurance: From Theory to Implementation*. Pearson/Addison-Wesley, 2004.

[Goodenough 2010]

Goodenough, J. B. *Evaluating Software's Impact on System and System of Systems Reliability* (white paper). Software Engineering Institute, Carnegie Mellon University, 2010. http://www.sei.cmu.edu/library/abstracts/whitepapers/swandreliability.cfm

[Gurfinkel 2008]

Gurfinkel A. & Chaki, S. "Combining Predicate and Numeric Abstraction for Software Model Checking." *In Proceedings of the Formal Methods in Computer-Aided Design International Conference (FMCAD 2008).* Portland, Oregon, November 2008. Curran Associates, 2008.

[Hayes 2003]

Hayes, J. H. "Building a Requirement Fault Taxonomy: Experiences from a NASA Verification and Validation Research Project," 49–59. *IEEE International Symposium on Software Reliability Engineering (ISSRE).* Denver, CO, Nov. 2003. IEEE Computer Society Press, 2003.

[Hecht 2010]

Hecht, M., Lam, A., Howes, R., & Vogl, C. "Automated Generation of Failure Modes and Effects Analyses from AADL Architectural and Error Models," Presented at the 22nd Annual Systems and Software Technology Conference, Salt Lake City, UT, Apr. 2010.

[Jackson 2007]

Jackson, Daniel, Thomas, Martyn, & Millett, Lynette I., eds. *Software for Dependable Systems: Sufficient Evidence?* National Research Council of the National Sciences, 2007.

[Katoen 2011]

Joost-Pieter Katoen. *Towards Trustworthy Aerospace Systems: An Experience Report.* In *16th International Workshop on Formal Methods for Industrial Critical Systems (FMICS).* pages 1–4. Volume 6959 of LNCS. Springer-Verlag, 2011.

[Kelly 2004]

Kelly, T. & Weaver, R. "The Goal Structuring Notation: A Safety Argument Notation." *Proceedings of International Workshop on Models and Processes for the Evaluation of COTS Components (MPEC 2004).* Edinburgh, Scotland, May 2004. IEEE Computer Society, 2004.

[Klein 1993]

Klein, M., Ralya, T., Pollak, B., Obenza, R., Gonzales Harbour, M., "A Practitioner's Handbook for Real-Time Analysis: Guide to Rate Monotonic Analysis for Real-Time Systems", Kluwer Academic, 1993.

[Leveson 2004]

Leveson, Nancy. "A New Accident Model for Engineering Safer Systems." *Safety Science 42*, 4 (April 2004): 237–270.

[Leveson 2009]

Leveson, Nancy G. *Engineering a Safer World: System Thinking Applied to Safety*. MIT Press, 2011. http://sunnyday.mit.edu/safer-world/safer-world.pdf

[Miller 2005]

Miller, S., Whalen, M., O'Brien, D., Heimdahl, M. P., & Joshi, A. *A Methodology for the Design and Verification of Globally Asynchronous/Locally Synchronous Architectures* (NASA/CR-2005-213912). NASA, 2005.

[Miller 2010]

Miller, S., Whalen, M., & Cofer, D. "Software Model Checking Takes Off." *Communications of the ACM 53*, 2 (February 2010): 58–64.

[NIST 2002]

National Institute of Standards and Technology. *The Economic Impacts of Inadequate Infrastructure for Software Testing* (Planning Report 02-3). NIST, 2002.

[Powell 1992]

Powell, D. "Failure Mode Assumptions and Assumption Coverage," 386–395. *Twenty-Second International Symposium on Fault-Tolerant Computing*. Boston, MA, July 1992. IEEE Computer Society Press, 1992.

[Prisaznuk 2008]

Prisaznuk, Paul J. "ARINC 653 Role in Integrated Modular Avionics (IMA)," 1.E.5-1–1.E.5-10. *Proceedings of the Digital Avionics Systems Conference*. St. Paul, MN, Oct. 2008. IEEE Computer Society Press, 2008.

[Redman 2010]

Redman, David, Ward, Donald, Chilenski, John, & Pollari, Greg. "Virtual Integration for Improved System Design," 57–64. *Proceedings of the First Analytic Virtual Integration of Cyber-Physical Systems Workshop*. San Diego, CA, Nov. 2010.

[RTCA 1992/2011]

RTCA, Inc. *Software Considerations in Airborne Systems and Equipment Certification* (Document No. RCTA/DO-178B). RTCA, 1992. (Document No. RCTA/DO-178C). RTCA, 2011.

[Rushby 1999]

Rushby, John. *Partitioning for Safety and Security: Requirements, Mechanisms, and Assurance* (DOT/FAA/AR-99/58, CR-1999-209347). NASA, 1999.

[SAE 1996]

SAE International. *Guidelines and Methods for Conducting the Safety Assessment Process on Civil Airborne Systems and Equipment* (Standard ARP 4761). SAE International, 1996.

[SAE 2006]

SAE International. *Architecture Analysis & Design Language (AADL) Annex Volume 1: AADL Meta Model & XML Interchange Format Annex, Error Model Annex, Programming Language Annex* (Standards Document AS5506/1).

[SAE 2010]

SAE International. Guidelines for Development of Civil Aircraft and Systems (Standard ARP 4754A). SAE International, 2010.

[SAE 2012]

SAE International. *Architecture Analysis & Design Language v2.1*(Standard AS5506B). SAE International, 2004–2012. http://www.sae.org/technical/standards/AS5506b

[Weinstock 2013]

Weinstock C., Goodenough J., Klein A., *Measuring Assurance Case Confidence using Baconian Probabilities*, 1st International Workshop on Assurance Cases for Software-intensive Systems (ASSURE 2013), at ICSE 2013.

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

DM-0000288