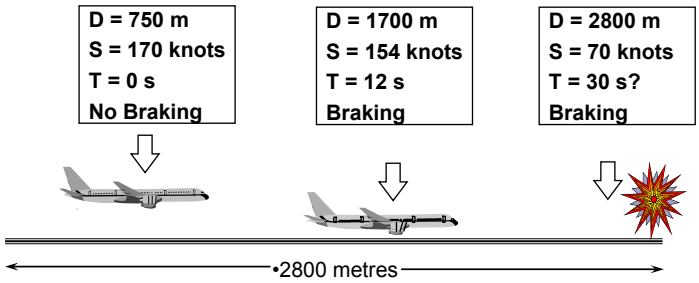
Fundamental Principles of Software Safety Assurance

Tim Kelly

THE UNIVERSITY of York

Example

- LH2904 Okecie, Warsaw, Sept. 14th 1993
  (A320 Warsaw — approximate analysis)



THE UNIVERSITY of York

Hazards & Accidents

- Hazard:
  - System (the airframe) - travelling down runway at circa 170 knots, without braking
- Environment:
  - weather - strong winds, veered from cross to tail winds in final approach; raining heavily
  - runway - standing water
  - ATC - didn't inform pilots of shift in wind direction
  - plane at 170 knots, tail-wind not cross-wind, standing water so aquaplaning, landed long, earth wall ...
  - may not have been credible - NB judgemental
- Hazard + Environment ➔ Accident
  - aircraft hit earth wall near end of runway (and ensuing fire)
  - 2 lives lost, 54 injured

Hazards & Failures

Hazards are caused by failures

- *failures* - unintended condition of system or structure (which can lead to a hazard), eg uncommanded movement (turning) of nosewheel by BSCU

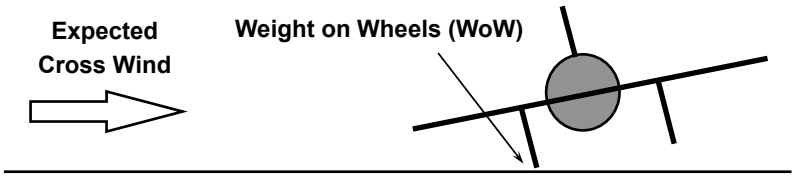
- For the hardware elements – random failures
  - e.g. "Will the bulb in the warning lamp blow?"
- For the software (& hardware) elements …
  - Software will never fail randomly
  - It will only 'do the wrong thing' if that is how we designed it!
  - These are systematic failures (w.r.t. *intent*)

So What Happened?

A320 Warsaw - Platform

Airframe - “on ground” at T=0

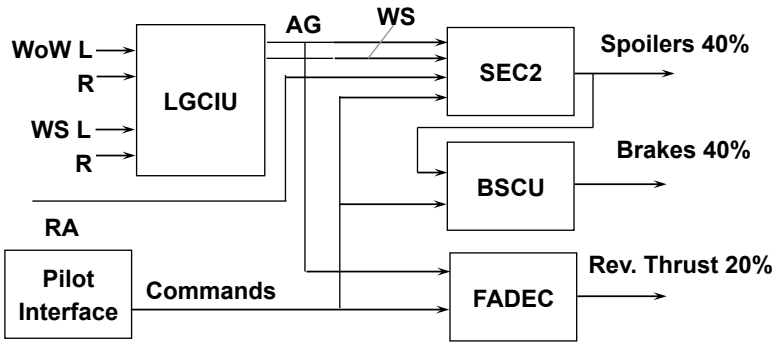
- one main landing gear compressed, other not; aircraft banked due to expected cross-wind



THE UNIVERSITY of York

Systems Controlling Ground Deceleration

A320 Warsaw — Systems (approx.)



THE UNIVERSITY of York

Systems Controlling Ground Deceleration

Landing Gear Control & Interface Unit (LGCIU)

- landing gear extension, retraction, etc.
- synthesises AG (Air/Ground Transition) & WS
- AG = WoW>12 tonnes (both LG)
- WS = Wheel Spinning > 72 knots (either LG)

Spoiler Elevator Computer Secondary (SEC2)

- deploys spoilers, etc

Full Authority Digital Engine Controller (FADEC)

- controls engine, & deploys reverse thrusters

Some in-built 'safeguards'

Brakes and Steering Control Unit (BSCU)

- nosewheel steering, all braking and ABS

Logic - distributed amongst systems

- apply reverse thrusters - AG true
- apply air and wheel brakes -
  WS true or (RA true (radio alt <10 feet)
  and AG true)

Basic logic - full system more complex

- pilot sets braking level, which affects timing ...

Everything functioned as designed

A320 Warsaw
- Systems Conditions and Failures

- AG (weight on both wheels) = False
- WS (wheels spinning > 72 knots) = False
- Alt (less than 10 feet) = True
- major systems — LGCIU, SEC2, BSCU
  — all functioned to specification
- no braking - air brakes, reverse thrusters or wheel brakes

Correctness != Safety

Logic was implemented *correctly* (to spec.)

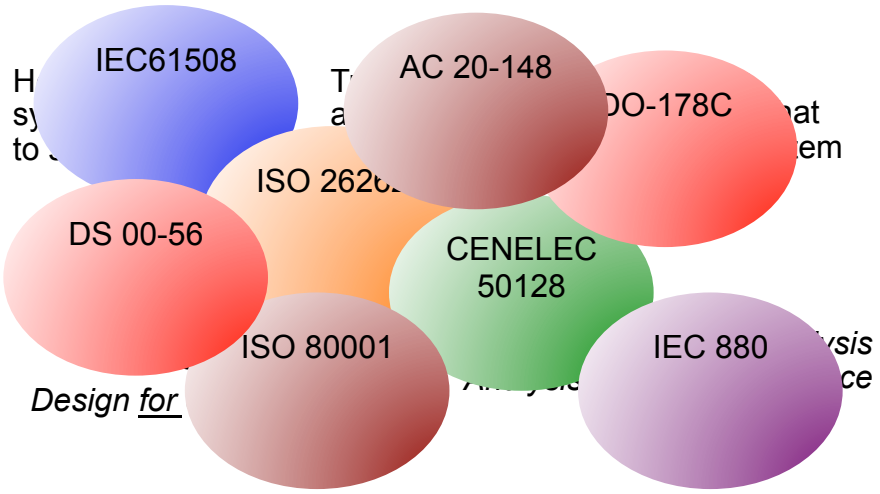
- Logic was undesirable in the situation
- LGCIU (and related systems) — failure — against intent

Operators

- pilot — possible failure — no manual override (but some courses of action prohibited)

Overall cause - *questionable* design
- in the circumstances

What can we do about it?



THE UNIVERSITY of York

There is hope …

- The 4+1 Principles
  - Why +1?
- Examples from past systems
- Methods and means of achieving the principles
- Potential Pitfalls

THE UNIVERSITY of York

Software safety requirements shall be defined to address the software contribution to system hazards

PRINCIPLE 1

THE UNIVERSITY of York

Principle 1

- The identification and management of (specific) risks is fundamental to system safety
- This is no different when considering software
- Many causes of system-level hazards
  - Mechanical
  - Human
  - Environmental
  - …
  - Software
- Need to ensure that we have identified, understood and captured the potential contribution of software to system level hazards

THE UNIVERSITY of York

Methods / Means

- Combination of deductive and inductive safety and hazard analysis techniques desirable
- Deductive (from the system 'top-down')
  - e.g. constructing system level fault trees that appropriately acknowledge the contribution of the Software Intensive System
  - … to the black box of the software outputs
  - Still may combine random and systematic causes
- Inductive (from the software to the system 'bottom-up')
  - e.g. Functional Hazard Analysis applied to the Functional Requirements Specification of the software system
- Both have problems
  - Combination is useful

Methods / Means 2

- Can be hard to determine 'hazardous' behaviour of S/W
- When manual techniques start to falter …
  - Simulation
  - Executable Prototypes
  - Model checking at the 'System Level'
- Ultimately this is the Requirements Validity issue
- Terminology:
  - Software 'Hazards'
    - Really a shorthand
  - Hazardous Software Failure Modes
    - Some don't like saying 'failure' for software
  - Hazardous Software Contributions
    - Non-judgmental (re: failure)

Pitfalls

- Not being specific about the contributions
  - E.g. Failure Event: “Control System Fails”
- Assuming software safety is simply about correctness
  - Remember Verification and Validation distinction

THE UNIVERSITY of York

Example

- Medical Device Domain historically without strict software safety assurance regime
- 2010, US FDA ordered Baxter Healthcare Corp to recall Colleague Volumetric Infusion Pumps
- Safety problems associated with these infusions pumps (amongst other models of infusion pumps)
  - FDA traced the sources of some of these safety problems to software defects.
- Since 2010, the FDA has placed increased attention to embedded software, particularly within its new guidance document that can be used for preparing premarket notification submissions for infusion pumps

THE UNIVERSITY of York

The intent of the software safety requirements shall be maintained throughout requirements decomposition

PRINCIPLE 2

THE UNIVERSITY of York

Principle 2

- Typical software development lifecycle: Progression from more abstract requirements to concrete implementation
- Necessarily requirements must be refined, decomposed, allocated, interpreted
- There's more …
  - … design commitment
  - … information
  - … defined behaviour
- … in the lower level requirements
- With regard to safety this could go well, or not …

THE UNIVERSITY of York

Principle 2

- Following principle 1, we believe the higher level requirement is OK
- Is the intent of the higher level requirement maintained in the lower level requirements?
- Notion of "*Intent*" important
  - What we *want* from / *meant* by the requirement
  - Covers implied semantics
  - (Unfortunately) a lot can remain unstated / deliberately undefined, even quantification
- Don't just think of requirements ➔requirements
  - Requirements ➔Verification Properties
  - Requirement ➔Test cases

THE UNIVERSITY of York

Methods / Means

- Requirements / design traceability
- Requirements review
  - Creativity applied to the 'have we got this right'? question
- Formal refinement
- Better expression of requirements
  - Clear semantics
  - Controlled expression
- 'Rich traceability'
  - Consideration of rationale for decomposition

THE UNIVERSITY of York

Pitfalls

- Believing traceability data is sufficient
- Need to be confident about the trace / mapping
- Not (creatively) reconsidering intent in requirements / design decomposition
  - Worrying about completeness
  - Identifying and resolving requirements ambiguity and conflict
- Allowing ambiguous requirements
- Using techniques with ambiguous semantics

Example

- Airbus A320-200 Lufthansa Flight 2904
- High Level Requirement (Paraphrased):
  - “Do not deploy Ground Deceleration unless on the ground”
- High Level Requirement is a “Good idea”
- Unfortunately, the good idea has to be implemented
  - The world has to be sensed, and cannot be known perfectly
  - Weight on Wheels (WoW), Wheel Speed (WS) and Radio Altimeter (RA)
- Air Ground Transition = True if WoW > 12 tonnes on both Landing Gear
- Wheels Spinning if WG > 72 knots on either Landing Gear

Software safety requirements shall be satisfied

PRINCIPLE 3

THE UNIVERSITY of York

Principle 3

- The (most) obvious one?
- Does the system actually do what we said it ought to do (as stated in the safety requirements)?
- Variety of means of achievement possible
  - Will discuss evidence selection later
- Consequence of earlier principles
  - Want specific evidence for specific safety requirements
- This is the Verification issue

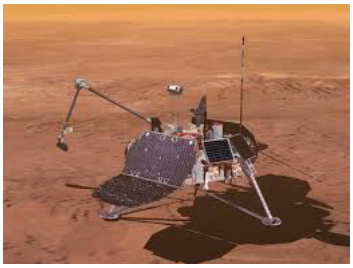
THE UNIVERSITY of York

Pitfalls

- Losing the scent of the trace from requirements to verification evidence
- Not compelling simply to say that there's a bucket of evidence and the answer's in there somewhere
- The final link from requirement to satisfaction shares the challenges of requirements and design decomposition discussed in Principle 2
  - E.g. a passed (relevant) test case does not mean the requirement is convincingly satisfied

Example

- Loss of NASA's Mars Polar Lander (MPL), Jan 1999
- Inadequate testing a contributory factor
  - (alongside inadequate requirements specification)
- Software error leading to the premature shutdown of the decent engines has been considered a probable cause of the loss of the lander
- Software fault-injection testing regime was considered inadequate to stress test the flight software, especially testing for transient surface touchdown sensor signals
- Test environment was deemed insufficient to detect flaws in the touchdown sensing software



Hazardous behaviour of the software has been identified and mitigated

PRINCIPLE 4

Principle 4

- Sister principle to Principle 2
- Principle 2 concerned about maintaining the intent of our safety requirements, in the presence of increasing design commitment
- Principle 4 also concerned with the consequence of increasing design commitment
- Rather than “Does it do what we required”? (Princ. 2)
- Now “Does it do anything else that is unsafe”?
  - i.e. Hazardous side-effects

Principle 4

- Hazardous software behaviours could result from:
  - unanticipated behaviours and interactions arising from software design decisions
    - e.g. preventing ground deceleration when required in the earlier A320-200 example
    - Concerned with where design is unsafe (under some conditions)
  - systematic errors introduced during the software development process
    - E.g. Coding errors, compilation errors, code-generation errors, modelling errors
    - (Specific) causality doesn't have to be proven to know that there are some errors to be avoided

Methods / Means

- Regarding systematic errors in the development process:
  - Trusted tools / Tool Qualification
  - Checking output of untrusted tools
  - Implementation guidelines & checkers
    - Modeling guidelines
    - Coding rules
  - Use models, languages etc. that don't allow mistakes so easily!
- Regarding safety implications of design decisions
  - (Similar to methods and means of Principle 2)
  - Requirements reviews
  - Design reviews walkthroughs
  - Architectural evaluation
  - Software HAZOPs
  - Common Theme: Reconsideration of the behaviour of the design

Example

Boeing 777-200 Perth to Kuala Lumpur flight in 2005

- Experienced a number of serious alerts, spurious indications and dangerous auto-pilot activity
- Pilot managed to return aircraft safely to Perth only by disengagement of auto-pilot, manually overriding warnings and automatic commands, and reliance on ATC
- Initiating event for this incident occurred four years previously when one of the aircraft accelerometers failed such that it provided erroneously high output
- S/W in the aircraft's Air Data Inertial Reference Unit (ADIRU), in accordance with its specification, disregarded the erroneous accelerometer and instead relied upon data from a back-up accelerometer

Example

- When back-up accelerometer also failed ADIRU software reverted to taking input from the accelerometer that had initially failed
- ADIRU software had been designed such that when it was shut down and restarted the accelerometer was no longer recognised as faulty and was assumed available for use should it be required

The confidence established in addressing the software safety principles shall be commensurate to the contribution of the software to system risk

PRINCIPLE 4+1

THE UNIVERSITY of York

Principle 4+1

- Why 4+1, not 5?
  - Because this principle cross-cuts the implementation of the other principles
- Perfect assurance of the achievement of the other principles is desirable, but unachievable
  - e.g. consider Principle 1, we cannot *prove* that the safety requirements are complete
  - Not even if "money no object"
- Instead, we must consider when is enough enough?
- Really a system principle
- Some challenges applying to software

THE UNIVERSITY of York

Methods / Means

- Safety Integrity Levels (SIL)
- Development Assurance Levels (DAL)
- Association between SILs / DALs and Software Assurance Techniques / Processes / Practice
- Software Criticality Assessment
  - E.g. Software Hazard Risk Indices
- Consideration of 'Assurance Deficits'
- Establishing a Risk vs. Confidence Split in the Software Safety Case

Pitfalls

- Treating all software assurance the same way
  - Can be accused of 'gold plating'
- Failing to acknowledge the confidence issue
  - e.g. "We've reviewed, therefore it's perfect"
- Hard to implement …
  - without good understanding of the system context and system level risks
  - without a means of describing software criticality (more later)
  - without understanding of effectiveness of software assurance techniques

Example

- Boeing 777 Fly-By-Wire Flight Control System (FCS)
- Central element of the FCS is the Primary Flight Computer (PFC)
- 777 FCS provides the single source of control of the aircraft in the pitch, roll and yaw axes. Should there be a failure of the PFC, then control of the aircraft could be lost, with no further mitigations available
- The PFC function clearly makes a very large contribution to risk for the aircraft, and as such was determined to require the highest level of assurance (DAL A)
  - Architecture and V&V Requirements Set Accordingly

Example 2

- DUST-EXPERT advisory system developed by Adelard
- Advises on the safe design and operation of plant that is subject to dust explosions
- Fault tree analysis conducted using data of reported explosions, the ratio of injuries to fatalities, etc.
- Concluded that DUST-EXPERT—as a potential contributing factor should be developed to SIL 2, assuming:
  - failure of explosion relief vents led to 1 death/100 000 workers/ year,
  - vent is at SIL 1

Summary of the Principles

1. Software safety requirements shall be defined to address the software contribution to system hazards
2. The intent of the software safety requirements shall be maintained throughout requirements decomposition
3. Software safety requirements shall be satisfied
4. Hazardous behaviour of the software has been identified and mitigated

4+1. The confidence established in addressing the software safety principles shall be commensurate to the contribution of the software to system risk

DO-178C

Principle 1

- Assumed starting point in DO-178B/C is that behavioural safety requirements allocated to software have already been derived by system level safety analysis performed in accordance with ARP 4754A
  - ARP4754A addresses the problem of validation of these requirements
  - ARP 4754A also defines the process for judging the criticality of the contribution of software to system level hazards and expresses this as an allocated software DAL

DO-178C

Principle 2

- strong emphasis on maintaining traceability through the stages of software development
- recognises problem of validation of decomposition, e.g. through requirements for review
- simply recording traceability information is necessary for but insufficient
  - need justification (cf. Rich Traceability)

THE UNIVERSITY of York

DO-178C

Principle 3

- well addressed - verification evidence that addresses the demonstration of requirements both under normal conditions and fault conditions
  - DO-178C admits a wider range of verification techniques

THE UNIVERSITY of York

DO-178C

Principle 4

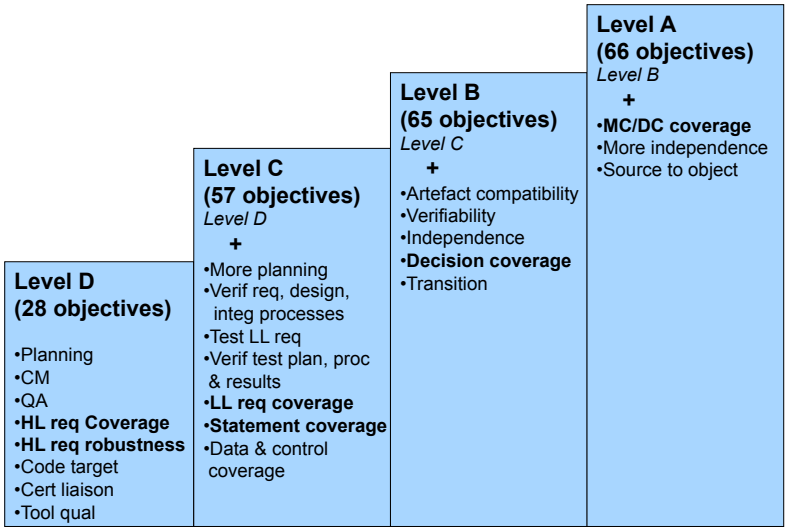
- recognises that 'Software design process activities could introduce possible modes of failure into the software or, conversely, preclude others' and 'In such cases, additional data should be defined as derived requirements and provided to the system safety assessment process'.
- Removal of errors leading to unacceptable failure conditions as an objective of testing
- Acknowledges that 'The effects of derived requirements on safety related requirements are determined by the system safety assessment process'.
- However, …

DO-178C

Principle 4+1

- captured through the mechanism of DALs that tailor requirement for the demonstration of the objectives of the standard according to criticality

Assurance vs. Objectives



THE UNIVERSITY of York

Observations

- P1-3 can be observed to be at the heart of many standards
- P4 is less well addressed
  - However, both discuss the potential for systematic error introduction within the software development lifecycle
- Many standards attempt to address P4+1 through SILs / DALs
  - differences in allocation and what is varied
  - lack of a significant evidence-base that demonstrates that either approach to varying confidence can be easily correlated with achieved risk reduction

THE UNIVERSITY of York

Generic vs. Specific Application of Principles

- intent of principles is not that they are addressed generically (e.g. by appeal to generic processes or adherence to standards)
- should be evidenced specifically
- requirements and processes of a standard may be capable of demonstrating principles, but may still fall short in practice
  - consider Requirements Review
- application of standards cannot be considered in a tokenistic sense, as a talisman of confidence
- An area where confidence can be lost, also where assurance cases can help

THE UNIVERSITY of York

Generic vs. Specific Application of Principles

- Significant issue re: P4+1
  - Standards established a general set of requirements for varying requirements, processes and techniques according to an abstract level of required confidence
  - Generality is potentially a problem
  - Is it what's required in a specific case - e.g. applicability of MCDC metrics?
  - Opportunity cost of doing something that doesn't add to confidence
- Some mechanisms to address:
  - PSAC, SAS in DO-178C, Justification of selection from amongst 'loose' SIL recommendations in IEC 61508

THE UNIVERSITY of York

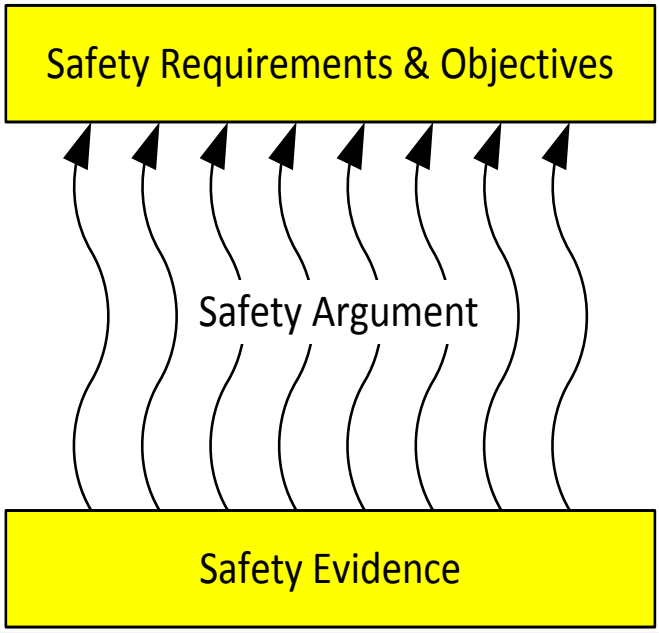
Safety (Assurance) Cases

- The purpose of a safety case can be defined in the following terms:

A safety case should communicate a clear, comprehensive and defensible argument (supported by evidence) that a system is acceptably safe to operate in a particular context

- This extends to software (in a system context)

THE UNIVERSITY of York

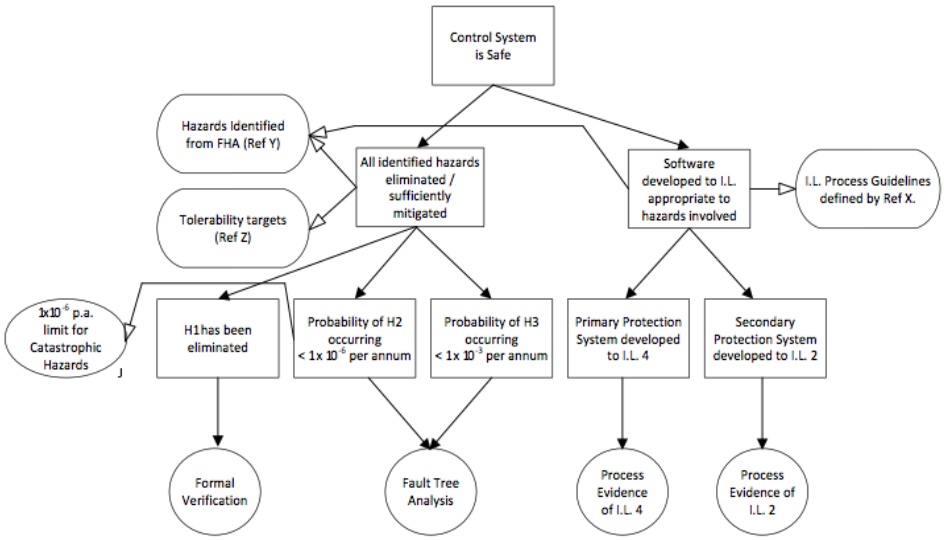


THE UNIVERSITY of York

Arguments

- Historically, narrative text has been commonly used
  - But problems …
- Structured Argumentation Approaches
  - GSN - Goal Structuring Notation
  - GSN clearly disambiguates the structure and elements of the argument, it cannot ensure that the argument itself is 'good' or sufficient for its purpose
- "Informal Logic"

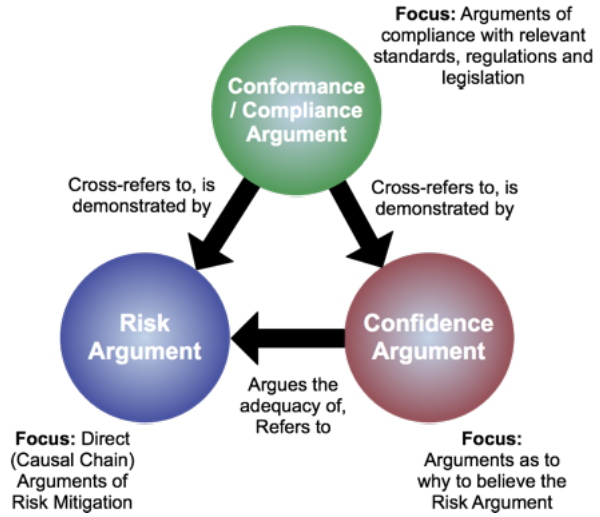
GSN Example



Three types of argument

- (Causal) Behavioural arguments of **risk** management, i.e. how the causes of hazards are eliminated or mitigated, or how the consequences of hazards are mitigated
  - Principle 1, 2, 3, 4
- Confidence arguments – arguments that provide confidence in the adequacy of the details of the risk management argument, e.g. justifying the adequacy of hazard identification techniques, or the sufficiency of verification results presented
  - Principle 4+1
- Arguments of conformance / compliance with safety standards, regulations, and legislation – where compliance is not straightforward it is necessary to justify how a project, system design and operation have addressed legal and regulatory obligations

Risk – Confidence – Compliance



Targeting Assurance Case Effort

- Many standards provide the template for the technical risk argument needed at the core of any software assurance case (forming central pillar of response to P1,2 & 3)
- Standards also provide general requirements and recommendations for the avoidance of potentially hazardous errors and anomalous behaviour (P4)
- Standards also provide general guidance on how effort should be tailored according to risk (P4+1)
- Confidence can be lost in the (lack of) justification of the specific instantiation of these template structures and general guidance
  - Assurance Cases can help here!

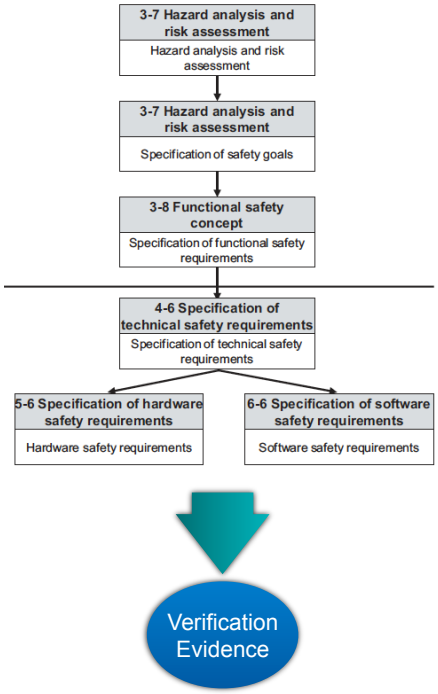
THE UNIVERSITY of York

Targeting Assurance Case Effort

- P1 - assurance cases are well suited to the (inevitably subjective) justification of the adequacy of the identified software safety requirements
- P2 - well suited to the hard problem of the justification of maintenance of intent in traceability structures
- P3 - well suited to the justification of the adequacy of evidence (e.g. the appropriateness and trustworthiness of specific forms of evidence for requirements satisfaction)
- P4 - usefully targeted at the justification of the management of unintentionally hazardous side effects of otherwise intentional design commitments
- P4+1 - directly relates to the notion of a confidence / meta argument

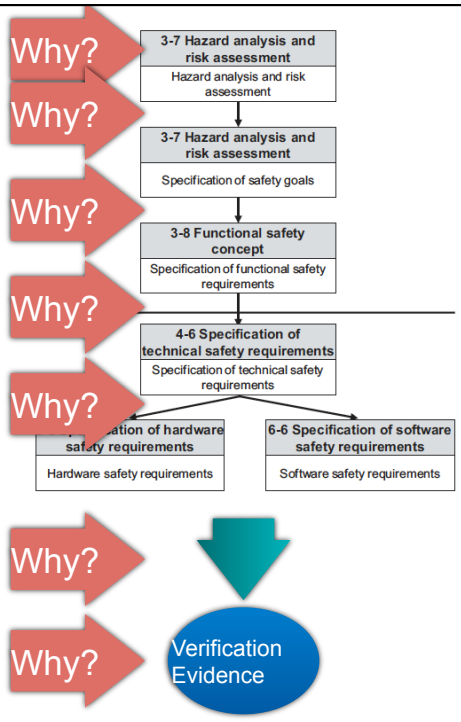
THE UNIVERSITY of York

What's the 'core' risk argument with a ISO 26262 project?



But, where's the confidence argument?

- Having a 26262 compliant 'structure' isn't enough
- Safety is 'won and lost' in the specific details of …
- safety goals, functional safety requirements, means of testing



Some Considerations

- Is formalisation a natural next step?
- Questions / Challenges:
  - How does formalisation of arguments address current needs in safety case practice?
  - Are all types of safety case argument equally amenable to formalisation?
  - Does the subject matter of a safety case argument affect the value of formalisation?

Problems of Current Practice

- Nimrod: Focus, Outsourcing, Scale
  - Specifically: lacked evidence -> validity
- Confirmation Bias (one of Leveson's criticisms)
- Formalisation?
  - Help with well-formedness, scale
  - Problems of validity and veracity remain
  - Reasoning about truth of assertions relies upon establishing a connection between formal models of the subject domain as well as formal models of the argument
    - Otherwise propositions are uninterpreted propositions

Problems (ctd.)

- Damer's five categories of argument fallacy:
  (1) structural flaw in the argument;
  (2) irrelevance of a premise;
  (3) unacceptability of a premise;
  (4) insufficiency of the combined premises of an argument to establish its conclusion; and
  (5) failure to give an effective rebuttal.
- Annotation with argumentation schemes (Walton) help consistency checking against prior knowledge of acceptable structure
  - Suggests author knows of the scheme (but has ignored it!)

Are all types of safety case argument equally amenable to formalisation?

- formalisation often involves axiomatising (informal) aspects of the argument at the 'edge' of our argument
  - e.g. 'all hazards identified' argument
  - Of course, could structure this further
    - Kicking the can down the road?
    - Further set of axioms covering the informal aspects of the formalised argument

Are all types of safety case argument equally amenable to formalisation?

- Valuable service has been performed by 'annexing' the informal arguments to an easily identified location (a form of reductionism)?
- Concern: illusion of formality created through hiding problematic informal and subjective arguments behind an abstraction
- formalised ‘core’ with informality pushed to the periphery of the formalisation is advantageous or dangerous for evaluation and review?
- formalisation will not reduce perhaps the most significant aspect of the review burden – namely individual review and acceptance of subjective (informal) assertion

THE UNIVERSITY of York

Does the subject matter of a safety case argument affect the value of formalisation?

- deductive arguments can form part of a safety case
  - when subject matter domain is itself logical
  - asserted inferences can become provable inferences
  - When safety case arguments (or at least portions of them can become provable) are they perhaps not better represented as evidence (i.e. proof), rather than as informal logic?
- value of a safety case is to represent the informal logical ‘glue’ that pulls together different forms of the evidence (including deductive results – proof being one such example)

THE UNIVERSITY of York

What are the tradeoffs between accessibility and precision in safety arguments?

- value of safety cases lies in facilitating accessibility of safety arguments amongst key stakeholders
  - Informal argumentation notations designed with this position in mind
- value of safety cases lies in removing ambiguity and improving precision in the description of safety arguments
  - Formalisation helps
- Positions can be reinforcing, but may in be conflict
  - e.g. precision at the expense of comprehensibility

Summary

- Principles underlie many (all?) software safety standards
  - P1-3 served well, P4 & 4+1 not so well
- Assurance Cases complement standards
- Standards suffer from problems relating to specific enactment and judgement
  - Standards can't remove (subjective) judgement
  - Assurance cases are good at explicitly representing and recording judgements
- But discipline is required:
  - Structured Argumentation
  - Clear separation of risk, confidence and compliance arguments
- Combining formal and informal arguments is part of the challenge