Assurance Cases, Evidence and Patterns

Tim Kelly

E-mail: tim.kelly@cs.york.ac.uk

High Integrity Systems Engineering Group
Department of Computer Science
THE UNIVERSITY of York

Safety Integrity

- Bell and Reinert Definition:

> ***"The likelihood of a safety related system satisfactorily performing the required [intended] safety functions under all the stated conditions within a stated period of time"***

- Random and **Systematic** Elements of this

Integrity Requirements

- Many safety standards require a quantitative approach to defining safety integrity requirements

> ***"Quantitative safety integrity requirements should be defined for safety related complex electronic elements"***
>
> (UK DefStan 00-56)

- Higher importance of the behaviour to system safety ➔ more stringent integrity requirements
- Different ways to express: MTTF, Probability of Failure Free Operation, PFD

Software Integrity Requirements

- ***Allocation*** of Integrity Requirements not difficult – e.g. see IEC 61508
- ***Quantisation*** of Integrity Requirements as ***Levels*** not difficult
- Following of rules *prescribed* for a given integrity level not difficult
- Arguing the ***achievement*** of integrity *is* difficult
  - Poor correlation between practices and achieved integrity
  - Problems of direct measurement

The Switch

- Difficulty is recognised (ignored?) by some standards

> ***"Development of software to a software [development assurance] level does not imply the assignment of a failure rate for that software. Thus, software levels or software reliability rates based on software levels cannot be used by the system safety assessment process as can hardware failure rates."***
>
> (DO178B)

- Demonstrating compliance with the standard ➔ sufficiently ***assured*** that the software is implemented correctly (against the requirements) and/or safely
  - **Integrity** ➔ **confidence** (the switch has taken place!)

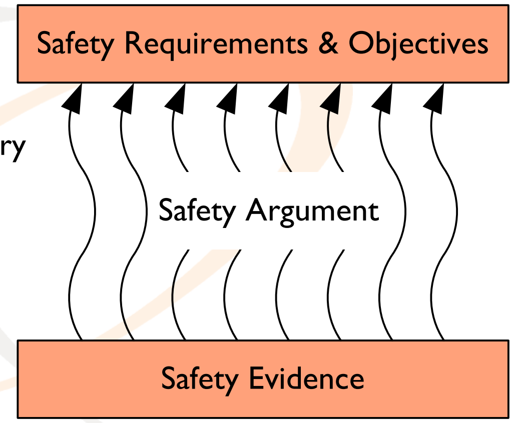
Uncertainty

- Integrity ⬄ ***aleatoric*** uncertainty
  - Having an element of *chance, randomness*
- Confidence ⬄ ***epistemic*** uncertainty
  - characterised by the *limitations of knowledge*
- Must justify that the epistemic uncertainty is *commensurate* with the aleatoric uncertainty

> ***"The Safety Case shall contain a structured argument demonstrating that the evidence contained therein is <u>sufficient</u> to show that the system is safe. The argument shall be commensurate with the potential risk posed by the system …" (UK DefStan 00-56)***

Gaining Assurance

- Assurance is the level of confidence which can be ***justified***
- To gain assurance – necessary to identify evidence that can (directly) demonstrate the achievement of specific software safety properties
- Structured, ***product-based,*** safety arguments provide a means of demonstration



The Purpose of a Safety Case

- **A safety case presents the argument that a system will be acceptably safe in a given context**
- 'System' could be …
  - Physical (e.g. aero-engines, reactor protection systems)
  - Procedural (e.g. railway operations, off-shore)
  - **Software!**
- Increasingly adopted in the defence (UK), automotive, rail, oil and gas, process, medical device domains
  - Including military aerospace (Eurofighter Typhoon, parts of Joint Strike Fighter, C130J Hercules)

Argument & Evidence

- ***Supporting Evidence***
  Results of observing, analysing, testing, simulating and estimating the properties of a system that provide the *fundamental* information from which safety can be inferred
- ***High Level Argument***
  Explanation of how the available evidence can be reasonably interpreted as indicating acceptable safety – usually by demonstrating compliance with requirements, sufficient mitigation / avoidance of hazards etc
- Argument without Evidence is **unfounded**
- Evidence without Argument is **unexplained**

The Structure of an Argument

- An **argument** reasons from *premises* to a *conclusion*
- **Proposition** = a statement which
  - **(a) must be either true or false, and**
  - **(b) cannot be *both* true and false**
- **"The sky is blue"** is a valid proposition
- An argument is a collection of propositions, one of which is the conclusion, the others being premises for that conclusion

- **If it is a Public Holiday, then it will rain**
- **Today is a Public Holiday**

} ***Premises***

- **It will rain today**

} ***Conclusion***

The Goal Structuring Notation (GSN)

Purpose of a Goal Structure

To show how **goals** □ are broken down into sub-goals,

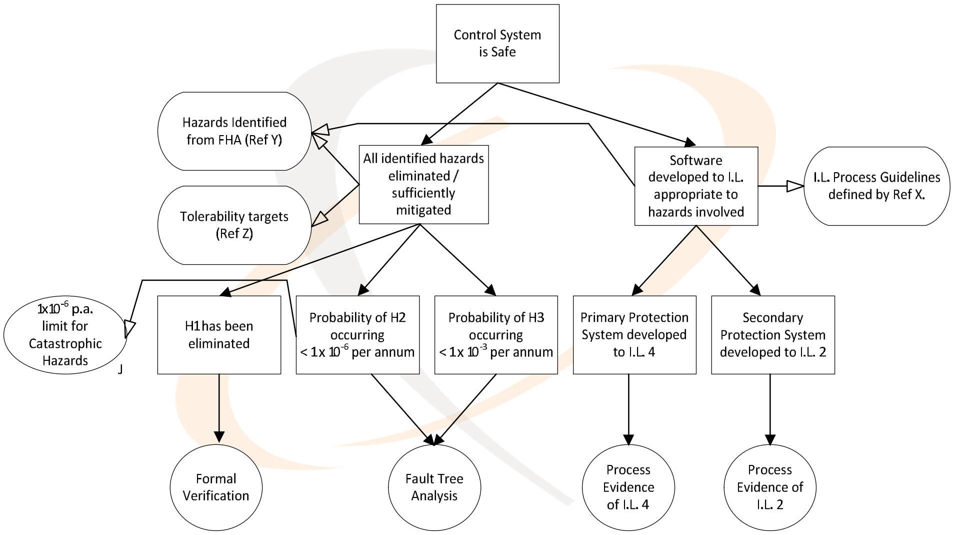
and eventually supported by evidence (**solutions**) ○

whilst making clear the **strategies** ▱ adopted,

the rationale for the approach (**assumptions**, **justifications**) ⬭ A/J

and the context ▢ in which goals are stated

A Simple Goal Structure



Control System is Safe

Hazards Identified from FHA (Ref Y)

Tolerability targets (Ref Z)

All identified hazards eliminated / sufficiently mitigated

Software developed to I.L. appropriate to hazards involved

I.L. Process Guidelines defined by Ref X.

$1x10^{-6}$ p.a. limit for Catastrophic Hazards

J

H1 has been eliminated

Probability of H2 occurring $< 1x\ 10^{-6}$ per annum

Probability of H3 occurring $< 1x\ 10^{-3}$ per annum

Primary Protection System developed to I.L. 4

Secondary Protection System developed to I.L. 2

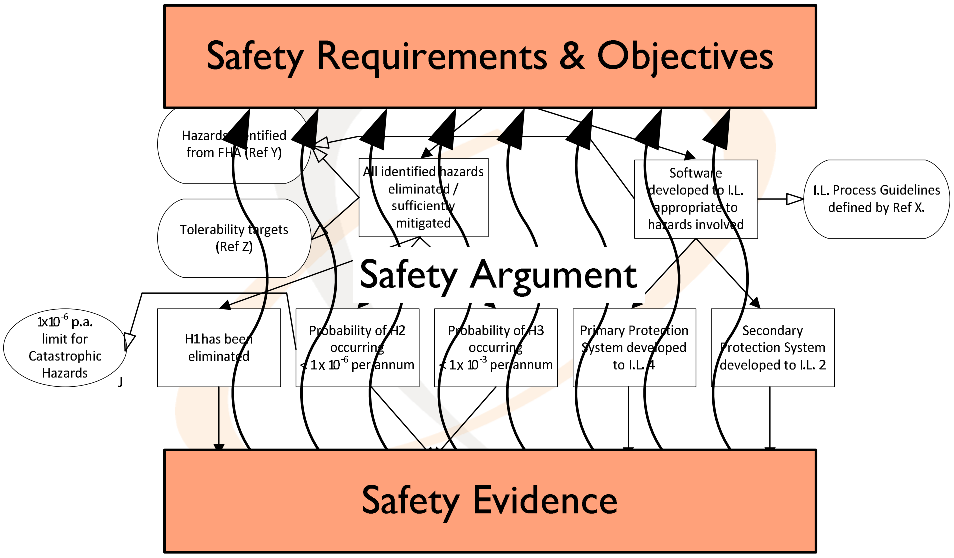
Formal Verification

Fault Tree Analysis

Process Evidence of I.L. 4

Process Evidence of I.L. 2

A Simple Goal Structure



Safety Requirements & Objectives

Hazards Identified from FHA (Ref Y)

Tolerability targets (Ref Z)

All identified hazards eliminated / sufficiently mitigated

Software developed to I.L. appropriate to hazards involved

I.L. Process Guidelines defined by Ref X.

Safety Argument

1x10^{-6} p.a. limit for Catastrophic Hazards

H1 has been eliminated

Probability of H2 occurring < 1x 10^{-6} per annum

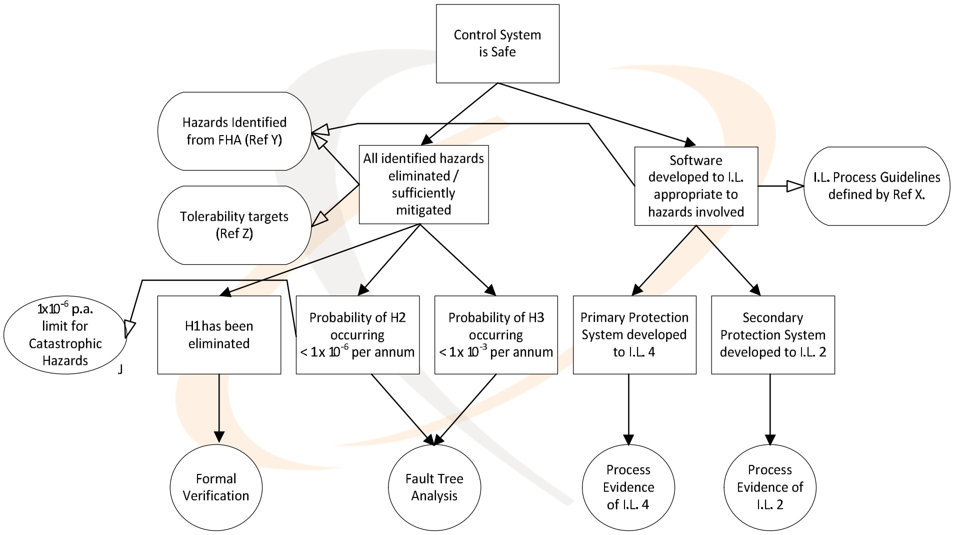
Probability of H3 occurring < 1x 10^{-3} per annum

Primary Protection System developed to I.L. 4

Secondary Protection System developed to I.L. 2

Safety Evidence

A Simple Goal Structure



Control System is Safe

Hazards Identified from FHA (Ref Y)

Tolerability targets (Ref Z)

All identified hazards eliminated / sufficiently mitigated

Software developed to I.L. appropriate to hazards involved

I.L. Process Guidelines defined by Ref X.

1x10^{-6} p.a. limit for Catastrophic Hazards

H1 has been eliminated

Probability of H2 occurring < 1x 10^{-6} per annum

Probability of H3 occurring < 1x 10^{-3} per annum

Primary Protection System developed to I.L. 4

Secondary Protection System developed to I.L. 2

Formal Verification

Fault Tree Analysis

Process Evidence of I.L. 4

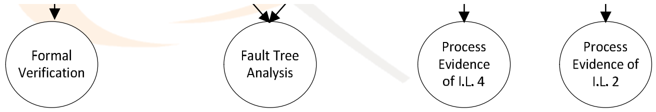
Process Evidence of I.L. 2

A Simple Goal Structure

Q: Take away the argument and what are you left with?

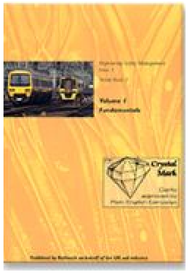
A: The BAG of Evidence

Note – This is not a safety case!

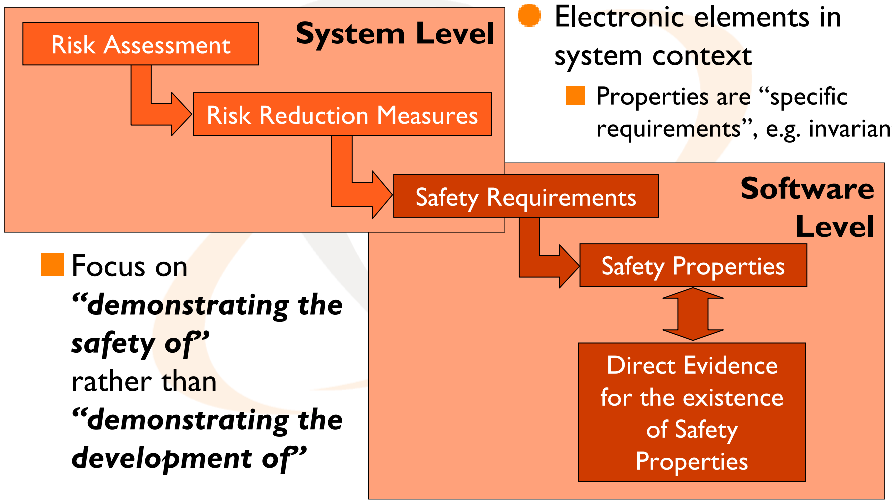


Structured Argumentation

- Safety Cases now used in a large number of domains
  - Chemical Industry, Off-shore Oil & Gas, Railways, Nuclear, Automotive, Aerospace, Defence, Air-Traffic Management, Medical Devices
- Structured Approaches to Safety Argumentation (e.g. using GSN) used widely, e.g.
  - Yellow Book (http://www.yellowbook-rail.org.uk/)
  - Eurocontrol
  - OMG Initiative
  - ISO 15026
  - DHS SWA



Intended System-Software Safety Case Relationship



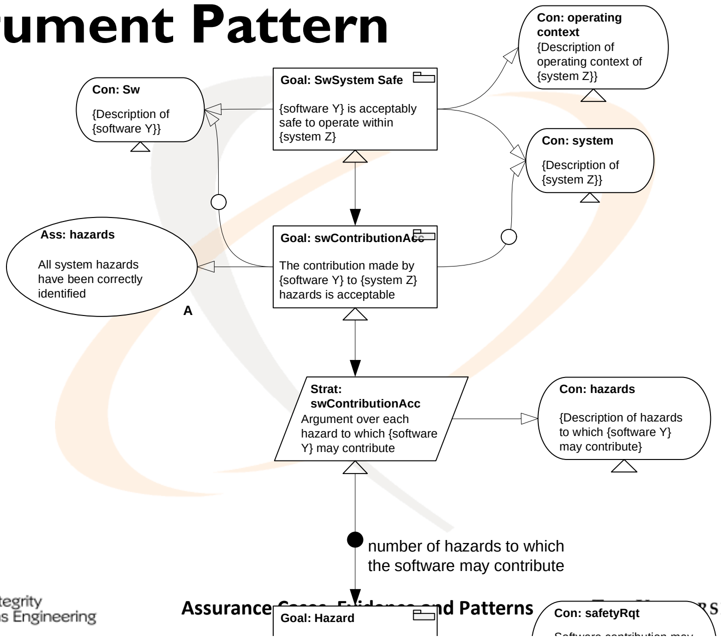
- Electronic elements in system context
  - Properties are "specific requirements", e.g. invariants

- Focus on ***"demonstrating the safety of"*** rather than ***"demonstrating the development of"***

Software Safety Cases – How?

- Two Issues when applied to software systems:
  - Constructing compelling software safety arguments
  - Justifying their sufficiency

High-level Software Safety Argument Pattern



Con: operating context
{Description of operating context of {system Z}}

Goal: SwSystem Safe
{software Y} is acceptably safe to operate within {system Z}

Con: Sw
{Description of {software Y}}

Con: system
{Description of {system Z}}

Ass: hazards
All system hazards have been correctly identified
A

Goal: swContributionAcc
The contribution made by {software Y} to {system Z} hazards is acceptable

Strat: swContributionAcc
Argument over each hazard to which {software Y} may contribute

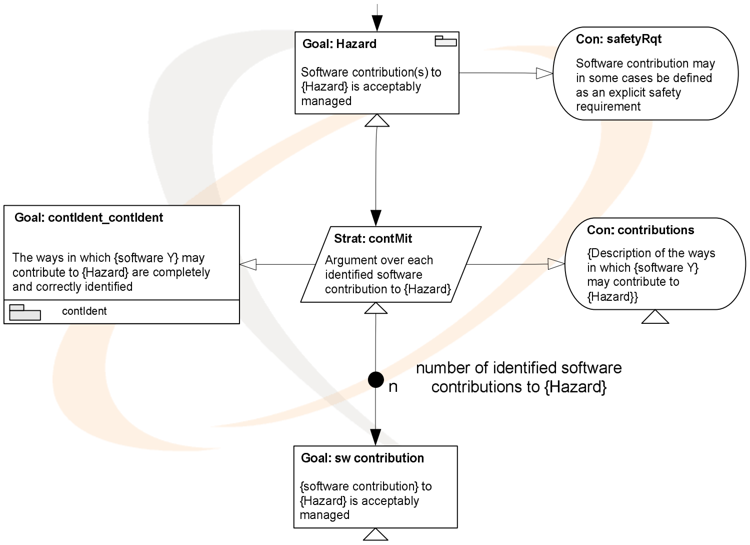
Con: hazards
{Description of hazards to which {software Y} may contribute}

number of hazards to which the software may contribute

Goal: Hazard

Con: safetyRqt

High-level Software Safety Argument Pattern



Goal: Hazard
Software contribution(s) to {Hazard} is acceptably managed

Con: safetyRqt
Software contribution may in some cases be defined as an explicit safety requirement

Goal: contIdent_contIdent
The ways in which {software Y} may contribute to {Hazard} are completely and correctly identified
contIdent

Strat: contMit
Argument over each identified software contribution to {Hazard}

Con: contributions
{Description of the ways in which {software Y} may contribute to {Hazard}}

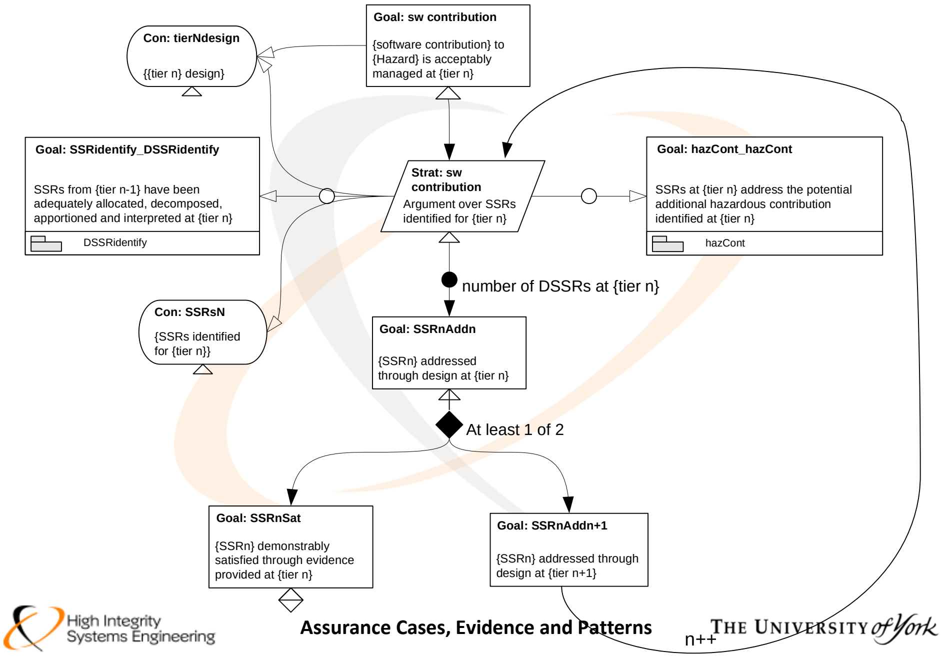
number of identified software contributions to {Hazard}
n

Goal: sw contribution
{software contribution} to {Hazard} is acceptably managed

Software Contribution Safety Argument Pattern

- Must consider all ways in which errors introduced into software which could lead to the software contribution
- Different development process used on different projects
  - Always have various 'tiers' of design
- At each tier must address requirements of the higher level
  - DSSRs from the previous tier must be adequately addressed
  - Consider additional hazardous contributions that may be introduced at each tier
- Instantiation decisions made here will have large impact on assurance

Software Contribution Safety Argument Pattern



Goal: sw contribution
{software contribution} to {Hazard} is acceptably managed at {tier n}

Con: tierNdesign
{{tier n} design}

Strat: sw contribution
Argument over SSRs identified for {tier n}

Goal: SSRidentify_DSSRidentify
SSRs from {tier n-1} have been adequately allocated, decomposed, apportioned and interpreted at {tier n}
DSSRidentify

Goal: hazCont_hazCont
SSRs at {tier n} address the potential additional hazardous contribution identified at {tier n}
hazCont

Con: SSRsN
{SSRs identified for {tier n}}

number of DSSRs at {tier n}

Goal: SSRnAddn
{SSRn} addressed through design at {tier n}

At least 1 of 2

Goal: SSRnSat
{SSRn} demonstrably satisfied through evidence provided at {tier n}

Goal: SSRnAddn+1
{SSRn} addressed through design at {tier n+1}

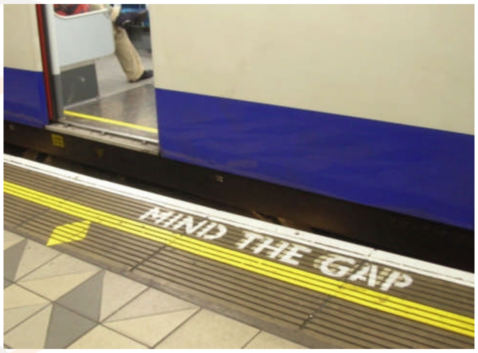
n++

Software Safety Arguments

- **Deductive arguments**
  - if the premises are true, then the conclusion must also be true
- **Inductive arguments**
  - the conclusion follows from the premises not with necessity, but only with probability
- (Predictive) software safety assurance arguments will always contain ***inductive*** elements

Mind the Gap

- Inductive nature ➔ determination of assurance ***subjective***
- Factors that can affect confidence:
  - Assumptions made
  - & scope drawn
  - The "inductive gap"
  - Trustworthiness of evidence
  - Visibility of information
- Reasoning about such factors can aid in successful acceptance of a safety case



Assurance Based Argument Development

- At every step in constructing the argument it is inevitable that information will be lost
  - Defining the safety claims
  - Deciding on strategy (argument approach)
  - Identifying assumptions and context
  - Providing evidence
- Losing information increases uncertainty, which affects assurance
  - **Assurance deficits**

Assurance Deficits I

- Recognised assurance deficits = Something we don't know (haven't addressed in the case)
  - A known unknown
  - Potential source of ***counter evidence***
- Increase assurance by addressing deficits
- Sufficiently?

"much of the effort only improves confidence that requirements have been met. In applying ALARP, the confidence achieved should be proportionate to the risk." (UK DefStan 00-56)

Assurance Deficits 2

- Are the identified assurance deficits acceptable?
- Necessary to determine 'consequences' of deficit
  - … on the software safety argument claims
- Which aspects of the claims are still assured, and which are not?
  - What are the worst implications of 'not knowing'?

Assurance Deficits 3

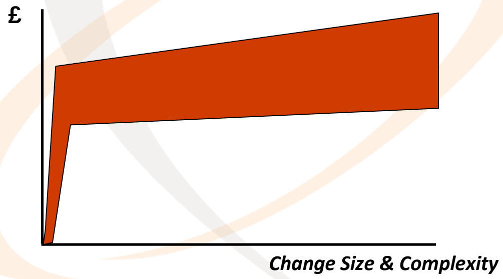
- Are we moved to act?
- Diminishing returns
- Can consider costs vs. benefits
- This leads us onto a consideration of ACARP

As Confident As Reasonably Practicable?

- Is the assurance deficit intolerable, negligible, or tolerable
  - Answer can involve a cost-benefit argument

Modular Certification – Why?

- The costs of *change* have become a major part of the cost of ownership of a system
- Currently, the costs of re-certification of a system following any change account for the greater part of the cost of change



Cost of Re-Certification is Related to **System** Size and Complexity

Safety Case Architecture

- Software architecture defined in the following terms (Bass et al., 1998):

> *"The structure or structures of the system, which comprise software components, the externally visible properties of those components, and the relationships among them"*

- Safety case architecture can be defined in similar terms:

> *The **high level organisation** of the safety case into components of arguments and evidence, the **externally visible properties** of these components, and the **interdependencies** that exist between them*

Modular Safety Case 'Interfaces'

Externally visible properties:

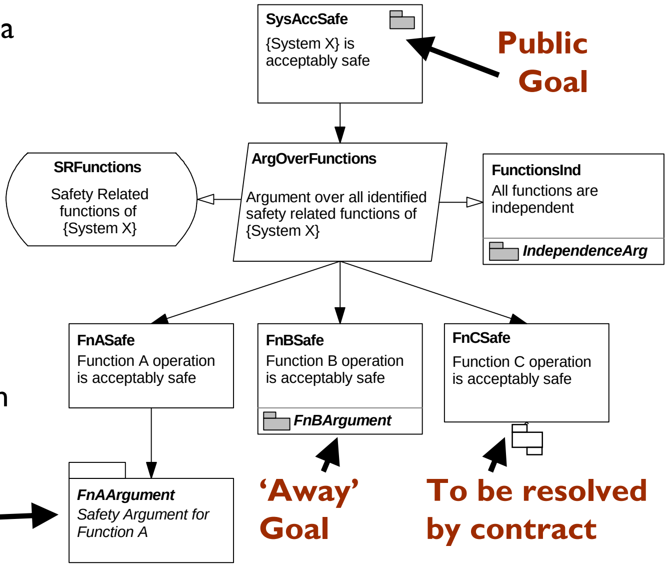
1. **Claims** 'publicly' addressed by the module
2. **Evidence** presented within the module
3. **Context** (e.g. assumptions) defined within the module

But also need to consider interdependencies …

4. **Claims** requiring support
5. Reliance on specific **claims** addressed elsewhere
6. Reliance on specific **evidence** presented elsewhere
7. Reliance on specific **context** defined elsewhere

'Modular' GSN Extensions

- Ability to mark a goal as *'public'*
- Ability to refer to goals defined in other modules
- Ability to refer to modules
- Ability to place one argument in the context of another


SysAccSafe
{System X} is acceptably safe
SRFunctions
Safety Related functions of {System X}
ArgOverFunctions
Argument over all identified safety related functions of {System X}
FunctionsInd
All functions are independent
IndependenceArg
FnASafe
Function A operation is acceptably safe
FnBSafe
Function B operation is acceptably safe
FnBArgument
FnCSafe
Function C operation is acceptably safe
FnAArgument
Safety Argument for Function A

Public Goal

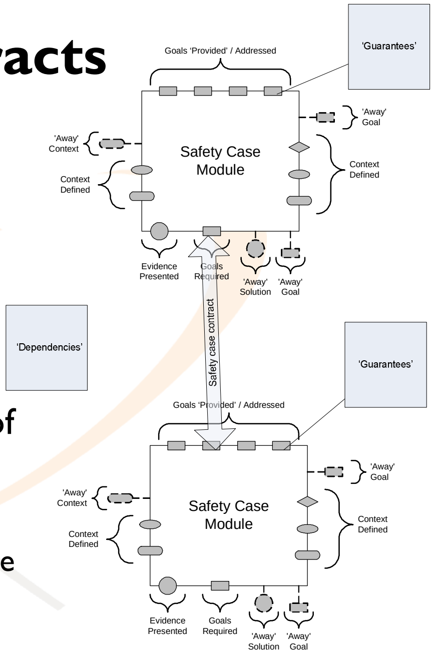
Module Reference

'Away' Goal

To be resolved by contract

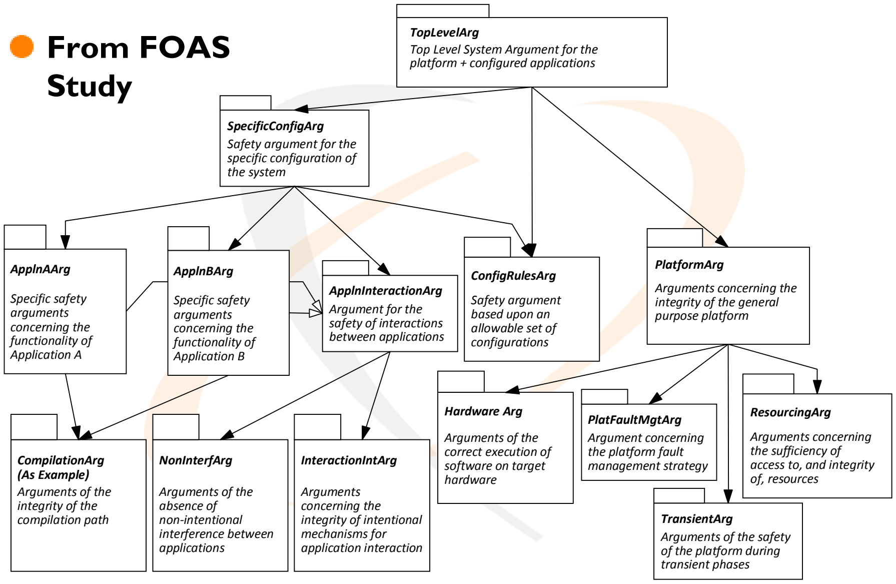
Safety Case Contracts

- Safety Case Modules can be composed if:
  - Goals Match (both ways)
  - **Context is compatible**
- Results recorded in a **safety case contract**
- Establishes a defined record of the inter safety case agreement
  - Supports management of change



Example: Safety Case Architecture for IMA

- **From FOAS Study**



TopLevelArg
Top Level System Argument for the platform + configured applications

SpecificConfigArg
Safety argument for the specific configuration of the system

ApplnAArg
Specific safety arguments concerning the functionality of Application A

ApplnBArg
Specific safety arguments concerning the functionality of Application B

ApplnInteractionArg
Argument for the safety of interactions between applications

ConfigRulesArg
Safety argument based upon an allowable set of configurations

PlatformArg
Arguments concerning the integrity of the general purpose platform

CompilationArg (As Example)
Arguments of the integrity of the compilation path

NonInterfArg
Arguments of the absence of non-intentional interference between applications

InteractionIntArg
Arguments concerning the integrity of intentional mechanisms for application interaction

Hardware Arg
Arguments of the correct execution of software on target hardware

PlatFaultMgtArg
Argument concerning the platform fault management strategy

ResourcingArg
Arguments concerning the sufficiency of access to, and integrity of, resources

TransientArg
Arguments of the safety of the platform during transient phases

Change Scenarios

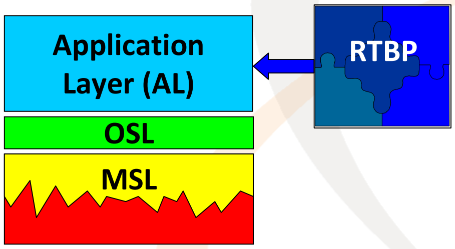
- Credible **change scenarios** include:
  - Hardware Vendor Change
  - Addition of a single application
  - Removal of a single application
  - Modification of existing application
  - Addition of extra processing nodes
  - Remove of processing nodes
  - Change of Databus
- Which safety case modules (arguments and evidence) would have to change in each case?
- Is the change local, non-local, architectural?

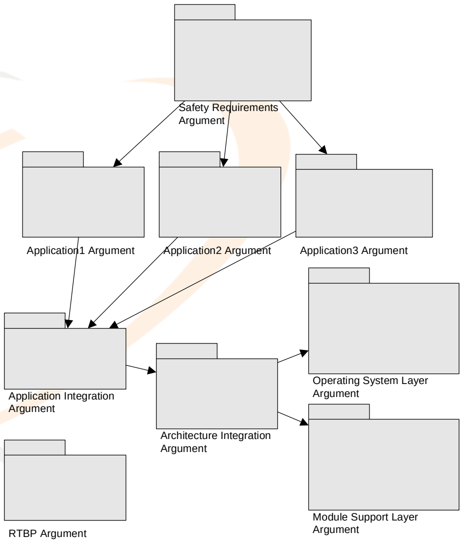
Hawk Parallel Study

- UK MoD funded a ‘hot’ research task
  - **Hawk AJT** aircraft chosen
  - Developing a modular safety case for a new system **in parallel** to monolithic project safey case
- Mission Computer is IMA using an ASAAC-compliant three-layer stack
- Study aims:
  - Show that a modular safety case can be produced for a representatively sized project
  - Demonstrate that the proposed benefits can be achieved
  - **Multi-party** modular safety case development
  - Involve MOD appointed safety assessors (QinetiQ TES) to assess viability

Safety Case ‘Architecture’

Design Architecture




Safety Requirements Argument
Application1 Argument
Application2 Argument
Application3 Argument
Application Integration Argument
Architecture Integration Argument
Operating System Layer Argument
Module Support Layer Argument
RTBP Argument

More Uses of Modular Assurance Cases

- Dealing with multi-attribute cases
  the **Dependability Case**
- Allows abstraction of **different** ‘top level’ arguments
- Maps to **common** properties of modules for architectural components
- + Arguments covering the (inevitable) trade-offs
- **Composition of Evidence**
  - All evidence has limitations
  - Box: “‘*all models are wrong*, some are useful”
  - Limitations can be mitigated in the composition of evidence
  - Inevitable that we have to appeal to multiple forms of evidence

Summary

- We want to reason about software safety **integrity**
  - But we find it difficult to do so directly (given nature of systematic errors)
- **Assurance case** approach introduced to address perceived problems with existing software safety assurance:
  - Poor correlation between processes and achieved integrity
- Safety cases require clearly articulated **argument**, supported by references to **evidence**
  - Evidence needs to be **contextualised** and **'glued' together**
- Guidance can be given on how to construct
- But, arguments still must be judged for **sufficiency**
  - Explicit consideration of ACARP useful
- **Modularity** can be exploited to manage complex cases

E-mail: tim.kelly@cs.york.ac.uk