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 Management
- DO-254-Design Assurance Guidance for Airborne Electronic Hardware
- DO-297-Integrated Modular Avionics (IMA) Development Guidance and Certification Considerations
- SAE-ARP4754-Certification Consideration for Highly Integrated or Complex Aircraft Systems
- SAE-ARP4671- Guidelines for Airworthiness
- FAA Advisory Circular AC 27-1B-Certification of Normal Category Rotorcraft
- FAA Advisory Circular AC 29-2C-Certification of Transport Category Rotorcraft
- ISO/IEC 12207-Software Life Cycle Processes
- ARINC 653-Specification Standard for Time and System Partitioning
- 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 Civil Aircraft in Instrument Meteorological Conditions and Civil Instrument Flight Rules
- ADS-64-SP-Airworthiness Requirements for Military Rotorcraft

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 a 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

Thirty-six passengers and crew had to be evacuated from a mid-air emergency event that left three people injured, according to Qantas newspapers.

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

The Airbus A330-300 was cruising at 37,000 feet (11,277 meters) when a computer fed incorrect information to the flight control system, the Australian Transport Safety Bureau said yesterday. The jet nosedived into the ocean 650 feet within seconds, slamming passengers and crew against the ceiling, before the pilots regained control.

"This appears to be a unique event," the bureau said, adding that Airbus had not confronted similar issues before. The company is based in Toulouse, France.

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 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

80% of faults discovered post unit test

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

Recertification cost is not proportional to system changes

Sources: Critical Code; NIST, NASA, INCOSE, and Aircraft Industry Studies
Mismatches in Assumptions in Safety-Critical System Interactions

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

Embedded software system as major source of hazards
Software Reliability

Observations

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

  ![Bathtub curve for hardware reliability](image1.png)

  ![Revised bathtub curve for software reliability](image2.png)

  Lifespan of single design and physical product

  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

Software Solutions Symposium 2017
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

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

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

- Architecture-led Requirement Specification
- Architecture-centric Virtual System Integration
- Static Analysis & Compositional Verification
- Incremental Assurance Plans & Cases throughout Life Cycle
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

The Physical System
- Aircraft, Car, Train

Command & Control

The Software System
- Embedded Operational Avionics & Mission Software
- SW Design & Runtime Architecture

Physical Interface Platform Component

The Computer System
- Computer System Hardware & OS

Deployed on Utilizes

Basis for Virtual System Integration and Analysis

SAE AS5506 International 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

- **RESOURCE CONSUMPTION**
  - Bandwidth
  - CPU Time
  - Power Consumption

- **REAL-TIME PERFORMANCE**
  - Deadlock/Starvation
  - Latency
  - Execution Time/Deadline

- **DATA QUALITY**
  - Temporal Correctness
  - Data Precision/Accuracy
  - Confidence

- **SAFETY & RELIABILITY**
  - MTBF
  - FMEA
  - Hazard Analysis

- **SECURITY**
  - Intrusion
  - Integrity
  - Confidentiality

SAE AS5506 AADL
Three Dimensions of Requirement Coverage

System interactions, state, behavior

Environment

Constraints/Controls

System

Input

Behavior

State

Output

Resources

Guarantees

Assumptions

Invariants

Implementation constraints

Design & operational quality attributes

Performance

Transaction

New products

Change COTS

Modifiability

Utility

Availability

COTS S/W failures

Security

Data confidentiality

Data integrity

Exceptional conditions

Invariants

Fault impact & contributors

Fault Propagation Ontology

Omission errors

Commission errors

Value errors

Sequence errors

Timing errors

Replication errors

Rate errors

Concurrency errors

Authentication errors

Authorization errors

Behavior

Output

Control System

State

Input

Behavior

System Under Control

State

Actuator

Sensor
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

Error Model Annex can be adapted to other ADLs
Automation of Safety Analysis Practice (SAVI)

A public Aircraft Wheel Brake System model

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)
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

Service errors
- Omission
- Commission

Omission: $\forall i, (ts_i \in ST_i) \lor (\forall j \geq i, ts_j = \lnot)$

Value errors
- Sequence errors
- Replication errors
- Concurrency errors

Extensions to Powell/Vasiliades Ontologies

Fault Lattice for Data streams
- Value errors
- Timing errors

Control System
- Behavior
- State

Output

Input

Actuator

System Under Control
- Behavior
- State

Cmd Input

Sensor

Output
Improvements in Safety Analysis for Safety-critical Software Systems

March 20–23, 2017

© 2017 Carnegie Mellon University

Software Engineering Institute | Carnegie Mellon University

Functional and 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

Virtual integration of architecture fault models recording SIL test observations detects unhandled fault.

Corrupted data shows airspeed of 2000 knots

Vibration causes data corruption through touching boards

Response to corrupted airspeed causes stall
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
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

```plaintext
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};
      outComputeErrorEffect: Incorrect-[]-> location{IncorrectData};
  end component;
properties
  emv2::OccurrenceDistribution => [ ProbabilityValue => 7.5e-4 ;
     Distribution => Poisson;
  ] applies to computeError;
**};
end GPSProcessing_computeError;
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

Consistency of abstract specification compositional specification and implementation fault models

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.

Root Cause of Data Loss Is Non-deterministic Temporal Buffer Read/Write Ordering

Fault propagation Effects Engine Control Mode to Issue Shut Down Engine Sequence

Reachability Analysis Of Unsafe States

Fault Impact Analysis

Detection of Unhandled Data Loss Fault

Model validation Requirements

Faults that can be tested Decision coverage

Faults that cannot be tested Race conditions

Faults that are unhandled Transient data loss in protocol

Improved documentation & design

Demonstrated in COMPASS project Use of text templates as formalism frontend

From PSSA to SSA

Model Validation Requirements

Faults that can be tested

Decision coverage

Faults that cannot be tested

Race conditions

Faults that are unhandled

Transient data loss in protocol

Improved documentation & design

Demonstrated in COMPASS project Use of text templates as formalism frontend

From PSSA to SSA
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

Increased Confidence Through Verification And Testing

Build the System

Assure the System

Reduced Cost through Early Discovery

80% Post Unit Test Discovery

Increased Confidence Through Verification And Testing
References

Website [www.aadl.info](http://www.aadl.info)
Public Wiki [https://wiki.sei.cmu.edu/aadl](https://wiki.sei.cmu.edu/aadl)


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