Integrated Environment for Development and Assurance

Software Engineering Institute
Carnegie Mellon University
Pittsburgh, PA  15213

Peter H. Feiler
Jan 26, 2015
**Integrated Environment for Development and Assurance**

1. **REPORT DATE**
   
   15 JAN 2015

2. **REPORT TYPE**
   
   N/A

3. **DATES COVERED**
   

4. **TITLE AND SUBTITLE**
   
   Integrated Environment for Development and Assurance

5. **AUTHOR(S)**
   
   Feiler /Peter

6. **PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES)**
   
   Software Engineering Institute
   Carnegie Mellon University
   Pittsburgh, PA 15213

7. **SPONSORING/MONITORING AGENCY NAME(S) AND ADDRESS(ES)**
   

8. **PERFORMING ORGANIZATION REPORT NUMBER**
   

9. **DISTRIBUTION/AVAILABILITY STATEMENT**
   
   Approved for public release, distribution unlimited.

10. **SUPPLEMENTARY NOTES**
    
    The original document contains color images.

11. **ABSTRACT**

12. **SUBJECT TERMS**

13. **SECURITY CLASSIFICATION OF:**

   a. **REPORT**
      
      unclassified

   b. **ABSTRACT**
      
      unclassified

   c. **THIS PAGE**
      
      unclassified

   d. **LIMITATION OF ABSTRACT**
      
      SAR

   e. **NUMBER OF PAGES**
      
      40

   f. **NAME OF RESPONSIBLE PERSON**
      
      unclassified

---

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 Hazard Analysis
Incremental Life-cycle Assurance of Systems
Summary and Conclusion
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." a 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."

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.
Mismatched Assumptions in System Interactions

- **System Engineer**
  - Hazards: Impact of system failures

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

- **System Under Control**
  - Physical Plant Characteristics: Lag, proximity
  - Data Stream Characteristics: Latency jitter affects control behavior
  - Potential event loss

- **System User/Environment**
  - Operator Error: Automation & human actions

- **Hardware Engineer**
  - Distribution & Redundancy: Virtualization, load balancing, mode confusion

- **Application Developer**
  - Concurrency Communication: ITunes crashes on dual-cores

- **Embedded SW System Engineer**
  - Why do system level failures still occur despite fault tolerance techniques being deployed in systems?

*Embedded software system as major source of hazards*
Multi-Fidelity End-to-end Latency in Control Systems

Operational Environment

System Engineer

System Under Control

Control Engineer

Control System

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
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

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.

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

Other Challenge Problems
- Aircraft wheel braking system
- Engine control power up
- Situational Awareness & health monitoring
Time-sensitive Auto-brake Mode Confusion

Auto-brake mode selection by push button

- Three buttons for three modes
- Each button acts as toggle switch

Event sampling in asynchronous system setting

- Dual channel COM/MON architecture
- Each COM, MON unit samples separately
  - Button push close to sampling rate results in asymmetric value error
  - COM/MON mode discrepancy votes channel out
  - Repeated button push does not correct problem
  - Operational work around (1 second push) is not fool proof

Avoidable complexity design issue

- Concept mismatches: desired state by event and sampled event processing
- Desirable solution: State communication by multi-position switch
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 Hazard Analysis
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.
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

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

Auto-generated analytical models

Security
- Intrusion
- Integrity
- Confidentiality

Resource Consumption
- Bandwidth
- CPU time
- Power consumption

Data Quality

Software Engineering Institute | Carnegie Mellon

Integrated Development and Assurance
Feller, Jan 26, 2015
© 2015 Carnegie Mellon University
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, …

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

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

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
Rapid Architecture Trade Study

Help designers to choose the best Architecture
   Best reliability, avoid potential failure/error
   Meet timing and performance requirements

Analyze operational quality attributes from three perspectives
   Safety/Reliability
   Latency
   Resources and Budgets
Latency Analysis results

Architecture Alternative 1

<table>
<thead>
<tr>
<th>Contributor</th>
<th>Min Value</th>
<th>Min Method</th>
<th>Max Value</th>
<th>Max Method</th>
</tr>
</thead>
<tbody>
<tr>
<td>Device obstacle_camera</td>
<td>0.0ms</td>
<td>first sampling</td>
<td>0.0ms</td>
<td>first sampling</td>
</tr>
<tr>
<td>Device obstacle_camera</td>
<td>20.0ms</td>
<td>processing time</td>
<td>50.0ms</td>
<td>processing time</td>
</tr>
<tr>
<td>Sampled Connection obstacle</td>
<td>0.0ms</td>
<td>no latency</td>
<td>0.0ms</td>
<td>no latency</td>
</tr>
<tr>
<td>Thread thr_acq</td>
<td>0 ms</td>
<td>sampling</td>
<td>50.0ms</td>
<td>sampling</td>
</tr>
<tr>
<td>Thread thr_acq</td>
<td>10.0ms</td>
<td>processing time</td>
<td>40.0ms</td>
<td>processing time</td>
</tr>
<tr>
<td>Sampled Connection image_acq</td>
<td>0.0ms</td>
<td>no latency</td>
<td>0.0ms</td>
<td>no latency</td>
</tr>
<tr>
<td>Thread thr</td>
<td>0 ms</td>
<td>sampling</td>
<td>100.0ms</td>
<td>sampling</td>
</tr>
<tr>
<td>Thread thr</td>
<td>20.0ms</td>
<td>processing time</td>
<td>50.0ms</td>
<td>processing time</td>
</tr>
<tr>
<td>Sampled Connection obstacle_det</td>
<td>0.0ms</td>
<td>no latency</td>
<td>0.0ms</td>
<td>no latency</td>
</tr>
<tr>
<td>Bus bus1</td>
<td>200.25ms</td>
<td>transmission time</td>
<td>500.625ms</td>
<td>transmission time</td>
</tr>
<tr>
<td>Thread thr</td>
<td>0 ms</td>
<td>sampling</td>
<td>10.0ms</td>
<td>sampling</td>
</tr>
<tr>
<td>Thread thr</td>
<td>0.0ms</td>
<td>no latency</td>
<td>0.0ms</td>
<td>no latency</td>
</tr>
<tr>
<td>Sampled Connection obstacle</td>
<td>0.0ms</td>
<td>no latency</td>
<td>0.0ms</td>
<td>no latency</td>
</tr>
<tr>
<td>Thread thr</td>
<td>0 ms</td>
<td>sampling</td>
<td>4.0ms</td>
<td>sampling</td>
</tr>
<tr>
<td>Thread thr</td>
<td>0.0ms</td>
<td>no latency</td>
<td>0.0ms</td>
<td>no latency</td>
</tr>
<tr>
<td>Sampled Connection emergency</td>
<td>0.0ms</td>
<td>no latency</td>
<td>0.0ms</td>
<td>no latency</td>
</tr>
<tr>
<td>Thread thr</td>
<td>0 ms</td>
<td>sampling</td>
<td>2.0ms</td>
<td>sampling</td>
</tr>
<tr>
<td>Thread thr</td>
<td>0.0ms</td>
<td>no latency</td>
<td>0.0ms</td>
<td>no latency</td>
</tr>
<tr>
<td>Sampled Connection warming</td>
<td>0.0ms</td>
<td>no latency</td>
<td>0.0ms</td>
<td>no latency</td>
</tr>
<tr>
<td>Device warning_alert</td>
<td>0 ms</td>
<td>sampling</td>
<td>500.0ms</td>
<td>sampling</td>
</tr>
<tr>
<td>Device warning_alert</td>
<td>20.0ms</td>
<td>processing time</td>
<td>50.0ms</td>
<td>processing time</td>
</tr>
</tbody>
</table>

Latency Total: 270.25ms, End to End Latency: 700.0ms

Architecture Alternative 2

<table>
<thead>
<tr>
<th>Contributor</th>
<th>Min Value</th>
<th>Min Method</th>
<th>Max Value</th>
<th>Max Method</th>
</tr>
</thead>
<tbody>
<tr>
<td>Device obstacle_camera</td>
<td>0.0ms</td>
<td>first sampling</td>
<td>0.0ms</td>
<td>first sampling</td>
</tr>
<tr>
<td>Device obstacle_camera</td>
<td>20.0ms</td>
<td>processing time</td>
<td>50.0ms</td>
<td>processing time</td>
</tr>
<tr>
<td>Sampled Connection obstacle</td>
<td>0.0ms</td>
<td>no latency</td>
<td>0.0ms</td>
<td>no latency</td>
</tr>
<tr>
<td>Thread thr_acq</td>
<td>0 ms</td>
<td>sampling</td>
<td>50.0ms</td>
<td>sampling</td>
</tr>
<tr>
<td>Thread thr_acq</td>
<td>10.0ms</td>
<td>processing time</td>
<td>40.0ms</td>
<td>processing time</td>
</tr>
<tr>
<td>Sampled Connection image_acq</td>
<td>0.0ms</td>
<td>no latency</td>
<td>0.0ms</td>
<td>no latency</td>
</tr>
<tr>
<td>Thread thr</td>
<td>0 ms</td>
<td>sampling</td>
<td>100.0ms</td>
<td>sampling</td>
</tr>
<tr>
<td>Thread thr</td>
<td>20.0ms</td>
<td>processing time</td>
<td>50.0ms</td>
<td>processing time</td>
</tr>
<tr>
<td>Sampled Connection obstacle_det</td>
<td>0.0ms</td>
<td>no latency</td>
<td>0.0ms</td>
<td>no latency</td>
</tr>
<tr>
<td>Bus bus1</td>
<td>10.000125ms</td>
<td>transmission time</td>
<td>30.00125ms</td>
<td>transmission time</td>
</tr>
<tr>
<td>Thread thr</td>
<td>0 ms</td>
<td>sampling</td>
<td>10.0ms</td>
<td>sampling</td>
</tr>
<tr>
<td>Thread thr</td>
<td>0.0ms</td>
<td>no latency</td>
<td>0.0ms</td>
<td>no latency</td>
</tr>
<tr>
<td>Sampled Connection obstacle</td>
<td>0.0ms</td>
<td>no latency</td>
<td>0.0ms</td>
<td>no latency</td>
</tr>
<tr>
<td>Thread thr</td>
<td>0 ms</td>
<td>sampling</td>
<td>4.0ms</td>
<td>sampling</td>
</tr>
<tr>
<td>Thread thr</td>
<td>0.0ms</td>
<td>no latency</td>
<td>0.0ms</td>
<td>no latency</td>
</tr>
<tr>
<td>Sampled Connection emergency</td>
<td>0.0ms</td>
<td>no latency</td>
<td>0.0ms</td>
<td>no latency</td>
</tr>
<tr>
<td>Thread thr</td>
<td>0 ms</td>
<td>sampling</td>
<td>2.0ms</td>
<td>sampling</td>
</tr>
<tr>
<td>Thread thr</td>
<td>0.0ms</td>
<td>no latency</td>
<td>0.0ms</td>
<td>no latency</td>
</tr>
<tr>
<td>Sampled Connection warming</td>
<td>0.0ms</td>
<td>no latency</td>
<td>0.0ms</td>
<td>no latency</td>
</tr>
<tr>
<td>Device warning_alert</td>
<td>0 ms</td>
<td>sampling</td>
<td>500.0ms</td>
<td>sampling</td>
</tr>
<tr>
<td>Device warning_alert</td>
<td>20.0ms</td>
<td>processing time</td>
<td>50.0ms</td>
<td>processing time</td>
</tr>
</tbody>
</table>

Latency Total: 80.000125ms, End to End Latency: 866.00125ms

Software Engineering Institute

Carnegie Mellon

Integrated Development and Assurance
Feller, Jan 26, 2015
© 2015 Carnegie Mellon University
## Analysis Summary

<table>
<thead>
<tr>
<th></th>
<th>Architecture 1</th>
<th>Architecture 2</th>
</tr>
</thead>
<tbody>
<tr>
<td>Latency</td>
<td>![Checkmark]</td>
<td>![X]</td>
</tr>
<tr>
<td>Resources Budgets</td>
<td>![X]</td>
<td>![Checkmark]</td>
</tr>
<tr>
<td>Safety</td>
<td>![X]</td>
<td>![Checkmark]</td>
</tr>
<tr>
<td>Cost</td>
<td>![Checkmark]</td>
<td>![X]</td>
</tr>
</tbody>
</table>

What is the “best” architecture?
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 Hazard Analysis
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**
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

- **Environmental Assumptions**
- **Requirements Guarantees Assumptions**
- **Precondition Postcondition Invariant**

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

- Developmental Requirements
  - Modifiability
  - Assurability

- Quality attribute utility tree

- Mission Requirements
  - Function
  - Behavior
  - Performance

- Dependability Requirements
  - Reliability
  - Safety
  - Security

- System Specification and Requirements Coverage

- Coverage
- Quality attribute utility tree

- System Constraints/Controls
- Environment
- Input
- State
- Output
- Resources

- Interaction contract: match input assumption with guarantee
Architecture-led Requirement & Hazard Specification

Error Propagation Ontology

Service errors
- Omission
- Commission

Omission: \( \forall i, (ts_i \in ST) \) \( \forall j \geq i, ts_j = \) \(<\)

Value errors
- Sequence errors
- Replication errors
- Concurrency errors

Extensions to Powell/Vasiliadis Ontologies

Fault Lattice for Data streams
- Value errors
- Timing errors

Leveson pattern

Controller
- Inadequate control
  - Algorithm flaws in creation, process changes, incorrect modification or adaptation

Actuator
- Inadequate operation
  - Inappropriate, ineffective or missing control action

Sensor
- Inadequate operation
  - Incorrect or no information provided

Controlled Process
- Component failures
  - Changes over time

Process input missing or wrong
- Process output contributes to system hazard

Inadequate or missing feedback
- Feedback delays

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
Safety Practice in Development Process Context

Focus on System Engineering Largely Ignores Software as Hazard Source

Labor-intensive
Early in system engineering
Rarely repeated due to cost
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
- Propagation to resource

Bound resources

“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.BadData
Path: processor.NoResource -> P2.NoData

NoData
ValueError
NoData
BadValue
NoResource
P1
P2
P3
Component C
Processor
Memory
Bus
Incoming
Outgoing

Software Engineering Institute
Carnegie Mellon
Integrated Development and Assurance
Feller, Jan 26, 2015
© 2015 Carnegie Mellon University
26
System engineering activity with focus on failing components.
Discovery of Unexpected PSSA Hazard through Repeated Virtual Integration

- **EGI**
  - EGI Logic
    - Oper'l
    - Failed
    - Corrupted
  - EGI HW
    - Oper'l
    - Failed

- **Flight Mgmt System**
  - NoData
  - Airspeed Data
  - CorruptedData

- **Auto Pilot**
  - Operational
  - Failed

- **FMS Processor**
  - Operational
  - Failed

- **FMS Power**
  - NoService

- **Actuator Cmd**
  - NoService
  - Stall

- **Anticipated**: No EGI data
- **Anticipated**: NoService
- **Anticipated**: No Stall Propagation

**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)
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 Hazard Analysis
Incremental Life-cycle Assurance of Systems
Summary and Conclusion
Quality & Certification Improvement Strategy

2010 SEI Study for AMRDEC Aviation Engineering Directorate

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

- **Architecture-led Requirement Specification**
  - Mission Requirements
    - Function
    - Behavior
    - Performance
  - Survivability Requirements
    - Reliability
    - Safety
    - Security

- **Architecture-centric Virtual System Integration**
  - Model Repository
    - Architecture Model
    - Component Models
    - System Implementation
    - System configuration

- **Static Analysis & Compositional Verification**
  - Operational & failure modes
  - Resource, Timing & Performance Analysis
  - Reliability, Safety, Security Analysis

- **Incremental Assurance Plans & Cases throughout Life Cycle**

**Four pillars for Improving Quality of Critical Software-reliant Systems**
Building the Assurance Case throughout the Life Cycle

- Requirements Engineering
  - Requirements Validation
  - System Design
  - System Architecture Validation
  - Software Architectural Design
  - Software Architecture Validation
  - Component Software Design
  - Design Validation
  - Code Development
  - Architecture Modeling, Analysis & Generation
  - Integration Build
  - Integration Test
  - Unit Test
  - Acceptance Test
  - System Test
  - Deployment Build
  - Build the System
  - Build the Assurance Case

Integrated Development and Assurance
Feiler, Jan 26, 2015
© 2015 Carnegie Mellon University
Virtual System Integration & Compositional Verification

Architecture-centric Virtual Integration

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

Incremental Evolution and Execution of Assurance Plans

Incremental Architecture & Requirement Evolution

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

Auto-generated Assurance Cases

Build the Assurance Case

Build the System

Design Validation by Virtual Integration

Virtual Architecture Integration & Analysis

System Integration Lab Testing

Code Coverage Testing

Flight Test

Early Discovery through Architecture Analysis leads to Assurance Related Rework Reduction

Compositional Verification

Increased Confidence

Cost-Effective Tests

C1: The system is safe
C3: Hazard B has been eliminated
Ev1: Evidence
Ev2: Evidence
Ev3: Evidence
Hazard A has been eliminated

Auto-generated Assurance Cases

Requirement Coverage

Design & Req Refinement

Compositional Verification

Incremental Contract-based Compositional Verification

Integrated Development and Assurance
Feller, Jan 26, 2015
© 2015 Carnegie Mellon University
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

Evidence records in terms of claims that requirements have been met

Linkage to automated test harnesses

Requirement coverage
Assumption evidence
Incremental Development and Assurance Practice

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

BUSINESS AND MISSION GOALS

ARCHITECTURE

SYSTEM

Architecture-centric virtual integration and compositional verification of requirements
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 Hazard Analysis
Incremental Life-cycle Assurance of Systems
Summary and Conclusion
Benefits of Incremental Life Cycle Assurance through Virtual System Integration

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
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