Complete reworking of talks based on TSE 2012 paper with Bev Littlewood, plus new material from others at City
Explaining **Software Certification**

John Rushby

Based on work with/by Bishop, Littlewood, Povyakalo, Strigini at City University UK

Computer Science Laboratory
SRI International
Menlo Park CA USA
Introduction

- Software certification seems to work
  - At least for industries and systems where public data are available
    - e.g., passenger aircraft, trains, nuclear power
  - No major software-induced calamity
    - Maybe not so well for medical devices

- But how and why does it work?

- Worth knowing before we change things

- Or try to extend to other areas
  - e.g., cars, security
Certification Goals

• Usually some variation on “nothing really bad will happen”

• But the world is an uncertain place and this cannot be guaranteed, so we need to bound the exposure and add “with high probability”

• E.g., no catastrophic failure in the lifetime of all airplanes of one type
  • Or no release of radioactivity in 10,000 years of operation

• By arithmetic on these, we derive acceptable rates and probabilities for critical failures
  • E.g., for aircraft software, catastrophic failure rate < $10^{-9}$ per hour sustained for duration of flight
  • Or for nuclear shutdown pfd < $10^{-3}$
Certification Based on Experimental Quantification

• This means statistically valid random testing
• Need the operational profile
• It’s difficult and you need a lot of tests
• Can just about get to $10^{-3}$, maybe $10^{-4}$ this way
• Butler and Finelli calculated 114,000 years on test for $10^{-9}$
• Actually the Airbus A320 family has about $10^8$ hours of operation with no catastrophic software failure
• So, based on this alone, how much confidence can we have in another $10^8$ hours?
Certification Based on Experimental Quantification (ctd.)

Roughly speaking, if $p_f$ is probability of failure per demand (a complete flight, say), then we are interested in probability of $n$ demands without failure

$$p_{srv}(n) = (1 - p_{fnp})^n$$
Certification Based on Experimental Quantif’n (ctd. 2)

• So, based on this alone, how much confidence can we have in another $10^8$ hours?

  ◦ About 50-50
    ◦ We have $n = 10^8$ and no failures, from this estimate $p_f$ and extrapolate to $p_{srv}(2 \times 10^8)$.

• And for the remaining lifetime of the fleet (say $10^9$ hours)?
  ◦ Very little

• Need additional information—i.e., “priors”

• Aha! That’s what software assurance does for us—but how?

John Rushby, SRI
Maybe It’s Perfect

- Given $10^8$ hours of operation for the A320 family, the best we can say with no priors is that its catastrophic failure rate is probably no worse than $10^{-8}$

- That’s an extremely low rate

- It is almost easier to believe that it has no faults
  - i.e., is perfect
    Than that it has faults that occur at a rate below $10^{-8}$

- No amount of failure-free operation can confirm perfection

- Need some priors

- Aha! Maybe that’s how software assurance works
System Safety

• Think of **everything** that could go wrong
  - Those are the **hazards**

Design them out, find ways to mitigate them
  - i.e., reduce consequences, frequency

This may add complexity (a source of hazards)

• **Iterate** until you’ve dealt with **everything**

• And then **recurse** down through subsystems

• Until you get to **widgets**
  - Build those **correctly**

• Provide **assurance** that you have done **all** this successfully
Software Safety

- **Software is a widget** in this scheme
- **We don’t analyze it for safety, we build it correctly**
- **In more detail...**
  - Systems development yields functional and **safety requirements** on a subsystem that will be implemented in software; call these (sub)system safety requirements
    - Often expressed as **constraints** or **goals**
  - From these, develop the **high level software requirements**
    - **How to achieve** those goals
  - Elaborate through more detailed levels of requirements
  - Until you get to **code** (or something that **generates code**)
- **Provide assurance** that you have done **all** this successfully

John Rushby, SRI  Explaining Certification 9
Aside: Software is a Mighty Big Widget

The example of aircraft

- safety goal
- safety
- validation
- correctness
- verification
- code
- aircraft–level requirements
- aircraft function requirements
- (sub)system requirements
- high–level software requirements
- code

- As more of the system design goes into software
- Maybe the widget boundary should move
- Safety vs. correctness analysis would move with it
- But has not done so yet

John Rushby, SRI
The Conundrum

- Cannot eliminate hazards with certainty (because the environment is uncertain), so top-level claims about the system are stated quantitatively
  - E.g., no catastrophic failure in the lifetime of all airplanes of one type (“in the life of the fleet”)
- And these lead to probabilistic systems-level requirements for software-intensive subsystems
  - E.g., probability of failure in flight control $< 10^{-9}$ per hour
- To assure this, do lots of software assurance
- But this is all about showing correctness
- For stronger subsystem claims, do more software assurance
- How does amount of correctness-based software assurance relate to probability of failure?
The Conundrum Illustrated: The Example of Aircraft

- 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 conditions to be less than $10^{-9}$ per hour, sustained for many hours

- And $10^{-7}$, $10^{-5}$, $10^{-3}$ for the lesser failure conditions
The Conundrum Illustrated: Example of Aircraft (ctd.)

- DO-178C identifies five Software Levels
- And 71 assurance objectives
  - E.g., documentation of requirements, analysis, traceability from requirements to code, test coverage, etc.
- More objectives (plus independence) at higher levels
  - 26 objectives at DO178C Level D \( (10^{-3}) \)
  - 62 objectives at DO178C Level C \( (10^{-5}) \)
  - 69 objectives at DO178C Level B \( (10^{-7}) \)
  - 71 objectives at DO178C Level A \( (10^{-9}) \)
- The Conundrum: how does doing more correctness-based objectives relate to lower probability of failure?
Some Background and Terminology
Aleatory and Epistemic Uncertainty

- **Aleatory** or *irreducible* uncertainty
  - is “uncertainty in the world”
  - e.g., if I have a coin with $P(\text{heads}) = p_h$, 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
Aleatory and Epistemic Uncertainty in Models

• In much scientific modeling, the **aleatory** 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
Software Reliability

- Not just software, any artifacts of comparably complex design
- 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)
Testing and Software Reliability

- The basic way to determine the reliability of given software is by experiment
  - Statistically valid random testing
  - Tests must reproduce the operational profile
  - Requires a lot of tests

- This is where we came in

- Note that the testing in DO-178C is not of this kind
  - It’s coverage-based unit testing: a local correctness check

- So how can we estimate reliability for software?
Back To The Main Thread
Assurance is About Confidence

- We do correctness-based software assurance
- And do more of it when higher reliability is required
- But the amount of correctness-based software assurance has no obvious relation to reliability
- And it certainly doesn’t make the software “more correct”
- Aha! What it does is make us more confident in its correctness
- And we can measure that as a subjective probability
  - More assurance, higher probability of correctness, roughly...
- But that still doesn’t connect to reliability
- And is it really correctness that we want?
Correct but Imperfect Software: Example

• Fuel emergency on Airbus A340-642, G-VATL, on 8 February 2005 (AAIB SPECIAL Bulletin S1/2005)

• Toward the end of a flight from Hong Kong to London: two engines flamed out, crew found certain tanks were critically low on fuel, declared an emergency, landed at Amsterdam

• Two Fuel Control Monitoring Computers (FCMCs) on this type of airplane; each a self-checking pair with a backup (so 6-fold redundant in total); they cross-compare and the “healthiest” one drives the outputs to the data bus

• Both FCMCs had fault indications, and one of them was unable to drive the data bus

• Unfortunately, this one was judged the healthiest and was given control of the bus even though it could not exercise it

• The backups were suppressed because the FCMCs indicated they were not both failed
Perfect Software

- Correctness is relative to software requirements, which themselves may be flawed
  - Actually, the main source of failure in aircraft software

- We want correctness relative to the critical claims in the (sub)system requirements
  - Or what those claims should have been

- Call that perfection (aka. fault-freeness)

- Software that will never experience a critical 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 assurance 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 assurance
  - The probability of perfection is then the probability that any software randomly selected from this class is perfect
Probabilities of Perfection and Failure

- Probability of perfection relates to software assurance
- But it also relates to reliability:
  By the formula for total probability
  \[
  P(\text{s/w fails} \mid \text{on a randomly selected demand}) = P(\text{s/w fails} \mid \text{s/w perfect}) \times P(\text{s/w perfect}) \\
  + P(\text{s/w fails} \mid \text{s/w imperfect}) \times P(\text{s/w imperfect}).
  \]

- 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 \(P(\text{software fails}) = p_{fnp} \times p_{np}\)
- This analysis is aleatoric, with parameters \(p_{fnp}\) and \(p_{np}\)

John Rushby, SRI
Explaining Certification 24
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
  \[
  \int_{0 \leq p_{fnp} \leq 1} \int_{0 \leq p_{np} \leq 1} p_{fnp} \times p_{np} \, dF(p_{fnp}, p_{np}).
  \]
  \[ (2) \]
• 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, nuclear protection systems take no credit for the software assurance they do and base their certification on 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 \( P_{fnp} < 10^{-4} \).
  - Conservative assumption that allows separation of beliefs.

- Any software assurance 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 the same pfd with \( P_{fnp} < 10^{-3} \).
  - Conservative methods available if beliefs not independent.

- This could reduce the total cost of certification.
Practical Application—Aircraft, Version 1

- Aircraft software is assured by processes such as DO-178C Level A, needs failure rate $< 10^{-9}$ per hour
- They also do a massive amount of all-up testing but do not take (software) certification credit for this
- Our analysis says software failure rate $\leq P_{fnp} \times P_{np}$
- So they are setting $P_{fnp} = 1$ and $P_{np} < 10^{-9}$
- No plane crashes due to software, enough operational exposure to validate software failure rate $< 10^{-7}$, even $10^{-8}$
- Does this mean flight software has probabilities of imperfection $< 10^{-7}$ or $10^{-8}$?
- And that DO178C delivers this?
Practical Application—Aircraft, Version 2

• That seems unlikely!
  ◦ Implies that of 10,000,000 software systems assured to Level A, just 1 would ever suffer a critical failure

• An alternative measure is $p_{srv}(n)$, the probability of surviving $n$ demands without failure, where

$$p_{srv}(n) = (1 - p_{np}) + p_{np} \times (1 - p_{fnp})^n$$

• This doesn’t help with $10^{-9}$

• But can make $n$ equal to “life of the fleet” and get there with modest $p_{np}$ and $p_{fnp}$

• Really?
Long Run Failure-Free Operation

- Recall

\[ p_{srv}(n) = (1 - p_{np}) + p_{np} \times (1 - p_{fnp})^n \]

- **First term** is independent of \( n \)

- **Second term** decays exponentially

\[ \text{prob of survival} \]

\[ \text{failure if imperfect} \]

\[ \text{perfect} \]

\[ \text{time} \]

\[ \text{now} \quad \text{future} \]

John Rushby, SRI

Explaining Certification 29
How Software Certification Works

- Let’s suppose Level A gives us $p_{np} < 10^{-2}$
- Not unreasonable
- Then we have a 99% chance of no catastrophic failures in the life of the fleet
- So we’re done!
- Aah, but what about the other 1% chance?
  - Might crash every time
- So the term $10^{-2} \times (1 - p_{fnp})^n$ needs to be small
- Which is difficult when $n$ is huge ($\approx 10^9$)
  - Looks like we’re back to demonstrating $p_{fnp} < 10^{-7}$
- But on Day 1 do we really need the full lifetime of all airplanes of the type
  - In the first 6 months, say, we’ll only have 10 planes
  - 10 planes for 6 months is quite a small $n$ ($\approx 10^4$)
How Software Certification Works (ctd.)

- With a smaller $n$, modest $p_{fnp}$ can do it (e.g., $p_{fnp} < 10^{-3}$)
- Could get that from the all-up system and flight tests
- Provides the “bootstrap” to have confidence in first few months of flight
- Thereafter, experience to date validates smaller $p_{fnp}$ and provides confidence for next increment of exposure
  - See SafeComp13 paper by Strigini and Povyakalo for math
- I think this matches intuition
  - I’ve heard certifiers say they’ll wait a while before flying
Aside: Monitoring

- In some systems, it’s feasible to have a simple monitor that can shut off a more complex operational component
  - Turns malfunction and unintended function into loss of function
  - Prevents transitions into unsafe states
- It is a theorem that the possible perfection of the monitor is independent of the reliability of the operational channel
- Reliability of the whole is the product of these
  - At aleatoric level, more complex for epistemic
  - Must also deal with undesired monitor activation
- So formally synthesize monitor from formal safety constraints
  - Example: A340 fuel management
  
Maybe $p_{np} < 10^{-3}$ is plausible
Further Aside: Representing and Analyzing Requirements

- One reason requirements are often flawed is there are no good notations or methods of analysis for very abstract descriptions.
- Typically just boxes and arrows on a whiteboard and BOGSAT analysis (bunch of guys sitting around a table).
- So often use things like Simulink: premature implementation.
- Want the abstractness of boxes and arrows with just enough semantics that it is feasible to express constraints and analyze for interesting properties.
- Aha! Infinite Bounded Model Checking (Inf-BMC) can do this.
  - Inf-BMC allows use of uninterpreted functions, e.g., $f(x)$.
  - Constraints can be encoded as synchronous observers.
  - Inf-BMC can do automated model checking (using SMT solvers) and cover the entire modeled space.

John Rushby, SRI
Explaining Certification 33
Summary

- I think this really is the explanation for certification
- It dissolves the conundrum
  - Connects assurance effort to confidence: $p_{np}$
  - And connects confidence to reliability
  - And long term survival
- It shows how testing/experience buttresses that
  - Without requiring extreme numbers
  - Because it works incrementally
- Observe this explanation works for lifetime of the fleet
  - US regulation
- But not for $10^{-9}$
  - EU regulation (just illustrative in US)
- What do you think?
Future Work

• We need to get good estimates for the \( p_{np} \) delivered by DO-178C Level A, B, C
  ○ Maybe “chunk” the 71 objectives
  ○ Michael Holloway’s retrospective safety case does this
  ○ Get expert opinions on each chunk
  ○ And use BBNs to combine them

• Also need estimates on \( p_{fnp} \) delivered by flight tests, first 6 months, first 2 years etc.
  ○ Recall, it is about \( 10^{-8} \) for A320 after 20 years

• Derive advice for new fields, certification regimes, assurance methods?