

What Is Software Assurance?

John Rushby

Based on joint work with Bev Littlewood (City University UK)

Computer Science Laboratory

SRI International

Menlo Park CA USA

A Conundrum

- Critical systems are those where **failures** can have unacceptable consequences: typically safety or security
- Cannot eliminate failures 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** requirements for software-intensive subsystems
 - E.g., **probability of failure in flight control $< 10^{-9}$ per hour**
- To assure this, do lots of **verification and validation (V&V)**
- But V&V is all about showing **correctness**
- And for **stronger claims**, we do **more V&V**
- So how does **amount** of V&V relate to **probability** of failure?

Background

The Basis For Assurance and Certification

- We have **claims** or **goals** that we want to substantiate
 - Typically claims about a critical property such as **security** or **safety**
 - Or some functional property, or a combinationE.g., no catastrophic failure condition in the life of the fleet
- We produce **evidence** about the **product** and its development **process** to support the claims
 - E.g., analysis and testing of the product and its design
 - And documentation for the process of its development
- And we have an **argument** that the evidence is **sufficient** to support the claims
- Surely, this is the intellectual basis for **all** certification regimes

Standards-Based Certification vs. Safety Cases

- Applicant follows a set **process**, delivers prescribed **outputs**
 - e.g., documented requirements, designs, analyses, testsThese provide **evidence**; **goals** and **argument** largely **implicit**
- Common Criteria (security), DO-178B (civil aircraft) do this
- **Works well in fields that are stable or change slowly**
 - Can institutionalize lessons learned, best practice
- **May be less suitable with novel problems, solutions, methods**
- Alternative is a **safety case**: applicant
 - Makes an **explicit** set of **goals** or **claims**
 - Provides supporting **evidence** for the claims
 - And **arguments** that **link the evidence to the claims**
- The case is evaluated by independent assessors
- The main novelty is the **explicit argument**
- Generalized to security, dependability, assurance cases

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)

Aleatory and Epistemic Uncertainty

- Aleatory 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

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

Back To The Main Thread

Measuring/Predicting Software Reliability

- For pfd's 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^{-6} ?
- Standards for system security or safety (e.g., Common Criteria, DO178B) **require you to do a lot of V&V**
 - e.g., **57** V&V “objectives” at DO178B **Level C** (10^{-5})
- And you have to do more for higher levels
 - **65** objectives at DO178B **Level B** (10^{-7})
 - **66** objectives at DO178B **Level A** (10^{-9})
- What's the connection between amount of V&V (mostly focused on **correctness**) and degree of software **reliability**?

Aleatory and Epistemic Uncertainty for Software

- The amount of correctness-based V&V relates poorly to reliability
- Maybe it relates better to **some other** probabilistic property of the software's behavior
- We are interested in a property of its **dynamic** behavior
 - There is aleatoric uncertainty in this property due to variability in the circumstances of the software's operation
- We examine the **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 how many 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 the critical claims**
 - Taken directly from the system's **assurance case**
- Call that **perfection**
- **Software that will never experience a 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**

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

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

- 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

$$\int_{\substack{0 \leq p_{fnp} \leq 1 \\ 0 \leq p_{np} \leq 1}} p_{fnp} \times p_{np} dF(p_{fnp}, p_{np}). \quad (2)$$

- 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 are assured by statistically valid random testing
- Very expensive to get to pfd of 10^{-4} this way
- Our analysis says $\text{pfd} \leq P_{fnp} \times P_{np}$
- 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 the same pfd with $P_{fnp} < 10^{-3}$
- This could reduce the total cost of assurance

Practical Application—Aircraft, Version 1

- No plane crashes 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
- They do a massive amount of all-up testing but do not take assurance credit for this
- Our analysis says software failure rate $\leq P_{fnp} \times P_{np}$
- So they are setting $P_{fnp} = 1$ and $P_{np} < 10^{-9}$
- Littlewood and Povyakalo show (under independence assumption) that large number of failure-free runs shifts assessment from imperfect but reliable toward perfect
- So flight software might indeed have probabilities of imperfection $< 10^{-9}$
- And DO-178B delivers this

Practical Application—Aircraft, Version 2

- Although no crashes 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 all the testing they do, 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: Two Channel Systems

- Many safety-critical systems have two (or more) diverse “channels” arranged in 1-out-of-2 (1oo2) structure
 - E.g., nuclear shutdown
- A primary protection system is responsible for plant safety
- A simpler secondary channel provides a **backup**
- **Cannot** simply multiply the pfd's 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

So, traditionally, difficult to assess the reliability delivered

Two Channel Systems and Possible Perfection

- But if the second channel is simple enough to support a plausible claim of possible perfection
 - Its imperfection is conditionally independent of failures in the first channel at the aleatory level
 - Hence, system pfd is conservatively bounded by product of pfd of first channel and probability of imperfection of the second
 - $P(\text{system fails on randomly selected demand}) \leq pfd_A \times pnp_B$

This is a theorem

- Epistemic assessment similar to previous case
 - But may be more difficult to separate beliefs
 - Conservative approximations are available

Type 1 and Type 2 Failures in 1oo2 Systems

- So far, considered only failures of omission
 - Type 1 failure: both channels fail to respond to a demand
- Must also consider failures of commission
 - Type 2 failure: either channel responds to a nondemand
- Demands are events at a point in time; nondemands are absence of demands over an interval of time
- So full model must unify these
- Details straightforward but lengthy

Monitored Architectures

- One **operational** channel does the business
- Simpler **monitor** channel can shut it down if things look bad
- Used in airplanes
- Analysis is a variant of 1oo2:
 - No Type 2 failures for operational channel
- Monitored architecture **risk** per unit time
$$\leq c_1 \times (M_1 + F_A \times P_{B1}) + c_2 \times (M_2 + F_{B2|np} \times P_{B2})$$
where the M s are due to mechanism shared between channels
- May provide justification for some of the architectures suggested in ARP 4754
 - e.g., 10^{-9} system made of Level C operational channel and Level A monitor

Monitors Do Fail

- Fuel emergency on [Airbus A340-642](#), G-VATL, 8 February 2005
 - Type 1 failure
- EFIS Reboot during spin recovery on [Airbus A300](#) (American Airlines Flight 903), 12 May 1997
 - Type 2 failure
- Current proposals are for **formally synthesized/verified monitors** for **properties in the safety case**

Back To The Main Thread

Application to Formal Verification

- We know DO-178B “works”
 - 10^{-9} by Version 1, or 10^{-4} by Version 2
- But it’s expensive
- Formal verification can be cheaper
 - Yes it can!
- But is often burdened by belief that it must support a claim of **absolute correctness** and must therefore itself be **infallible**
 - Leads to inappropriate allocation of resources or choice of techniques (e.g., no decision procedures)
- We now know it needs to support a claim of **possible perfection**
- So let’s see where that goes

Formal Verification and the Probability of Perfection

- We want to assess P_{np} for something like a monitor
- Context is an **assurance case** in which **claims** about a system are justified by an **argument** based on **evidence** about the system and its development
- **Suppose part of the evidence is formal verification**
- What is the probability of perfection of formally verified software?
- Surely a function of the ways in which formal verification can fail
 - i.e., the **hazards** to formal verification
- So let's enumerate these and look for techniques that can provide assurance those hazards are eliminated

The Basic Requirements For The Software Are Wrong

- This error is made before any formalization
- It seems to be the **dominant** source of errors in flight software
- But monitoring and backup software are built to requirements taken directly from the safety case
 - If these are wrong, we have **big** problems
- **So this concern belongs at a higher level**

The Requirements etc. are Formalized Incorrectly

- Could also be the **assumptions**, or the **design** that are formalized incorrectly
- Formalization may be **inconsistent**
 - i.e., meaningless

Can be **eliminated** using constructive specifications

- In a tool-supported framework
- That guarantees **conservative extension**

But that's not always appropriate

- Prefer to state assumptions as **axioms**
- Consistency can then be guaranteed by exhibiting a constructive model (interpretation)
- PVS can do this
- So we can **eliminate** concern about inconsistency

The Requirements etc. are Formalized Incorrectly (ctd.)

- Formalization may be consistent, but **wrong**
- Formal specifications that have not been subjected to analysis are no more likely to be correct than programs that have never been run
 - In fact, less so: engineers have better intuitions about programs than specifications
- Should **challenge** formal specifications
 - Prove putative theorems
 - Get counterexamples for deliberately false conjectures
 - Directly execute them on test cases
- **Social process operates on widely used theories**
- **In my experience, incorrect formalization is the chief source of errors in formal verification**
 - There are papers on errors in my specifications

The Requirements etc. are Formalized Incorrectly (ctd. 2)

- Even if a theory or specification is formalized incorrectly, it does not necessarily invalidate all theorems that use it
- Only if the verification actually exploits the incorrectness will the validity of the theorem be in doubt
 - Even then, it could still be true, but unproven
- Some verification systems identify all the axioms and definitions on which a formally verified conclusion depends
 - PVS does this

If these are correct, then logical validity of the verified conclusion follows by soundness of the verification system

- Can apply special scrutiny to them
- So concern about incorrect formalization can be managed

The Formal Specification and Verification is Discontinuous or Incomplete

- **Discontinuities** arise when several analysis tools are applied in the same specification
 - e.g., static analyzer, model checker, timing analyzerConcern is that different tools ascribe different semantics
- Increasing issue as specialized tools outstrip monolithic ones
 - Need integrating frameworks such as a tool bus
- Most significant **incompleteness** is generally the gap between the most detailed model and the real thing
 - Algorithms vs. code, libraries, OS callsThat's one reason why we still need testing
 - Driven from the formal specification
 - Cf. penetration tests for security: probe the assumptions
- **Concerns about incompleteness need to be managed**

Unsoundness In the Verification System

- All verification systems have had soundness bugs
- But none have been exploited to prove a false theorem
- Many efforts to guarantee soundness are costly
 - e.g., reduction to elementary steps, proof objects
 - What does soundness matter if you cannot do the proof?
- A better approach is KOT: the Kernel Of Truth (Shankar)
 - A ladder of increasingly powerful verified checkers
 - Untrusted prover leaves a trail, blessed by verified checker
 - More powerful checkers guaranteed by one-time check of its verification by the one below
 - The more powerful the verified checker, the more economical the trail can be (little more than hints)
- So concern about unsoundness can be reduced

Example

- Suppose we can get $P_{fnp} < 10^{-3}$ by testing, want P_{np} of 10^{-3}
 - So system will then be $< 10^{-6}$
- Through sufficiently careful and comprehensive formal challenges, it is plausible an assessor can assign a subjective posterior probability of imperfection on the order of 10^{-3} to the **formal statements on which a formal verification depends**
- Through testing and other scrutiny, a similar figure can be assigned to the probability of imperfection due to **discontinuities and incompleteness in the formal analysis**
- By use of a verification system with a trusted or verified kernel, or trusted, verified, or diverse checkers, assessor can assign probability of 10^{-4} or smaller that the theorem prover **incorrectly verified the theorems that attest to perfection**
- We're done!

Discussion

- These numbers are **feasible** and **plausible**
 - **Really?** Why 10^{-3} and not 10^{-2} or 10^{-4} ?
 - Need to develop basis for numerical estimates
 - If you believe my analysis, historical record suggests DO-178B Level A does justify very strong estimates
- Formal methods and their tools do not need to be held to (much) higher standards than the systems they assure
- Remember Fetzer's jeremiad?
- This is the first analysis that supports a measured response

Conclusion

- **Probability of perfection** is a radical and valuable idea
 - It's due to Bev Littlewood, and Lorenzo Strigini
- **Provides the bridge between correctness-based verification activities and probabilistic claims needed at the system level**
- Relieves formal verification, and its tools, of the burden of infallibility
 - Allows rational allocations of resources to hazards
- Could help in rebalancing the assurance activities at higher EALs of the Common Criteria
- Likely to work well in an assurance case framework
- Explains what software assurance **is**