Software assurance in an era of complexity

David Peterson
Principal Airworthiness Standards Officer
Civil Aviation Safety Authority (CASA)
How do we assure the airworthiness of systems incorporating software?
Ariane 501

From the report:

- **Disintegration** at H0 + 39 (T+39 sec) due to aerodynamic loads
- Full deflection of Vulcain engine & SRB nozzles (commanded by on-board computer)
- OBC command based on INS value containing ‘diagnostic data’ (not measured data!)
- INS 2 (SNI 2) failure (and diagnostic triggered) due to a software exception
- Exception due to overflow during data conversion from 64 bit float $\rightarrow$ 16 bit signed integer
- Conversion **unguarded** due to performance **constraints** of embedded system
- Overflow due to unexpectedly high value of BH parameter (horizontal bias) (vs. Ariane 4)
- Also: INS 1 (SNI 1) unavailable – it failed (common mode) on the prior OBC data cycle

> “the primary technical causes are the Operand Error when converting the horizontal bias variable BH, and the lack of protection of this conversion which caused the SRI computer to stop”

$\rightarrow$ Error arose in INS ‘strap-down’ code – these results were **unused** after launch !!!

http://sunnyday.mit.edu/accidents/Ariane5accidentreport.html
What this talk is about

- Why do systems incorporating software fail?
- What can we do to prevent this from happening?
- What does it mean to assure systems (and software) used in civil aviation?
- What must we remain mindful of when certifying systems incorporating software?
The era of complexity

- Each new generation of aircraft incorporates $\sim3x$ to $\sim10x$ the amount of software as the previous generation.

\begin{itemize}
  \item \textbf{Q: How to transform and scale our approach to software assurance to meet this complexity challenge?}
\end{itemize}
1. Why do software(-based) systems fail?

- Wrong (or incomplete) **requirements** (system or software -level)
- Wrong **implementations** – algorithmic correctness
- Wrong **behaviours** – behavioural correctness / robustness
- Inadequate **V&V** – code coverage, decision coverage, data bounds
- Tested ≠ **Proven**
- **Hardware** issues – systematic / one-off (e.g. SEU)
- **Other** – non-determinism / interference, (lack of) dissimilarity, (lack of) redundancy, (lack of) monitoring / filtering
“… most errors found in operational software can be traced to requirement flaws, particularly incompleteness. Completeness is a quality often associated with requirements but rarely defined.”

“Software requirement specifications are complete if they are sufficient to distinguish the desired behavior of the software from that of any other undesired program that might be designed.”

“Nearly all the serious accidents in which software has been involved in the past twenty years can be traced to requirements flaws, not coding errors.”

Source: ESW, pg. 49.

https://mitpress.mit.edu/books/engineering-safer-world
2. What can we do to prevent it?

- Maintain a **rigorous focus on requirements** – correct & complete? (vs. incomplete, i.e. underspecified)
- **Establish and maintain** process discipline
- **Learn** from the past – shift focus from *causes* to *reasons* (ESW)
- Keep it **simple** (avoid unnecessary complexity)
- **Test extensively** and with *real-world data* – abnormals / failures.
- **Achieve assurance at scale** using advanced tools – FM, MBE, etc.
- **Design for resilience** (degradation) & **redundancy** (failure)
3. How do we “assure” aviation software?

- ARP 4754A – Guidelines For Development of Civil Aircraft and Systems (→ System Certification Plan)
- ARP4761 – Safety Assessment Process (→ SSA)
- DO-178C – Software Considerations in Airborne Systems and Equipment Certification
- Supplements: DO-330, DO-331, DO-332, DO-333

Also …
- DO-297 – IMA
- DO-254 – CEH
- DO-200 – Data
- DO-160 – Environmental
Aviation system/software assurance model

- ARP 4761 Safety Assessment Process
- ARP 4754A Aircraft & System Development Process
- DO-178C Software Assurance
- DO-254 Electronic Hardware
- IDAL (DO- objectives) [item level]
- FDAL (Appendix A) [functional level]
- DO-330
- DO-331
- DO-332
- DO-333

Relationships:
- Failure/criticality
- SW reqs
- HW reqs
1.1 Purpose

The rapid increase in the use of software in airborne systems and equipment used on aircraft and engines in the early 1980s resulted in a need for industry-accepted guidance for satisfying airworthiness requirements. (…)

This document, now revised [to DO-178C] in light of experience, provides the aviation community with guidance for determining, in a consistent manner and with an acceptable level of confidence, that the software aspects of airborne systems comply with airworthiness requirements.

As software use increases, technology evolves, and experience is gained in the application of this document, this document will be reviewed and revised.
6.1 Purpose of Software Verification  \textit{RTCA DO-178C pg. 39}

The purpose of the software verification process is to detect and report \textcolor{red}{errors that may have been introduced during the software development process}. Removal of the errors is an activity of the software development process. The software verification process verifies that:

a. The \textcolor{red}{system requirements} allocated to software have been developed into \textcolor{red}{high-level requirements} that satisfy those system requirements.

b. The \textcolor{red}{high-level requirements} have been developed into \textcolor{red}{software architecture and low-level requirements} that satisfy the high-level requirements. ...

c. The \textcolor{red}{software architecture and low-level requirements} have been developed into \textcolor{red}{Source Code} that satisfies the low-level requirements and software architecture.

d. The \textcolor{red}{Executable Object Code} satisfies the software requirements (that is, intended function), and provides confidence in the \textcolor{red}{absence of unintended functionality}.

e. The \textcolor{red}{Executable Object Code} is \textcolor{red}{robust} with respect to the software requirements such that \textcolor{red}{it can respond correctly to abnormal inputs and conditions}.

f. The \textcolor{red}{means used} to perform this verification are technically \textcolor{red}{correct and complete for the software level}. 
6.1 Purpose of Software Verification

The purpose of the software verification process is to detect and report errors that may have been introduced during the software development process. Removal of the errors is an activity of the software development process. The software verification process verifies that:

a. The system requirements allocated to software have been developed into high-level requirements that satisfy those system requirements.

b. The high-level requirements have been developed into software architecture and low-level requirements that satisfy the high-level requirements. ...

c. The software architecture and low-level requirements have been developed into Source Code that satisfies the low-level requirements and software architecture.

d. The Executable Object Code satisfies the software requirements (that is, intended function), and provides confidence in the absence of unintended functionality.

e. The Executable Object Code is robust with respect to the software requirements such that it can respond correctly to abnormal inputs and conditions.

f. The means used to perform this verification are technically correct and complete for the software level.
### DO-178C - example

<table>
<thead>
<tr>
<th>Objective</th>
<th>Activity</th>
<th>Applicability by Software Level</th>
<th>Output</th>
<th>Control Category by Software Level</th>
</tr>
</thead>
<tbody>
<tr>
<td>1. High-level requirements comply with system requirements.</td>
<td>6.3.1.a 6.3.1</td>
<td>● ● ○ ○</td>
<td>Software Verification Results</td>
<td>11.14 (2 2 2 2)</td>
</tr>
<tr>
<td>2. High-level requirements are accurate and consistent.</td>
<td>6.3.1.b 6.3.1</td>
<td>● ● ○ ○</td>
<td>Software Verification Results</td>
<td>11.14 (2 2 2 2)</td>
</tr>
<tr>
<td>3. High-level requirements are compatible with target computer.</td>
<td>6.3.1.c 6.3.1</td>
<td>○ ○</td>
<td>Software Verification Results</td>
<td>11.14 (2 2)</td>
</tr>
<tr>
<td>4. High-level requirements are verifiable.</td>
<td>6.3.1.d 6.3.1</td>
<td>○ ○ ○</td>
<td>Software Verification Results</td>
<td>11.14 (2 2 2)</td>
</tr>
<tr>
<td>5. High-level requirements conform to standards.</td>
<td>6.3.1.e 6.3.1</td>
<td>○ ○ ○</td>
<td>Software Verification Results</td>
<td>11.14 (2 2 2)</td>
</tr>
<tr>
<td>6. High-level requirements are traceable to system requirements.</td>
<td>6.3.1.f 6.3.1</td>
<td>○ ○ ○ ○</td>
<td>Software Verification Results</td>
<td>11.14 (2 2 2 2)</td>
</tr>
<tr>
<td>7. Algorithms are accurate.</td>
<td>6.3.1.g 6.3.1</td>
<td>● ● ○</td>
<td>Software Verification Results</td>
<td>11.14 (2 2 2)</td>
</tr>
</tbody>
</table>

*DO 178C Table A-3 (Verification of outputs of Software Requirements Process)*
6.3.1 Reviews and Analyses of High-Level Requirements

These review and analysis activities detect and report requirements errors that may have been introduced during the software requirements process. These review and analysis activities confirm that the high-level requirements satisfy these objectives:

a. **Compliance with system requirements:** The objective is to ensure that the system functions to be performed by the software are defined, that the functional, performance, and safety-related requirements of the system are satisfied by the high-level requirements, and that derived requirements and the reason for their existence are correctly defined.

f. **Traceability:** The objective is to ensure that the functional, performance, and safety-related requirements of the system that are allocated to software were developed into the high-level requirements.

g. **Algorithm aspects:** The objective is to ensure the accuracy and behavior of the proposed algorithms, especially in the area of discontinuities.
Figure 6.1 Software Testing Activities
The DO-178C “interface”

For the regulator:
• Start of project: **PSAC** – Plan for Software Aspects of Certification (identifies DAL), NAA LOI determined.
• End of project: **SAS** – Software Accomplishment Summary (out) and **SCI** – Software Configuration Index (akin to a MDL).

For the applicant:
• A large number of artifacts underpin the above – SDP (software development plan), SVP (software verification plan), SCM plan, SQA plan, SW Req Stds, SW Design Stds, SW Code Stds, SVR (software verification results), EOC, trace data, SQA records, …
• … and supporting processes – QA, CM, change control (CC), problem reports (PR)
CAST papers

Certification Authorities Software Team (CAST) position papers raise issues around emerging topics of significance.

- Topics typically covered by an (FAA) issue paper or (EASA) CRI (Certification Review Item).
- [https://www.faa.gov/aircraft/air_cert/design_approvals/air_software/cast/cast_papers/](https://www.faa.gov/aircraft/air_cert/design_approvals/air_software/cast/cast_papers/)

Recent notable CAST papers:

- CAST 32A – Multi-core processors (MCP) (note: this supersedes earlier CAST 32 paper)
  - Addresses sources of non-determinism in MCP

- CAST 29 – COTS graphical processors (CGPs) [strictly, this relates to AEH, not SW]
  - Addresses CGP “closed” hardware. Normal COTS part management practices not enough for CGP.
  - The consideration is that a common mode failure (of a single type of CGP) may render multiple redundant displays inoperative.
Supplements to DO-178C:

- DO-330 – Tool Qualification
- DO-331 – Model-based Development (MBD)
- DO-332 – Object Oriented Technology and Related Techniques (OOT)
- DO-333 – Formal Methods (FM)

Supplements work in conjunction with DO-178C by:

- Adding or modifying objectives
- Providing guidance on how to apply methods to fulfil objectives
Model Based Development (DO-331)

- Automated generation of code from **qualified tool**
- Example: Esterel (Ansys) **SCADE** (→ Safety-Critical App. Dev. Env.)
- Qualified to TQL-1 (under DO-330)

Example in use:
- A380 Cockpit displays (Thales)
- Similar to MATLAB “Simulink”
- Heavy use in Europe (particularly rail)

Image Credit: ANSYS SCADE
Object Oriented Technology (DO-332)

- Object-oriented technology (OOT) and ‘related techniques’
  - OOT → classes, inheritance
  - ‘related techniques’ → generics / templates (i.e. use of parametric polymorphism), exceptions, virtualization, dynamic memory management, overloading.

- **Only adds 2 objectives**: local type consistency (dynamic dispatch), dynamic memory. Remaining guidance relates to additional activities for existing DO-178C objectives.

- Much of the guidance in DO-332 may be relevant even when OOT is not being used in the project.
Formal Methods (DO-333)

• What? Proving the correctness of algorithms (and implementations) with respect to a formal specification.

• How? Using *formal proof* in mathematics.

• Two main approaches:
  • **Model checking** – create an abstract mathematical model of system and systematically check all states and transitions (e.g. TLA+, Alloy, Atelier B / B-method, Z, Petri Nets)

  • **Formal (deductive) verification** – generate and discharge proof obligations using interactive theorem provers (e.g. HOL, Isabelle, Coq, PVS) or SMT solvers.

• **Limitations** – Typically, formal methods do not “verify” the software per se. Formal methods verify the correctness of the safety and liveness properties specified for the software. (Exception is when *refinement* is used e.g. seL4)
Model checking example – TLA+

- Use of model checking (“lightweight FM”) can lead to useful detection of subtle errors in code.
- Experience with TLA+ at Amazon:

<table>
<thead>
<tr>
<th>System</th>
<th>Components</th>
<th>Line count (excl. comments)</th>
<th>Benefit</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Background redistribution of data</td>
<td>645 PlusCal</td>
<td>Found 1 bug, and found a bug in the first proposed fix.</td>
</tr>
<tr>
<td>DynamoDB</td>
<td>Replication &amp; group-membership system</td>
<td>939 TLA+</td>
<td>Found 3 bugs, some requiring traces of 35 steps</td>
</tr>
<tr>
<td>EBS</td>
<td>Volume management</td>
<td>102 PlusCal</td>
<td>Found 3 bugs.</td>
</tr>
<tr>
<td>Internal distributed lock manager</td>
<td>Lock-free data structure</td>
<td>223 PlusCal</td>
<td>Improved confidence. Failed to find a liveness bug as we did not check liveness.</td>
</tr>
<tr>
<td></td>
<td>Fault tolerant replication and reconfiguration algorithm</td>
<td>318 TLA+</td>
<td>Found 1 bug. Verified an aggressive optimization.</td>
</tr>
</tbody>
</table>

See also: https://github.com/tlaplus/Examples  http://lamport.azurewebsites.net/tla/formal-methods-amazon.pdf
**Formal verification example - SPARK**

- **SPARK** is a subset of the Ada programming language that is amenable to formal verification.
  - Commercially supported by AdaCore / Altran. Open source GPL version available.

- SPARK 2014 extends Ada 2012 with verification conditions (VCs) that may be discharged (proved) with a theorem prover.
  - Uses the “Why3” verification suite to provide a number of SMT solvers: Alt-Ergo, CVC3, YICES and Z3.

- Example in use:
  - 2011 – NATS iFACTS enroute air traffic control system. (529 KSLOC of SPARK, 152K VCs, 98.7% automatically proven by solvers, ~200 VCs discharged manually). (Note: this pre-dates SPARK 2014 and the latest solvers, so the numbers would be better today!)
Formal verification example - SPARK

```plaintext
pragma Spark_Mode (On);
package Sorters is
    type Array_Type is array ( Positive range <> ) of Integer;

    function Perm (A: in Array_Type; B: in Array_Type) return Boolean
      with Globals => null, Ghost => True, Import => True;

    procedure Selection_Sort ( Values : in out Array_Type)
      with Depends => ( Values => Values ),
      Pre => Values’Length >=1 and then Values’Last <= Positive’Last,
      Post => ( for all J in Values’First .. Values’Last-1 =>
        Values (J) <= Values (J+1) ) and then Perm (Values’Old, Values);
end Sorters;
```

Source: Building High Integrity Applications with SPARK, by McCormick & Chapin (2015)
There were six hours during the night of April 10, 2014, when the entire population of Washington state had no 911 service. People who called for help got a busy signal. One Seattle woman dialed 911 at least 37 times while a stranger was trying to break into her house. When he finally crawled into her living room through a window, she picked up a kitchen knife. The man fled.

4. Regulatory take-away

• Assuring the safety of software systems is HARD
• Certification is much more than a *paperwork* exercise
• What’s missing (particularly in terms of software and system requirements) can be *just as important* as what’s present!

• An important regulatory challenge is to make software certification **scalable**.

  • Have we been asking *too much* for small aircraft certification (→ e.g. see recent draft FAA policy for STC avionics in Pt 23 Level I/II aircraft – i.e. below 6 000 lb.)?
  • Are we asking *too little* for large aircraft certification (→ currently no *requirement* for model checking or formal verification, even at DAL A – only tests)?
A400M accident

- 3 of 4 engines experienced a “power freeze” after take-off. (No response to command changes in power). Resultant LOC-I.
- Fault traced to missing “torque calibration parameter data” (i.e. PDI) in FADEC
- Working scenario: accidental erasure of the PDI during software installation.
- No cockpit or other indication of missing data until after take-off.

- Presumably the FADEC software (& PDI) met the DO-178C objectives?
Discussion