# Software Safety Case Management

Dr Tim Kelly
Department of Computer Science
University of York

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



## Tutorial Overview

- Part One: 1300-1515
  - Introduction to Safety Cases
  - The Importance of Safety Arguments
  - The Goal Structuring Notation (GSN)
- Part Two: 1530-1745
  - Software Safety Cases
  - Problems with Current Approaches
  - The Structure of a Typical Software Safety Argument
  - Establishing Hazard-Directed Software Safety Arguments
  - Linking Process and Product Arguments



## Part 1: Overview

- Safety Case concept and purpose
- Requirements from standards
- Safety Case contents
- Safety arguments
  - presenting clear arguments
  - Goal Structuring Notation (GSN)
- Creating Arguments in GSN
- Where, When and How to Create Assurance Arguments





## Motivation

- Many (UK) standards establish the need for production of a safety case, e.g.

*"Safety Cases are required for all new ships and equipment as a means of formally documenting the adequate control of Risk and demonstrating that levels of risk achieved are As Low As Reasonably Practicable (ALARP)." (JSP430)*

*A person in control of any railway infrastructure shall not use or permit it to be used for the operation of trains unless*
*(a) he has prepared a safety case …*
*(b) the Executive has accepted that safety case …"*
*(HSE Railway Safety Case Regulations)*

*"The Software Design Authority shall provide a Software Safety Case …"*
*(U.K. Defence Standard 00-55)*



## The Purpose of a Safety Case

*Principal Objective:*

- 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)
- Safety Cases can be prepared for ..
  - commissioning
  - maintenance
  - decommissioning ...





## Some Definitions

- *"A safety case is a comprehensive and structured set of safety documentation which is aimed to ensure that the safety of a specific vessel or equipment can be demonstrated by reference to:*
  - *safety arrangements and organisation*
  - *safety analyses*
  - *compliance with the standards and best practice*
  - *acceptance tests*
  - *audits*
  - *inspections*
  - *feedback*
  - *provision made for safe use including emergency arrangements"*

  **(JSP 430)**
- *"The software safety case shall present a well-organised and reasoned justification based on objective evidence, that the software does or will satisfy the safety aspects of the Statement of Technical Requirements and the Software Requirements specification."*

  **(DS 00-55)**



## Argument & Evidence

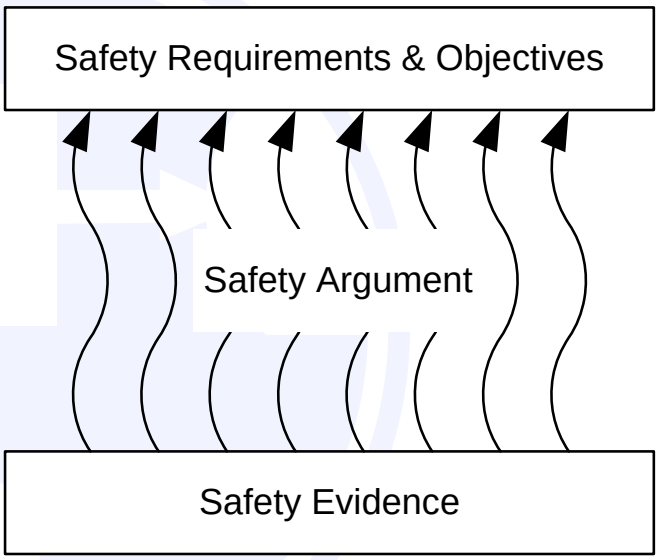
**A safety case requires two elements:**

- ***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***
  ***E*xplanation 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**





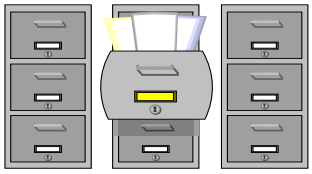
## Argument & Evidence 2






## Safety Cases vs. Safety Case Reports

- **Safety Case** is the totality of the safety justification + all the supporting material: testing reports, validation reports, relevant design information etc



- **Safety Case Report** is the document that summarises all the key components of the Safety Case and *references* all supporting documentation in a clear and concise format





## Safety Case Reports

- Exact contents depends on regulatory environment
- The following are key elements of most standards:
  - Scope
  - System Description
  - System Hazards
  - Safety Requirements
  - Risk Assessment
  - Hazard Control / Risk Reduction Measures
  - Safety Analysis / Test
  - Safety Management System
  - Development Process Justification
  - Conclusions



## Safety Arguments

- The Safety Case is not just a collection of disparate pieces of information
- The Safety Argument should form the 'spine' of the Safety Case showing how these elements are related and combined to provide assurance of safety
  - within the limits defined [*Scope*], the system [*System Description*] is SAFE because all identified hazards [*System Hazards*] and requirements [*Safety Requirements*] have been addressed. Hazards have been sufficiently controlled and mitigated [*Hazard Control / Risk Reduction Measures*] according to the safety risk posed [*Risk Assessment*]. Evidence [*Safety Analysis / Test*] is provided that demonstrates the effectiveness and sufficiency of these measures. Appropriate roles, responsibilities and methods were defined throughout the development of this system [*Development Process Justification*] [*Safety Management System*] and defined future operation





## Safety Arguments – Text Example

```
The Defence in Depth principle (P65) has been
addressed in this system through the provision of
the following:
• Multiple physical barriers between hazard source
  and the environment (see Section X)
• A protection system to prevent breach of these
  barriers and to mitigate the effects of a barrier
  being breached (see Section Y)
...
```

- Safety Arguments should clearly describe how a safety objective / requirement / claim has been achieved in the system as proposed
  - how it has been interpreted
  - ultimately, what evidence supports the requirements



## Safety Arguments – Text Problems

> For hazards associated with warnings,the assumptions of [7] Section 3.4 associated with the requirement to present a warning when no equipment failure has occurred are carried forward. In particular, with respect to hazard 17 in section 5.7 [4] that for test operation, operating limits will need to be introduced to protect against the hazard, whilst further data is gathered to determine the extent of the problem.

- not everyone can write clear English
- can take many readings to decipher meaning
- multiple cross-references in text can be awkward
- is there a clear shared understanding of the argument?





## Presenting Clear Arguments

- It is possible in text – at least sometimes
  - use simple language and short sentences
  - use bullet points for key statements
  - break down the argument one step at a time
    - and refer to following sub-sections
  - structure document sub-sections around separate concepts
    - e.g. Section 6.2 – Control of Hazard – 'Inadvertent Chaff Release'
- But it is easier with pictures!
  - use a *graphical notation* to summarise argument
    - Goal Structuring Notation (GSN)
    - Claims Argument Evidence (CAE)



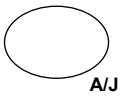
# The Goal Structuring Notation

## 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,

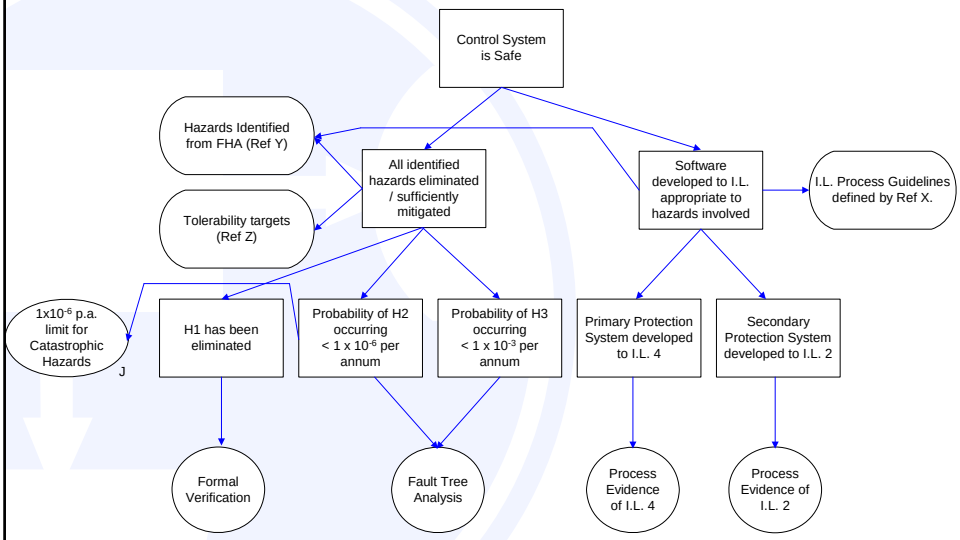
the rationale for the approach (assumptions, justifications) 


and the context  in which goals are stated



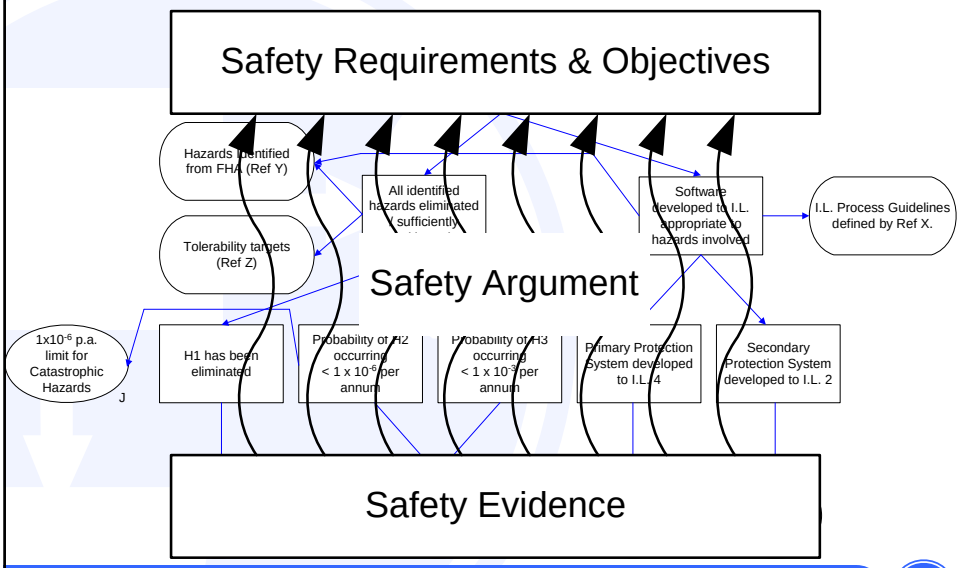


# A Simple Goal Structure






# A Simple Goal Structure





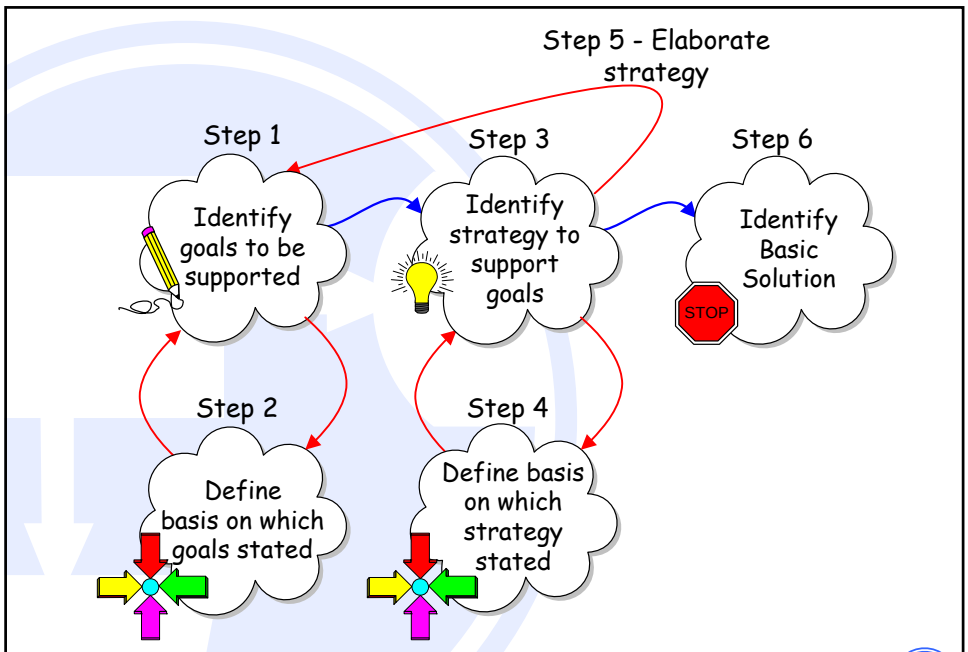



# Group Exercise

- **Create a goal structure to support the claim:**

**Top**

This is a good pint of beer



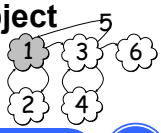





## Step 1 - Identify Goals: Phrasing

- Goals should be phrased as propositions
  - Statements that can be said to be TRUE / FALSE (e.g. "The sky is blue" or "York is a beautiful city")
  - NB: not limited to statements that can be objectively proven
  - Statement should be expressed as a single statement (1 sentence) of in the form:

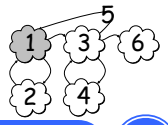
    <NOUN-PHRASE><VERB-PHRASE>
  - Noun-Phrase identifies the subject of the goal
  - Verb-Phrase defines a predicate over the subject



## Step 1 - Identify Goals: Phrasing

- **The following are examples of correctly stated goals:**

| ***Subject*** <Noun-Phrase> | ***Predicate*** <Verb-Phrase> |
|---|---|
| Component X | has no 'critical' failure modes |
| All identified hazards for System Y | have been sufficiently mitigated |
| Non-destructive examination of weld-site Z | has been performed |
| Design A | employs triple modular redundancy |





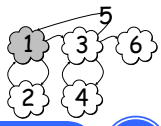



## Step 1 - Identify Goals: Phrasing

- **The following are examples of *incorrectly* stated goals:**

| | ***Reason:*** |
|---|---|
| • "Hazard Log for System Y" | *NP – describes an entity - not a statement* |
| • "Fault Tree for Hazard H1" | *As above* |
| • "Perform Fault Tree Analysis of Hazard H1" | *VP - an action - not a statement* |
| • "How many failure modes does component X have?" | *Question - not a statement* |

- **Test: can we say goal is TRUE / FALSE?**



## Step 1 – Example

**G1**

Press is acceptably safe to operate within CCC Whatford Plant

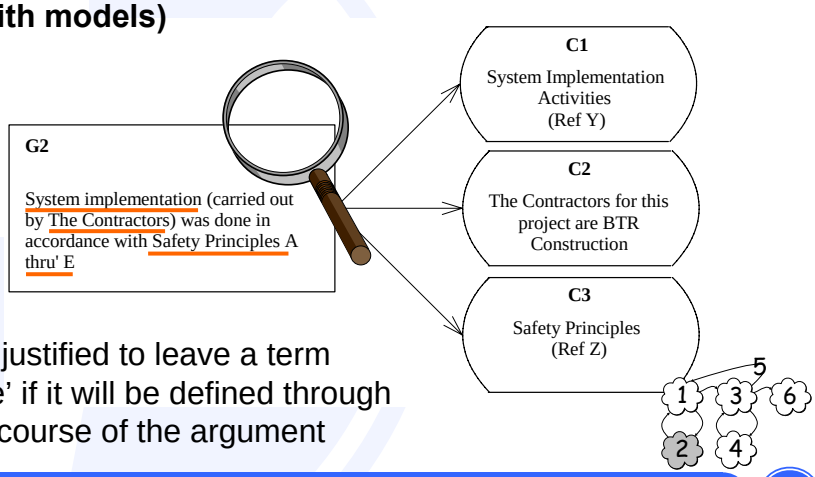
- **As with conventional safety case report – we wish to clearly set out the objective and scope of the safety argument being presented**





## Step 2 - Define basis for goals: *Context*

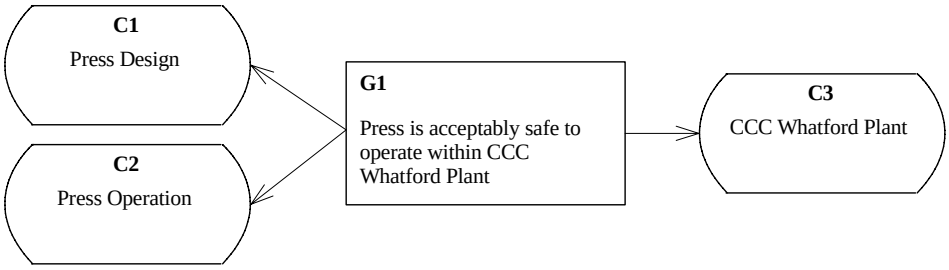
- **Examine goal statement for terms / concepts not clearly defined within scope (context inherited as with models)**




It is justified to leave a term 'free' if it will be defined through the course of the argument



## Step 2 – Example



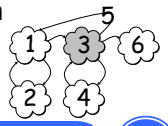

- Terms – *Press*, *Operate* and *CCC Whatford Plant* drawn out explicitly as contextual information
- *Acceptably Safe* left for expansion through the supporting argument





## Step 3 - Identify strategy

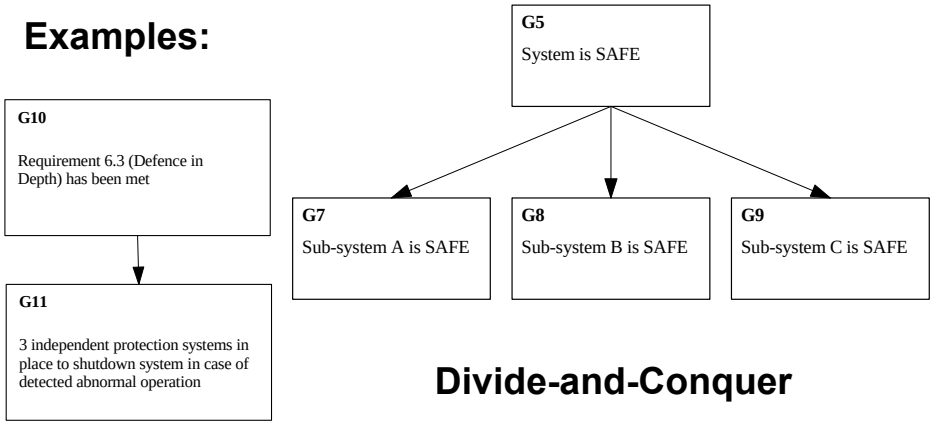
- Next step is to work out how to substantiate the stated goal
  - "What reasons are there for saying the goal is TRUE?"
  - "What statements would convince the reader that the goal is TRUE?"
- Aiming for statements that are *easier* to support than the larger goal
  - Breaking into a number of smaller goals - i.e. Divide-and-Conquer
  - Relating goal more closely to specific application in question (e.g. for a generic requirement)





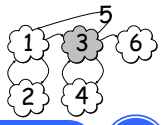

## Step 3 - Identify strategy

**Examples:**




**Divide-and-Conquer**

**Interpretation / Particularisation**








## Step 3 - Identify strategy: *Phrasing*

- The role of a strategy node is to clearly explain the relationship between a goal and a set of sub-goals
- An analogy:

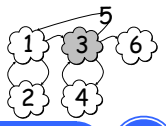
***Strategy***

$3xy^3 + 2x^2y^2 + 5xy = 17y$ *(Divide both sides by y)*

$3xy^2 + 2x^2y + 5x = 17$

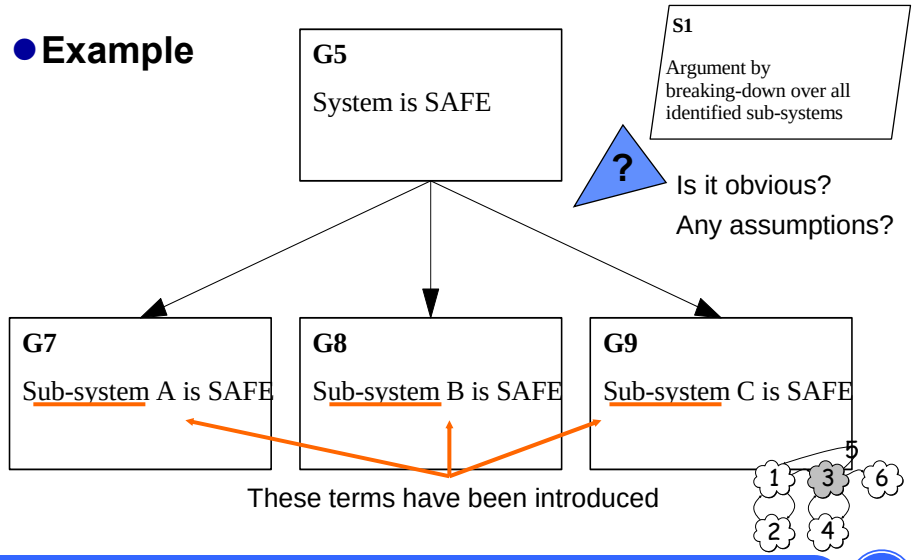
- Strategy statement should succinctly describe the argument approach adopted, ideally in the form:
  - ***"Argument by … <approach>"***

- **Example statements:**
  - ***"Argument by appeal to test results"***
  - ***"Argument by consideration of historical data"***
  - ***"Quantitative argument using simulated run data"***



## Step 3 - Identify strategy

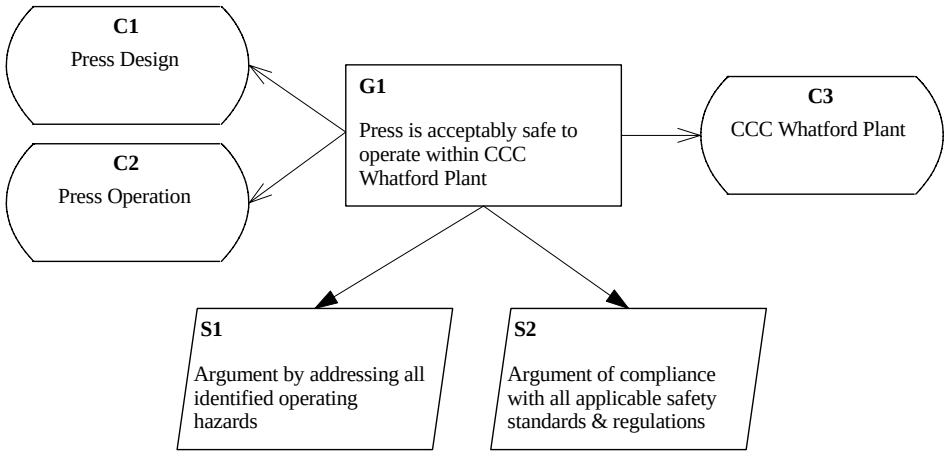
- Example







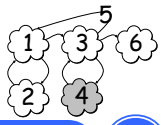

## Goal Structure Build-Up 3




- Two separate strategies – for reader's benefit

# Step 4 - Define basis for strategy

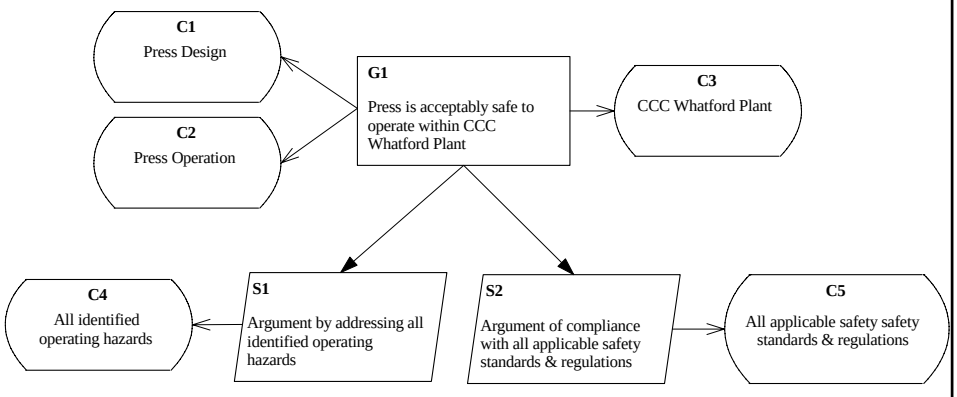
- In the same way as is necessary for goals, must examine what contextual information (including models) is required
- Same process - examine strategy for terms / concepts introduced but not 'bound'
  - e.g. for sub-system breakdown strategy the term '*All Identified sub-systems*" is used
- Ask what information is required in order to expand / fulfill strategy outlined



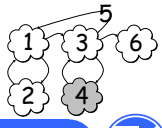





# Step 4 - Example

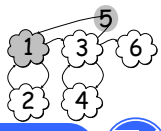



- Needs Justification?
- Any Assumptions?



# Step 5 - Elaborate strategy

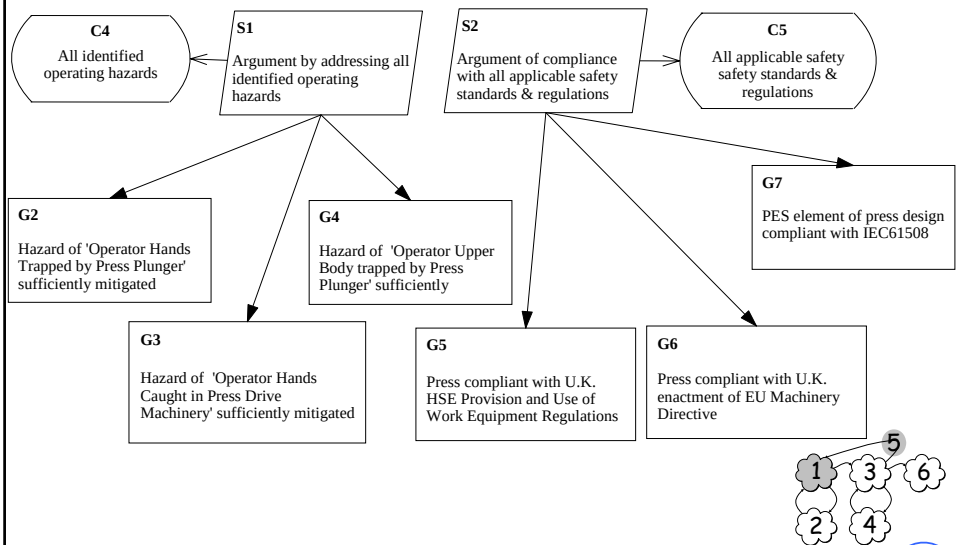
- Having identified an approach, it is necessary to lay out the goals that fulfill that approach, e.g.
  - e.g., for strategy ranging over all sub-systems - expand for goals over each individual sub-system
  - e.g. for quantitative results strategy - provide quantitative goal statements
- In elaborating the strategy, again defining goals (back to Step 1)
- If strategy, and basis of strategy, are clear - this step can be straightforward
  - E.g. see next slide





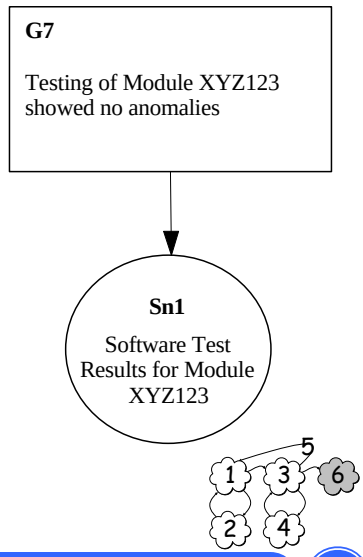



# Step 5 - Example



# Step 6 - Identify Solutions

- Eventually, faced with a goal that doesn't need further expansion, refinement, explanation …
- In such cases, simply have to reference out to information that supports claim by means of solution
- As references, solutions should be NOUN-PHRASEs



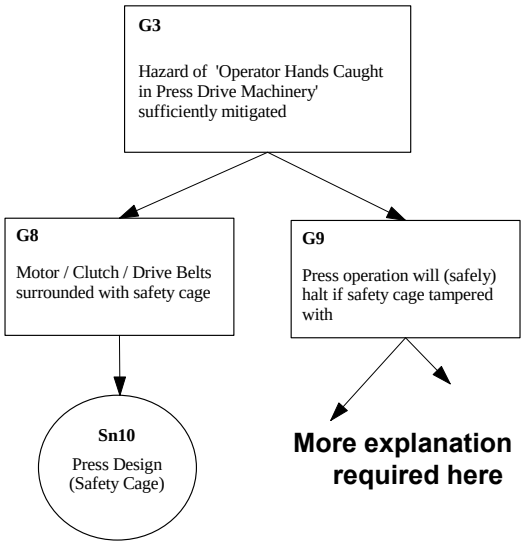








# Step 6 – Example




- Reference to source of information that would substantiate claim

## How to Create Goal Structures

- Two main approaches:
  - As an individual
  - As a group
    - Where aim is to reach common understanding and agreement of structure of the safety argument
      - Attendance: Safety Argument Owner (Principal Stakeholder), Experts, GSN Facilitator, Secretary (Optional)
      - Requires Key Documents to be circulated beforehand





## Where to Brainstorm the Argument?

- Q: Where is it worth brainstorming an argument?
- Answers:
  - Wherever there is most uncertainty about the argument (key claims, evidence)
  - Wherever the argument is currently confused or is over-complex
  - Wherever there is disagreement about the argument
  - Wherever the consequences of having a *wrong* argument are high (in terms of rework, delays etc.)



## When to Visualise the Argument?

- Q: At what stage in a project is it worth visualising the argument?
- Answers:
  - Early on (high level) to get a clear picture (and gain agreement) of argument structure
    - Useful as a scoping exercise and effort allocation
  - As project is progressing, in order to monitor status towards completion of an acceptable argument
  - At end of project in order to present the final argument and evidence that exists





## How to Present Goal Structures

- Customers are keen to see goal structures within safety documents
- Possible approaches to inclusion of GSN:
  - In full as Appendix / Annex to document
  - Integrated within body of document
    - Goal structure (1 level), Text, Goal structure, Text …
    - See 'Nuclear Trip System Safety Case Example'
  - As 'Executive Summary' at beginning of document
    - Maximum 2 pages of structure, 2-3 levels of decomposition
  - As separate, stand-alone, Index Document
    - e.g. to explain argument distributed across many safety case documents



## Potential GSN Benefits

- Improving comprehension of existing arguments
- Improving discussion and reducing time-to-agreement on argument approaches being adopted
- (Having identified argument structure up front) focusing activities towards the specific end-objectives
- Recognition and exploitation of successful (convincing) arguments becomes possible
- Supports 'light-weight' evolution of an argument
- Supports monitoring of project progress towards a successful safety case





## Conclusions

- Within conventional safety case reports the 'chain of argument' can often get lost
  - The argument is more important than the document!
- GSN has been found to be a useful basis for *mapping out* and *evolving* the structure of the *Safety Arguments*
  - *Provides a Road-map for a document / set of documents*
  - *Provides a basis for discussion amongst engineers and between developers and assessors*
  - *Creating an outline arguments towards beginning of project can be seen as making progress towards a final solution*



# Software Safety Case Management – Part Two

Dr Tim Kelly
Department of Computer Science
University of York

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

Acknowledgements: Rob Weaver, Ibrahim Habli



## Part 2 Overview

- Current approaches (and the problems) for Software Safety Case development
- An Evidence-based Framework for software safety arguments
  - Software Level Safety Requirements
  - Different types of Evidence
  - Different types of Hazardous Failure Modes To provide an overview of a 'generic' computer system safety argument structure

We will look at:

- Process Arguments
- Product Arguments
- Hazard-Based vs. Requirements-Based Arguments
- Functional vs. Non-Functional Issues



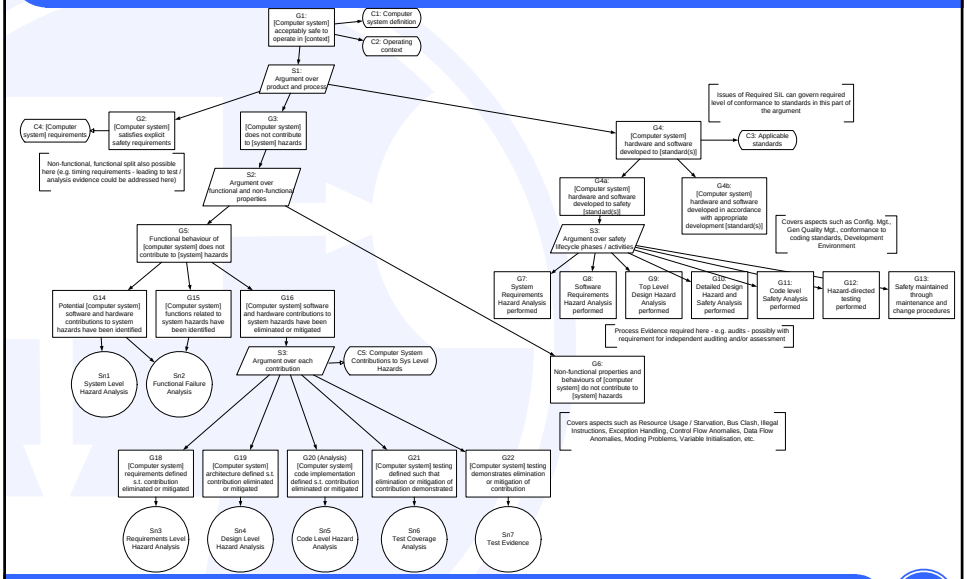
# Problems with Current Practice

- Discontinuity in the safety case
  - System Argument has a Product Focus
  - Software Argument has a Process Focus
- Does Good Process = Good Product ?
  - Questionable Assumption
- Application to Legacy and Commercial Off The Shelf (COTS) software
- Modern Practices – e.g. Code Generators
- Standards prescribe Software Development Process
  - Discourages intelligent thought about what evidence is useful or relevant
- Responsibility for safety lies with those that set the standard rather than those that build the software





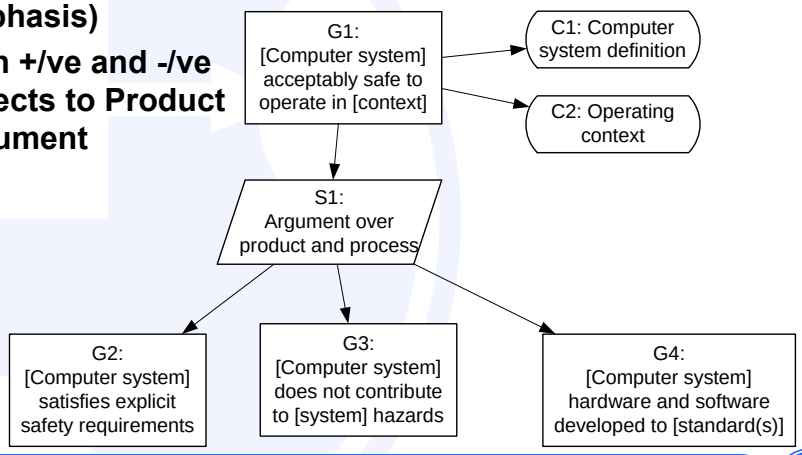
# Overall S/W SC Structure






# Top Level Argument

- Important of Context and Clear System Definition
- Process and Product (Historically more Process Emphasis)
- Both +/ve and -/ve aspects to Product Argument

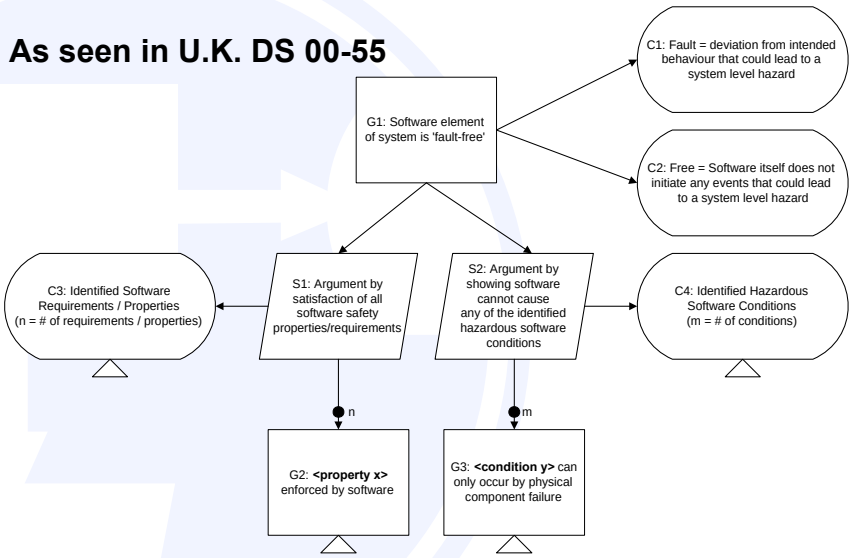






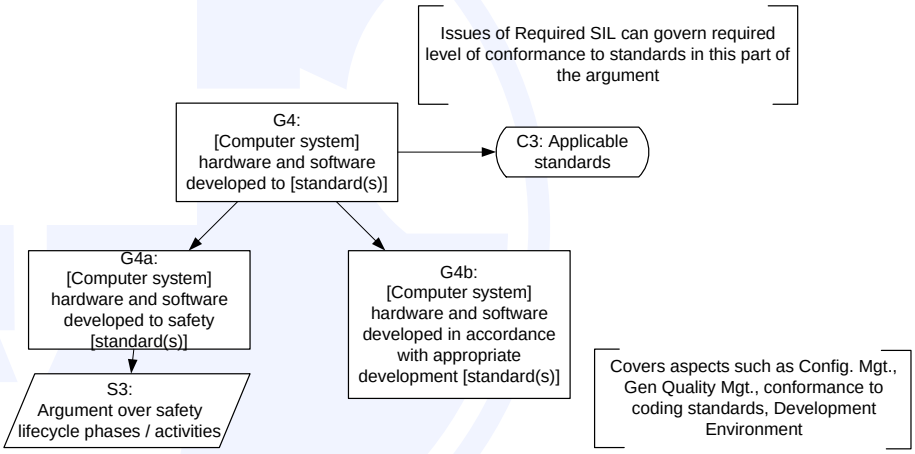

# Software 'Fault Free' Pattern

- As seen in U.K. DS 00-55



# Conformance to Standards Argument

- SILS can govern required level of conformance
- Role for both conventional dev't and 'safety' standards








# SIL Tailoring of Process Arguments

## SIL (Process) Justifications:

***Annex D - Tailoring Guide Across Differing Safety Integrity Levels***

| Clause | S1 | S2 | S3 | S4 | Comments |
|---|---|---|---|---|---|
| 36 Coding Process | | | | | |
| 36.1 Coding Standards | J1 | J1 | M | M | |
| 36.2 | M | M | M | M | |
| 36.3 | J2 | J1 | M | M | |
| 36.4 | J2 | J1 | M | M | |
| 36.5 Static Analysis and Formal Verification | | | | | |
| 36.5.1 | J1 | J2 | M | M | |
| … | … | … | … | … | … |

- M = Must be Applied
- J1 = Justification of not following clause
  - Inapplicability
  - Cost-benefit (ALARP)
- J2 = Less detailed / rigourous justification

# DO-178B Example

***Table A-3 Verification of Output of Software Requirements Process***

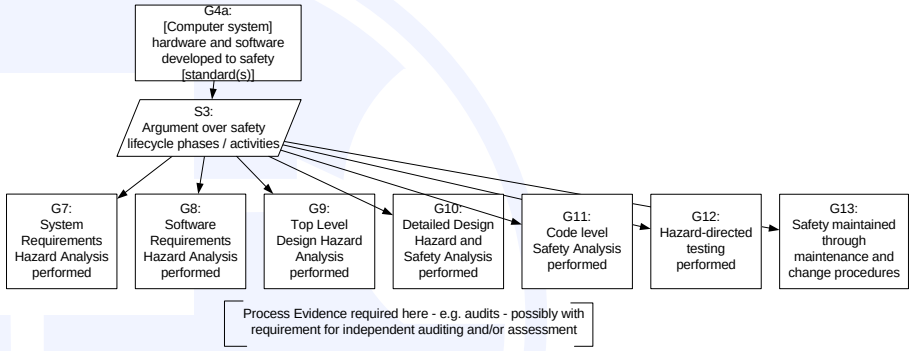
| | Objective | | Applicability by SW Level | | | | Output | | Control Category by SW Level | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | Description | Ref. | A | B | C | D | Description | Ref. | A | B | C | D |
| 1 | Software high-level requirements comply with system requirements | 6.3.1a | ● | ● | ❍ | ❍ | Software Verification Results | 11.14 | ② | ② | ② | ② |
| 2 | High-level requirements are accurate and consistent | 6.3.1b | ● | ● | ❍ | ❍ | Software Verification Results | 11.14 | ② | ② | ② | ② |
| 3 | High-level requirements are compatible with target computer | 6.3.1c | ❍ | ❍ | | | Software Verification Results | 11.14 | ② | ② | | |
| 4 | High-level requirements are verifiable | 6.3.1d | ❍ | ❍ | ❍ | | Software Verification Results | 11.14 | ② | ② | ② | |
| 5 | High-level requirements conform to standards | 6.3.1e | ❍ | ❍ | ❍ | | Software Verification Results | 11.14 | ② | ② | ② | |
| 6 | High-level requirements are traceable to system requirements | 6.3.1f | ❍ | ❍ | ❍ | ❍ | Software Verification Results | 11.14 | ② | ② | ② | ② |
| 7 | Algorithms are accurate | 6.3.1g | ● | ● | ❍ | | Software Verification Results | 11.14 | ② | ② | ② | |

| | | |
|---|---|---|
| **LEGEND**: | ● | The objective should be satisfied with independence |
| | ❍ | The objective should be satisfied |
| | Blank | Satisfaction of objective is at applicant's discretion |
| | ② | Data satisfies the objectives of Control Category 2 |





# Safety Standards Argument

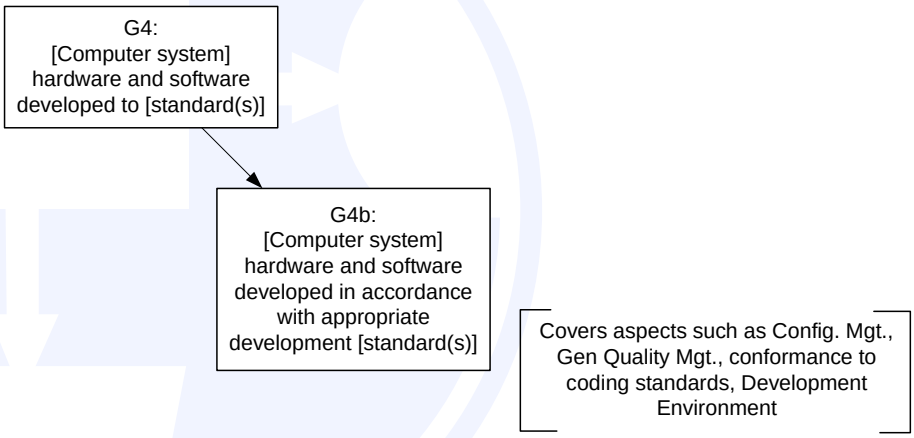



- **Structured here by lifecycle phase**
- **Argument that safety considered appropriately at various points in the traditional development lifecycle**
- **Process Evidence, possibly with requirement for independence, needed here**



## ‘Traditional’ Development Standards Arg.

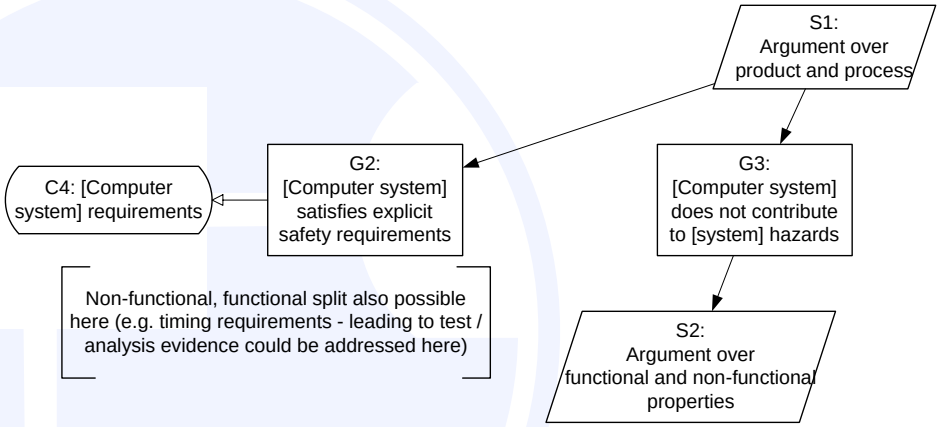
- **Necessary but not sufficient elements**
- **Still concerns general ‘integrity’ of finished product**







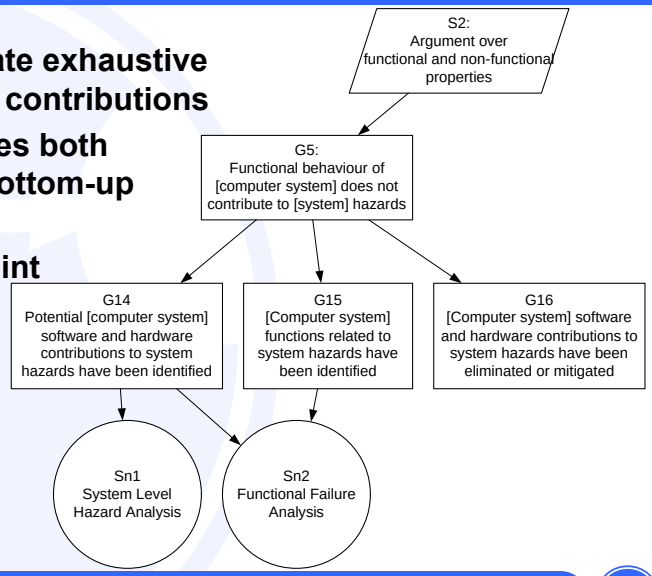

## Product Argument




- **Functional and Non-functional split possible both under hazard based argument and requirements conformance argument**
- **Requirements based argument can deal with cascade from system level**
- **Can be overlap between requirements and hazard based arguments**

# Functional Behaviour Argument

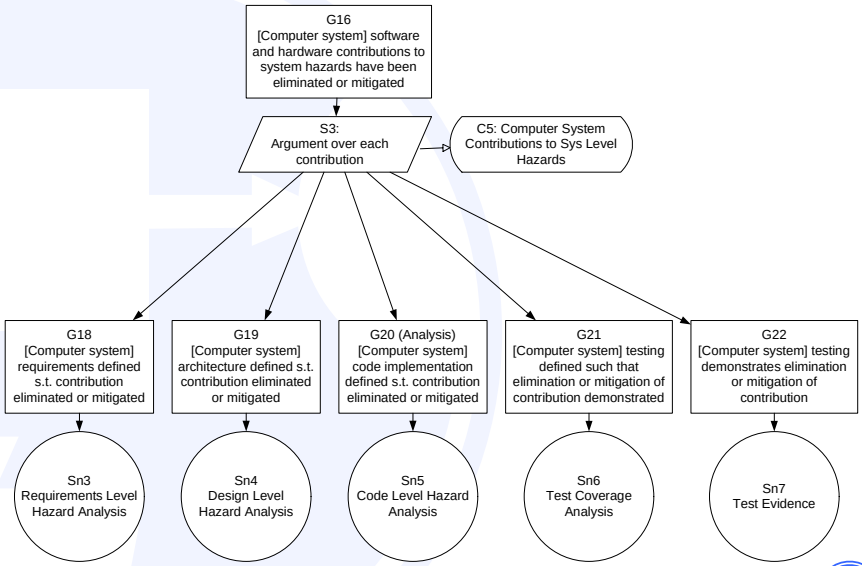
- Must demonstrate exhaustive identification of contributions
- Typically requires both top-down and bottom-up approaches
- Useful to pin-point contributions to dev't items in software structure








# Functional Contribution Argument 1



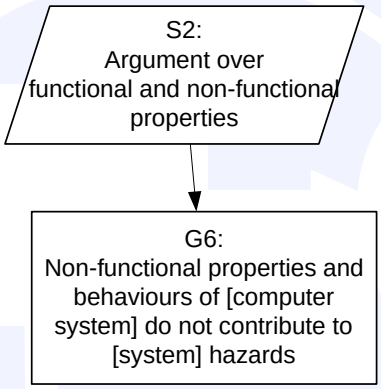
# Functional Contribution Argument 2

- Need to demonstrate systematic consideration of each contribution
- Ideally, wish to see consideration through lifecycle
- Often best solutions to potential contribution are early lifecycle
- Argument can be based on both analysis (e.g. of specification) and test (of implementation)
- For Testing – must demonstrate coverage of contributions
  - May be possible to integrate with conventional acceptance testing





# Non-Functional Contribution Argument




- Remember, non-functional requirements dealt with elsewhere
- Dealing here with non-intentional implementation 'hazards'
  - Sometimes described at 'Computing' Hazards
- Problem of Completeness
  - Standard Lists exist (e.g. 882C)

Covers aspects such as Resource Usage / Starvation, Bus Clash, Illegal Instructions, Exception Handling, Control Flow Anomalies, Data Flow Anomalies, Moding Problems, Variable Initialisation, etc.



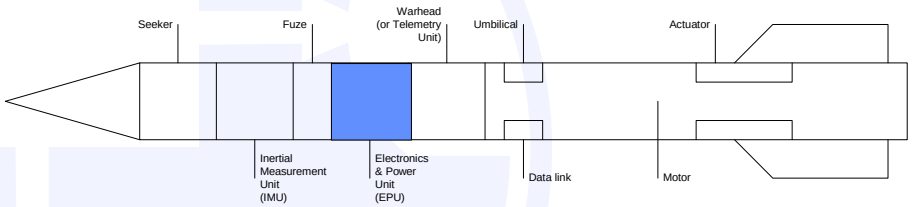
# Evidence-Based Framework

- System Level Requirements
- Requirement for System Level Evidence
  - Validation, Satisfaction, Traceability
- Software Level Safety Requirements
  - Focus on Software Contributions to System Level Hazards
- Requirement for System level Evidence
  - Validation, Satisfaction, (Traceability)
- Classification of Hazardous Failure Modes
  - Omission, Commission, Early, Late, Value
- Generic Arguments
- Targeted Selection of Evidence





# Example Case Study – Air to Air Missile






# System Level Safety Requirements

- System Level Hazard Analysis
  - Handling Hazards
  - Hazards to Launcher
    - Premature Detonation (or Break-up, on telemetry rounds)
    - Premature Launch
    - Disintegration near or in front of launcher due to high-g or high-roll manoeuvres
    - Hitting the launcher due to incorrect trajectory
    - Premature fin movement (i.e. prior to launch, perhaps damaging missile or launcher)
    - Hang-fire (excessive delay between ignition and thrust)
    - Hang-up (missile remains on launcher but thrusts)
  - Hazards to Friendly-Forces
  - Hazards of Mission Failure



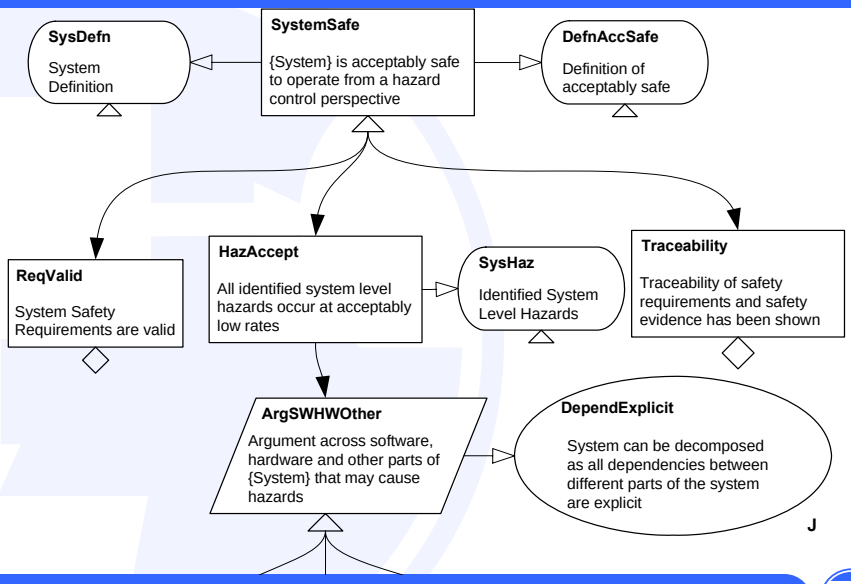


# System Level Evidence Categorisation

- Validation
  - *Demonstration that the set of System Safety Requirements is complete*
- Satisfaction
  - *Demonstration that all System Safety Requirements have been met*
- Traceability
  - *Demonstration that all System Safety Requirements have been tracked throughout System Development and Safety Analysis*
- Next stage is to consider Hardware, Software and Other contributions to System-level Hazards



# Pattern – System-level SR



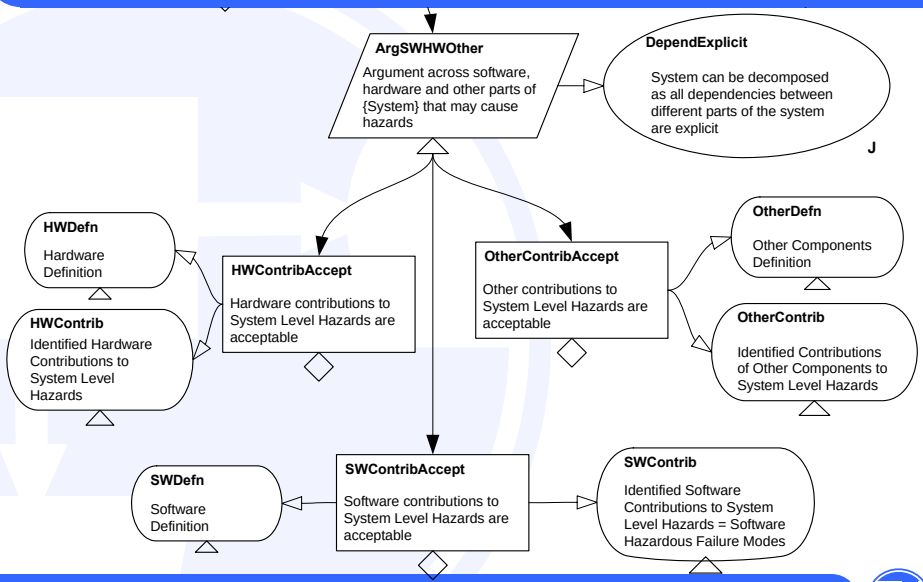





# Evidence-Based Framework

- System Level Requirements
- Requirement for System Level Evidence
  - Validation, Satisfaction, Traceability
- Software Level Safety Requirements
  - Focus on Software Contributions to System Level Hazards
- Requirement for System level Evidence
  - Validation, Satisfaction, (Traceability)
- Classification of Hazardous Failure Modes
  - Omission, Commission, Early, Late, Value
- Generic Arguments
- Targeted Selection of Evidence

# Pattern – Software Level SR








# Software Level Safety Requirements

- Derived from System Level Safety Requirements
  - From System Level Perspective can be seen as a "Basic Event"
  - From Software Level Perspective can be seen as the "Top Level"
- Hazard Based
  - Potential Failures within the Software that can lead to System Level Hazards
- Example – Software Safety Requirement
  - Acceptability of Software Failure Mode - Software Fails to block premature launch

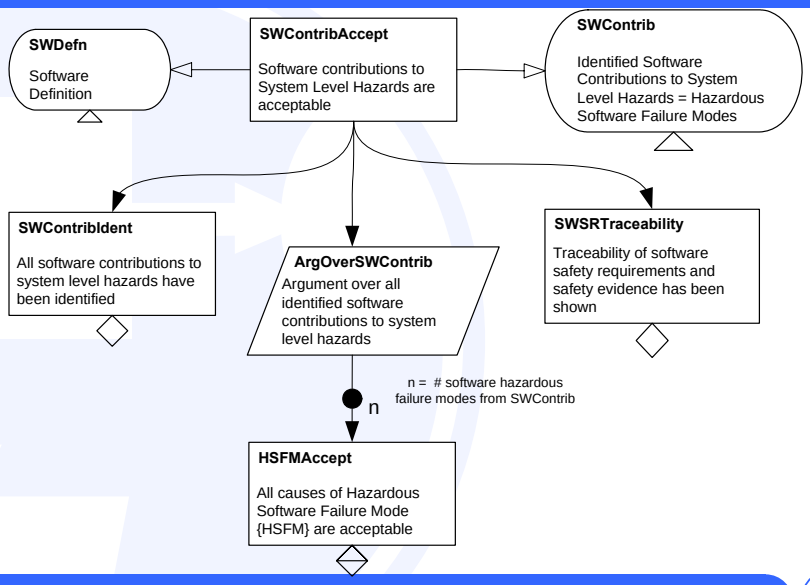
# Software Level Evidence Categorisation

- **Validation**
  - ***Demonstration that the set of Software Safety Requirements is complete***
- **Satisfaction**
  - ***Demonstration that all Software Safety Requirements have been met***
- **Traceability**
  - ***Demonstration that all Software Safety Requirements have been tracked throughout System Development and Safety Analysis***





# Pattern – Software Level SR






## Validation and Traceability

- Validation
  - Completeness
    - Functional Failure Analysis/HAZOP
- Traceability
  - DO-178B
    - Traceability between the system requirements and software requirements.
    - Traceability between the low-level requirements and high-level requirements
    - Traceability between the Source Code and the low-level requirements
  - Traceability Matrix





## Evidence-Based Framework

- System Level Requirements
- Requirement for System Level Evidence
  - Validation, Satisfaction, Traceability
- Software Level Safety Requirements
  - Focus on Software Contributions to System Level Hazards
- Requirement for System level Evidence
  - Validation, Satisfaction, (Traceability)
- Classification of Hazardous Failure Modes
  - Omission, Commission, Early, Late, Value
- Generic Arguments
- Evidence Characteristics
  - Relevance, Coverage, Independence



# Hazardous Failure Mode Classification

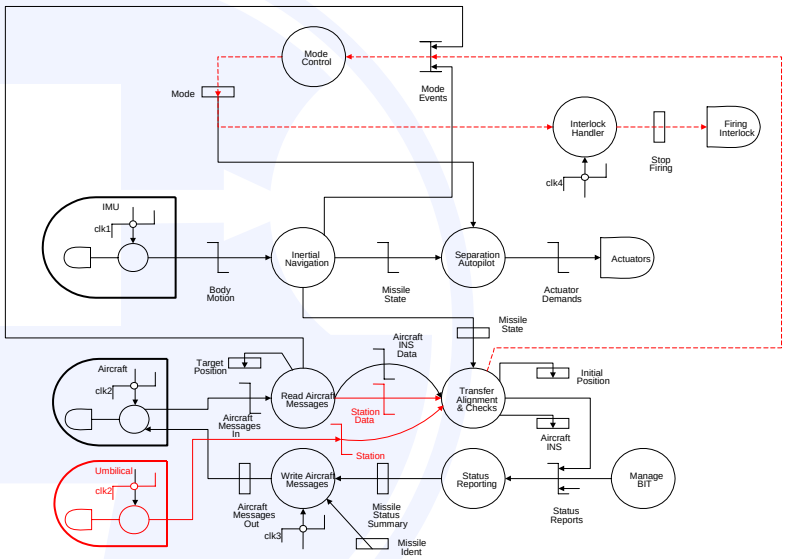
- Omission, Commission, Early, Late, Value
  - Service Provision, Service Timing, Value
- Common arguments for different classes
  - Certain arguments (and evidence types) can be used for different failure types
- It is possible to produce generic safety case arguments that can be reused.
- Example
  - Software Failure Mode - Software Fails to block premature launch



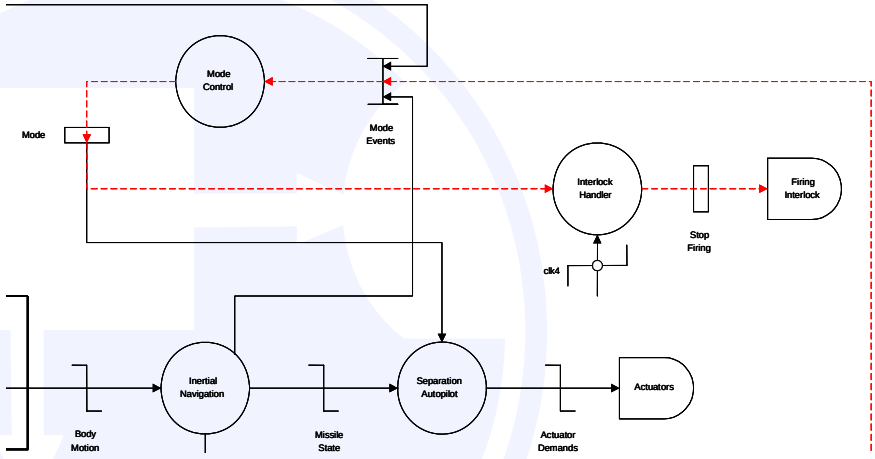




# Example - Software Architecture 1



# Example - Software Architecture 2

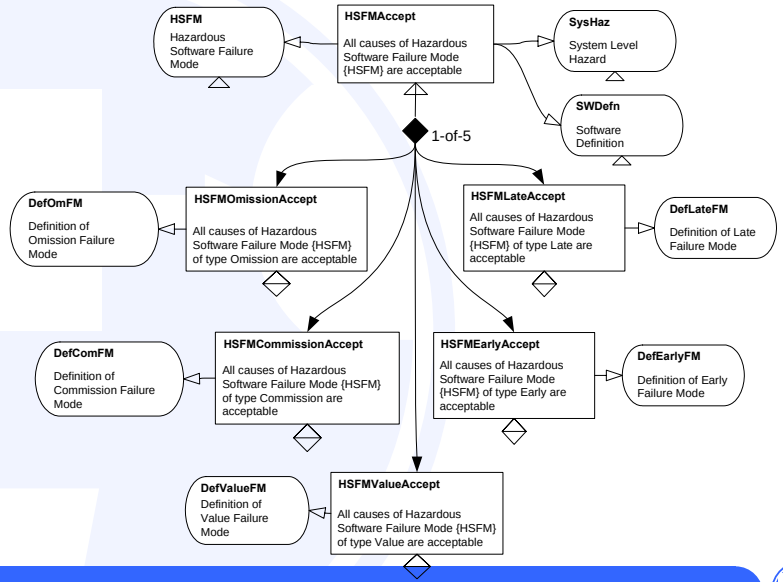



- Software Fails to block premature launch caused by:
  - Software interlock handler fails to write to interlock pool – Omission Failure Type (Service Provision Failure Type)





# Pattern –Failure Mode Classification






## Evidence-Based Framework

- System Level Requirements
- Requirement for System Level Evidence
  - Validation, Satisfaction, Traceability
- Software Level Safety Requirements
  - Focus on Software Contributions to System Level Hazards
- Requirement for System level Evidence
  - Validation, Satisfaction, (Traceability)
- Classification of Hazardous Failure Modes
  - Omission, Commission, Early, Late, Value
- Generic Arguments
- Targeted Selection of Evidence



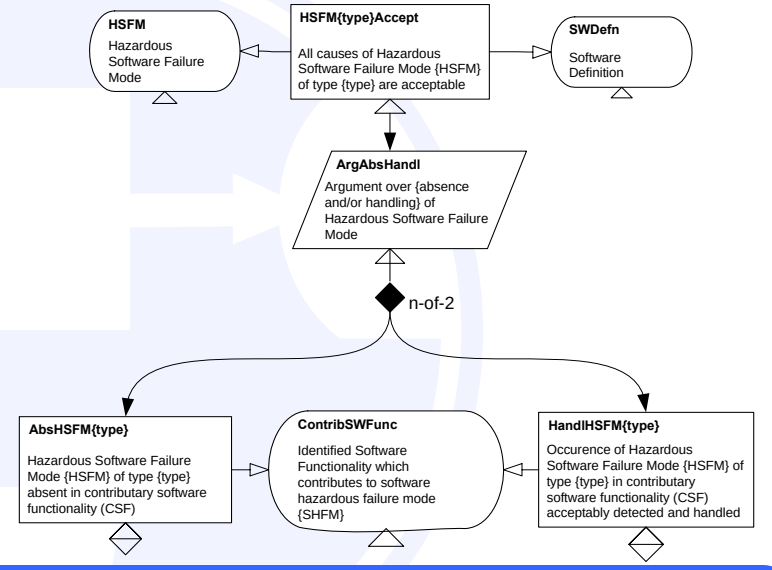


## Arguing about Requirements

- Satisfaction of a Typed Hazardous Failure Requirement
- Demonstration that the requirement has been met shown through combination of
  - Absence
  - Handling
  - (Probability – we can't show this for software)
- Example
  - Show Absence of Omission Hazardous Failure Mode - Software output module fails to provide braking value to actuator



# Pattern – Argument Approach



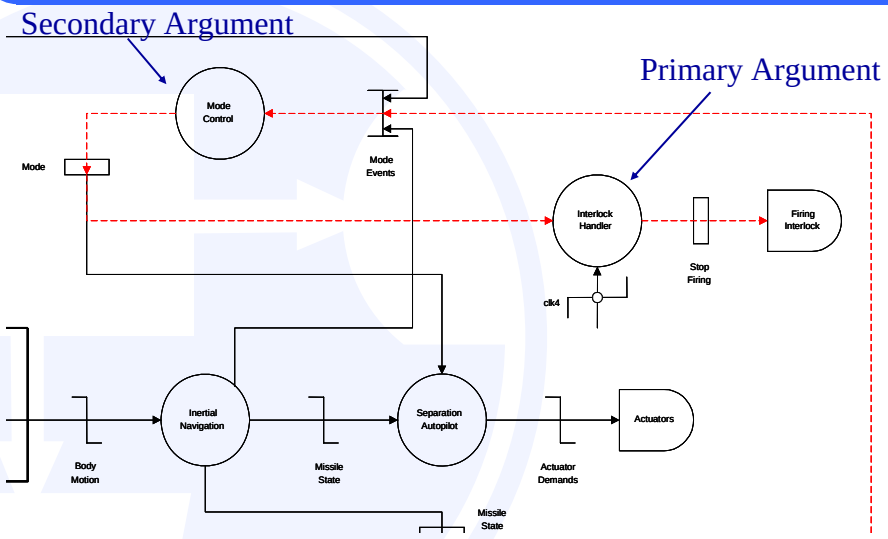





# Argument for Absence Omission Failure

- Software Failure Mode (of type Omission) is absent if:
- Primary Argument
  - All feasible paths through software functionality contain a unique output statement
- Secondary Argument
  - Failure of other software functionality which could lead to a failure of primary software functionality does not occur
  - All necessary resources exist to support correct operation of primary software functionality
- Control Argument
  - Primary software functionality is scheduled and allowed to run (at least) once

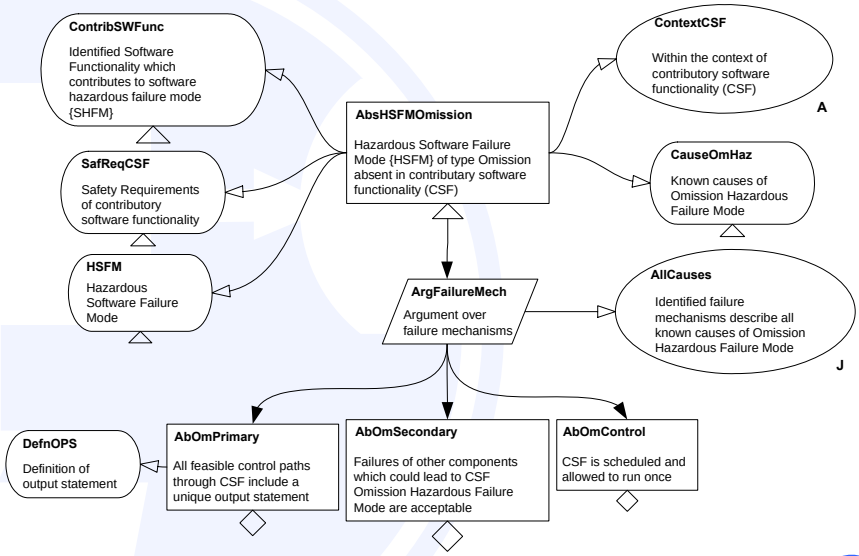
# Software Architecture



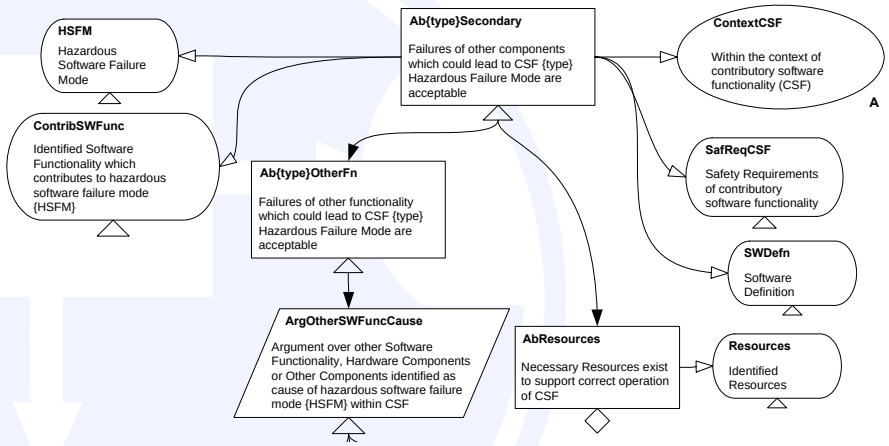





# Product Orientated Decomposition



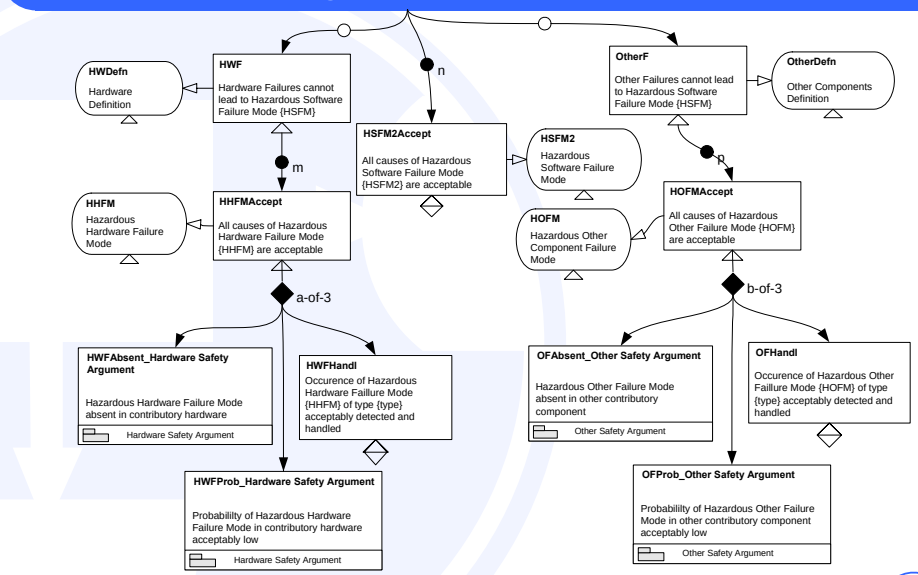
# Secondary Causes Decomposition








# Secondary Causes Decomposition 2



# Targeted Selection of Evidence

- IEC 61508

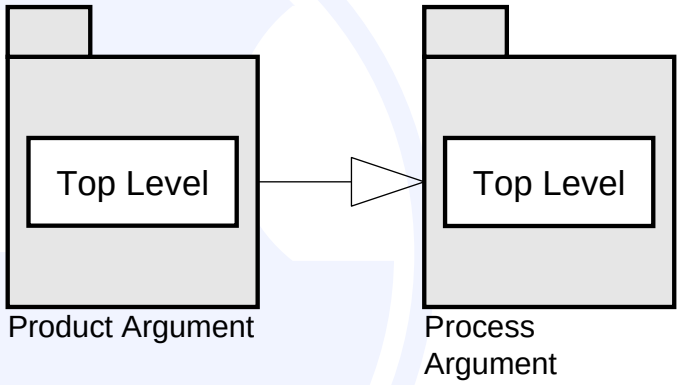
| Technique/Measure | Ref | SIL1 | SIL2 | SIL3 | SIL4 |
|---|---|---|---|---|---|
| 1 Formal proof | C.5.13 | --- | R | R | HR |
| 2 Probabilistic testing | C.5.1 | --- | R | R | HR |
| 3 Static analysis | B.6.4 Table B.8 | R | HR | HR | HR |
| 4 Dynamic analysis and testing | B.6.5 Table B.2 | R | HR | HR | HR |
| 5 Software complexity metrics | C.5.14 | R | R | R | R |
| Software module testing and integration | See table A.5 | | | | |
| Programmable electronics integration testing | See table A.6 | | | | |
| Software system testing (validation) | See table A.7 | | | | |

- Selection of techniques is difficult except with respect to the Safety Case Objectives
  - A problem with the current standards
  - Approach drives the argument down to low-level objectives first





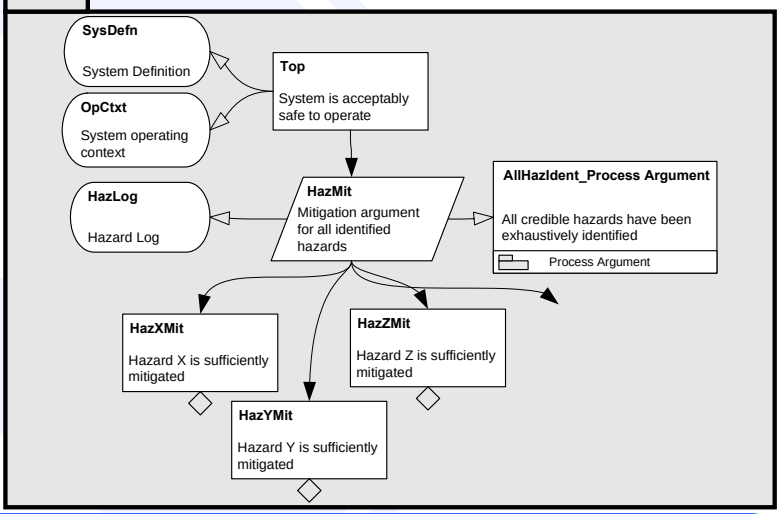
# Process and Product Argument 1






# Process and Product Argument 2

## Product Argument

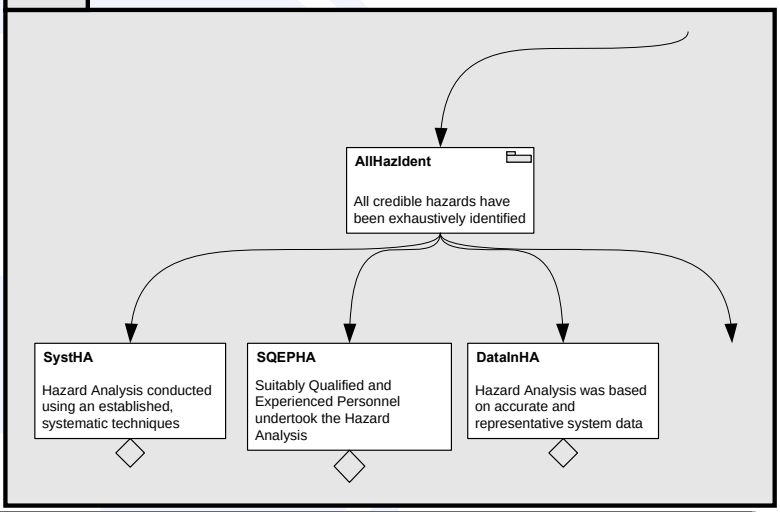







# Process and Product Argument 3

## Process Argument



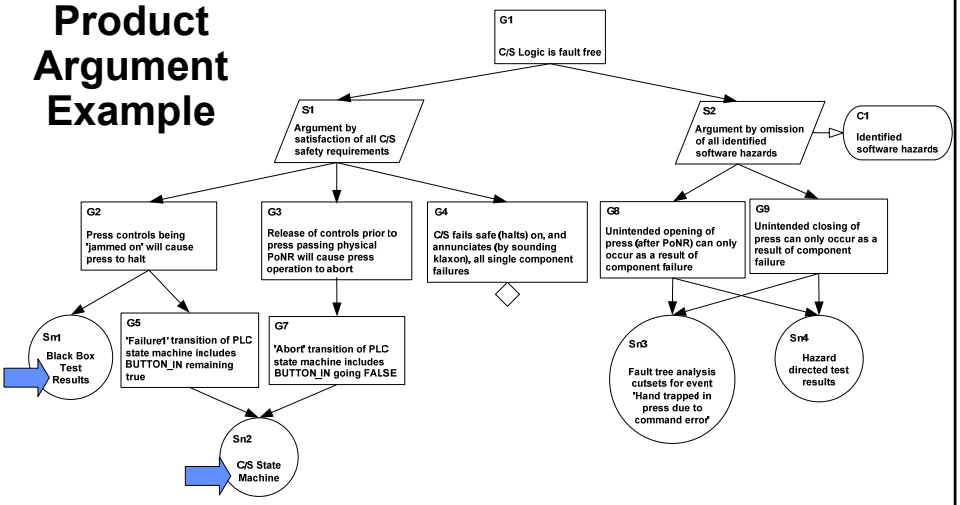
# Elements of Process Argument

- Traceability of artefacts
  - E.g. traceability of requirements to design, implementation and verification
- Competency of personnel
  - E.g. experts, practitioners or supervised
- Suitability and reliability of Methods
  - E.g. testing or analysis, formal or informal
- Qualification of tools
  - E.g. Development or verification tools, qualified or unqualified
- Suitability and clarity of notations
  - E.g. formal, informal or structured
- Independence
  - E.g. independence at the personnel and organisation level





# Product Argument Example




- Black box testing and state machine analysis provide explicit and independent evidence (solutions) directly related to system artefact rather than appealing to quality of development process



## Product Argument Example: Process Uncertainty

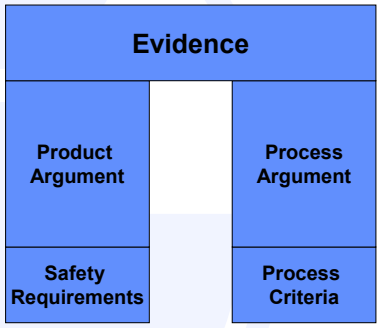
- Black box testing process
  - Is testing team independent from design team?
  - Is process of generating, executing and analysing test cases carried out systematically and thoroughly?
  - Is traceability between safety requirements and test cases well established and documented?
- State machine analysis process
  - How accurate is correspondence between the mathematical model and software behaviour at run-time?
  - Is analysis carried out by mathematically qualified engineers?





## Integrating Process Arguments into Product Arguments

- Process uncertainty should be addressed by linking process arguments to items of evidence used in product argument

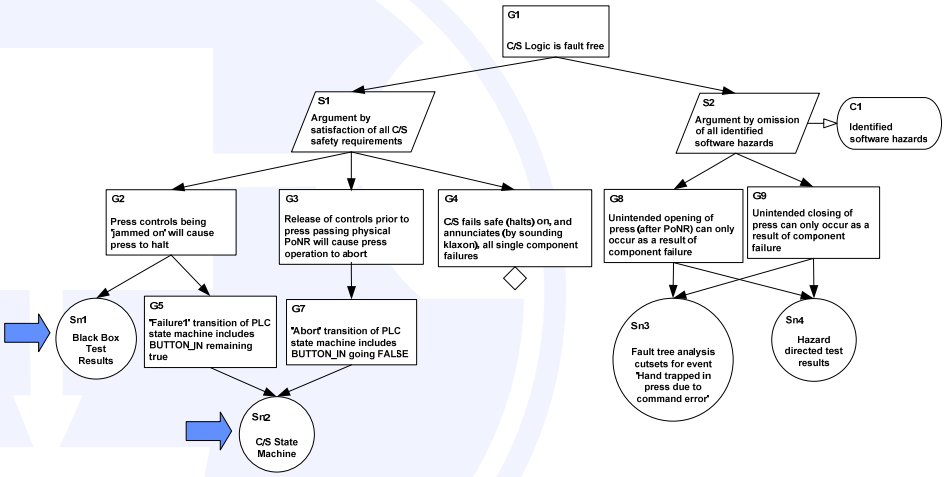



Useful in absence of prescribed process

Evidence-based ≠ Product-Based



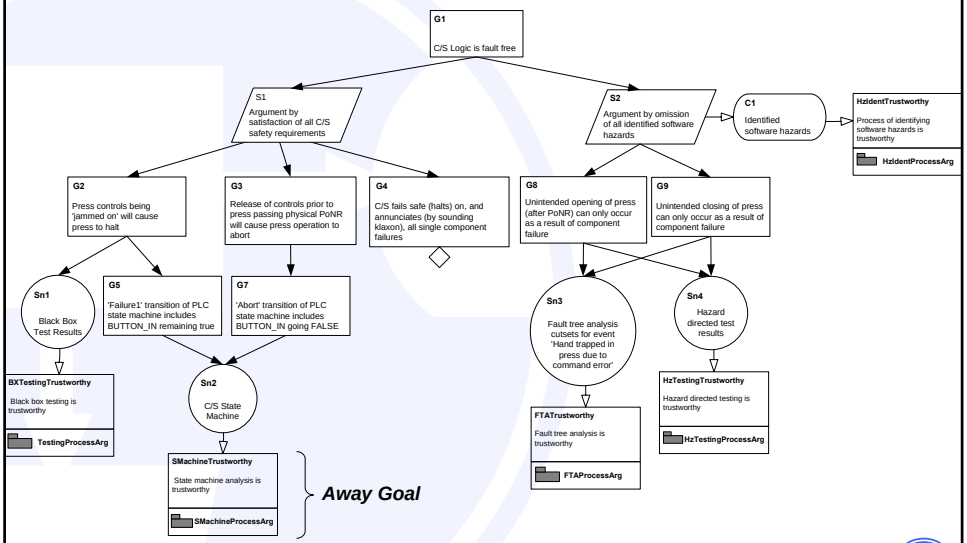
# Product Argument Example Revisited

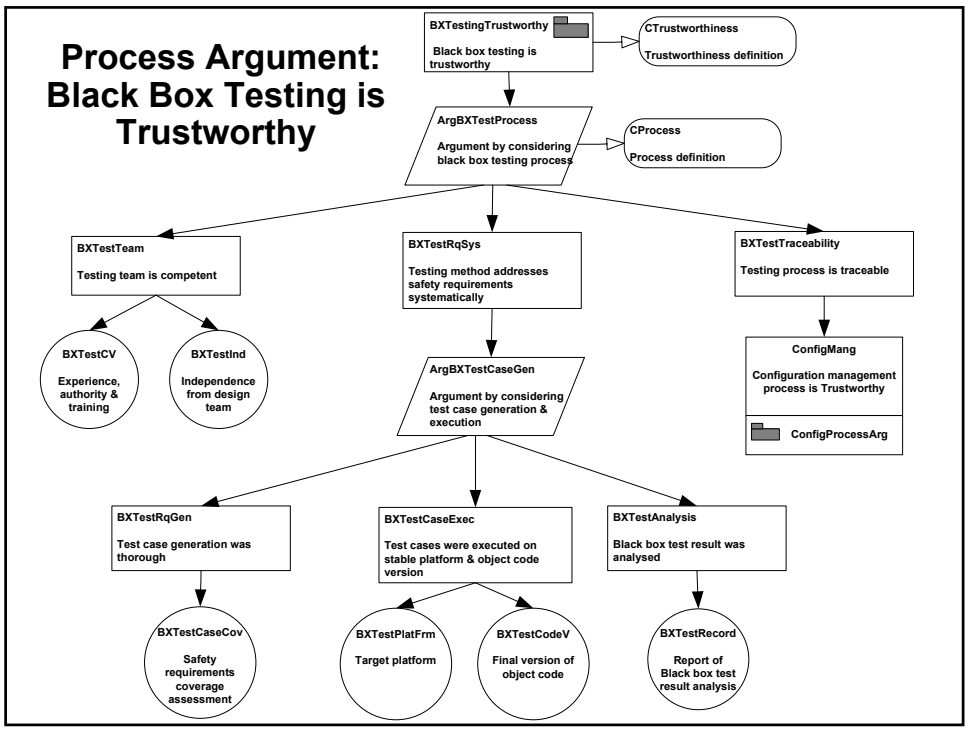







# Product & Process Argument in Modular GSN






## Summary

- Generic Computer System Safety Argument shown
- Both process and product elements required
  - However, historically perhaps over-emphasis on process
- Evidence-based S/W Safety Argument Framework
  - Traceable to system-level hazards
  - Based upon taxonomy of software failure mode types
  - Validation, Satisfaction and Traceability
  - Patterns of required arguments
  - Leading to targeted evidence selection
- Linking Process and Product Arguments
- Importance of a Structured Approach to Assurance Case Construction and Presentation!