An Incremental Life-cycle Assurance Strategy for Critical System Certification

Software Engineering Institute
Carnegie Mellon University
Pittsburgh, PA 15213

Peter H. Feiler
Nov 4, 2014
Outline

Challenges in Safety-critical Software-intensive systems
An Architecture-centric Virtual Integration Strategy with SAE AADL
Improving the Quality of Requirements
Architecture Fault Modeling and Safety
Incremental Life-cycle Assurance of Systems
Summary and Conclusion
We Rely on Software for Safe Aircraft Operation

Even with the autopilot off, flight control computers still "command control surfaces to protect the aircraft from unsafe conditions such as a stall," the investigators said.

The unit continued to send false stall and speed warnings to the aircraft's primary computer and about 2 minutes after the initial fault "generated very high, random and incorrect values for the aircraft's angle of attack."

"This appears to be a unique event," the bureau said, adding that it was cruising at 37,000 feet (11,277 meters) when the computer fed incorrect information to the flight control system, the Australian Transport Safety Bureau said yesterday. The aircraft dropped 650 feet within seconds, slamming passengers and crew into the cabin ceiling, before the pilots regained control.

"Embedded software systems introduce a new class of problems not addressed by traditional system modeling & analysis."

Autopilot Off
A "preliminary analysis" of the Qantas plunge showed the error occurred in one of the jet's three air data inertial reference units, which caused the autopilot to disconnect, the ATSB said in a statement on its Web site.

The crew flew the aircraft manually to the end of the flight, except for a period of a few seconds, the bureau said.

Even with the autopilot off, flight control computers still "command control surfaces to protect the aircraft from unsafe conditions such as a stall," the investigators said.

The unit continued to send false stall and speed warnings to the aircraft's primary computer and about 2 minutes after the initial fault "generated very high, random and incorrect values for the aircraft's angle of attack."

The flight control computer then commanded a "nose-down aircraft movement, which resulted in the aircraft pitching down to a maximum of about 8.5 degrees," it said.

No "Similar Event"
"Airbus has advised that it is not aware of any similar event over the many years of operation of the Airbus," the bureau added, saying it will continue investigating.
Software Problems not just in Aircraft

Lexus GX 460 passes retest; Consumer Reports lifts "Don't Buy" label

Consumer Reports is lifting the Don't Buy: Safety Risk designation from the 2010 Lexus GX 460 SUV after recall work corrected the problem it displayed in one of our emergency handling tests. (See the original report and video: "Don't Buy: Safety Risk—2010 Lexus GX 460.")

We originally experienced the problem in a test that we use to evaluate what's called lift-off oversteer. In this test, as the vehicle is driven through a turn, the driver quickly lifts his foot off the accelerator pedal to see how the vehicle reacts. When we did this with our GX 460, its rear end slid out until the vehicle was almost sideways. Although the GX 460 has electronic stability control, which is designed to prevent a vehicle from sliding, the system wasn't interning quickly enough to stop the slide. We consider this a safety risk because in a real-world situation this could cause a rear tire to strike a curb or slide off the pavement, possibly causing the vehicle to roll over. Tall vehicles with a high center of gravity, such as the GX 460, heighten our concern. We are not aware, however, of any reports of injury related to this problem.

Lexus recently duplicated the problem on its own test track and developed a software upgrade for the vehicle's ESC system that would prevent the problem from happening. Dealers received the software fix last week and began notifying GX 460 owners to bring their vehicles in for repair.

Many appliances now rely on electronic controls and operating software. May 2010 Consumer Reports Magazine. But it turned out to be a problem for the Kenmore 4027 front-loader, which scored near the bottom in our February 2010 report.

Our tests found that the rinse cycles on some models worked improperly, resulting in unimpressive cleaning.

When Sears, which sells the washer, saw our February 2010 Ratings (available to subscribers), it worked with LG, which makes the washer, to figure out what was wrong. They quickly determined that a software problem was causing short or missing rinse and wash cycles, affecting wash performance. Sears and LG say they have reprogrammed the software on the models in their warehouses and on about 65 percent of the washers already sold, including the ones we had purchased.

Our retests of the reprogrammed Kenmore 4027 found that the cycles now worked properly, and the machine excelled. It now tops our Ratings (available to subscribers) of more than 50 front-loaders and we've made it a CR Best Buy.

If you own the washer, or a related model such as the Kenmore 4044 or Kenmore Elite 4051 or 4219, you should get a letter from Sears for a free service call. Or you can call 800-733-2299.

How do you upgrade washing machine software?
High Fault Leakage Drives Major Increase in Rework Cost

Aircraft industry has reached limits of affordability due to exponential growth in SW size and complexity.

70% Requirements & system interaction errors

80% late error discovery at high rework cost

Major cost savings through rework avoidance by early discovery and correction
A $10k architecture phase correction saves $3M

Where faults are introduced
Where faults are found

The estimated nominal cost for fault removal

Sources:

Software as % of total system cost
1997: 45% → 2010: 66% → 2024: 88%

Post-unit test software rework cost 50% of total system cost and growing

Total System Cost
Boeing 777 $12B
Boeing 787 $24B

Unit Test
Integration Test
System Test
Acceptance Test

Requirements Engineering
System Design
Software Architectural Design
Component Software Design
Code Development
Unit Test
Integration Test
System Test
Acceptance Test

10%, 50.5% 20x
0%, 9% 80x
20.5% 300-1000x
70%, 3.5% 1x
20%, 16% 5x
Mismatched Assumptions in System Interactions

System Engineer
- System Under Control
- Physical Plant Characteristics
  - Lag, proximity
- Hazards
  - Impact of system failures
- Operator Error
  - Automation & human actions

Control Engineer
- Control System
- Measurement Units, value range
  - Boolean/Integer abstraction
  - Air Canada, Ariane, 7500 Boolean variable architecture

Application Developer
- Application Software
- Concurrency
  - Communication
  - iTunes crashes on dual-cores

Hardware Engineer
- Compute Platform
- Runtime Architecture
- Distribution & Redundancy
  - Virtualization, load balancing, mode confusion

Embedded SW System Engineer
- Embedded software system as major source of hazards

Why do system level failures still occur despite fault tolerance techniques being deployed in systems?
Inconsistency between independently developed analytical models

Confidence that model reflects implementation

This aircraft industry experience has led to the System Architecture Virtual Integration (SAVI) initiative
Why UML, SysML Are Not Sufficient

- **System engineering**
  - Focus on system architecture and operational environment
  - SysML developed to capture interactions with outside world, as a standardized UML profile
  - 4 pillars/diagrams: requirements, parameterics (added in SysML), structure, behavior
- **Conceptual architecture**
  - UML-based component model
  - Architecture views (DoDAF, IEEE 1471)
  - Platform Independent model (PIM)
- **Embedded software system engineering**
  - OMG Modeling and Analysis of Real Time Embedded systems (MARTE) as UML profile
    - Borrowed Meta model concepts from AADL
    - Focus on modeling implementations
  - xUML insufficient for PSM (Kennedy-Carter, NATO ALWI study)
Impact of Three Step Data Request Protocol

Data Provider

- Request Sensor Data
  - «request»
  - Receive Sensor Data
  - «response»

- Request Current State
  - «request»
  - Receive Current State
  - «response»

- Request Target State
  - «request»
  - Receive Target State
  - «response»

Data Consumer

- Apply algorithm
- Publish Updated State
Operating as ARINC653 Partitioned System

Data Consumer Requirement

• Process data in 1 second

Partitions

• Provide space and time boundary enforcement
• Execute periodically on a static timeline at 1 second rate

Data request protocols across partitions

How much time does consumer actually have to process the data?
Who pays for the communication overhead?
Modeling is used in practice

- Modeling, analysis, and simulation in mechanical, control, computer hardware engineering

Current practice: software modeling close to source code

- Remember software through pictures
- MDE and MDA with UML
- Automatically generated documents

We need language for architecture modeling and analysis

- Strongly typed
- Well-defined execution and communication timing semantics
- Systematic approach to dealing with exceptional conditions
- Support for large-scale development
Outline

Challenges in Safety-critical Software-intensive systems
An Architecture-centric Virtual Integration Strategy with SAE AADL
Improving the Quality of Requirements
Architecture Fault Modeling and Safety
Incremental Life-cycle Assurance of Systems
Summary and Conclusion
AADL focuses on interaction between the three elements of a software-reliant mission and safety-critical systems.
The SAE AADL Standard Suite (AS-5506 series)

Core AADL language standard (V2.1-Sep 2012, V1-Nov 2004)
- Strongly typed language with well-defined execution and communication semantics
- Textual and graphical notation
- Standardized XMI interchange format

<table>
<thead>
<tr>
<th>Standardized AADL Extensions</th>
</tr>
</thead>
<tbody>
<tr>
<td>Error Model language for safety, reliability, security analysis</td>
</tr>
<tr>
<td>ARINC653 extension for partitioned architectures</td>
</tr>
<tr>
<td>Behavior Specification Language for modes and interaction behavior</td>
</tr>
<tr>
<td>Data Modeling extension for interfacing with data models (UML, ASN.1, …)</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>AADL Annex Extensions in Progress</th>
</tr>
</thead>
<tbody>
<tr>
<td>Requirements Definition and Assurance Annex</td>
</tr>
<tr>
<td>Synchronous System Specification Annex</td>
</tr>
<tr>
<td>Hybrid System Specification Annex</td>
</tr>
<tr>
<td>System Constraint Specification Annex</td>
</tr>
<tr>
<td>Network Specification Annex</td>
</tr>
</tbody>
</table>
AADL: The Language

Precise execution semantics for components
• Thread, process, data, subprogram, system, processor, memory, bus, device, virtual processor, virtual bus

Continuous control & event response processing
• Data and event flow, call/return, shared access
• End-to-End flow specifications

Operational modes & fault tolerant configurations
• Modes & mode transition

Modeling of large-scale systems
• Component variants, layered system modeling, packages, abstract, prototype, parameterized templates, arrays of components, connection patterns

Accommodation of diverse analysis needs
• Extension mechanism, standardized extensions
Architecture-Centric Quality Attribute Analysis

Single Annotated Architecture Model Addresses Impact Across Operational Quality Attributes

Safety & Reliability
- MTBF
- FMEA
- Hazard analysis

Security
- Intrusion
- Integrity
- Confidentiality

Data Quality
- Data precision/accuracy
- Temporal correctness
- Confidence

Resource Consumption
- Bandwidth
- CPU time
- Power consumption

Real-time Performance
- Execution time/Deadline
- Deadlock/starvation
- Latency

Architecture Model

Auto-generated analytical models
Multi-Fidelity End-to-end Latency in Control Systems

Common latency data from system engineering:
- Processing latency
- Sampling latency
- Physical signal latency

Impact of Scheduler Choice on Controller Stability
A. Cervin, Lund U., CCACSD 2006
Software-Based Latency Contributors

- Execution time variation: algorithm, use of cache
- Processor speed
- Resource contention
- Preemption
- Legacy & shared variable communication
- Rate group optimization
- Protocol specific communication delay
- Partitioned architecture
- Migration of functionality
- Fault tolerance strategy
Early Discovery and Incremental V&V through System Architecture Virtual Integration (SAVI)

- **Aircraft: (Tier 0)**
  - Aircraft system: (Tier 1)
    - Engine, Landing Gear, Cockpit, ...
    - Weight, Electrical, Fuel, Hydraulics,...
  - System & SW Engineering:
    - Mechatronics: Actuator & Wings
    - Safety Analysis (FHA, FMEA)
    - Reliability Analysis (MTTF)
  - OEM & Subcontractor:
    - Subsystem proposal validation
    - Functional integration consistency
    - Data bus protocol mappings
  - Repeated Virtual Integration Analyses:
    - Power/weight
    - MIPS/RAM, Scheduling
    - End-to-end latency
    - Network bandwidth

- **LRU/IMA System: (Tier 2)**
  - Hardware platform, software partitions
  - Power, MIPS, RAM capacity & budgets
  - End-to-end flow latency

- **Subcontracted software subsystem: (Tier 3)**
  - Tasks, periods, execution time
  - Software allocation, schedulability
  - Generated executables

- **Proof of Concept Demonstration and Transition by Aerospace industry initiative**
  - Architecture-centric model-based software and system engineering
  - Architecture-centric model-based acquisition and development process
  - Multi notation, multi team model repository & standardized model interchange

- Multi-tier system & software architecture (in AADL)
- Incremental end-to-end validation of system properties
Multi-Notation Approach to Architecture-centric Virtual System and Software Integration

AADL

Application Software Runtime Architecture (task & communication)

Physical System Architecture (interface with embedded SW/HW)

Computer Platform Architecture (processors & networks)

Physical Components (mechanical, electrical, heat)

Hardware Components (circuits & logic)

VHDL

Application Software Components (source code)

Java, UML, Simulink

Simulink, Modelica

Operational Environment (People, Use scenarios)

UML

Embedded Software Engineering

System Engineering

Control Engineering

Application Software Engineering

Mechanical Engineering

Electrical Engineering

SAVI Approach

Model delivery with interchange standards

Model repository content with intra and inter-model consistency

Tool chain flexibility for contractor
Architecture-centric Virtual Integration Practice (ACVIP)

Iterative architecture design, safety analysis, and requirement decomposition

Stakeholder and Quality Attribute (QA) driven architecture-centric requirement specification

Model-based architecture specifications & multidimensional QA analysis

Transformation and code generation based on verified architecture specifications

Testing against verified specifications and models

Assurance plan and execution

Architecture-centric virtual integration and compositional verification of requirements

BUSINESS AND MISSION GOALS

ARCHITECTURE

SYSTEM
Outline

Challenges in Safety-critical Software-intensive systems
An Architecture-centric Virtual Integration Strategy with SAE AADL
Improving the Quality of Requirements
Architecture Fault Modeling and Safety
Incremental Life-cycle Assurance of Systems
Summary and Conclusion
Certification & Recertification Challenges

Certification: assure the quality of the delivered system

- **Sufficient evidence** that a **system implementation** meets **system requirements**
- **Quality of requirements** and **quality of evidence** determines quality of system

Certification related rework cost

- Currently 50% of total system cost and growing

Recertification Challenge

- Desired cost of recertification in **proportion** to change

**Improve quality of requirements and evidence**

**Perform verification compositionally throughout the life cycle**
## Industry Practice in DO-178B Compliant Requirements Capture

### Industry Survey in 2009 FAA Requirements Engineering Study

<table>
<thead>
<tr>
<th>Notation</th>
<th>System Requirements</th>
<th>Data Interconnect (ICD)</th>
<th>High-Level Software Requirements</th>
<th>Low-Level Software Requirements</th>
<th>Hardware Requirements</th>
</tr>
</thead>
<tbody>
<tr>
<td>English Text or Shall Statements</td>
<td>39</td>
<td>27</td>
<td>36</td>
<td>32</td>
<td>29</td>
</tr>
<tr>
<td>Tables and Diagrams</td>
<td>31</td>
<td>30</td>
<td>30</td>
<td>19</td>
<td>18</td>
</tr>
<tr>
<td>UML Use Cases</td>
<td>1</td>
<td>2</td>
<td>4</td>
<td></td>
<td></td>
</tr>
<tr>
<td>UML Sequence Diagrams</td>
<td>3</td>
<td>6</td>
<td>1</td>
<td></td>
<td></td>
</tr>
<tr>
<td>UML State Diagrams</td>
<td>1</td>
<td>7</td>
<td>1</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Executable Models (e.g. Simulink, SCADE Suite, etc.)</td>
<td>7</td>
<td>1</td>
<td>8</td>
<td>8</td>
<td>1</td>
</tr>
<tr>
<td>Data Flow Diagrams (e.g. Yourdon)</td>
<td>4</td>
<td>6</td>
<td>9</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

### Need analyzable & executable specifications

- Other (Specify)XML: 1
- Operational models or prototypes: 1 1 1
- UML: 1 1
Requirement Quality Challenge

<table>
<thead>
<tr>
<th>Requirements error</th>
<th>%</th>
</tr>
</thead>
<tbody>
<tr>
<td>Incomplete</td>
<td>21%</td>
</tr>
<tr>
<td>Missing</td>
<td>33%</td>
</tr>
<tr>
<td>Incorrect</td>
<td>24%</td>
</tr>
<tr>
<td>Ambiguous</td>
<td>6%</td>
</tr>
<tr>
<td>Inconsistent</td>
<td>5%</td>
</tr>
</tbody>
</table>

There is more to requirements quality than “shall”s and stakeholder traceability.


<table>
<thead>
<tr>
<th>User Reqts</th>
<th>Technical Reqts</th>
<th>Design</th>
<th>Test Cases</th>
</tr>
</thead>
</table>

Browsable links/Coverage metrics

IEEE Std 830-1998 characteristics of a good requirements specification:
- Correct
- Unambiguous
- Complete
- Consistent
- Ranked for importance and/or stability
- Verifiable
- Modifiable
- Traceable

System to SW requirements gap [Boehm 2006]

How do we verify low level SW requirements against system requirements?

When StartUpComplete is TRUE in both FADECs and SlowStartupComplete is FALSE, the FADECStartupSW shall set SlowStartupInComplete to TRUE.
Stakeholder Needs and Requirement Categories

Table 2. Example of Stakeholder Requirements Classification. (SEBoK Original)

<table>
<thead>
<tr>
<th>Type of Stakeholder Requirement</th>
<th>Types of System Requirement</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Service or Functional</td>
<td>Functional Requirements</td>
<td>Describe qualitatively the system functions or tasks to be performed in operation.</td>
</tr>
<tr>
<td>Operational</td>
<td>Performance Requirements</td>
<td>Define quantitatively the extent of system performance and are task.</td>
</tr>
<tr>
<td></td>
<td>Usability Requirements</td>
<td>Define the quality of system use by users or end-users.</td>
</tr>
<tr>
<td>Interface</td>
<td>Interface Requirements</td>
<td>Define how the system is required to interact with the system, including human and machine interaction.</td>
</tr>
<tr>
<td>Environmental</td>
<td>Operational Requirements</td>
<td>Define the operational conditions, time, and the environment in which the system operates.</td>
</tr>
<tr>
<td>Utilization Characteristics</td>
<td>Mode and/or State Requirements</td>
<td>Define the various operational states of the system.</td>
</tr>
<tr>
<td>Human Factors</td>
<td>Logistical Requirements</td>
<td>Define potential extension, growth, or change.</td>
</tr>
<tr>
<td></td>
<td>Design and Realization</td>
<td>Define constraints on weights, dimensions, and structural interfaces.</td>
</tr>
<tr>
<td>Process Constraints</td>
<td>Design Constraints</td>
<td>Define the limits on the options that can be selected by the provider system element, or system interface.</td>
</tr>
<tr>
<td>Project Constraints</td>
<td>Project Constraints</td>
<td>Define relevant and applicable laws, regulations, and standards.</td>
</tr>
<tr>
<td>Business Model Constraints</td>
<td>Cost and Schedule Constraints</td>
<td>Define, for example, the cost and schedule constraints.</td>
</tr>
</tbody>
</table>

Leveson System Theoretic Framework

System, operational environment, development and V&V process
Requirements for a Patient Therapy System

The patient shall never be infused with a single air bubble more than 5ml volume.

When a single air bubble more than 5ml volume is detected, the system shall stop infusion within 0.2 seconds.

When piston stop is received, the system shall stop piston movement within 0.01 seconds.

The system shall always stop the piston at the bottom or top of the chamber.

Requirements and Design Information

1. The patient shall never be infused with a single air bubble more than 5ml volume.

2. When a single air bubble more than 5ml volume is detected, the system shall stop infusion within 0.2 seconds.

3. When piston stop is received, the system shall stop piston movement within 0.01 seconds.

4. The system shall always stop the piston at the bottom or top of the chamber.

Typical requirement documents span multiple levels of a system architecture

We have made architecture design decisions.

We have effectively specified a partial architecture

Adapted from M. Whalen presentation
System Specification and Requirements Coverage

- Requirements
- Guarantees
- Assumptions

- Environmental Assumptions
- Precondition
- Postcondition
- Invariant

- Developmental Requirements
  - Modular
  - Assurability

- Quality attribute utility tree

- Interaction contract:
  - match input assumption
  - with guarantee

- Implementation constraints

- Mission Requirements
  - Function
  - Behavior
  - Performance

- Dependability Requirements
  - Reliability
  - Safety
  - Security

- Exceptional condition
Architecture-led Requirement & Hazard Specification

Error Propagation Ontology

Leveson pattern
Outline

Challenges in Safety-critical Software-intensive systems
An Architecture-centric Virtual Integration Strategy with SAE AADL
Improving the Quality of Requirements
Architecture Fault Modeling and Safety
Incremental Life-cycle Assurance of Systems
Summary and Conclusion
AADL Error Model Scope and Purpose

System safety process uses many individual methods and analyses, e.g.

- hazard analysis
- failure modes and effects analysis
- fault trees
- Markov processes

Goal: a general facility for modeling fault/error/failure behaviors that can be used for several modeling and analysis activities.

Annotated architecture model permits checking for consistency and completeness between these various declarations.

Related analyses are also useful for other purposes, e.g.

- maintainability
- availability
- Integrity
- Security

SAE ARP 4761 Guidelines and Methods for Conducting the Safety Assessment Process on Civil Airborne Systems and Equipment

Demonstrated in SAVI Wheel Braking System Example

Error Model Annex can be adapted to other ADLs
Error Propagation Contracts

Incoming/Assumed
- Error Propagation
  Propagated errors
- Error Containment:
  Errors not propagated

Outgoing/Contract
- Error Propagation
- Error Containment

Bound resources
- Error Propagation
- Error Containment
- Propagation to resource

"Not" on propagated indicates that this error type is intended to be contained. This allows us to determine whether propagation specification is complete.

Legend
- Propagation of Error Types
- Direction
- Processor
- HW Binding
- Not propagated

Error Flow through component
Path P1.NoData -> P2.NoData
Source P2.BadValue
Path processor.NoResource -> P2.NoData
System engineering activity with focus on failing components.
Discovery of Unexpected PSSA Hazard through Repeated Virtual Integration

Unexpected propagation of corrupted Airspeed data results in Stall due to miss-correction.

Vibration causes boards to touch which causes EGI data corruption.

EGI maintainer adds corrupted data hazard to model. Error Model analysis of integrated model detects unhandled propagation.

Anticipated: No EGI data

Anticipated: NoService

Anticipated: NoService

Anticipated: No Stall Propagation
Recent Automated FMEA Experience

Failure Modes and Effects Analyses are rigorous and comprehensive reliability and safety design evaluations

- Required by industry standards and Government policies
- When performed manually are usually done once due to cost and schedule
- If automated allows for
  - multiple iterations from conceptual to detailed design
  - Tradeoff studies and evaluation of alternatives
  - Early identification of potential problems

Largest analysis of satellite to date consists of 26,000 failure modes

- Includes detailed model of satellite bus
- 20 states perform failure mode
- Longest failure mode sequences have 25 transitions (i.e., 25 effects)
Outline

Challenges in Safety-critical Software-intensive systems
An Architecture-centric Virtual Integration Strategy with SAE AADL
Improving the Quality of Requirements
Architecture Fault Modeling and Safety
Incremental Life-cycle Assurance of Systems
Summary and Conclusion
### Reliability & Qualification Improvement Strategy

#### Four pillars for Improving Quality of Critical Software-reliant Systems

<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Mission Requirements</td>
<td>Model Repository</td>
<td>Operational &amp; failure modes</td>
<td>Resource, Timing &amp; Performance Analysis</td>
</tr>
<tr>
<td>Function</td>
<td>Architecture Model</td>
<td></td>
<td>Reliability, Safety, Security Analysis</td>
</tr>
<tr>
<td>Behavior</td>
<td>Component Models</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Performance</td>
<td>System Implementation</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Survivability Requirements</td>
<td>System configuration</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Reliability</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Safety</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Security</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
## Verification Actions

**Table 2. Main Ontology Elements as Handled within Verification. (SEBoK Original)**

<table>
<thead>
<tr>
<th>Element</th>
<th>Definition</th>
</tr>
</thead>
<tbody>
<tr>
<td>Verification Action</td>
<td>A verification action describes what must be verified (the element as reference), on which element, the expected result, the verification technique to apply, on which level of decomposition.</td>
</tr>
<tr>
<td></td>
<td>Identifier, name, description</td>
</tr>
<tr>
<td>Verification Procedure</td>
<td>A verification procedure is a process to be followed for verification actions that have been identified. It is associated with a verification tool.</td>
</tr>
<tr>
<td>Verification Tool</td>
<td>A verification tool is a means for executing a verification procedure. It is associated with a verification configuration.</td>
</tr>
<tr>
<td>Verification Configuration</td>
<td>An element that contains information about the verification tool and associated verification procedure.  It is also associated with verification actions.</td>
</tr>
</tbody>
</table>

**Table 3. Verification Techniques. (SEBoK Original)**

<table>
<thead>
<tr>
<th>Verification Technique</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Inspection</td>
<td>Technique based on visual or dimensional examination of an element; the verification relies on the human senses or uses simple methods of measurement and handling. Inspection is generally non-destructive, and typically includes the use of sight, hearing, smell, touch, and taste, simple physical manipulation, mechanical and electrical gauging, and measurement. No stimuli (tests) are necessary. The technique is used to check properties or characteristics best determined by observation (e.g. - paint color, weight, documentation, listing of code, etc.).</td>
</tr>
<tr>
<td>Analysis</td>
<td>Technique based on analytical evidence obtained without any intervention on the submitted element using mathematical or probabilistic calculation, logical reasoning (including the theory of predicates), modeling and/or simulation under defined conditions to show theoretical compliance. Mainly used where testing to realistic conditions cannot be achieved or is not cost-effective.</td>
</tr>
<tr>
<td>Analogy or Similarity</td>
<td>Technique based on evidence of similar elements to the submitted element or on experience feedback. It is absolutely necessary to show by prediction that the context is invariant that the outcomes are transposable (models, investigations, experience feedback, etc.). Simplicity can only be used if the submitted element is similar in design, manufacture, and use; equivalent or more stringent verification actions were used for the similar element, and the intended operational environment is identical to or less rigorous than the similar element.</td>
</tr>
<tr>
<td>Demonstration</td>
<td>Technique used to demonstrate correct operation of the submitted element against operational and observable characteristics without using physical measurements (no or minimal instrumentation or test equipment). Demonstration is sometimes called ‘field testing’. It generally consists of a set of tests selected by the supplier to show that the element response to stimuli is suitable or to show that operators can perform their assigned tasks when using the element. Observations are made and compared with predetermined/expected responses. Demonstration may be appropriate when requirements or specification are given in statistical terms (e.g. meant time to repair, average power consumption, etc.).</td>
</tr>
<tr>
<td>Test</td>
<td>Technique performed onto the submitted element by which functional, measurable characteristics, operability, supportability, or performance capability is quantitatively verified when subjected to controlled conditions that are real or simulated. Testing often uses special test equipment or instrumentation to obtain accurate quantitative data to be analyzed.</td>
</tr>
<tr>
<td>Sampling</td>
<td>Technique based on verification of characteristics using samples. The number, tolerance, and other characteristics must be specified to be in agreement with the experience feedback.</td>
</tr>
</tbody>
</table>
Integrated Approach to Requirement V&V through Assurance Automation

- Safety hazards are part of the picture
- Requirement coverage
- Assumption evidence
- Evidence records in terms of claims that requirements have been met
- Linkage to automated test harnesses
- Generated assurance cases
Secure Mathematically-Assured Composition of Control Models

Key Problem
Many vulnerabilities occur at component interfaces. How can we use formal methods to detect these vulnerabilities and build provably secure systems?

Technical Approach
- Develop a complete, formal architecture model for UAVs that provides robustness against cyber attack
- Develop compositional verification tools driven from the architecture model for combining formal evidence from multiple sources, components, and subsystems
- Develop synthesis tools to generate flight software for UAVs directly from the architecture model, verified components, and verified operation system

Accomplishments
- Created AADL model of vehicle hardware & software architecture
- Identified system-level requirements to be verified based on input from Red Team evaluations
- Developed Resolute analysis tool for capturing and evaluating assurance case arguments linked to AADL model
- Developed example assurance cases for two security requirements
- Developed synthesis tool for auto-generation of configuration data and glue code for OS and platform hardware
Building the Assurance Case throughout the Life Cycle

Continuous Confidence Measure throughout Life Cycle that a System Meets its Requirements

Incremental Life-Cycle Assurance
Feiler, Nov 4, 2014
© 2014 Carnegie Mellon University

Incremental Evolution and Execution of Assurance Plans

Incremental Architecture & Requirement Evolution

Auto-generation from verified models
AADL&SCADE/Simulink
Ada SPARK/Ravenscar
MISRA C

Confidence = Requirement Quality + Evidence Quality

Need for Multi-valued Argumentation Logic

Confidence = Requirement Quality + Evidence Quality

Auto-generated Assurance Cases

FY15/16 line funded project
Outline

Challenges in Safety-critical Software-intensive Systems
An Architecture-centric Virtual Integration Strategy with SAE AADL
Improving the Quality of Requirements
Architecture Fault Modeling and Safety
Incremental Life-cycle Assurance of Systems
Summary and Conclusion
Architecture-centric Virtual System Integration & Incremental Life-cycle Assurance

Reduce risks
- Analyze system early and throughout life cycle
- Understand system wide impact
- Validate assumptions across system

Increase confidence
- Validate models to complement integration testing
- Validate model assumptions in operational system
- Evolve system models in increasing fidelity

Reduce cost
- Fewer system integration problems
- Incremental evidence through compositional verification
- Fewer verification steps through generation from single source and verified models
References

AADL Website [www.aadl.info](http://www.aadl.info) and AADL Wiki [www.aadl.info/wiki](http://www.aadl.info/wiki)

Blog entries and podcasts on AADL at [www.sei.cmu.edu](http://www.sei.cmu.edu)

AADL Book in SEI Series of Addison-Wesley

On AADL and Model-based Engineering
[http://www.sei.cmu.edu/library/assets/ResearchandTechnology_AADLandMBE.pdf](http://www.sei.cmu.edu/library/assets/ResearchandTechnology_AADLandMBE.pdf)

On an architecture-centric virtual integration practice and SAVI

On an a four pillar improvement strategy for software system verification and qualification

Contact Information

Peter H. Feiler  
Principal Researcher  
RTSS  
Telephone: +1 412-268-7790  
Email: phf@sei.cmu.edu

U.S. Mail  
Software Engineering Institute  
Customer Relations  
4500 Fifth Avenue  
Pittsburgh, PA 15213-2612  
USA

Web  
Wiki.sei.cmu.edu/aadl  
www.aadl.info

Customer Relations  
Email: info@sei.cmu.edu  
SEI Phone: +1 412-268-5800  
SEI Fax: +1 412-268-6257