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

NO WARRANTY. THIS CARNEGIE MELLON UNIVERSITY AND SOFTWARE ENGINEERING INSTITUTE MATERIAL IS FURNISHED ON AN “AS-IS” BASIS. CARNEGIE MELLON UNIVERSITY MAKES NO WARRANTIES OF ANY KIND, EITHER EXPRESSED OR IMPLIED, AS TO ANY MATTER INCLUDING, BUT NOT LIMITED TO, WARRANTY OF FITNESS FOR PURPOSE OR MERCHANTABILITY, EXCLUSIVITY, OR RESULTS OBTAINED FROM USE OF THE MATERIAL. CARNEGIE MELLON UNIVERSITY DOES NOT MAKE ANY WARRANTY OF ANY KIND WITH RESPECT TO FREEDOM FROM PATENT, TRADEMARK, OR COPYRIGHT INFRINGEMENT.

This material has been approved for public release and unlimited distribution.

This material may be reproduced in its entirety, without modification, and freely distributed in written or electronic form without requesting formal permission. Permission is required for any other use. Requests for permission should be directed to the Software Engineering Institute at permission@sei.cmu.edu.

DM-0000087
Outline

Challenges in Safety-critical Software-intensive systems
SAE AADL and the Error Model Annex
Hazards & Fault Impact Analysis
Compositional Analysis
Operational and Failure Modes
The Intricacies of Desired Timing Behavior
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 fitted with the same air-data computer. The advisory is "aimed at minimizing the risk in the unlikely event of a similar occurrence."

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 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
- 0%, 9% 80x

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

Rework and certification is 70% of SW cost, and SW is 70% of system cost.

Sources:

Delivery Delays Not Known Until Late into Project Schedule
Mismatched Assumptions in Embedded SW

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

Challenges in Safety-critical Software-intensive systems
SAE AADL and the Error Model Annex
Hazards & Fault Impact Analysis
Compositional Analysis
Operational and Failure Modes
The Intricacies of Desired Timing Behavior
Summary and Conclusion
SAE Architecture Analysis & Design Language (AADL) for Software-reliant Systems

AADL focuses on interaction between the three elements of a software-intensive system based on architectural abstractions of each.
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

Partitioned architecture models
Model compliance

Virtual processors & buses
Synchronization domains

Fault propagation
Security analysis
Architectural redundancy patterns

Resource budget analysis
& task roll-up analysis
Resource allocation & deployment configurations
Architecture-Centric Modeling Approach

Single Annotated Architecture Model Addresses Impact Across Non-Functional Properties

Safety & Reliability
- MTBF
- FMEA
- Hazard analysis

Security
- Intrusion
- Integrity
- Confidentiality

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

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

Resource Consumption
- Bandwidth
- CPU time
- Power consumption

Architecture Model

Auto-generated analytical models
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

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

- maintainability
- availability
- Integrity

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

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

Annotated architecture model permits checking for consistency and completeness between these various declarations.
Error Model and the Architecture

Propagation of errors of different types from error sources along propagation paths between architecture components.

Error flows as abstractions of propagation through components.
Component error behavior as transitions between states triggered by error events and incoming propagations.

Composite error behavior in terms of component error behavior states.
A Common Set of Error Propagation Types

Independent hierarchies of error types
Can be combined to characterize failure modes, resulting error states, and types of error propagation

Use aliases to adapt to domain
NoPower = ServiceOmission
Introduce user defined types
Formal Error Type Specifications

Service errors with respect to the service as a whole rather than individual service items

• Service Omission is perceived as a permanent fault in that no service items are provided after the point of failure.
  Service Omission: $\exists s_i \in S' \subseteq S: (st_i = \infty)$

• Service Commission is perceived as an impromptu service in that service items are provided before the point service is expected.
  Service Commission: $\exists s_i \in S' \supseteq S: (st_i \notin ST)$

• Other forms of service error can be defined: early service start, late service start, early service termination, late service termination.

Value errors with respect to the value of an individual service item

• Out Of Range error indicating that the value is outside the expected range of values, a detectable error.
  Out Of Range error: $s_i: sv_i \notin SV$
Error Types & Error Type Sets

Error type declarations

ServiceError: type;
Omission: type extends ServiceError;
Commission: type extends ServiceError;
Early: type extends TimingError;
Late: type extends TimingError;

An error type set represents the combination of error types that can occur or be propagated simultaneously.

- An error type set is defined as the product of error types.
- Example: an error propagation may involve both a late value and an incorrect value.

InputOutputError : type set {TimingError, ValueError};
StreamError : type set {TimingError, ValueError, SequenceError, RateError};

An error tuple represents a typed token instance

- Represents actual event, propagation, or state types

{LateValue + BadValue} or {LateValue}
Analyzable Architecture Fault Models

Through model-based analysis identify architecture induced unhandled, testable, and untestable faults and understand the root cause, impact, and potential mitigation options.
Analyzable Architecture Fault Models

Through model-based analysis identify architecture induced unhandled, testable, and untestable faults and understand the root cause, impact, and potential mitigation options.
Analyzable Architecture Fault Models

Through model-based analysis identify architecture induced unhandled, testable, and untestable faults and understand the root cause, impact, and potential mitigation options.
Analyzable Architecture Fault Models

Through model-based analysis identify architecture induced unhandled, testable, and untestable faults and understand the root cause, impact, and potential mitigation options.
Outline

Challenges in Safety-critical Software-intensive systems
SAE AADL and the Error Model Annex
Hazards & Fault Impact Analysis
Compositional Analysis
Operational and Failure Modes
The Intricacies of Desired Timing Behavior
Summary and Conclusion
Hazard Information in AADL Error Model

Hazards modeled as error propagations in the AADL Error Model

```plaintext
error model HazardList

features
Guidance_Loss: out error propagation {hazard =>
  (reference => "1.1.1",
  failure => "Loss of guidance values",
  phase => "approach",
  description => "Presence of no computed data should signal FD and AP disconnect.",
  criticality => "minor",
  comment => "Becomes major hazard, equivalent to incorrect guidance, if disconnect fails.");

Guidance_Incorrect: out error propagation {hazard =>
  (reference => "1.1.2",
  failure => "Incorrect guidance values",
  phase => "approach",
  description => "Gradual departure from references until detected by flight crew during check of primary flight data resulting in manual disconnect and approach",
  criticality => "major",
  comment => "No difference to the AP between loss of guidance and incorrect guidance.");

Transfer_Control_Loss: out error propagation {hazard =>
  (reference => "4.1.1",
  failure => "Loss of transfer control of flight guidance data to AP",
  phase => "all",
  description => "Flight crew unable to change 'Pilot Flying' side FGS. Manual disconnect",
  criticality => "minor",
  comment => ",

end FlightGuidance.subsystems;
```
Sample FHA in Spreadsheet View

Hazard information exported to spreadsheet format

<table>
<thead>
<tr>
<th>A</th>
<th>B</th>
<th>C</th>
<th>D</th>
<th>E</th>
<th>F</th>
<th>G</th>
<th>H</th>
</tr>
</thead>
<tbody>
<tr>
<td>Component</td>
<td>Error</td>
<td>Reference</td>
<td>Functional Failure (Hazard)</td>
<td>Critical Operational Phase</td>
<td>Aircraft Manifestation</td>
<td>Criticality</td>
<td>Comment</td>
</tr>
<tr>
<td>FlightGuidance_FlightGuidance_subsystem_Instance.PR_FGS_L1</td>
<td>Guidance_Loss</td>
<td>1.1.1</td>
<td>Loss of guidance values</td>
<td>approach</td>
<td>Presence of no computed data should signal FD and AP disconnect.</td>
<td>minor</td>
<td>Becomes major hazard, equivalent to incorrect guidance, if disconnect fails.</td>
</tr>
<tr>
<td></td>
<td>Guidance_Incorrect</td>
<td>1.1.2</td>
<td>Incorrect guidance values</td>
<td>approach</td>
<td>Gradual departure from references until detected by flight crew during check of primary flight data resulting in manual disconnect and manual flying.</td>
<td>major</td>
<td>No difference to the AP between loss of guidance and incorrect guidance.</td>
</tr>
<tr>
<td>Transfer_Control_Loss</td>
<td>4.1.1</td>
<td>Loss of transfer control of flight guidance data to AP</td>
<td>all</td>
<td>Flight crew unable to minor change 'Pilot Flying' side FGS. Manual disconnect and manual flying.</td>
<td>-</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Architecture Fault Model allows hazards to be recorded for use throughout all development phases
Component Error Propagation

Error Flow:
Path P1.NoData -> P3.NoData
Source P2.BadData;
Path processor.NoResource -> P2.NoData

Incoming/Assumed
- Propagated errors
- Errors not propagated

Outgoing/Intention
- Propagated errors
- Errors not propagated

Bound resources
- Propagated errors
- Errors not propagated
- Propagation from/to resource

Error propagation and flow specification supports fault impact analysis based on a Fault Propagation and Transformation Calculus (FPTC).
Discovery of Unexpected PSSA Hazard

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

Transient hardware failure corrupts EGI data.

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

Anticipated: No EGI data

Flight Mgmt System

NoData

FMS Power

Oper'l

Oper'l

NoData

NoData

Anticipated: No Service

Stall

NoService

Oper’l

Oper’l

Oper’l

NoService

FMS Processor

Operational

Failed

Auto Pilot

Operational

Failed

Actuator Cmd

Anticipated: No Stall Propagation

system [EGI]

features
trueairspeed: out data port DataDictionary::Velocity;
flows
f1: flow source latency = 0;
annex EMV2 {error process use types use behaviors [true];
flows ef1: flow source latency = 0;
properties EMV2:hazard [crossRef: failure phase = description severity criticality comment];

system implementation components
PilotGrip
PositionSensor
sensor
EGI: system
FMS: processor
Actuator1: device Actuator
Actuator2: device Actuator
FMSProcessor: processor PowerPC
connections
pilotCmd: port PilotGrip.DesiredPosition -> FMS.Setpoint;
sensedPosition: port PositionSensor.PositionReading -> FMS.Position;
Actuator1Cmd: port FMS.ActCmd -> Actuator1.ActCmd;
Actuator2Cmd: port FMS.ActCmd -> Actuator2.ActCmd;
ytx: port EGI.IndicAirSpeed -> FMS.IndicAirSpeed;

Outgoing propagation (Failure; CorruptedData) is not handled. Expected incoming (Failure)

from: Actuator1 vs: EGI;
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
SAE AADL and the Error Model Annex
Hazards & Fault Impact Analysis
Compositional Analysis
Operational and Failure Modes
The Intricacies of Desired Timing Behavior
Summary and Conclusion
Error Model at Each Architecture Level

- Abstracted error behavior of FMS
  - Error behavior and propagation specification

- Composite error behavior specification of FMS
  - State in terms of subcomponent states
    
    $$[\text{1 ormore} (\text{FG1.Failed} \text{ or } \text{AP1.Failed}) \text{ and } \text{1 ormore} (\text{FG2.Failed} \text{ or } \text{AP2.Failed}) \text{ or } \text{AC.Failed}] \rightarrow \text{Failed}$$

$$\text{FG1} \rightarrow \text{FG2} \rightarrow \text{AP1} \rightarrow \text{AP2} \rightarrow \text{FMS} \rightarrow \text{AC}$$

Consistency Checking Across Levels of the Hierarchy

Fault occurrence probability
Impact of Deployment Configuration Changes

FMS Failure on 2 or 3 processor configuration (CPU failure rate = $10^{-5}$)

<table>
<thead>
<tr>
<th>FMS Failure Rate</th>
<th>0</th>
<th>5*10^{-6}</th>
<th>5*10^{-5}</th>
</tr>
</thead>
<tbody>
<tr>
<td>MTTF – One CPU operational</td>
<td>112,000</td>
<td>67,000</td>
<td>14,000</td>
</tr>
<tr>
<td>MTTF – Two CPU operational</td>
<td>48,000</td>
<td>31,000</td>
<td>7,000</td>
</tr>
</tbody>
</table>

Side effects of design and deployment decisions in reliability predictions

Workload balancing of partitions later in development affects reliability
3 processor configuration can be less reliable than 2 processor configuration
Example: AP and FG distributed across two processors
SAFEbus with Dual Communication Channel

- Network is aware of dual host nature
  - Reflected in replication factor properties for connections and bus access
  - Dual channel communication as abstraction reflected in Replication Errors
- Integrity gate keeper for host system
  - Maps inconsistent value/omission errors into item omissions
  - *Expressed by error path from incoming to outgoing binding propagation*

SAFEbus as Application Gate
Keeper and Source of Error

- Fail-op/fail-stop tolerance of SAFEbus faults
  - Operational, SingleErrorOp, Failed states: SAFEbus error events cause state change
  - Operational: gate keeper mappings
  - SingleErrorOp: no error source & full gate keeper mappings
  - Failed: service omission as sole outgoing propagation
  - *Modeled by error behavior state machine and component behavior conditions*
Does SAFEbus Meet Application Needs?

- SAFEbus as Integrity Gate Keeper
  - Transform outgoing propagations to match incoming application integrity assumptions
- SAFEbus communication transport mechanism as error source
  - Map SAFEbus error events into incoming error propagations that must meet integrity assumptions
Assumptions & Hazards in Use of SAFEbus

- Assumption: no replicated Host stand-by operation
  - Item/service omission on one Host-S channel => Item/service omission for both Host-T channels
  - Modeled as *Inconsistent Omission propagation in stand-by operational mode*
- Assumption: no identical corrupt/bad value propagation from sender
  - Need to ensure no replication of corrupt/bad value (e.g., sensor fan-out)
  - Modeled as *no outgoing symmetric value error from sending host*
- SAFEbus as shared resource
  - Manage babbling host (item/service commission)
  - Modeled as *propagated commission that is expected to be masked*
Outline

Challenges in Safety-critical Software-intensive systems
SAE AADL and the Error Model Annex
Hazards & Fault Impact Analysis
Compositional Analysis
Operational and Failure Modes
The Intricacies of Desired Timing Behavior
Summary and Conclusion
Operational Modes and Failure Modes

• Nominal system behavior
  • Operational modes
  • Functional behavior
  • Represented by AADL modes and Behavior Annex

• Anomalous behavior reflecting failure modes
  • Deviation from nominal behavior
  • Due to design error
  • Due to physical failure (operational error)
  • Represented by AADL Error Model Annex

• System awareness of anomalous behavior
  • Detection, reporting/recording
  • Transformation, masking
  • Represented by AADL Error Model Annex
Example: GPS

- Operational modes: Hi-Precision, Lo-Precision, Off
  - User initiated transitions
- Failures: Sensor failure, Processing failure
- Error states of GPS:
  - Dual Sensor Op (operational)
    - Sensor1{NoError} and Sensor2{NoError} and Processing{NoError}
  - Single Sensor Op (Degraded)
    - 1 orless(Sensor1{Failed}, Sensor2{Failed})
  - Dual sensor failure or processing failure (FailStop)
    - 1 ormore(Sensor1{Failed} and Sensor2{Failed}, Processing{Failed})

- Mapping of Error States onto Operational Modes
Combined GPS Behavior Model

- Error states constrain operational modes
  - Degraded supports LoP and Off
  - FailStop shows Off behavior
- Forced mode transitions
  - New error state that excludes current mode
  - Explicit reflection of forced transition in mode behavior
    - Detection event and “Emergency” mode transition
- Behavioral record of error state (Mode state/Error state pairs)
  - Specify detection and reporting of error state
  - Allows for detection and reporting of limits on user initiated transitions

<table>
<thead>
<tr>
<th>Expected mode</th>
<th>Operational</th>
<th>Degraded</th>
<th>Fail Stop</th>
</tr>
</thead>
<tbody>
<tr>
<td>HiP</td>
<td>HiP</td>
<td>Force transition to LoP {ValueError}</td>
<td>Force transition to Off {Omission}</td>
</tr>
<tr>
<td>LoP</td>
<td>LoP</td>
<td>LoP Ok: LoP &lt;-&gt; Off</td>
<td>Force transition to Off {Omission}</td>
</tr>
<tr>
<td>Off</td>
<td>Off</td>
<td>Off</td>
<td>Off Perceived {NoError}</td>
</tr>
</tbody>
</table>
Outline

Challenges in Safety-critical Software-intensive systems
SAE AADL and the Error Model Annex
Hazards & Fault Impact Analysis
Compositional Analysis
Operational and Failure Modes
The Intricacies of Desired Timing Behavior
Summary and Conclusion
End-to-end Latency in Control Systems

- Processing latency
- Sampling latency
- Physical signal latency

Impact of Software Implemented Tasks
Jitter is typically managed by Periodic I/O
AADL immediate & delayed connections specify deterministic sampling

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

Flow Use Scenario through Subsystem Architecture

AADL supports modeling of latency contributor timing behavior
Input sampling by application code virtualizes time
Migration of Dual Fault Tolerant Control System

Highly unstable system being controlled
Control software fault tolerance
  • Simplex: Baseline, experimental, recovery controllers
  • Monitor experimental, recover to baseline control
Dual redundancy to address hardware failures
  • Two instances of Simplex control system
Asynchronous dual processor hardware
  • Bounded clock drift
  • Distributed leadership decision making
  • Each processor shared with unrelated higher priority task
Migration to dual core processors
  • One core dedicated to Simplex control system
  • Unrelated task on second core
  • Failure to provide control
Outline

Challenges in Safety-critical Software-intensive systems
SAE AADL and the Error Model Annex
Hazards & Fault Impact Analysis
Compositional Analysis
Operational and Failure Modes
The Intricacies of Desired Timing Behavior
Summary and Conclusion
Architecture Fault Modeling Summary

Architecture Fault Modeling with AADL

- SAE AADL V2.1 published in Sept 2012 (V1 published in 2004)
- Error Model Annex was published in 2006
  - Supported in AADL V1 and AADL V2
- Error Model concepts and ontology not specific to AADL, can be applied to other modeling notations
- Revised Error Model Annex (V2) based on user experiences currently in review

Safety Analysis and Verification

- Error Model Annex front-end available in OSATE open source toolset
  - Allows for integration with in-house safety analysis tools
- Multiple tool chains support various forms of safety analysis (Honeywell, Aerospace Corp., AVSI SAVI, ESA COMPASS)
- FHA, FMEA, fault tree, Markov models, stochastic Petri net generation from AADL>Error Model
  - Open source implementation as part of Error Model V2 publication
Early Discovery and Incremental V&V through 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

Proof of Concept Demonstration and Transition by Aerospace industry initiative
- Propagate requirements and constraints
- Higher level model down to suppliers' lower level models
- Verification of lower level models satisfies higher level requirements and constraints

- Multi-tier system & software architecture (in AADL)
- Incremental end-to-end validation of system properties
Increased Confidence through Model-based Analysis and Testing Evidence throughout Life Cycle

- **Requirements Engineering**
- **System & SW Architectural Design**
- **Component Software Design**
- **Design Validation**
- **Virtual Architecture Integration & Analysis**
- **Architecture Focused Requirements Analysis**
- **Architecture Modeling and Analysis**
- **Deployment Build**
- **Acceptance Test**
- **Flight Test**
- **System Test**
- **System Integration Lab Testing**
- **Integration Test**
- **Unit Test**
- **Code Coverage Testing**
- **Build the System**
- **Build the Assurance Case**

- **Requirements Validation**
- **System & SW Architecture Validation**
- **Design Validation**
- **Integration Build**
- **Target Build**
- **Build the System & SW Architecture**
- **Validation**
- **Design Validation by Virtual Integration**
- **Code Coverage Testing**
- **Build the Assurance Case**

**Increased Confidence through Model-based Analysis and Testing Evidence throughout Life Cycle**
References

Website [www.aadl.info](http://www.aadl.info)

Public Wiki [https://wiki.sei.cmu.edu/aadl](https://wiki.sei.cmu.edu/aadl)

AADL Book in SEI Series of Addison-Wesley
Contact Information

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

Web
www.sei.cmu.edu
www.sei.cmu.edu/contact.cfm

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