<table>
<thead>
<tr>
<th>1. REPORT DATE</th>
<th>2. REPORT TYPE</th>
<th>3. DATES COVERED</th>
</tr>
</thead>
<tbody>
<tr>
<td>20 OCT 2014</td>
<td>N/A</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>4. TITLE AND SUBTITLE</th>
<th>5a. CONTRACT NUMBER</th>
</tr>
</thead>
<tbody>
<tr>
<td>AADL and Model-based Engineering</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>6. AUTHOR(S)</th>
<th>5b. GRANT NUMBER</th>
</tr>
</thead>
<tbody>
<tr>
<td>Feiler /Peter</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES)</th>
<th>5c. PROGRAM ELEMENT NUMBER</th>
</tr>
</thead>
<tbody>
<tr>
<td>Software Engineering Institute Carnegie Mellon University Pittsburgh, PA 15213</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>8. PERFORMING ORGANIZATION REPORT NUMBER</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>12. DISTRIBUTION/AVAILABILITY STATEMENT</th>
</tr>
</thead>
<tbody>
<tr>
<td>Approved for public release, distribution unlimited.</td>
</tr>
</tbody>
</table>

13. SUPPLEMENTARY NOTES
The original document contains color images.

14. ABSTRACT

15. SUBJECT TERMS

16. SECURITY CLASSIFICATION OF:

<table>
<thead>
<tr>
<th>17. LIMITATION OF ABSTRACT</th>
</tr>
</thead>
<tbody>
<tr>
<td>SAR</td>
</tr>
</tbody>
</table>

18. NUMBER OF PAGES
51

19a. NAME OF RESPONSIBLE PERSON

Standard Form 298 (Rev. 8-98)
Prescribed by ANSI Std Z39-18
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."

Mayday call when it suddenly changed altitude during a flight from Singapore to Perth, Qantas said.

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."

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

May 7, 2010

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 intervening 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 of 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.

We contacted the Lexus dealership from which we had anonymously bought the vehicle and made an appointment to have the recall work performed. The work took about an hour and a half.

Following that, we again put the SUV through our full series of emergency handling tests. This time, the ESC system intervened earlier and its rear did not slide out in the lift-off oversteer test. Instead, the vehicle understeered—or plowed—when it exceeded its limits of traction, which is a more common result and makes the vehicle more predictable and less likely to roll over. Overall, we did not experience any safety concerns with the corrected GX 460 in our handling tests.

How do you upgrade washing machine software?

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 an 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.
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

70%, 3.5% 1x

10%, 50.5% 20x

20%, 16% 5x

20.5% 300-1000x

Where faults are introduced

Where faults are found

The estimated nominal cost for fault removal

Sources:

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

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

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

Where faults are introduced

Where faults are found

The estimated nominal cost for fault removal

Sources:
Mismatched Assumptions in System Interactions

- System Engineer
  - System Under Control
  - Physical Plant Characteristics
    - Lag, proximity
  - System User/Environment
    - 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
  - System User/Environment
    - Hazards

- Application Developer
  - Application Software
    - Concurrency
    - Communication
      - ITunes crashes on dual-cores
  - System User/Environment
    - Hazards

- Hardware Engineer
  - Compute Platform
    - Distribution & Redundancy
      - Virtualization, load balancing, mode confusion
  - System User/Environment
    - Hazards

Why do system level failures still occur despite fault tolerance techniques being deployed in systems?

Embedded software system as major source of hazards
Model-based Engineering Pitfalls

The system

Inconsistency between independently developed analytical models

System models

Confidence that model reflects implementation

System 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

Data Consumer

Request Sensor Data

Request Current State

Request Target State

Receive Sensor Data

Receive Current State

Receive Target State

Apply algorithm

Publish Updated State

loop
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?
Model-based Engineering in Practice

Modeling is used in practice

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

Current practice: modeling and software

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

We need language for architecture modeling

- 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
The Roots of AADL

- Strong typing, syntax borrowed from Ada
- Data and event ports, Operational modes
- Scheduling analysis and code generation

1994: Application to Missile Guidance System by Vestal and Lewis
- Three Week Port to Dual Processor Hardware

1997: MetaH Style for ACME by Peter Feiler and Jun Li
- CMU ECE Ph.D. on multi-dimensional analysis for Simplex architectures

1998: Error Model added to MetaH by Steve Vestal
- Generation of fault trees and Markov models

1999: Requirements Document for AADL Standard
- Industry input: packages, messages
- The best of MetaH and ACME
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 semantics
- Textual and graphical notation
- Standardized XMI interchange format

Standardized AADL Extensions
- Error Model language for safety, reliability, security analysis
- ARINC653 extension for partitioned architectures
- Behavior Specification Language for modes and interaction behavior
- Data Modeling extension for interfacing with data models (UML, ASN.1, …)

AADL Annex Extensions in Progress
- Requirements Definition and Assurance Annex
- Synchronous System Specification Annex
- Hybrid System Specification Annex
- System Constraint Specification Annex
- Network Specification Annex
System Level Fault Root Causes

Violation of data stream assumptions
- Stream miss rates, Mismatched data representation, Latency jitter & age

Partitions as Isolation Regions
- Space, time, and bandwidth partitioning
- Isolation not guaranteed due to undocumented resource sharing
- fault containment, security levels, safety levels, distribution

Virtualization of time & resources
- Logical vs. physical redundancy
- Time stamping of data & asynchronous systems

Inconsistent System States & Interactions
- Modal systems with modal components
- Concurrency & redundancy management
- Application level interaction protocols

Performance impedance mismatches
- Processor, memory & network resources
- Compositional & replacement performance mismatches
- Unmanaged computer system resources

End-to-end latency analysis
- Port connection consistency

Process and virtual processor to model partitioned architectures

Virtual processors & buses
- Multiple time domains

Operational and failure modes
- Interaction behavior specification
- Dynamic reconfiguration
- Fault detection, isolation, recovery

Resource allocation & deployment configurations
- Resource budget analysis
- & scheduling analysis

Codified in Virtual Upgrade Validation method
Architecture-Centric Quality Attribute Analysis

Single Annotated Architecture Model Addresses Impact Across Operational Quality Attributes

Safety & Reliability
- MTBF
- FMEA
- Hazard analysis

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

Security
- Intrusion
- Integrity
- Confidentiality

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

Resource Consumption
- Bandwidth
- CPU time
- Power consumption

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
Sampling of International Efforts Leveraging SAE AADL

**Compositional Timing Framework**
- OSD 2014

**P Project**
- Auto Code Gen
- 2011-2014

**OPEES**
- Formal analysis
- 2011-2014

**OpenGroup**
- Real-Time Forum
- EU + US partners
- 2008-current

**ITEA SPICES**
- Model-Driven Embedded Systems Engineering
- 15 partners €16M 2006-2009

**TOPCASED**
- Open Source Embedded Systems Tool Framework
- 28 partners €20+M 2005-2009

**IST ARTIST2**
- Embedded Systems Center of Excellence
- 2007-2012

**IST P Project**
- Auto Code Generation
- 2011-2014

**ESA TASTE**
- System & SW Validation & Generation
- 2007-2010

**Flex-eWare**
- Auto Code Generation
- 2007-2010

**EC ASSERT**
- Proof-based Satellite Architectures

**ESA TASTE**
- System & SW Validation & Generation
- 2010-current

**PARSEC**
- Safety/security
- 2010-2013

**AADL Inspector**
- Ellidiss
- 2010-current

**RAMSES**
- Avionics Workbench
- 2011-current
- $2M per year

**PROARTIS**
- Partitioned RT systems
- 2010-2013 € 1.8M

**MASIW**
- Avionics Workbench
- 2011-current

**DARPA META**
- Complex System Engineering
- 2010-2012

**DARPA HACMS**
- Security in CPS
- RC formal methods
- 2013-2015

**Integrated Clinical Environment Device Certification**
- FDA KSU
- 2011-current

**D-MILS**
- Design of Secure Systems
- 2013 - $4.9M

**AVSI SAVI**
- Analysis-based System Validation
- 12 partners $20M 2008-current

**DARPA COMPASS**
- System SW Co-engineering
- 2008-current

**OpenGroup Real-Time Forum**
- EU + US partners
- 2008-current

**OpenGroup**
- Real-Time Forum
- EU + US partners
- 2008-current
Architecture-centric Virtual System Integration
Evolution, Maturation and Transition

Pilot Projects in US, Europe, Japan, China

System Architecture Virtual Integration (SAVI) Software & Systems Engineering

SAE AADL Standard & Tool Support: Research Transition Platform

US & European Research Initiatives

Other Standards and Regulatory Guidance


ESA ASSERT
JPL Mission Data System
Apache Model-based ATAM
COMPASS SE-SW Co-engineering

AADLV1 Timing
Software & System Co-engineering
Multi-team Safety
Requirements Assurance

AADLV1 Error Model
European Commission SLIM/FIACRE
DARPA META
DARPA HACMS Security
AADL Engineering Workbench

OMG MARTE Embedded Systems
ARINC653 Partitions
Avionics Network Standards
System Safety Practice Standards
Regulatory Guidance NRC, FDA, UL

DARPA MetaH ACME
European Commission SLIM/FIACRE
DARPA META
DARPA HACMS Security
AADL Engineering Workbench

US & European Research Initiatives

Other Standards and Regulatory Guidance

AADLV1 Timing
Software & System Co-engineering
Multi-team Safety
Requirements Assurance

AADLV1 Error Model
European Commission SLIM/FIACRE
DARPA META
DARPA HACMS Security
AADL Engineering Workbench

OMG MARTE Embedded Systems
ARINC653 Partitions
Avionics Network Standards
System Safety Practice Standards
Regulatory Guidance NRC, FDA, UL

DARPA MetaH ACME
European Commission SLIM/FIACRE
DARPA META
DARPA HACMS Security
AADL Engineering Workbench

US & European Research Initiatives

Other Standards and Regulatory Guidance

OADL and MBE
Feller, Oct 20, 2014
© 2014 Carnegie Mellon University
22
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

**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

**Repeated Virtual Integration Analyses:**
- Power/weight
- MIPS/RAM, Scheduling
- End-to-end latency
- Network bandwidth

**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)
  - Application Software Components (source code)
    - Java, UML, Simulink

- **SysML**: Physical System Architecture (interface with embedded SW/HW)
  - Physical Components (mechanical, electrical, heat)
    - Simulink, Modelica

- **Operational Environment**: People, Use scenarios
  - UML

- **SAVI Approach**
  - Model delivery with interchange standards
  - Model repository content with intra and inter-model consistency
  - Tool chain flexibility for contractor

- **Computer Platform Architecture**: (processors & networks)
  - Hardware Components (circuits & logic)
    - VHDL

- **Embedded Software Engineering**
- **System Engineering**
- **Application Software Engineering**
- **Mechanical Engineering**
- **Control Engineering**
- **Application Software Engineering**
- **Electrical Engineering**
Architecture-centric Virtual Integration Practice (ACVIP)

- Iterative architecture design, safety analysis, and requirement decomposition
- Stakeholder and Quality Attribute (QA) driven architecture-centric requirement specification
- Architecture-centric virtual integration and compositional verification of requirements
- Model-based architecture specifications & multi-dimensional QA analysis
- Testing against verified specifications and models
- Assurance plan and execution
- Transformation and code generation based on verified architecture specifications

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
## Current Industry Practice in DO-178B Compliant Requirements Capture

### Industry Survey in 2009 FAA Requirements Engineering Study

#### Notation

Enter an “x” in every row/column cell that applies

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

**Need analyzable & executable specifications**

<table>
<thead>
<tr>
<th>Notation</th>
<th>System Requirements</th>
<th>Data Interconnect (ICD)</th>
<th>High-Level Software Requirement</th>
<th>Low-Level Software Requirement</th>
<th>Hardware Requirements</th>
</tr>
</thead>
<tbody>
<tr>
<td>Other (Specify)XML</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Operational models or prototypes</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td></td>
<td></td>
</tr>
<tr>
<td>UML</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

### Tool

Enter an “x” in every row/column cell that applies

<table>
<thead>
<tr>
<th>Tool</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>DOORS (e.g., Microsoft Access)</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Rational ROSE®</td>
<td></td>
<td></td>
<td></td>
<td>1</td>
<td></td>
</tr>
<tr>
<td>RDD-100®</td>
<td></td>
<td></td>
<td>1</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Requisite Pro®</td>
<td></td>
<td></td>
<td></td>
<td>5</td>
<td>5</td>
</tr>
<tr>
<td>Rhapsody</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>1</td>
</tr>
<tr>
<td>SCADE Suite</td>
<td></td>
<td></td>
<td>2</td>
<td>3</td>
<td>1</td>
</tr>
<tr>
<td>Simulink</td>
<td></td>
<td></td>
<td>5</td>
<td>1</td>
<td>3</td>
</tr>
<tr>
<td>Slate</td>
<td></td>
<td></td>
<td>1</td>
<td>1</td>
<td></td>
</tr>
<tr>
<td>Spreadsheet (e.g., Microsoft Excel)</td>
<td></td>
<td></td>
<td></td>
<td>5</td>
<td>5</td>
</tr>
<tr>
<td>Statemate</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>3</td>
</tr>
<tr>
<td>Word Processor (e.g., Microsoft Word)</td>
<td></td>
<td></td>
<td></td>
<td>19</td>
<td>20</td>
</tr>
<tr>
<td>YAPS™</td>
<td></td>
<td></td>
<td></td>
<td>1</td>
<td>3</td>
</tr>
<tr>
<td>Designer’s Workbench™</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>1</td>
</tr>
<tr>
<td>Proprietary Database, SCADE like pic tool</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>1</td>
</tr>
<tr>
<td>Interleaf</td>
<td></td>
<td></td>
<td></td>
<td>1</td>
<td></td>
</tr>
<tr>
<td>BEACON</td>
<td></td>
<td></td>
<td>1</td>
<td>1</td>
<td></td>
</tr>
<tr>
<td>CaliberRM</td>
<td></td>
<td></td>
<td></td>
<td>1</td>
<td></td>
</tr>
<tr>
<td>XM</td>
<td></td>
<td></td>
<td></td>
<td>1</td>
<td></td>
</tr>
<tr>
<td>Wiring diagram</td>
<td></td>
<td></td>
<td></td>
<td>1</td>
<td></td>
</tr>
</tbody>
</table>

© 2014 Carnegie Mellon University
There is more to requirements quality than “shall”’s and stakeholder traceability

**IEEE 830-1998 Recommended Practice for SW Requirements Specification**

<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>

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
Mixture of Requirements & Architecture Design Constraints

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.

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

- Developmental Requirements
  - Modifiability
  - Assurability

- Environmental Assumptions
- Requirements Guarantees Assumptions
- Precondition Postcondition Invariant

- Interaction contract: match input assumption with guarantee
- Implementation constraints

- Quality attribute utility tree

- Mission Requirements
  - Function
  - Behavior
  - Performance
  - Dependability Requirements
    - Safety
    - Reliability
    - Security

- Assurability

- Implementation constraints

- 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 Model V2: Abstraction and Refinement

Four levels of abstraction:

- Focus on fault interaction with other components
  - Probabilistic error sources, sinks, paths and transformations
  - Fault propagation and Transformation Calculus (FPTC) from York U.
- Focus on fault behavior of components
  - Probabilistic typed error events, error states, propagations
  - Voting logic, error detection, recovery, repair
- Focus on fault behavior in terms of subcomponent fault behaviors
  - Composite error behavior state logic maps states of parts into (abstracted) states of composite
- Types of malfunctions and propagations
  - Common fault ontology
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
  - Propagated Error Type
  - Not propagated

Error Flow through component
Path P1.NoData -> P2.NoData
Source P2.BadData
Path processor.NoResource -> P2.NoData

Incoming
- NoData
- ValueError
- NoData
- BadValue

Outgoing
- NoData
- BadValue
- LateData

Component C

Direction

Port

Legend
- Port
- Processor
- HW Binding

Propagated Error Type

Not propagated

Direction

Legend
- Propagation of Error Types
  - Propagated Error Type
  - Not propagated

Error Flow through component
Path P1.NoData -> P2.NoData
Source P2.BadData
Path processor.NoResource -> P2.NoData

“Not” on propagated indicates that this error type is intended to be contained. This allows us to determine whether propagation specification is complete.
System engineering activity with focus on failing components.
Discovery of Unexpected PSSA Hazard through Repeated Virtual Integration

Expected: No EGI data

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.
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)
Support of SAE ARP4761 System Safety Assessment Practice

FHA
Spreadsheet
Uses error sources

AADL & EMV2

FTA
CAFTA, OpenFTA
Uses composite error behavior

FMEA
Spreadsheet
Uses error flows & propagations

Markov Chain
PRISM
Uses error flows & behavior

RBD/DD
OSATE plugin
Uses composite error behavior

<table>
<thead>
<tr>
<th>Component</th>
<th>Error</th>
<th>Hazard Description</th>
<th>Controller</th>
<th>Functional Failure</th>
<th>Operational</th>
</tr>
</thead>
<tbody>
<tr>
<td>StabilizerPosition</td>
<td>ServiceOmission</td>
<td>“No stabilizer position readings due to...” “1.1.3”</td>
<td>“Loss of sensor readings”</td>
<td>“All”</td>
<td></td>
</tr>
<tr>
<td>StabilizerAct</td>
<td>ServiceOmission</td>
<td>“Failure to move stabilizer into desired...” “1.1.2”</td>
<td>“Loss of actuator functionality”</td>
<td>“All”</td>
<td></td>
</tr>
<tr>
<td>StabilizerAct</td>
<td></td>
<td>“Failure to move stabilizer into desired...” “1.1.2”</td>
<td>“Loss of actuator functionality”</td>
<td>“All”</td>
<td></td>
</tr>
<tr>
<td>StabilizerController</td>
<td>Null on ActEnd</td>
<td>“Absence of computed data should signs...” “1.1.1”</td>
<td>“Loss of guidance values”</td>
<td>“Approach”</td>
<td></td>
</tr>
<tr>
<td>StabilizerController</td>
<td>Null on ActEnd</td>
<td>“Absence of computed data should signs...” “1.1.1”</td>
<td>“Loss of guidance values”</td>
<td>“Approach”</td>
<td></td>
</tr>
</tbody>
</table>

Software Engineering Institute
Carnegie Mellon

AADL and MBE
Feiler, Oct 20, 2014
© 2014 Carnegie Mellon University
The Symptom: Missed Stepper Motor Steps

Stepper motor (SM) controls a valve

- Commanded to achieve a specified valve position
  - Fixed position range mapped into units of SM steps
- New target positions can arrive at any time
  - SM immediately responds to the new desired position

Safety hazard due to software design

- Execution time variation results in missed steps
- Leads to misaligned stepper motor position and control system states
- Sensor feedback not granular enough to detect individual step misses

Two Customer Proposed Solutions

- Sending of data at 12ms offset from dispatch
- Buffering of command by SM interface

No analytical confidence that the problem will be addressed

Software modeled and verified in SCADE

Full reliance on SCADE of SM & all functionality

Problems with missing steps not detected

Software tests did not discover the issue

Time sensitive systems are hard to test for.

Other Challenge Problems

- Aircraft wheel braking system
- Engine control power up
- Situational Awareness & health monitoring
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

2010 SEI Study for AMRDEC
Aviation Engineering Directorate

|-------------------------------------------|-----------------------------------------------|---------------------------------------------|----------------------------------------------------------|

Four pillars for Improving Quality of Critical Software-reliant Systems

- **Mission Requirements**
  - Function
  - Behavior
  - Performance

- **Survivability Requirements**
  - Reliability
  - Safety
  - Security

- **Model Repository**
  - Architecture Model
  - Component Models
  - System Implementation
  - System configuration

- **Operational & failure modes**
- **Resource, Timing & Performance Analysis**
- **Reliability, Safety, Security Analysis**
Secure Mathematically-Assured Composition of Control Models

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

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

16 months into the project
Draper Labs could not hack into the system in 6 weeks
Had access to source code
Integrated Approach to Requirement V&V through Assurance Automation

Safety hazards are part of the picture

Generated assurance cases

Requirement coverage

Assumption evidence

Evidence records in terms of claims that requirements have been met

Linkage to automated test harnesses
Building the Assurance Case throughout the Life Cycle

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

Architecture-centric Virtual Integration

Architecture Led Requirements Specification

Virtual Architecture Integration & Analysis

System Integration Lab Testing

Design Validation by Virtual Integration

Code Coverage Testing

Early Discovery through Architecture Analysis leads to Assurance Related Rework Reduction

Incremental Evolution and Execution of Assurance Plans

Incremental Architecture & Requirement Evolution

Incremental Contract-based Compositional Verification

Build the System

Build the Assurance Case

Auto-generated Assurance Cases

Auto-generation from verified models AADL&SCADE/Simulink Ada SPARK/Ravenscar MISRA C
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
Benefits of Architecture-centric Engineering

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
- Fewer validation steps through use of validated generators
References

AADL Website www.aadl.info and AADL Wiki www.aadl.info/wiki
Blog entries and podcasts on AADL at 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
On an architecture-centric virtual integration practice and SAVI
http://www.sei.cmu.edu/architecture/research/model-based-engineering/virtual_system_integration.cfm
On an a four pillar improvement strategy for software system verification and qualification
15 Years of the SAE AS-2C AADL Committee
10 Years since the first publication of the SAE AADL standard
And many more 😊
Contact Information

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

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

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

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