Software Solutions Symposium 2017

March 20–23, 2017

Improvements in Safety Analysis for Safety-critical Software Systems

Peter Feiler

Software Engineering Institute
Carnegie Mellon University
Pittsburgh, PA 15213

Challenges in Existing System Safety Practices

Current Reliance on Engineering Process

Guidelines for Robust & Reliable Aircraft

DO-178B-Software Considerations in Airborne Systems and Equipment Certification

DO-248B-Final Report for the Clarification of DO-178B

DO-278-Guidelines for Communications, Navigation, Surveillance, and Air Traffic Manager

DO-254-Design Assurance Guidance for Airborne Electronic Hardware

DO-297-Integrated Modular Avionics (IMA) Development Guidance and Certification Consi

SAE-ARP4754-Certification Consideration for Highly Integrated or Complex Aircraft Syster

SAE-ARP4671- Guidelines

FAA Advisory Circular AC

FAA Advisory Circular AC

ISO/IEC 12207-Software Lif

ARINC 653-Specification St

MIL-STD-882D-DoD System Safety

ADS-51-HDBK-Rotorcraft and Aircraft Qualification Handbook

AR-70-62-Airworthiness Release Standard

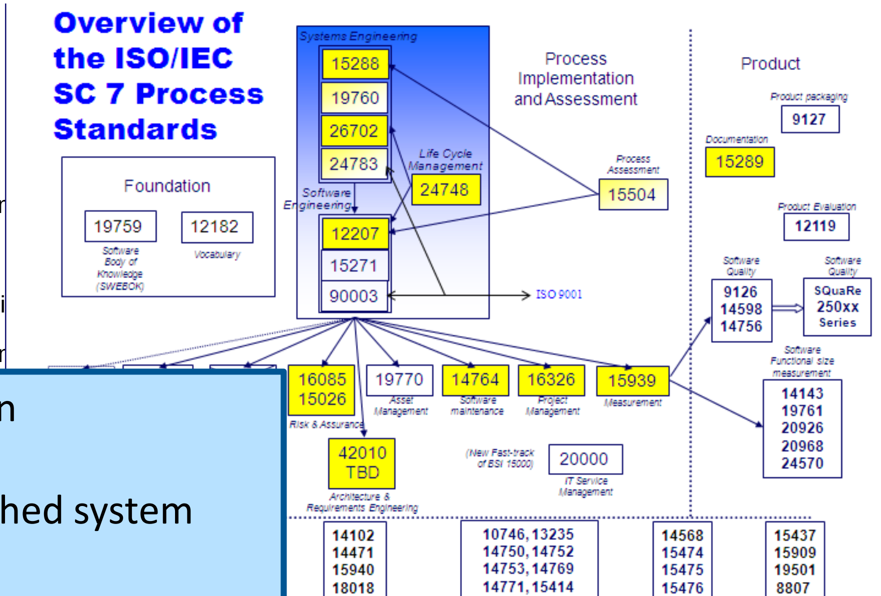
ADS-75-SS-Army Aviation System Safety Assessments and Analyses

ADS-48-PRF-Performance Specification for Airworthiness Qualification Requirements for (
Instrument Flight Rules

ADS-64-SP-Airworthiness Requirements for Military Rotorcraft

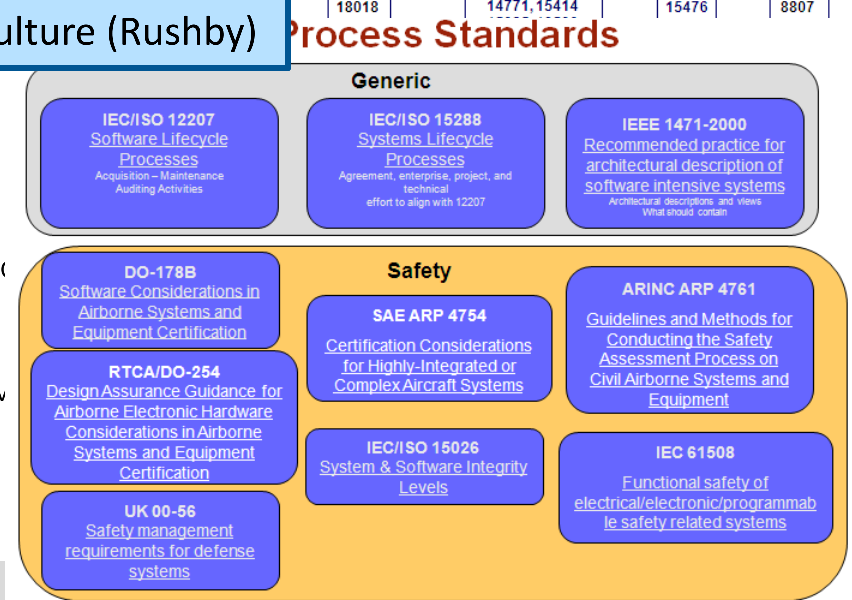
SED-SES-PMHFSA001-Software Engineering Directorate (SED) Software Engineering Ev
Airworthiness

SED-SES-PMHSS001 SED SEES Program Manager Handbook for Software Safety

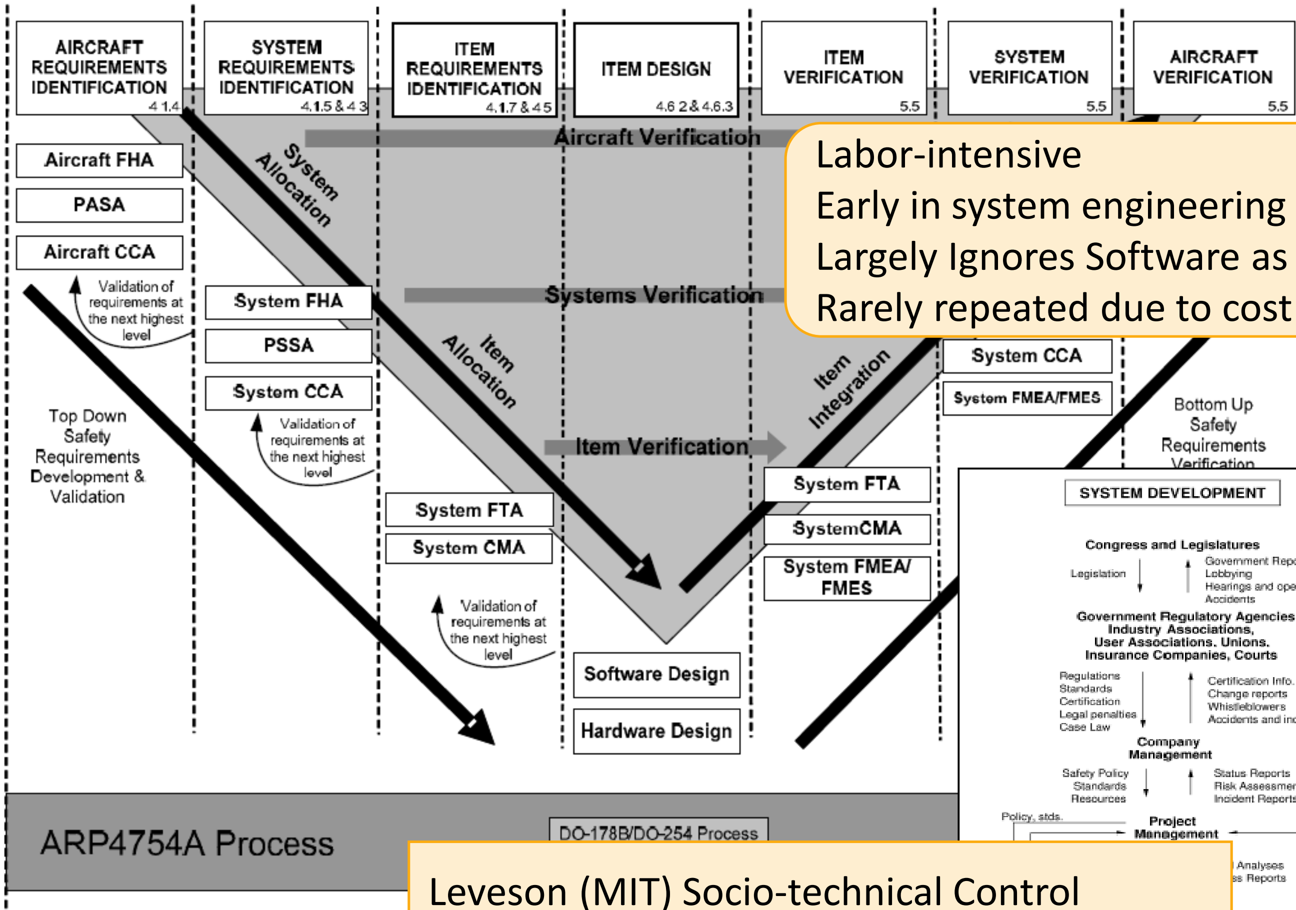


Current methods explicitly depend on
- standards and regulations
- rigorous examination of whole finished system

and implicitly depend on
- conservative practices and safety culture (Rushby)



Safety Practice in Development Process Context

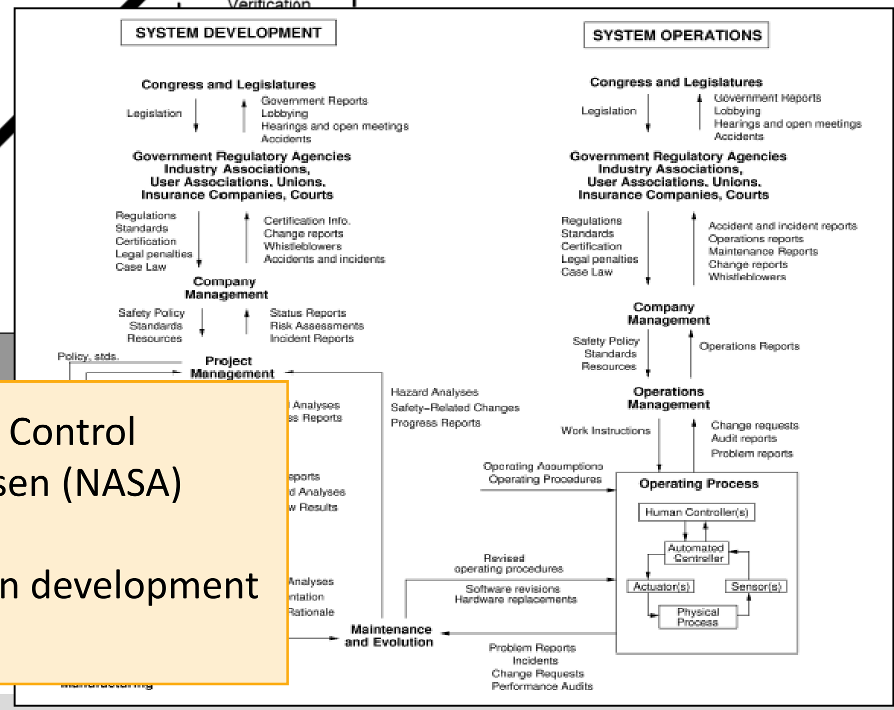


Labor-intensive

Early in system engineering

Largely Ignores Software as Hazard Source

Rarely repeated due to cost



Leveson (MIT) Socio-technical Control Framework based on Rasmussen (NASA) model of risk management

Multiple hazard contributors in development and operational context

We Rely on Software for Safe Aircraft Operation

Quantas Airbus A330-300 Forced to make Emergency Landing - 36 Injured

Written by htbw on Oct-7-08 1:48pm

From: soyawannaknow.blogspot.com

Thirty-si… in a mid… emerge… Tuesday…

The terrifying incident saw the Airbus A330-300 issue 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 safety analysis

Oct. 15 (Bloomberg) -- **Airbus SAS** issued an alert to airli… after Australian investigators said a computer fault on a … **Ltd.** flight switched off the autopilot and generated false … jet to nosedive.

The Airbus A330-300 was cruising at 37,000 feet (11,277… computer fed incorrect information to the flight control sy… **Australian Transport Safety Bureau** said yesterday. The… 650 feet within seconds, slamming passengers and crew… ceiling, before the pilots regained control.

``This appears to be a unique event,'' the bureau said, a… Toulouse, France-based Airbus, the world's largest maker… aircraft, issued a telex late yesterday to airlines that fly A… fitted with the same air-data computer. The advisory is `… minimizing the risk in the unlikely event of a similar occurr…

FAA says software problem with Boeing 787s could be catastrophic

By **Dan Catchpole**

@dcatchpole

The Federal Aviation Administration says a software problem with Boeing 787 Dreamliners could lead to one of the advanced jetliners losing electrical power in flight, which could lead to loss of control.

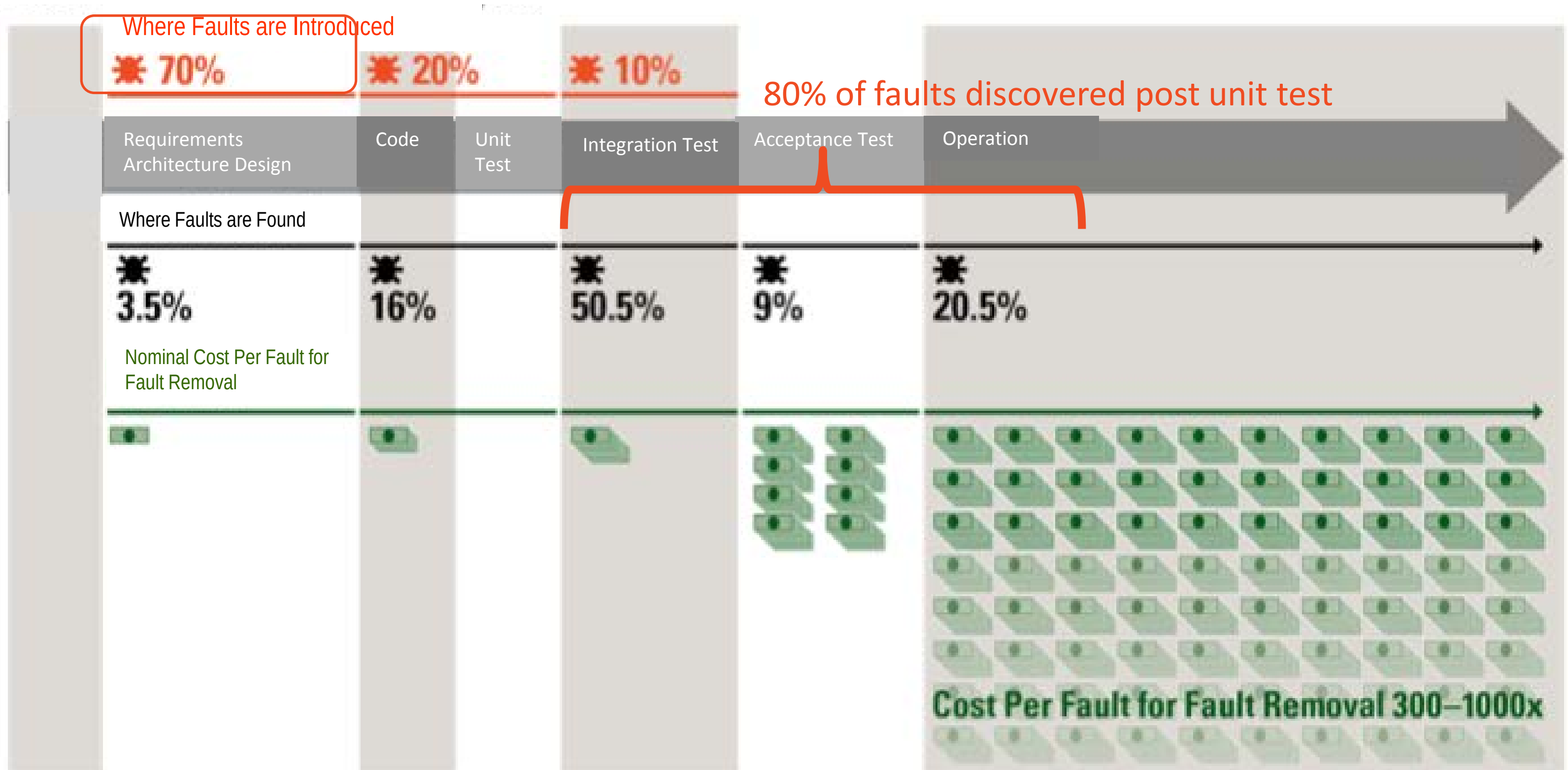
The Buzz: Hipster's dilemma

Boeing & aerospace news

Aerospace blog

The FAA notified operators of the airplane Friday that if a 787 is powered continuously for 248 days, the plane will automatically shut down its alternating current (AC) electrical power.

Safety Critical Software System Challenges

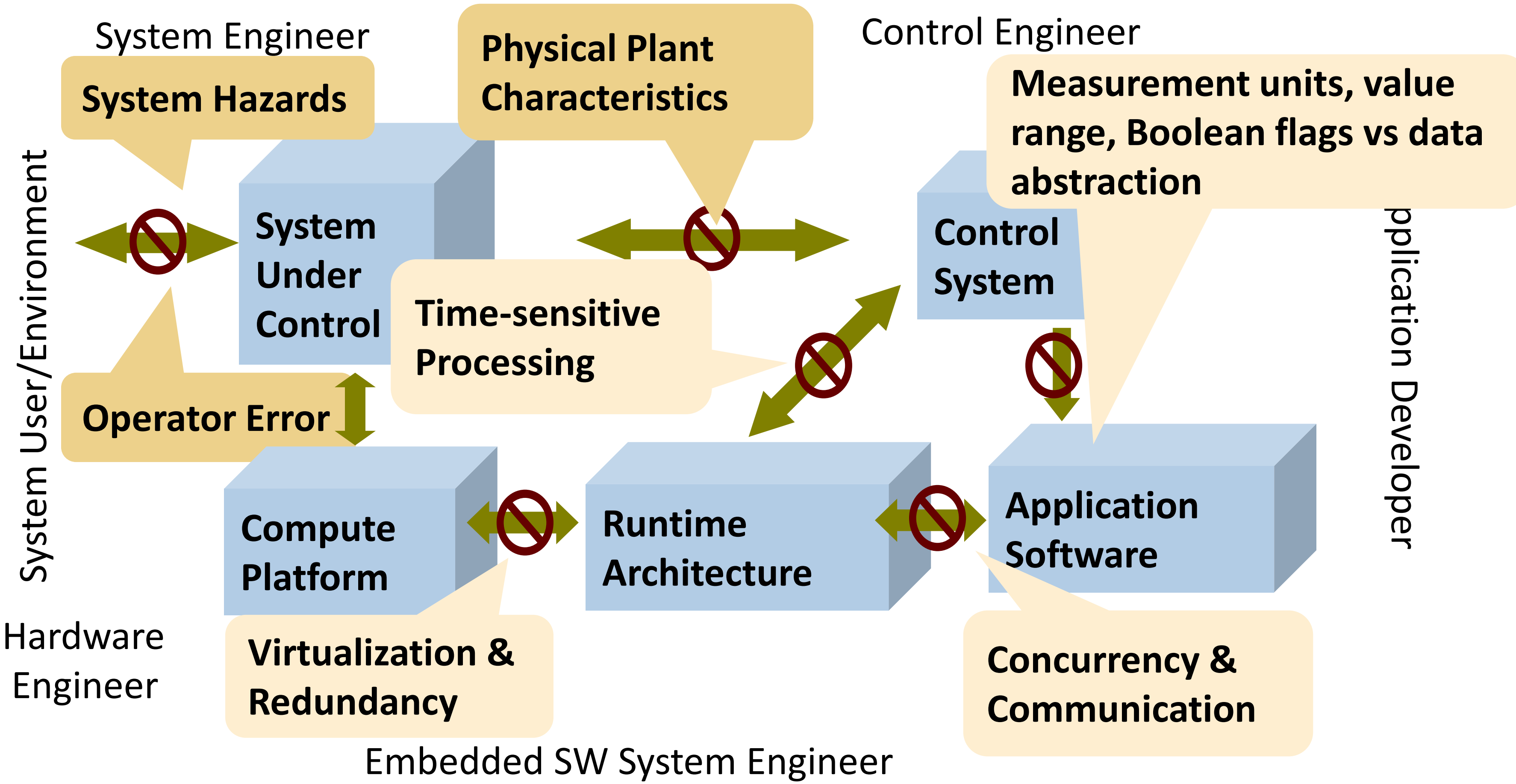


Sources: *Critical Code*; NIST, NASA, INCOSE, and Aircraft Industry Studies

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

Recertification cost is not proportional to system changes

Mismatched Assumptions in Safety-Critical System Interactions



Embedded software system as major source of hazards

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

Software Reliability

Observations

- Software reliability does not adhere to the bathtub failure rate curve for hardware

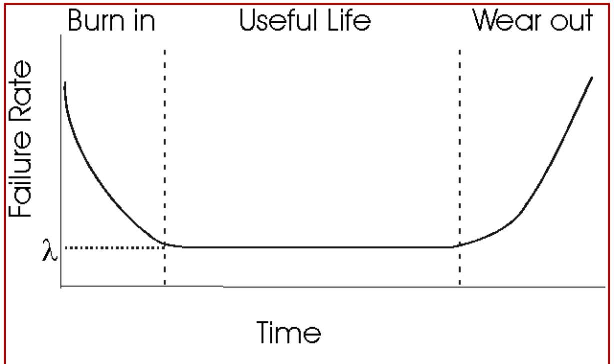


Figure 1. Bathtub curve for hardware reliability

Lifespan of single design and physical product

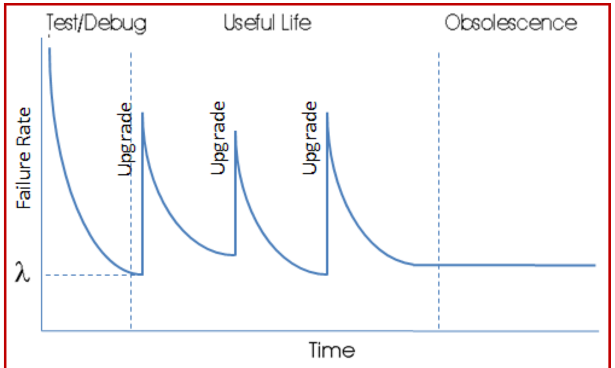


Figure 2. Revised bathtub curve for software reliability

Multi-release error rate of operating systems

- Software errors are design errors
- Software is not perfectable (unreasonable Zero defect assumption)
- Software is sensitive to operational context; testing has limited effectiveness

In a given use scenario the software defect is triggered every time

Improve Quality
Analytical verification
Coverage of exceptional conditions
Resilience to software defects

Operator Error Statistics

80% of accidents identified as due to pilot/operator errors

- References: http://www.vtol.org/safety.html (AF, Army), Leveson & other studies
- Result of single root cause event chain & focus on blame
- Operational procedures are not always in line with actual system operation
- Up to 75% of time dealing with operational work-around procedures instead of correcting the problem in software

Need for re-certification cost reduction

Challenges in Safety-Critical Digital Systems

Embedded software system as major hazard source

- High interaction complexity, mismatched assumptions, mode confusion
- Accidents due to combinations of major and minor hazard contributors

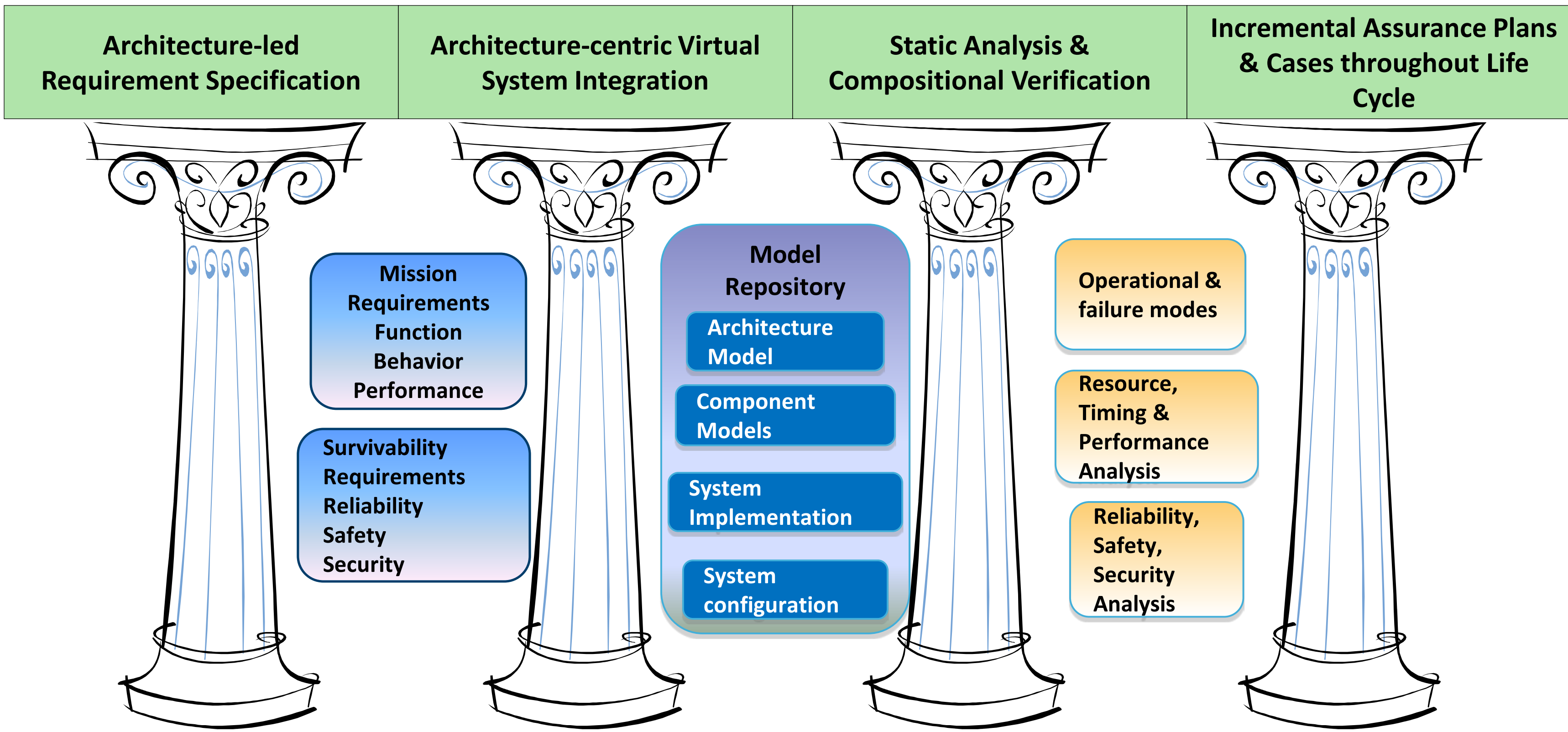
System safety analysis

- Safety engineering largely viewed as a system engineering practice
- Safety analysis processes are labor-intensive
- Consistency between evolving architecture design and safety analysis models

Virtual System Integration and Verification

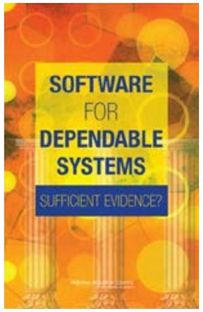
Reliability & Qualification Improvement Strategy

2010 SEI Study for AMRDEC
Aviation Engineering Directorate



Four pillars for Improving Quality of Critical Software-reliant Systems

Software for Dependable Systems: Sufficient Evidence? (National Research Council Study)



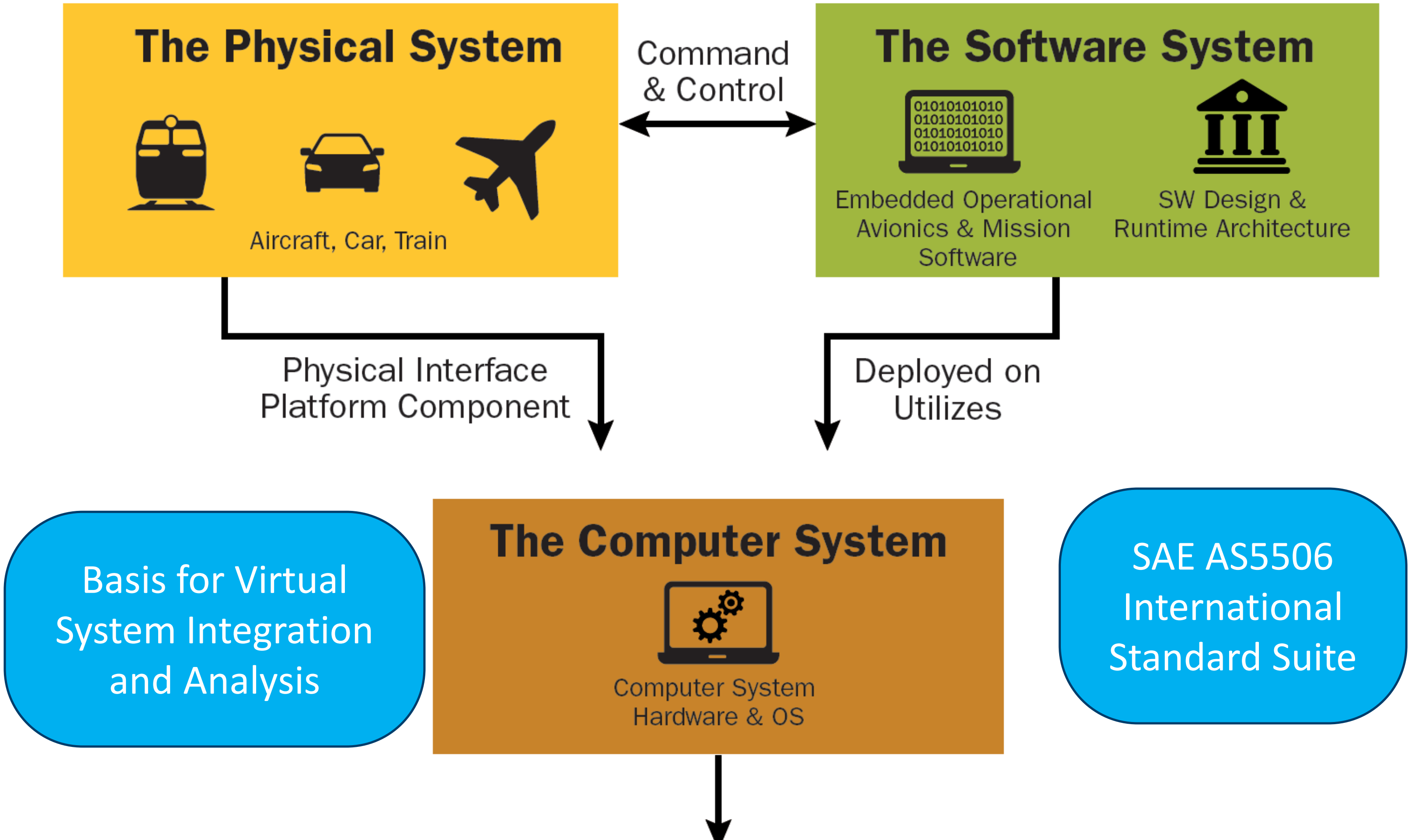
Testing is indispensable **BUT**

- “A rigorous development process in which testing and code review are the only verification techniques cannot justify claims of extraordinarily high levels of dependability”
- “Execution of even a large set of end-to-end tests, even with high levels of code coverage, in itself says little about the dependability of the system as a whole.”
- “For testing to be a credible component of a [case for dependability], the relation between testing and properties claimed will need to be explicitly justified”
- “Credible claims of dependability are usually impossible or impractically expensive to demonstrate after design and development are complete”

Assurance that a system is dependable requires the construction and evaluation of a “dependability case”

- Claims, arguments, evidence, expertise

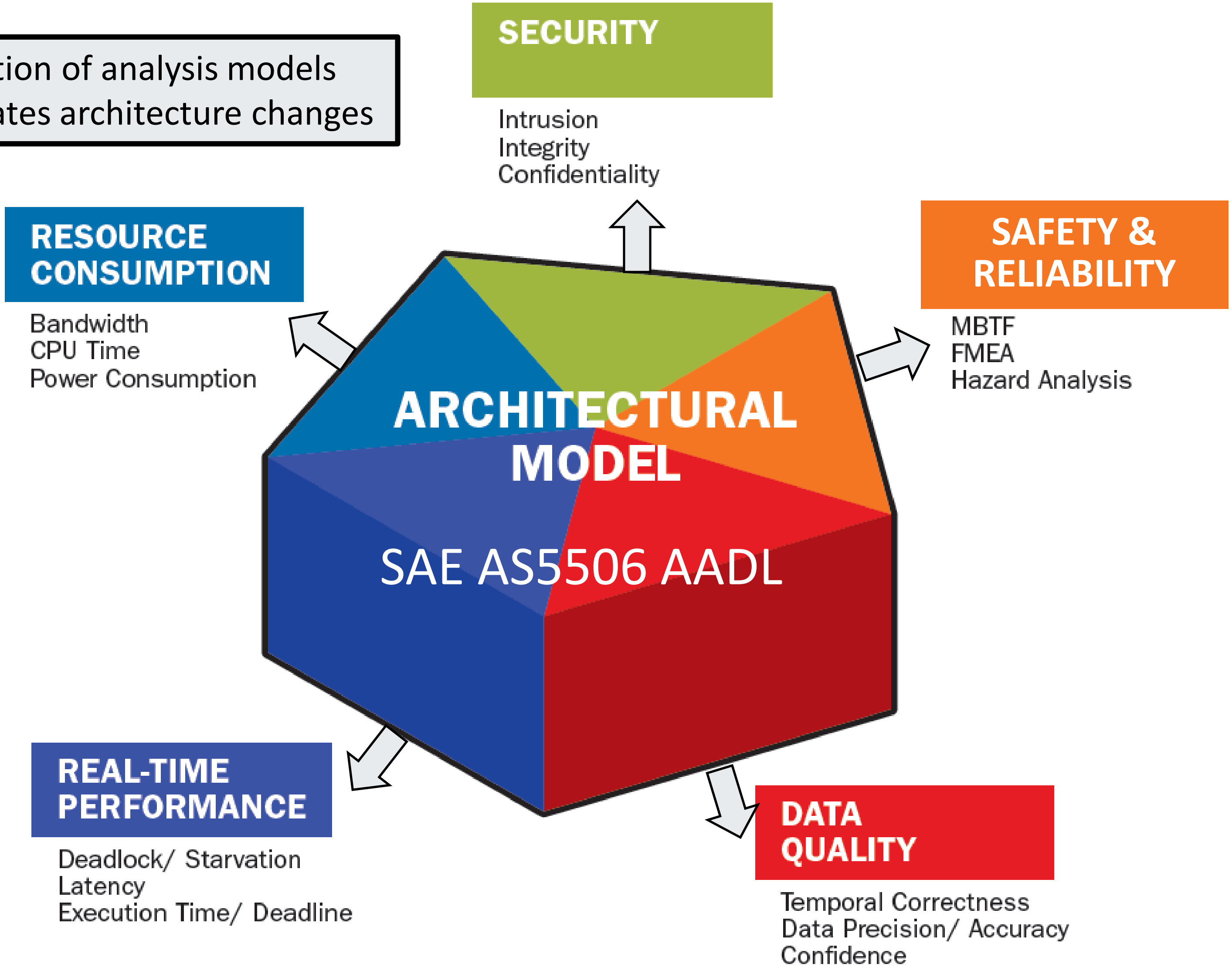
SAE Architecture Analysis & Design Language (AADL) Standard Suite



Analysis of AADL models results in early discovery of mismatched interaction assumptions and system level problems

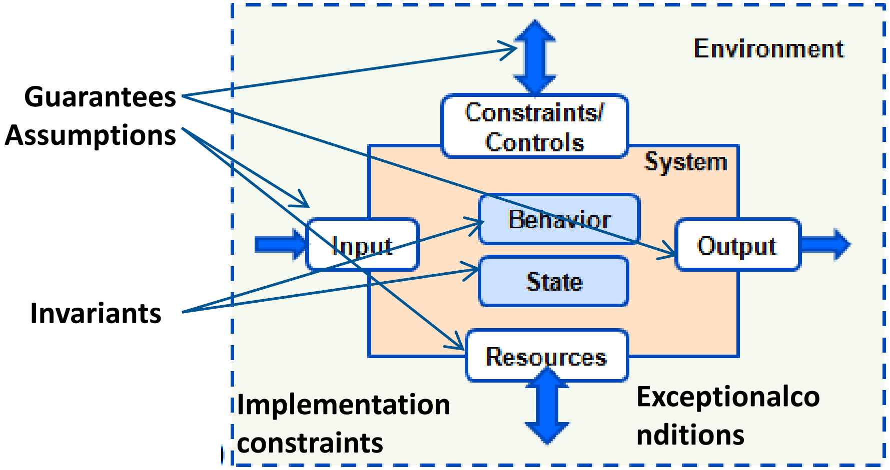
Analyzable Architecture Models Discover System Level Issues Early in Development

Generation of analysis models propagates architecture changes

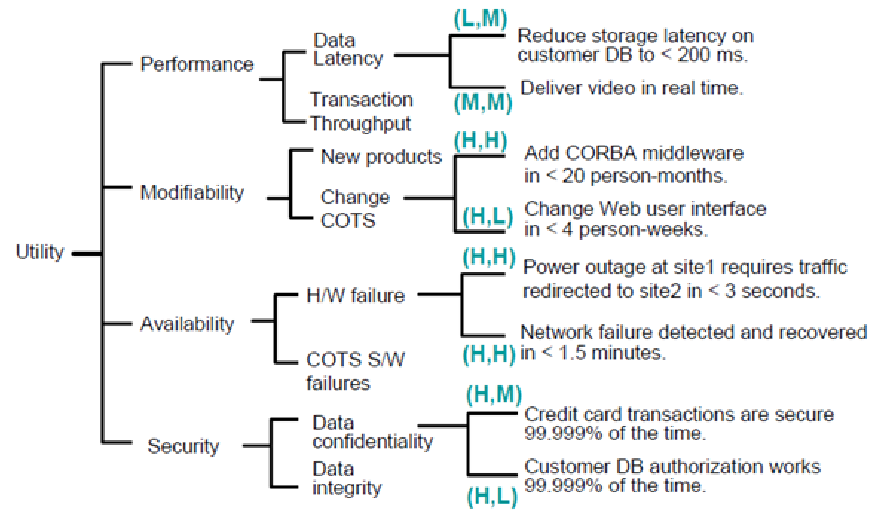


Three Dimensions of Requirement Coverage

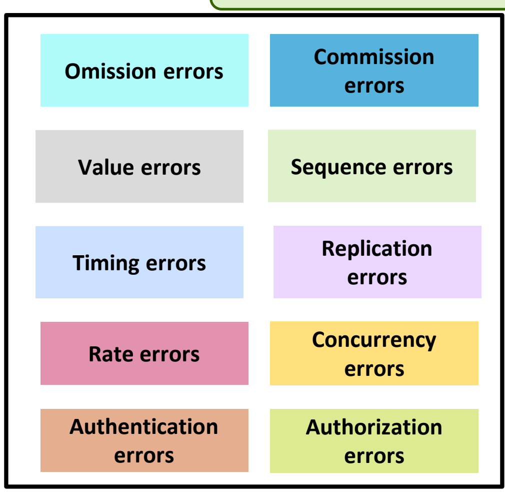
System interactions, state, behavior



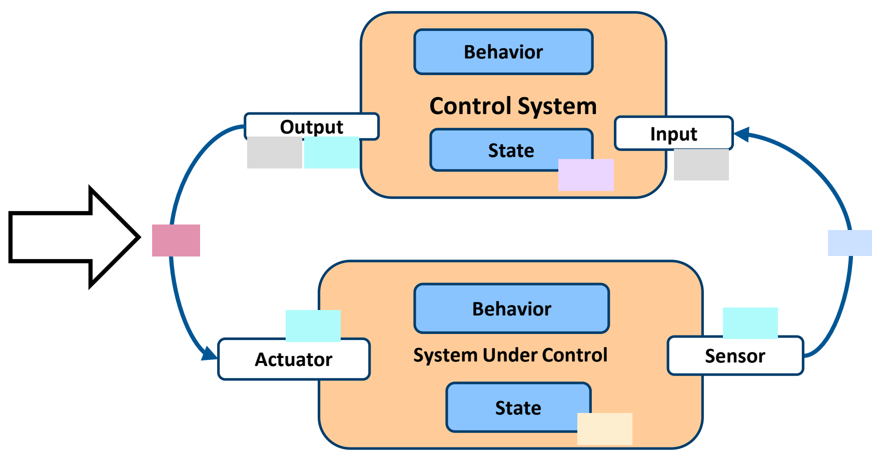
Design & operational quality attributes



Fault impact & contributors



Fault Propagation Ontology



Software Solutions Symposium 2017

Automation of Safety Analysis

Why Safety & Reliability Analysis Automation

Current process is

- Labor-intensive, years between repetition
- Prone to inconsistencies with evolving architecture and other analyses
- Requires knowledge of Markov, Petri net, and other notations

Early automation experiments with AADL

Steven Vestal, Honeywell, MetaH, Error Model, AADL committee, Avionics system trade studies during bidding (1999-)

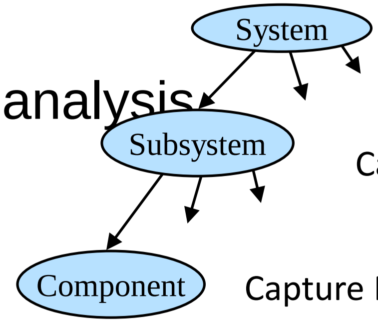
Myron Hecht, Aerospace Corp., member of AADL & DO-178C committee, Automated safety analysis of several satellite systems for JPL (2009-)
FMEA with 26,000 failure modes and 25 levels of effects

Thomas Noll, University of Aachen, COMPASS project, Automated safety analysis and verification of satellite systems for ESA (2008-)

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

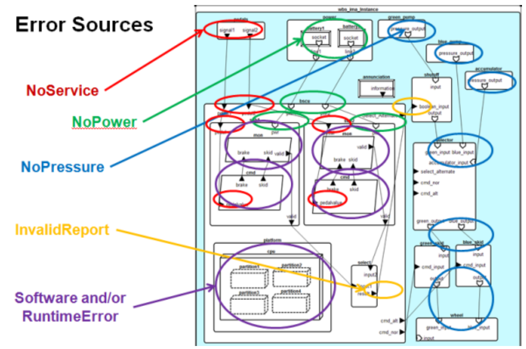
SAE ARP 4761 *Guidelines and Methods for Conducting the Safety Assessment Process on Civil Airborne Systems and Equipment*
Demonstrated in SAVI Wheel Braking System Example

Error Model Annex can be adapted to other ADLs

Automation of Safety Analysis Practice (SAVI)

A public Aircraft Wheel Brake System model

https://wiki.sei.cmu.edu/aadl/index.php/ARP4761_-_Wheel_Brake_System_%28WBS%29_Example



Use of Error-Model and ARINC653 annexes

- Relevance for the avionics community

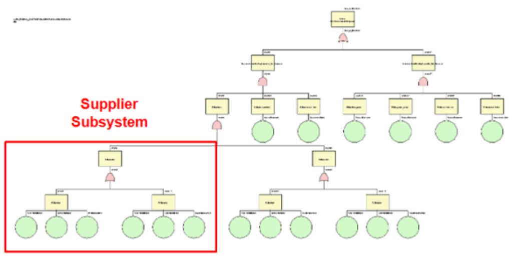
Comparative study

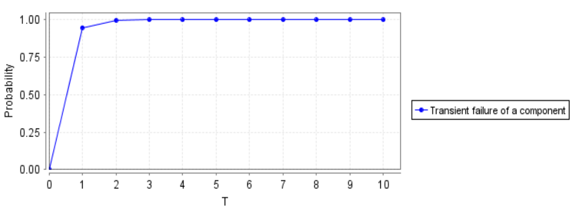
- Federated vs. Integrated Modular Avionics (IMA) architecture

Support of SAE ARP 4761 System Safety Assessment Practice

- Hazards (FHA), Fault Trees (FTA), Fault Impact (FMEA)
- Reliability/Availability (Markov Chain/Dependence Diagram)

Function Name	Failure Mode	Failure Rate (E-6)	Flight Phase	Failure Effect	Detection Method	Comments
+5 Volt	+5V out of spec.	0.2143	All	Possible P/S shutdown	Power Supply Monitor trips, shuts down supply and passes "invalid power supply (P/S)" to other BSCU system	BSCU channel fails
	+5V short to ground	0.2857	All	P/S shutdown	Power supply monitor passes invalid P/S to other BSCU system	BSCU channel fails
	Loss of / reduced filtering	0.3571	All	Increase Ripple	May pass out of spec voltage to rest of BSCU if ripple is such that it is not detected by the P/S monitor	May cause spurious P/S monitor trip
	+5V open	0.5714	All	P/S shutdown	Power supply monitor passes invalid P/S to other BSCU system	BSCU channel fails
	No Effect	0.1429	All	No Effect	None/No Effect	No Effect
Total Failure Rate of +5V Supply		1.5714				



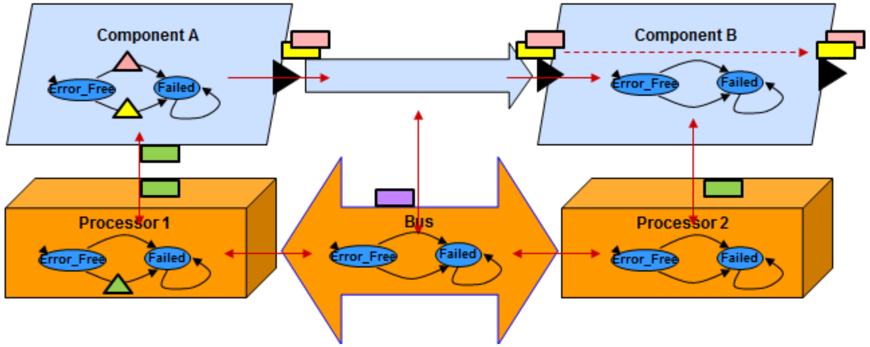


Error Model V2 Annotations of AADL Model

Three levels of abstraction expressed by EMV2

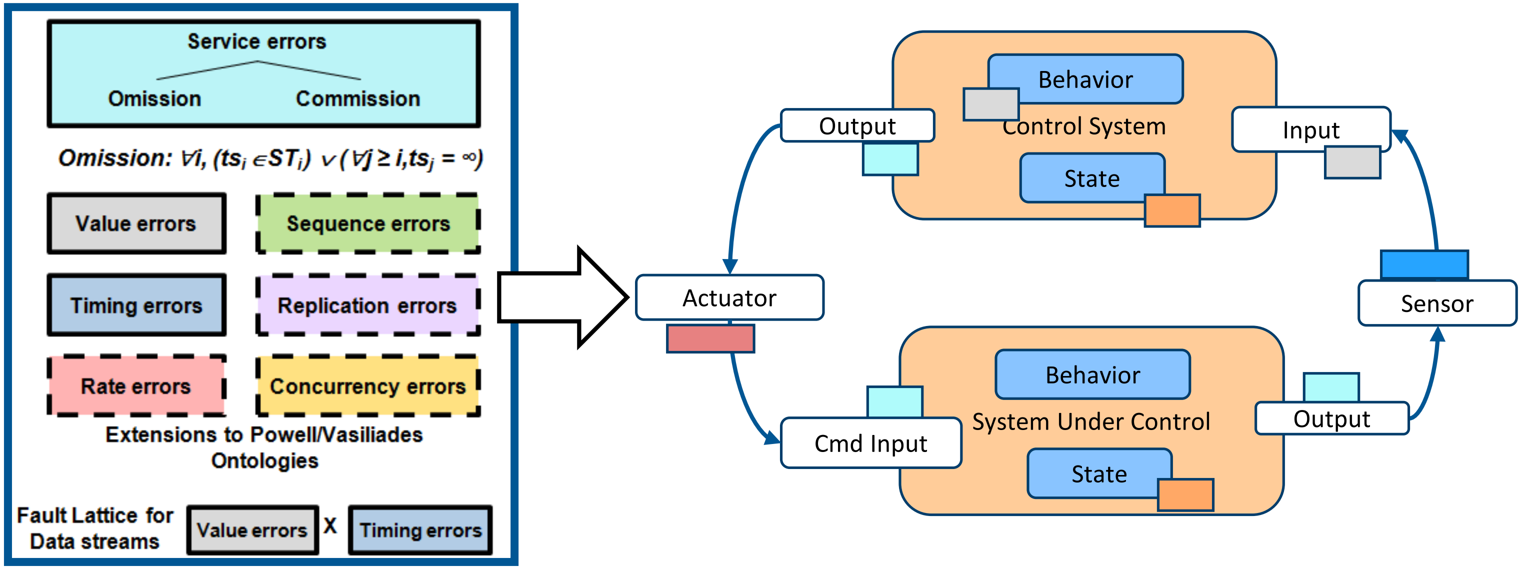
- Focus on fault propagation across components
  - Probabilistic error sources, sinks, paths and transformations
  - Fault propagation and Transformation Calculus (FPTC) from York U.
- Focus on fault behavior of components
  - Probabilistic typed error events, error states, propagations
  - Voting logic, 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

Fault tree generated from EMV2 annotations and propagation paths inferred from AADL model



Coverage of Fault Propagation Taxonomy

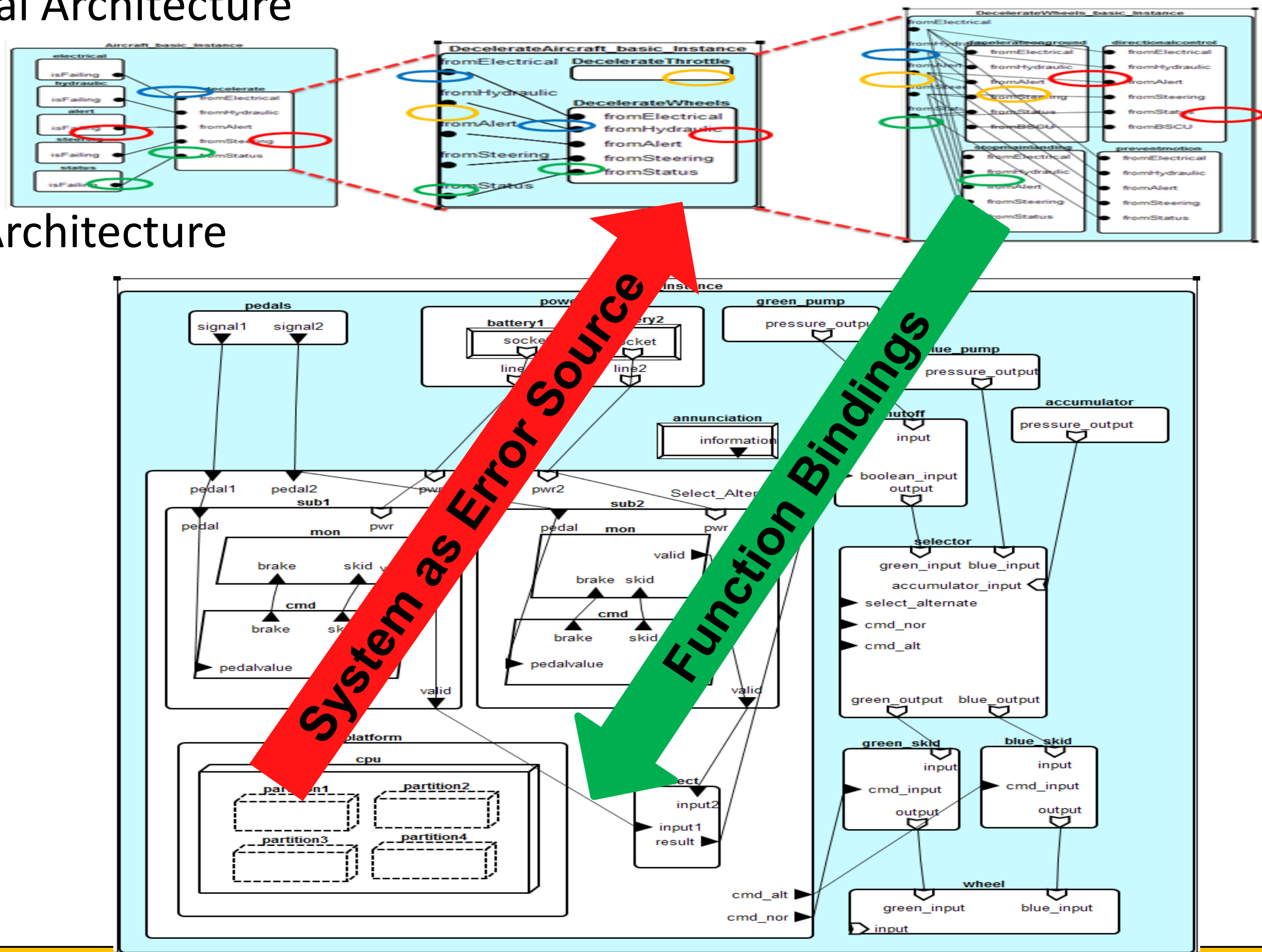
Fault Propagation Taxonomy



Functional and System Architecture

Functional Architecture

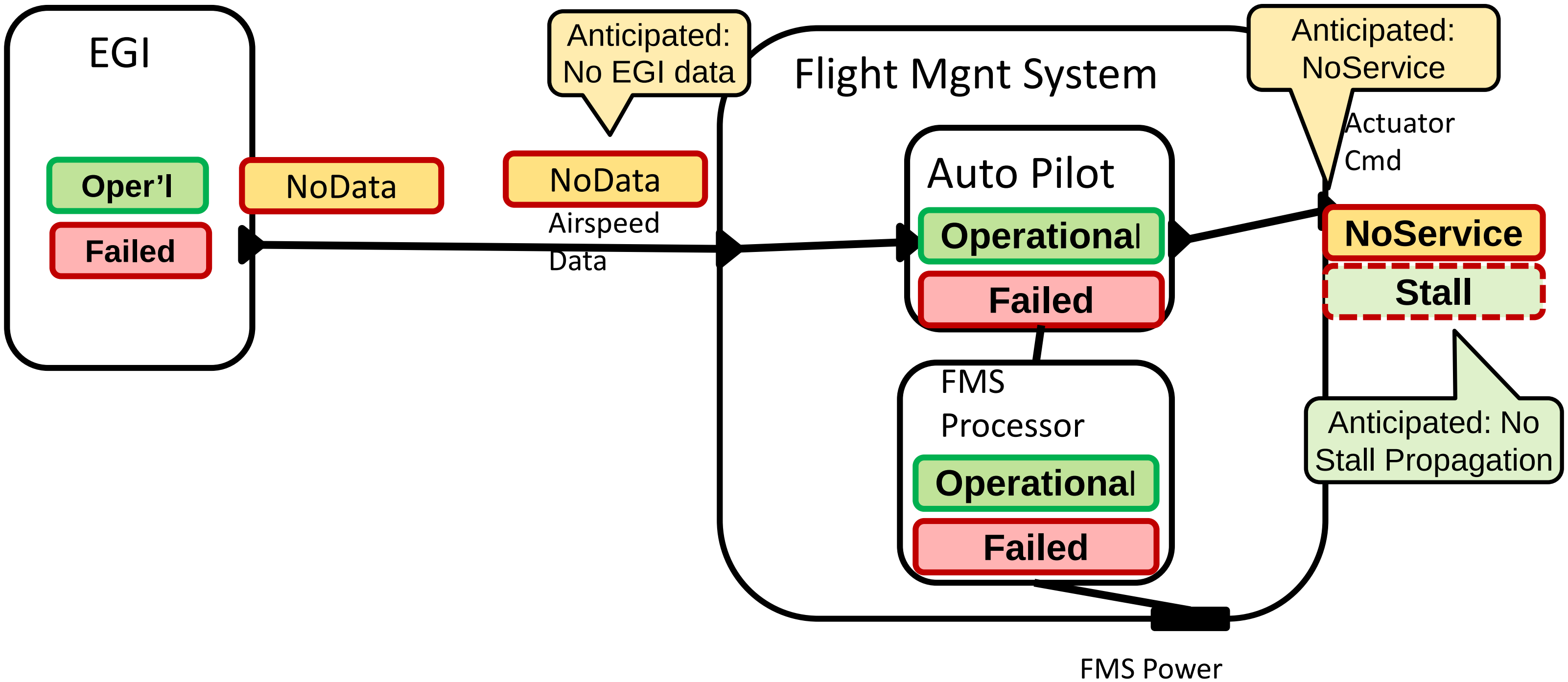
System Architecture



Consistency of Functional and System Fault Models

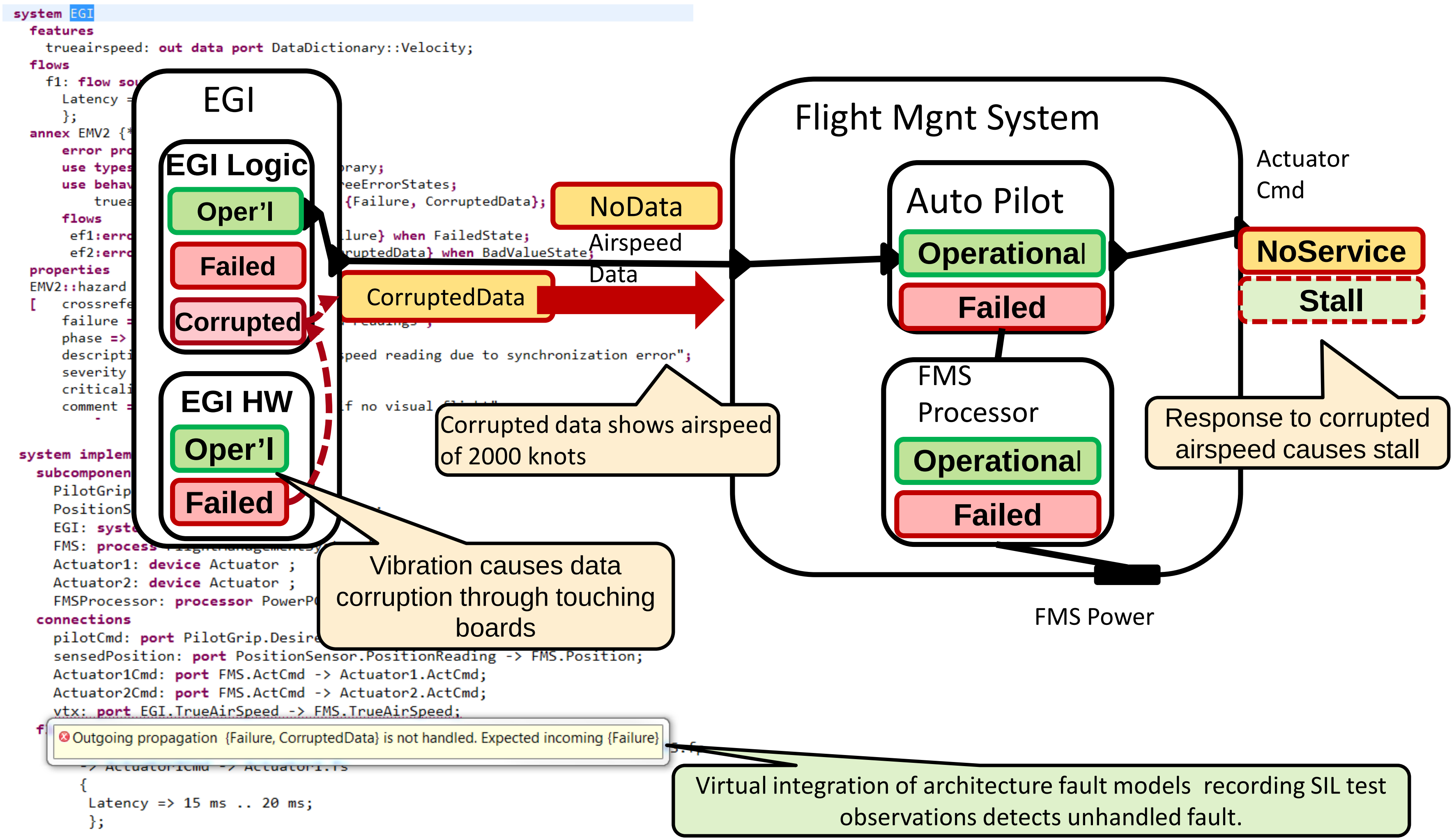
Function Mappings Imply System Components as Common Error Source

Software Induced Flight Safety Issue



Original Preliminary System Safety Analysis (PSSA)
System engineering activity with focus on failing components.

Unhandled Hazard Discovery through Virtual Integration



Software Solutions Symposium 2017

Automated Fault Tree and Common Cause Analysis

Automated Fault Tree Analysis from AADL Models

Fault tree generation from annotated AADL model

- AADL focuses on embedded software systems
- Error Model V2 Annex specifies error behavior at three levels of abstraction
- Architecture design changes are consistently reflected in fault tree

Use of fault propagation taxonomy

- Bounded set of failure effect types
- Taxonomy coverage
- Error propagation contracts and unhandled faults

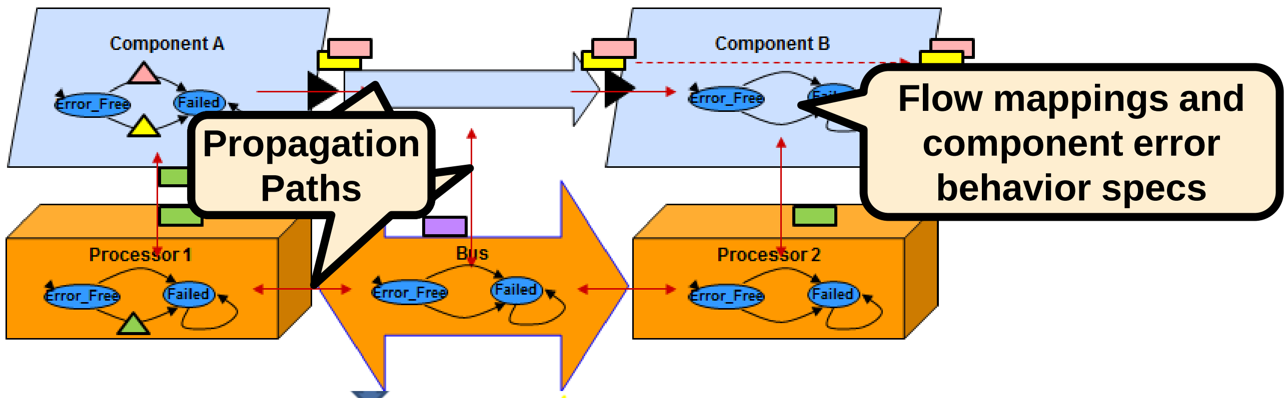
Common cause failure contributors

- Identification of fan-out in propagation paths
- Transformation of generated fault graph to eliminate/reduce dependent fault tree events

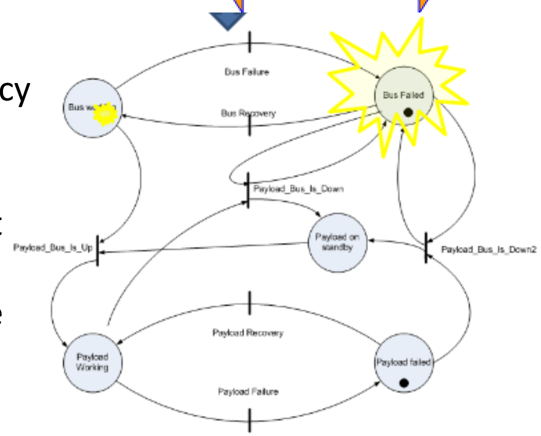
Propagation and Recovery Dependency Graph

AADL core model provides propagation paths

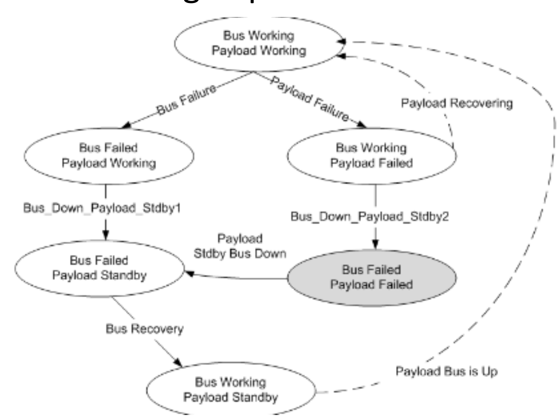
- Port connections, access connections, remote service calls
- Deployment binding of SW to HW



Abstracted dependency graph
Example: Petri net
OSATE2 uses compact propagation graph derived from instance model



Resulting impact trace



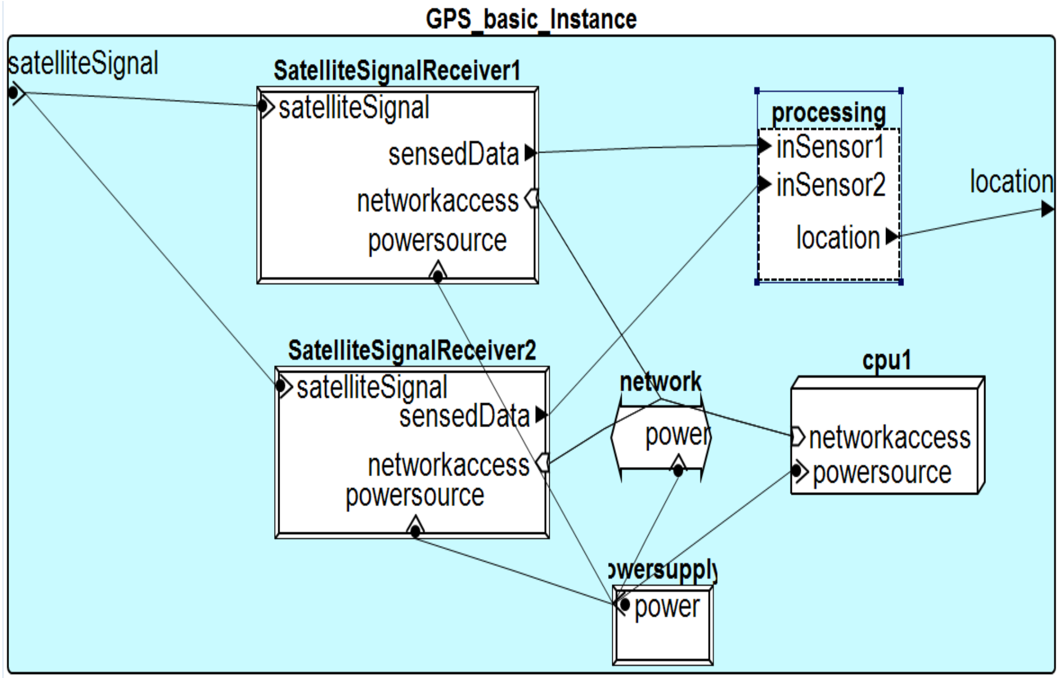
Example System: GPS

Dual redundant satellite signal receivers

- One is sufficient for less precise location output

Single power supply

- Common cause failure source



Error Propagation Specification

```
abstract GPSProcessing
features
  inSensor1: in data port;
  inSensor2: in data port;
  location: out data port;
annex EMV2 {**
use types ErrorLibrary, GPSErrorLibrary;
error propagations
 inSensor1 : in propagation {ServiceOmission};
 inSensor2 : in propagation {ServiceOmission};
 location : out propagation {ServiceOmission, LowPrecisionData,IncorrectData};
 processor: in propagation {ServiceOmission};
flows
 s1toloc: error path inSensor1{ServiceOmission} -> location{ServiceOmission};
 s2toloc: error path inSensor2{ServiceOmission} -> location{ServiceOmission};
 ptoloc: error path processor{ServiceOmission} -> location{ServiceOmission};
 gpssrc: error source location{LowPrecisionData, IncorrectData};
end propagations;
**};
end GPSProcessing;
```

Out propagation of different error types

Error paths include propagation from processor binding

Component Error Behavior Specification

```
abstract GPSProcessing_computeError extends GPSProcessing
annex EMV2 {**
  use types ErrorLibrary, GPSErrorLibrary;
  use behavior GPSErrorLibrary::GPSProcessingFailed;
  component error behavior
  events
    computeError: error Event;
  transitions
    internal: Operational -[computeError]-> Incorrect;
    lowPrecision: Operational -[inSensor1{ServiceOmission}
        or inSensor2{ServiceOmission}]-> LowPrecision;
    inputNoService: all -[inSensor1{ServiceOmission}
        and inSensor2{ServiceOmission}]-> NoService;
    CPUNoService: all -[processor {ServiceOmission}]-> NoService;
  propagations
    outNoService: NoService-[]-> location{ServiceOmission};
    outLowPrecision: LowPrecision-[]-> location{LowPrecisionData};
    outComputeErrorEfect: Incorrect-[]-> location{IncorrectData};
  end component;
  properties
    emv2::OccurrenceDistribution => [ ProbabilityValue => 7.5e-4 ;
        Distribution => Poisson;
    ] applies to computeError;
**};
end GPSProcessing_computeError;
```

Failure modes and handling of different error types

Handling Common Cause Failure Source

Propagation path fan out identifies common cause source

Transformations on common cause elements

Move common event up

(SSR1 or PS) and (SSR2 or PS)

=> PS or (SSR1 and SSR2)

Absorb subgate with common event

(SSR1 or PS) and PS => PS

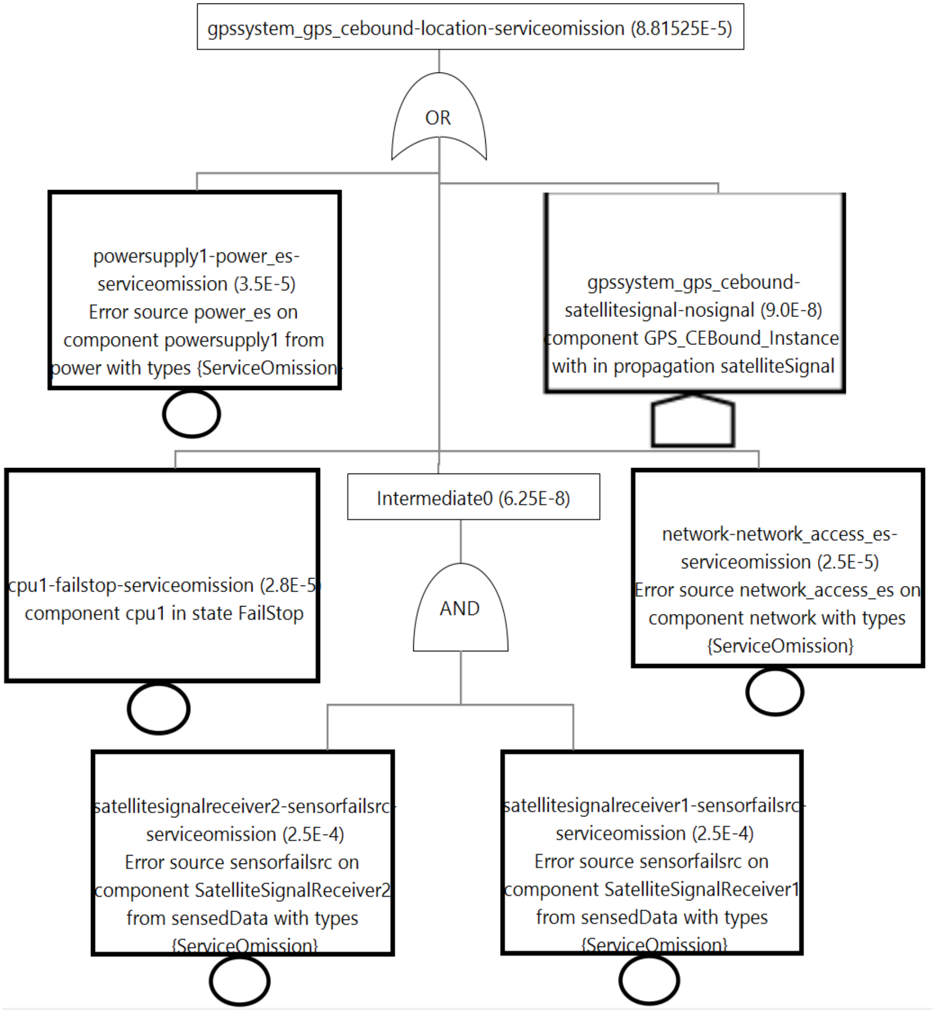
Eliminate replicate events

PS or PS => PS

Flatten nested gates

E1 or (E2 or E3) => or{E1, E2, E3}

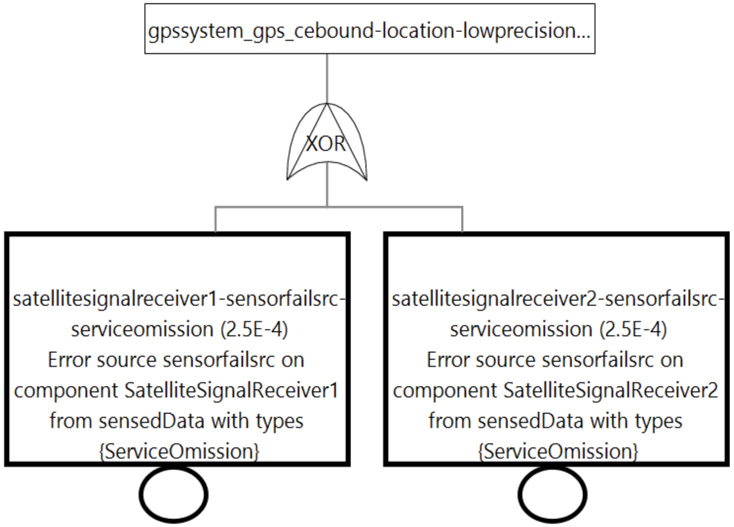
Elimination of dependent events simplifies occurrence probability calculation



Fault Tree for Degraded Mode

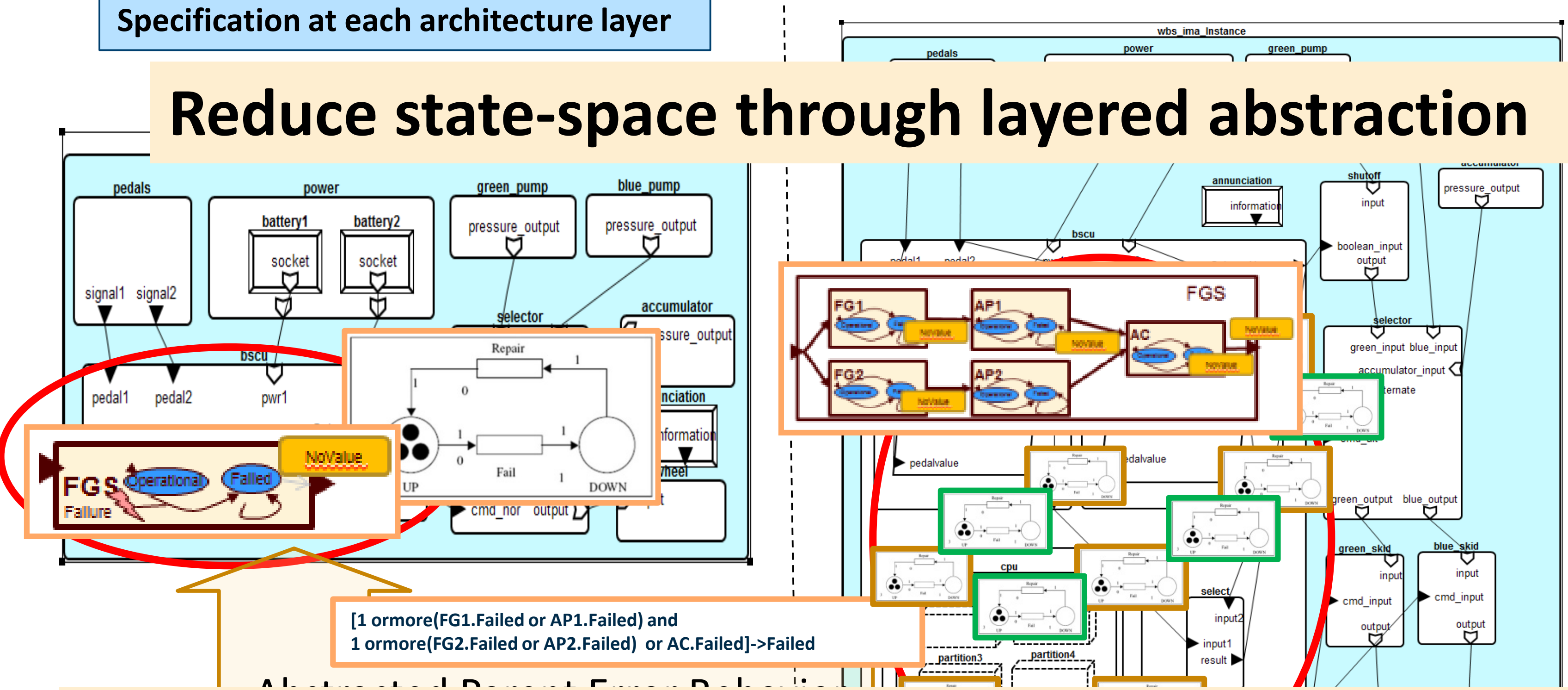
Condition: only one receiver fails

- Cannot include common cause contributors



Scalability and Incremental Safety Analysis

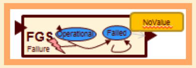
Abstract and Composite Error Model Specification at each architecture layer



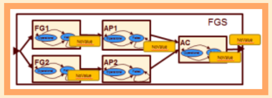
Reduce state-space through layered abstraction

[1 ormore(FG1.Failed or AP1.Failed) and
1 ormore(FG2.Failed or AP2.Failed) or AC.Failed]->Failed

Consistency of abstract specification compositional specification and implementation fault models



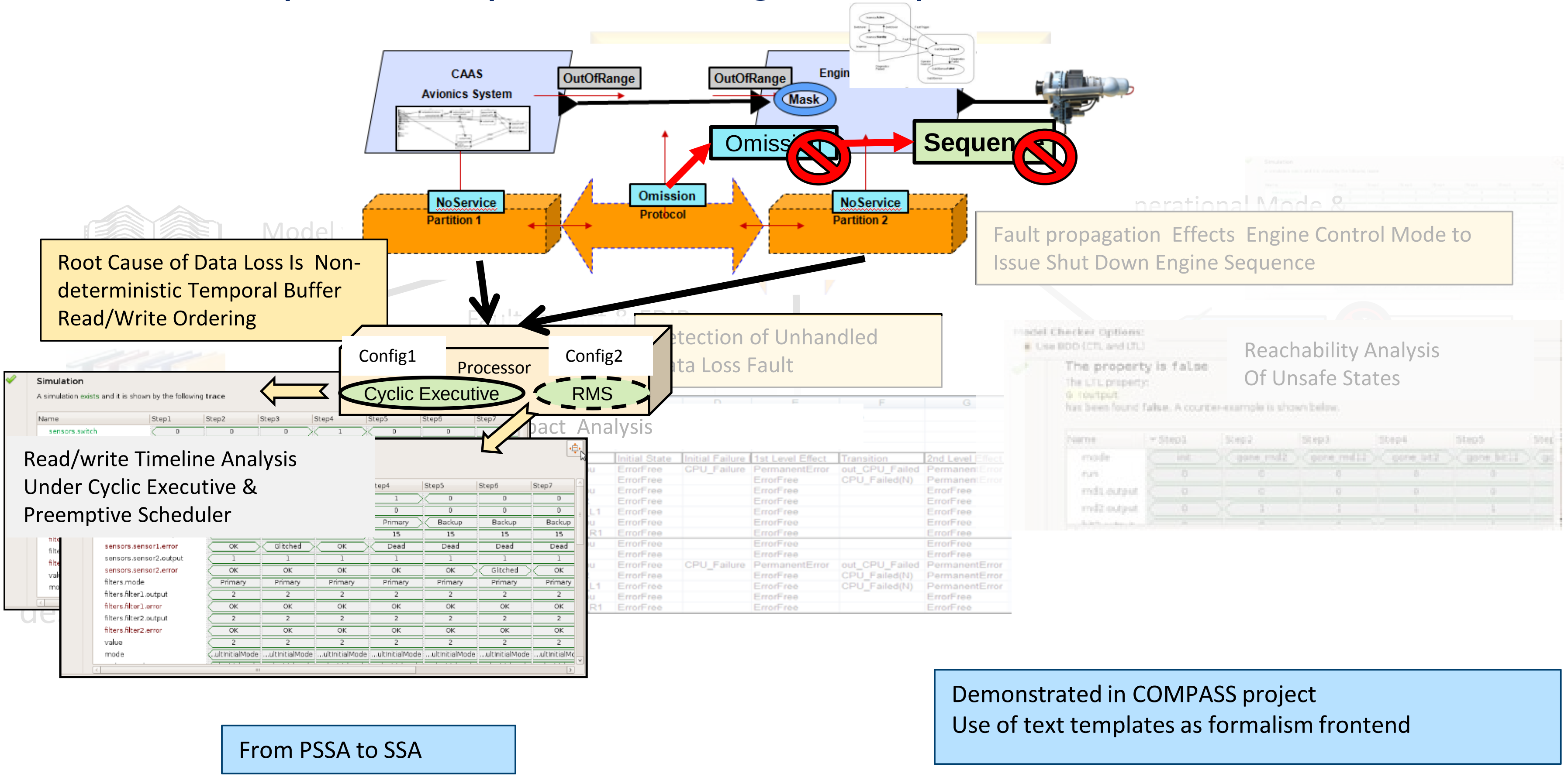
[1 ormore(FG1.Failed or AP1.Failed) and
1 ormore(FG2.Failed or AP2.Failed) or AC.Failed]->Failed



Abstraction across one or more architecture layers/tiers

Understanding the Cause of Faults

Through model-based analysis identify architecture induced unhandled, testable, and untestable faults and understand root causes, contributing factors, impact, and potential mitigation options.

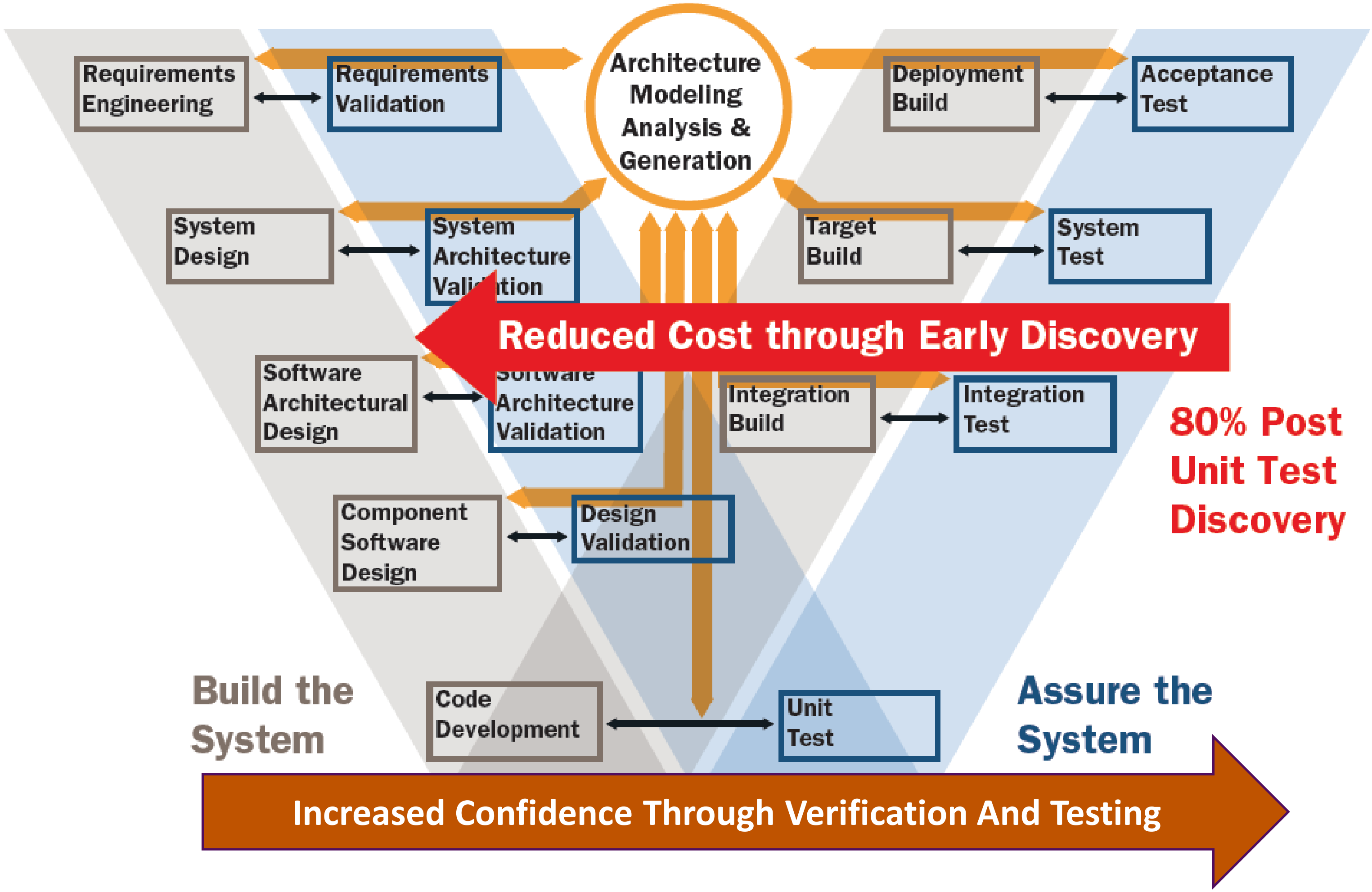


Benefits of Safety Analysis Automation

Automation allows for

- Early identification of potential problems
  - Single points of failure
  - Unanticipated effects
- Larger set of failure modes and combinations
- More levels of effects
- Safety analysis of system and software architecture
- More frequent re-analysis
- Architecture trade studies
- Consistency across analysis results

Benefits of Virtual System Integration & Incremental Lifecycle Assurance



References

Website www.aadl.info

Public Wiki https://wiki.sei.cmu.edu/aadl

Error Model V2 Annex Standard, SAE AS-5506/1A, Sept 2015.

AADL Fault Modeling and Analysis Within an ARP4761 Safety Assessment, SEI Technical Report, CMU/SEI-2014-TR-020, 2014.

Architecture Fault Modeling and Analysis with the Error Model Annex V2, SEI Technical Report, CMU/SEI-2016-TR-009, 2016.

Improving Quality Using Architecture Fault Analysis with Confidence Arguments, SEI Technical Report, CMU/SEI-2015-TR-006, 2015.

Automated Fault-Tree Analysis from AADL Models, Feiler & Delange, ACM High Integrity Language Technology International Workshop (HILT) 2016.

Julien Delange, Peter Feiler, Neil Ernst, Incremental Life Cycle Assurance of Safety-Critical Systems, 8th European Congress on Embedded Real Time Software and Systems (ERTS 2016), Jan 2016.

Contact Information

Peter Feiler

SEI Fellow

Telephone: +1 412.268.7790

Email: phf@sei.cmu.edu

U.S. Mail:

Software Engineering Institute
Carnegie Mellon University
4500 Fifth Avenue
Pittsburgh, PA 15213-3890

World Wide Web:

http://www.sei.cmu.edu/architecture/research/model-based-engineering/

www.aadl.info

www.aadl.info/wiki

osate.org

www.github.org/osate