New Challenges In Certification For Aircraft Software

John Rushby

Computer Science Laboratory
SRI International
Menlo Park CA USA

Overview

- The basics of aircraft certification
- The basics of aircraft software certification
- A theory of software assurance
- How well does aircraft certification work?
- And why does it work?
- How to improve it, and new challenges

Aircraft-Level Safety Requirements

- Aircraft failure conditions are classified in terms of the severity of their consequences
- Catastrophic failure conditions are those that could prevent continued safe flight and landing
- And so on through severe major, major, minor, to no effect
- Severity and probability/frequency must be inversely related
- AC 25.1309: No catastrophic failure conditions in the operational life of all aircraft of one type
- Arithmetic and regulation require the probability of catastrophic failure to be less than 10^{-9} per hour, sustained for many hours

Aircraft-Level Safety Analysis and Assurance

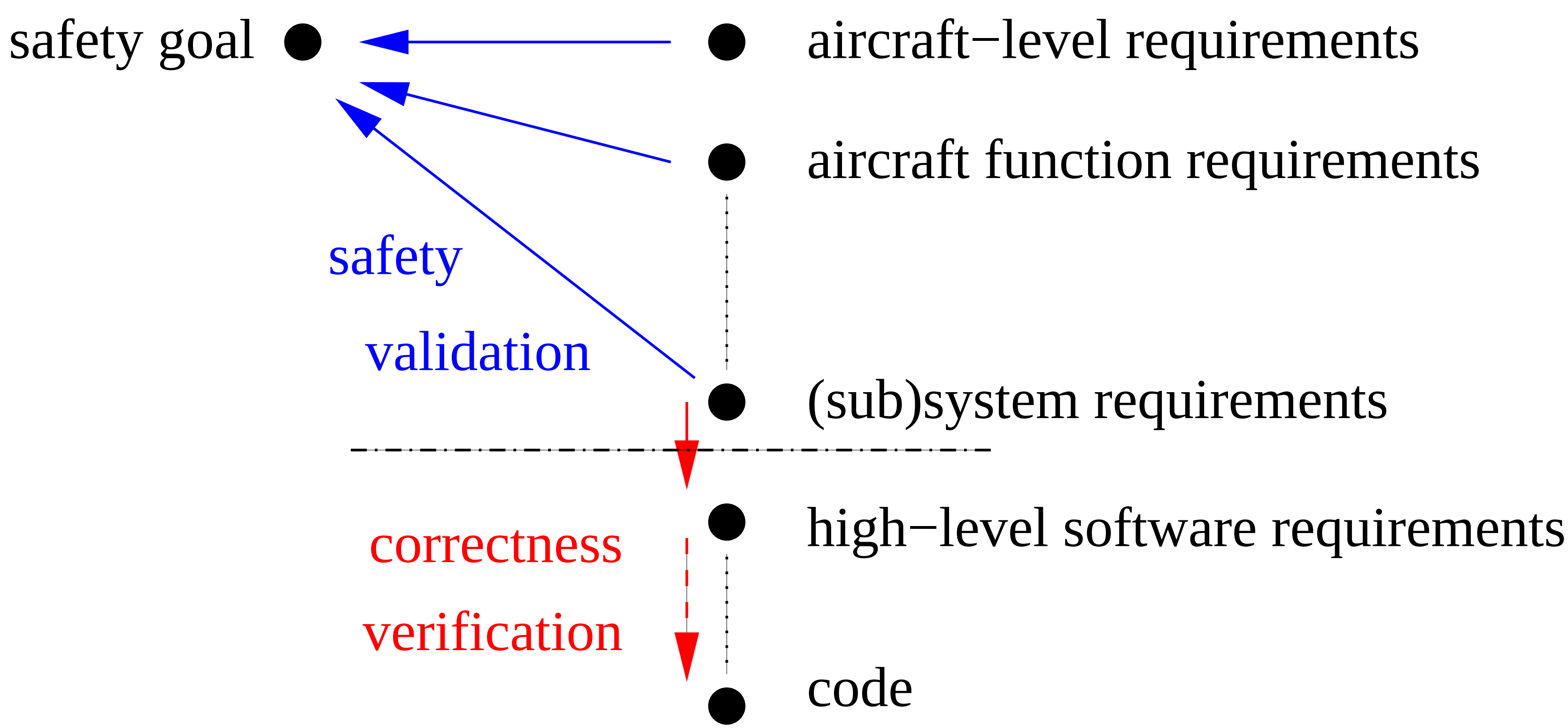
- This is spelled out in ARP 4761, ARP 4754A
- Basically, hazard analysis, hazard elimination and mitigation, applied iteratively and recursively through subsystems
- When we get to software components, must consider malfunction and unintended function as well as loss of function
- Assign design assurance levels (DALs) to software components: Level A corresponds to potential for catastrophic failures, through B, C, D, to E
- Can use architectural mitigation (e.g., monitors) to reduce DALs (e.g., instead of a Level A operational system, may be able to use a Level C system plus a Level A monitor)

Software Assurance

- Safety analysis recurses down through subsystems and components until you reach widgets
- For widgets, just build them right (i.e., correct wrt. specs)
- Software is a widget in this sense
- Hence, DO-178B is about correctness, not safety
- Safety analysis ends at the (sub)system requirements
- Show the high-level software requirements comply with and are traceable to system requirements, thereafter it's all about correct implementation (apart from derived requirements)

System vs. Software Assurance

- Safety analysis ends at the (sub)system requirements
- Thereafter it's all about correctness: DO-178B



DO-178B

- These are the current guidelines for airborne software
- DO-178B identifies 66 assurance objectives
  - E.g., documentation of requirements, traceability of requirements to code, test coverage, etc.)
- More objectives (plus independence) at higher DALs
  - 28 objectives at DO178B Level D (10^{-3})
  - 57 objectives at DO178B Level C (10^{-5})
  - 65 objectives at DO178B Level B (10^{-7})
  - 66 objectives at DO178B Level A (10^{-9})
- **The Conundrum**:

  What's the connection between the number of objectives
  - i.e., amount of correctness-focused V&V

  And probability of failure (or, dually, reliability)?

Software Reliability

- Software contributes to system failures through faults in its requirements, design, implementation—bugs
- A bug that leads to failure is certain to do so whenever it is encountered in similar circumstances
    - There's nothing probabilistic about it!
- Aaah, but the circumstances of the system are a stochastic process
- So there is a probability of encountering the circumstances that activate the bug
- Hence, probabilistic statements about software reliability or failure are perfectly reasonable
- Typically speak of probability of failure on demand (pfd), or failure rate (per hour, say)

Aleatoric and Epistemic Uncertainty

- Aleatoric or irreducible uncertainty
    - is “uncertainty in the world”
    - e.g., if I have a coin with $P(heads) = p_h$, I cannot predict exactly how many heads will occur in 100 trials because of randomness in the world

  Frequentist interpretation of probability needed here

- Epistemic or reducible uncertainty
    - is “uncertainty about the world”
    - e.g., if I give you the coin, you will not know p_h; you can estimate it, and can try to improve your estimate by doing experiments, learning something about its manufacture, the historical record of similar coins etc.

  Frequentist and subjective interpretations OK here

Aleatoric and Epistemic Uncertainty in Models

- In much scientific modeling, the aleatoric uncertainty is captured conditionally in a model with parameters
- And the epistemic uncertainty centers upon the values of these parameters
- As in the coin tossing example: p_h is the parameter

Measuring/Predicting Software Reliability

- For pfds down to about 10^{-4}, it is feasible to measure software reliability by statistically valid random testing
- But 10^{-9} would need 114,000 years on test
- So how do we establish that a piece of software is adequately reliable for a system that requires, say, 10^{-9}?
- Standards for system security or safety (e.g., Common Criteria, DO178B) require you to do a lot of V&V
- Which brings us back to The Conundrum

Aleatoric and Epistemic Uncertainty for Software

- The amount of correctness-based V&V does not relate to reliability in any obvious way
- Maybe it relates better to some other probabilistic property of the software's behavior
- Recap of the process:
    - We are interested in a property of s/w dynamic behavior
        - There is aleatoric uncertainty in this property due to variability in the circumstances of the software's operation
    - We examine static attributes of the software to form an epistemic estimate of the property
        - More examination refines the estimate
- For what kinds of properties could this work?

Perfect Software

- Property cannot be about some executions of the software
  - Like what proportion fail
  - Because the epistemic examination is static (i.e., global)
  - This is the disconnect with reliability
- Must be a property about all executions, like correctness
- But correctness is relative to specifications, which themselves may be flawed
- We want correctness relative to safety claims
  - Found in the system (not software) requirements
- Call that perfection
- Software that will never experience a safety failure in operation, no matter how much operational exposure it has

Possibly Perfect Software

- You might not believe a given piece of software is perfect
- But you might concede it has a possibility of being perfect
- And the more V&V it has had, the greater that possibility
- So we can speak of a (subjective) probability of perfection
- For a frequentist interpretation: think of all the software that might have been developed by comparable engineering processes to solve the same design problem
  - And that has had the same degree of V&V
  - The probability of perfection is then the probability that any software randomly selected from this class is perfect
- This idea is due to Bev Littlewood and Lorenzo Strigini

Probabilities of Perfection and Failure

- Probability of perfection relates to correctness-based V&V
- But it also relates to reliability:

  By the formula for total probability

$$
\begin{aligned}
& P(\text{s/w fails [on a randomly selected demand]}) \qquad (1)\\
= \; & P(\text{s/w fails} \mid \text{s/w perfect}) \times P(\text{s/w perfect})\\
& + P(\text{s/w fails} \mid \text{s/w imperfect}) \times P(\text{s/w imperfect}).
\end{aligned}
$$

- The first term in this sum is zero, because the software does not fail if it is perfect (other properties won't do)
- Hence, define
  - p_{np} probability the software is imperfect
  - p_{fnp} probability that it fails, if it is imperfect
- Then $P(\text{software fails}) \leq p_{fnp} \times p_{np}$
- This analysis is aleatoric, with parameters p_{fnp} and p_{np}

Epistemic Estimation

- To apply this result, we need to assess values for p_{fnp} and p_{np}
- These are most likely subjective probabilities
  - i.e., degrees of belief
- Beliefs about p_{fnp} and p_{np} may not be independent
- So will be represented by some joint distribution $F(p_{fnp}, p_{np})$
- Probability of software failure will be given by the Riemann-Stieltjes integral

$$\int\limits_{\substack{0 \le p_{fnp} \le 1 \\ 0 \le p_{np} \le 1}} p_{fnp} \times p_{np} \, dF(p_{fnp}, p_{np}). \qquad (2)$$

- If beliefs can be separated F factorizes as $F(p_{fnp}) \times F(p_{np})$
- And (2) becomes $P_{fnp} \times P_{np}$

  Where these are the means of the posterior distributions representing the assessor's beliefs about the two parameters

Practical Application—Nuclear

- Traditionally, UK nuclear protection systems are assured by statistically valid random testing
- Very expensive to get to pfd of 10^{-4} this way
- Our analysis says pfd $\leq P_{fnp} \times P_{np}$
- They are essentially setting P_{np} to 1 and doing the work to assess $P_{fnp} < 10^{-4}$
- Any V&V process that could give them $P_{np} < 1$
- Would reduce the amount of testing they need to do
  - e.g., $P_{np} < 10^{-1}$, which seems very plausible
  - Would deliver the same pfd with $P_{fnp} < 10^{-3}$
- This could reduce the total cost of assurance

Practical Application—Aircraft, Version 1

- No aircraft accidents due to software, and enough operational exposure to validate software failure rate $< 10^{-9}$
- Aircraft software is assured by V&V processes such as DO-178B Level A
- As well as DO-178B, they also do a massive amount of all-up testing but do not take assurance credit for this
- Littlewood and Povyakalo show (under independence assumption) that large number of failure-free runs shifts assessment from imperfect but reliable toward perfect
- Our analysis says software failure rate $\leq P_{fnp} \times P_{np}$
- So they are setting $P_{fnp} = 1$ and $P_{np} < 10^{-9}$
- So flight software might indeed have probabilities of imperfection $< 10^{-9}$
- And DO-178B delivers this

Practical Application—Aircraft, Version 2

- Although no accidents due to software, there have been several incidents
- So actual failure rate may be only around 10^{-7}
- Although they don't take credit for their all-up testing, this may be where a lot of the assurance is really coming from
- Our analysis says software failure rate $\leq P_{fnp} \times P_{np}$
- So perhaps testing is implicitly delivering, say, $P_{fnp} < 10^{-3}$
- And DO-178B is delivering only $P_{np} < 10^{-4}$
- I do not know which of Version 1 or 2 is true
- But there are provocative questions here

Aside: Dual and Monitored Systems

- Many safety-critical systems have two (or more) diverse "channels" arranged in 1-out-of-2 or primary/monitor architectures
- Cannot simply multiply the pfds of the two channels to get pfd for the system
    - Failures are unlikely to be independent
    - E.g., failure of one channel suggests this is a difficult case, so failure of the other is more likely
    - Infeasible to measure amount of dependence
- But the probability of imperfection of one channel is conditionally independent of the pfd of the other
    - So you can multiply these together to get system pfd

  See forthcoming IEEE TSE paper with Bev Littlewood

How Well Does DO-178B Work?

- There is one accident likely to be attributed to software
  - A330 in-flight upset near Learmonth, WA, 2008
  - Gust rejection in sensor fusion for angle of attack passed faulty values through
- And numerous incidents, some egregious
  - Fuel emergency on A340 near Amsterdam, 2005
  - Predator crash near Nogales, 2007
  - Threatened grounding of a widebody fleet
- Problems are always traced to flawed requirements
  - Compounded by unexpected interactions following the initial failure

Improving DO-178B

- It looks like the scrutiny of high level software requirements should be improved
- Beyond that, it is difficult to propose ways to improve DO-178B
- Because we do not know how well it works
  - cf. Versions 1 and 2 of my analysis
- Nor why it works
  - In the sense of what each objective "does"
  - In ways that would let us change or replace some of them
- We need a framework to help us understand this

Safety Cases

- All certification rests on a common intellectual basis
  - We have safety claims or goals we want to substantiate
  - We produce evidence about the product and its development process
  - And we have an argument that the evidence is sufficient to support the claims
- In a safety case, we have to produce all three parts
- In a standards-based approach, such as DO-178B, the claims and argument are implicit
  - They were presumably hashed out in the committee meetings that produced the standard

  And the standard/guidelines tell us what evidence to produce

Alternative Methods Of Compliance

- Can substitute an alternative method for an objective, provided it meets the "intent" of the objective
- The intent surely relates to the argument supported by the objective
- But these arguments are not documented
- Example: MC/DC testing
  - Tests generated from requirements must achieve a structural coverage criterion on the code called Modified Condition/Decision Coverage
  - Can we substitute some kind of formal analysis for this?
  - It depends on the intent of MC/DC

Intent of MC/DC

- Ensures reasonably thorough unit testing of the code
  - Valuable because we do not trust the compiler
- Because the tests are generated from requirements, code not covered by the tests indicates the presence of unintended functionality
- Because the tests are generated from the requirements, and must achieve rather demanding coverage of the internal program branching structure, it forces very detailed requirements

DO-178C

- A ten-year effort to update DO-178B
- Aircraft design evolves slowly, so do the system-oriented aspects of software (requirements etc.)
- But methods for software development and analysis change much more rapidly: DO-178C updates focused here
- DO-178C adds guidelines for model-based development and autocoding, object-oriented-languages, formal methods, etc.
- Required reverse-engineering the intent, and hence the argument, for many objectives
- But still does not document them
- Surely, it would be sensible and worthwhile to do so

New Challenges

- Size of software doubles every two years
  - Bug density is constant

  So failures may grow exponentially
- Furthermore, much of that additional code is integrating previously federated systems
  - Integration on board (IMA)
  - Integration with other aircraft and with air traffic management (NextGen)
  - "Integration" with flightcrew
    (shifting authority and autonomy)
- Federation provided natural barriers to fault propagation
  - Have to restore this by partitioning
- Integration may precipitate emergent misbehavior

New Challenges (2)

- The structure and practices of the industry are changing
  - Massive outsourcing, reduced oversight
  - DERs as consultants

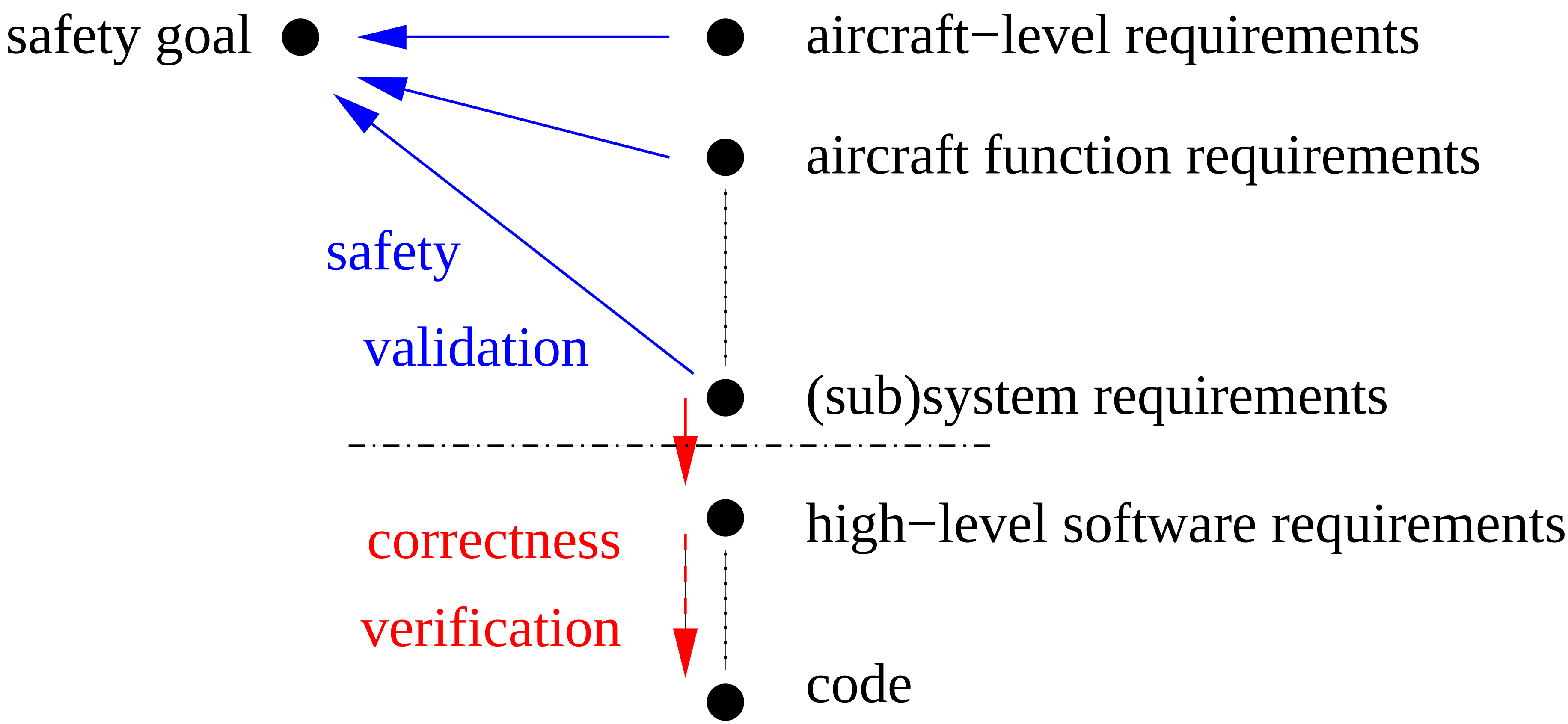
  May be losing the safety culture

  Can automation replace this?
- DO-178B costs a lot
  - This should be addressed by automation
  - Rather than by lobbying

Summary and Suggestions

- The standards-based approach (DO-178B/C) seems to work fairly well for aircraft
  - Possibly better than a safety case (cf. Nimrod)
- Updating and improving DO-178B/C (e.g., for increased automation) would be eased if the arguments supported by each objective were made explicit
- The main weakness seems to be in the transition from system to software processes
  - Perhaps safety analysis should be driven down to the high-level software requirements
  - Certainly they should be subjected to more analysis

Apply Safety Analysis To Software Requirements



Research Agenda

- Safety is not compositional
  - That's why the FAA certifies only aircraft and engines
- But it should be
  - So We need to better understand emergent misbehavior
  - And how to control it
- And we need to better understand software assurance
  - Probability of perfection explains how assurance works
  - But what values can we assess: 10^{-9}, 10^{-5}?

Research Agenda (2)

- What do the individual objectives of DO-178B accomplish?
- What can we do better?
- What is the relation between verification and safety argumentation?
- Closing thought:
  Let's develop certification as a research topic