The 4+1 Software Safety Principles and their relation to building safety cases

Richard Hawkins and Tim Kelly

Department of Computer Science

The University of York

What are the 4+1 Software Safety Assurance Principles?

Why 4+1?

4+1 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

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
  - …
  - <u>Software</u>
- Need to ensure that we have identified, understood and captured the potential contribution of software to system level hazards

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 …

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

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
- Consequence of earlier principles
  - Want specific evidence for specific safety requirements
- This is the Verification issue

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 <u>anything else </u>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
    - Concerned with where design is unsafe (under some conditions)
    - Reconsideration of the behaviour of the design
  - 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

Principle 4+1

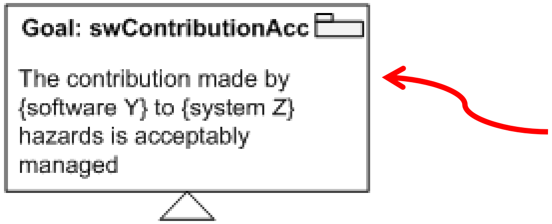
- Perfect assurance of the achievement of the other principles is desirable, but unachievable
  - e.g. consider Principle 1, we cannot *prove* that the safety requirements are complete
  - Not even if “money no object”
- Instead, we must consider when is enough enough?
- Really a system principle
- Some challenges applying to software

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

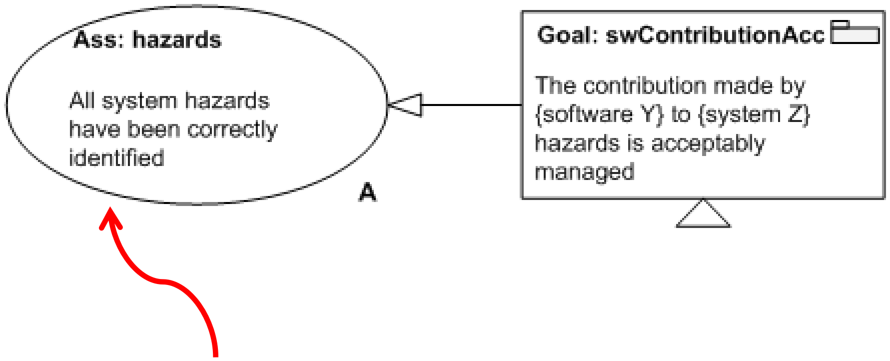
4+1. The confidence established in addressing the software safety principles shall be commensurate to the contribution of the software to system risk

Principle 1



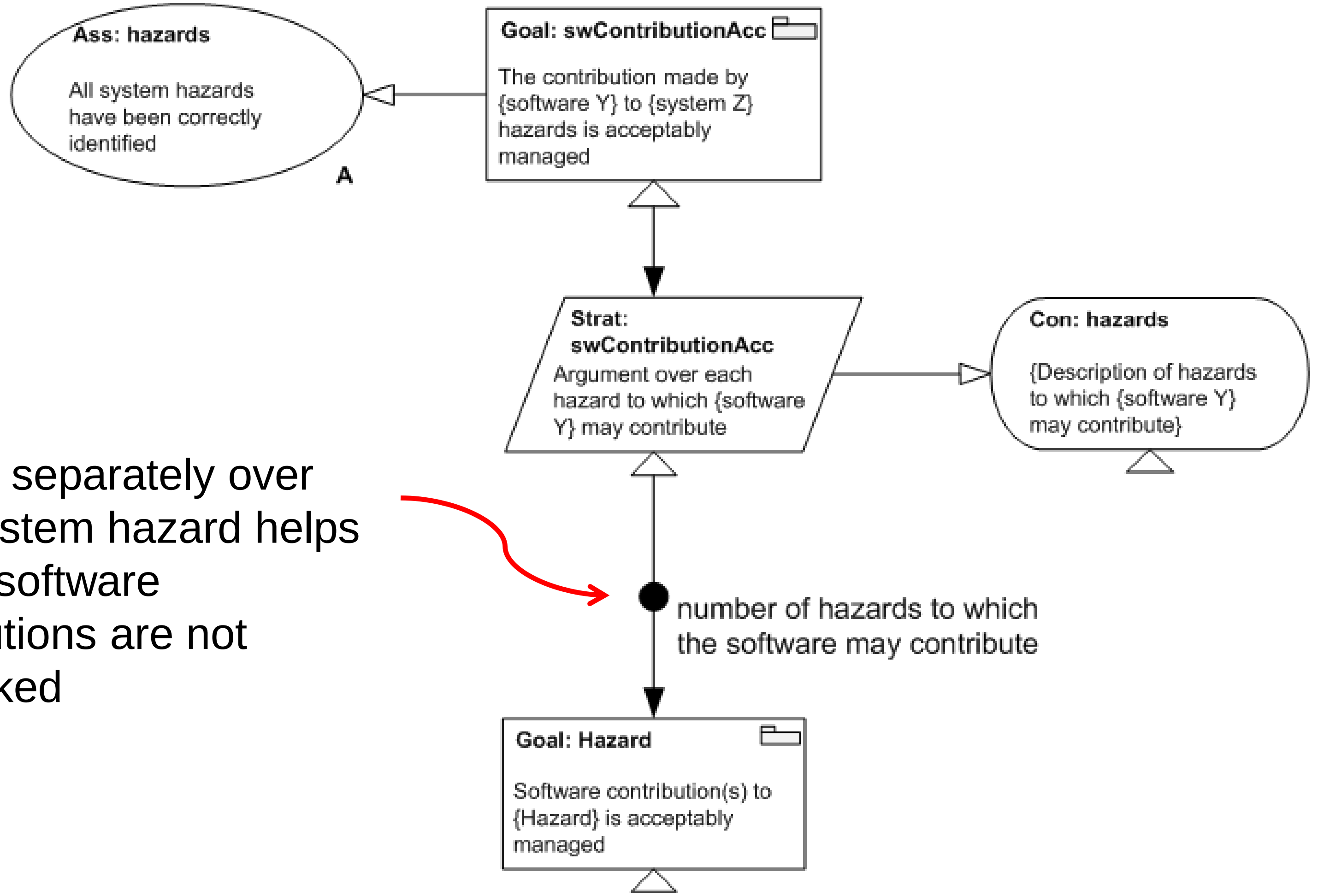
Must be able to argue that any contributions the software could make to system hazards are managed (through SSRs)

Principle 1



We need to know what all the hazards are at the system level – this is not a software issue (part of system safety process)

Principle 1

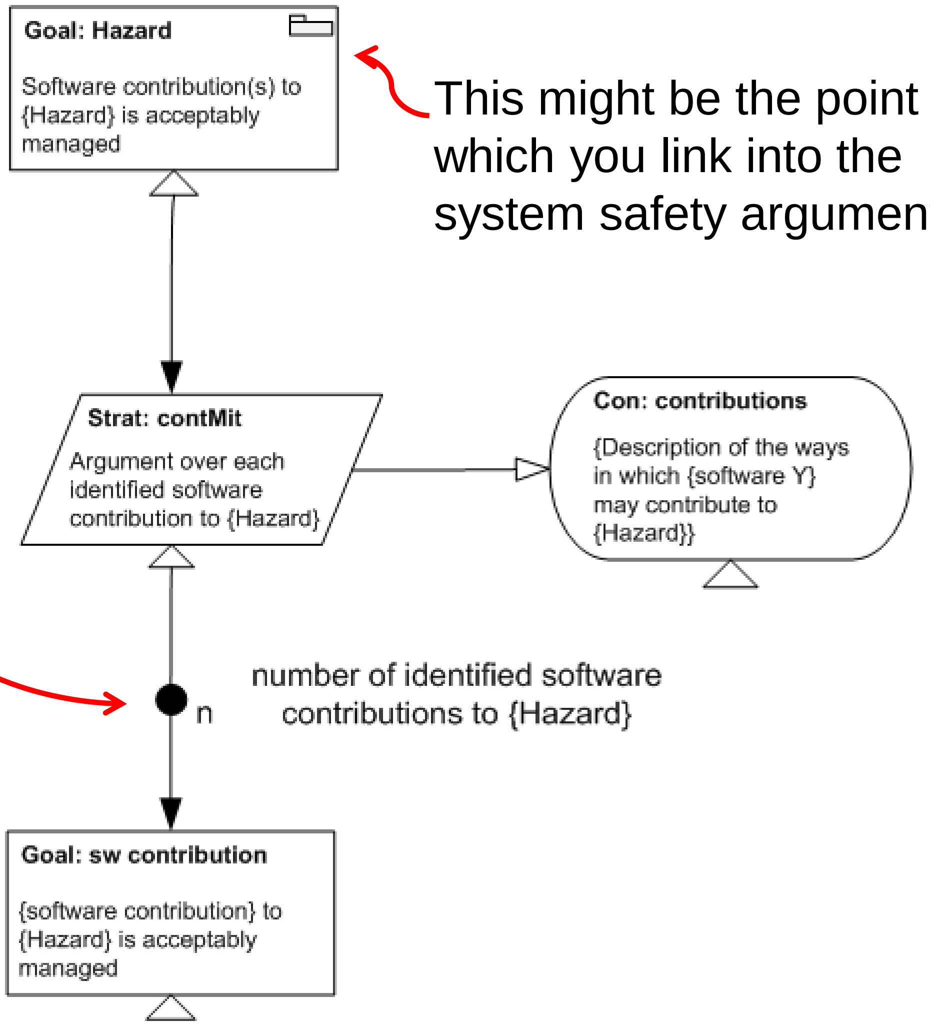


Arguing separately over each system hazard helps ensure software contributions are not overlooked

Principle 1

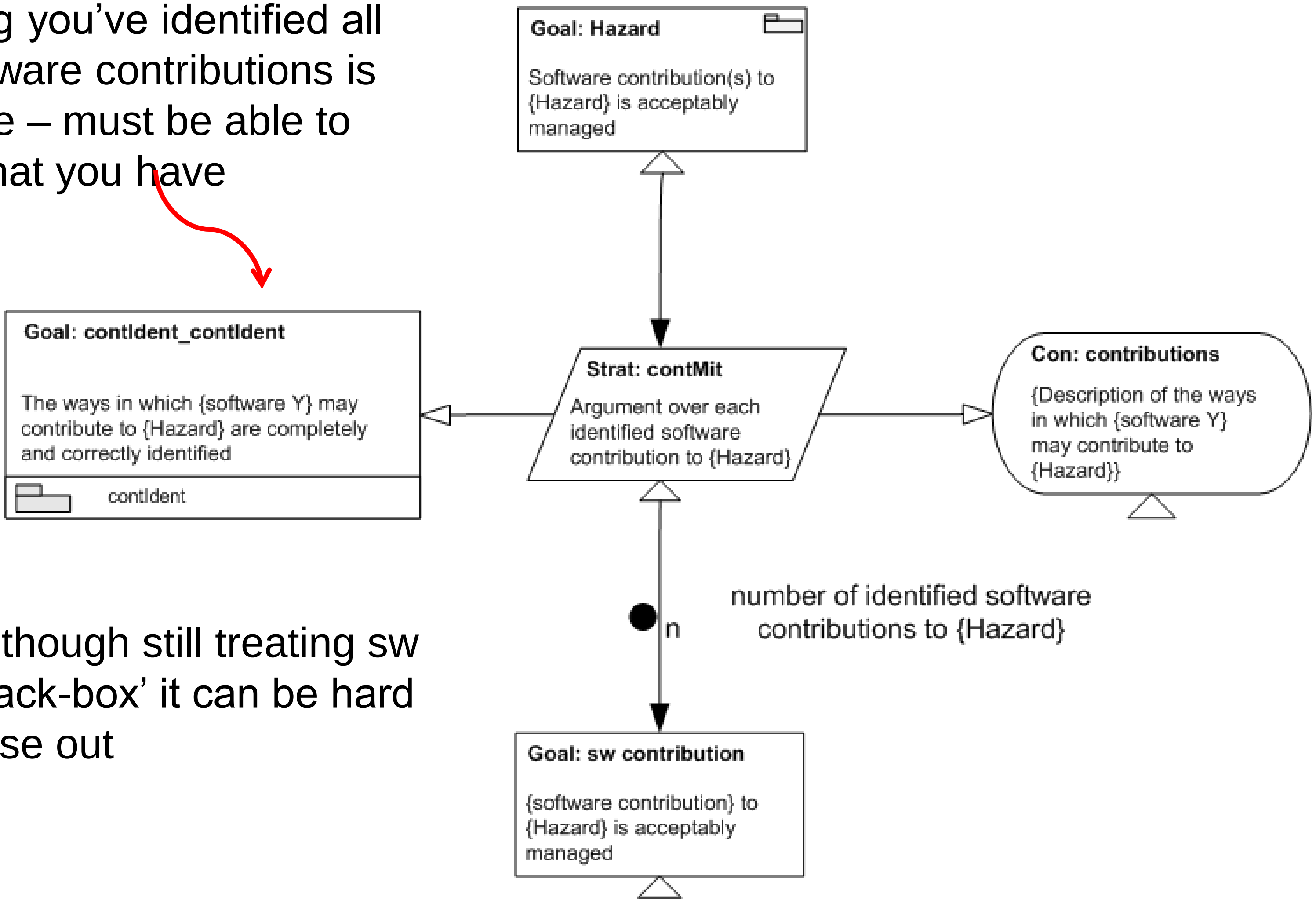
This might be the point at which you link into the system safety argument

- Then argue for each system hazard that each software contribution to that hazard is managed



Principle 1

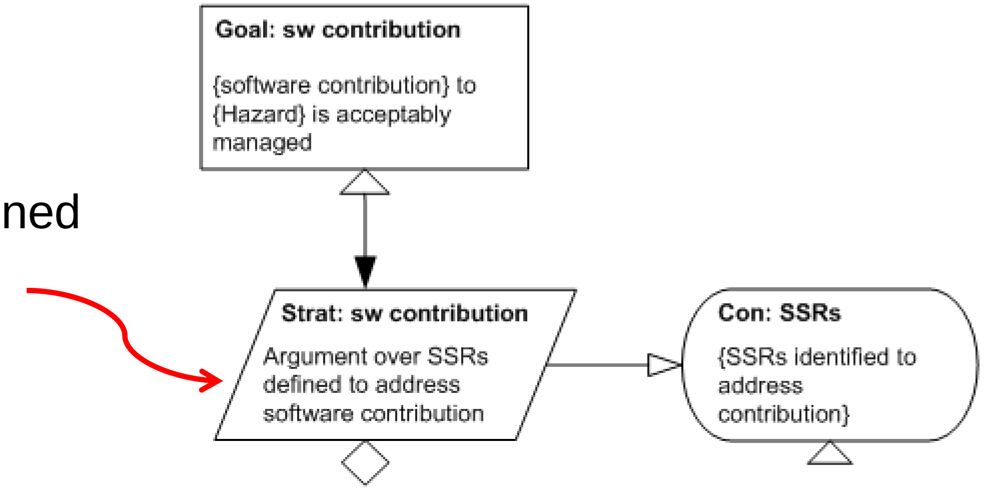
Knowing you've identified all the software contributions is key here – must be able to argue that you have



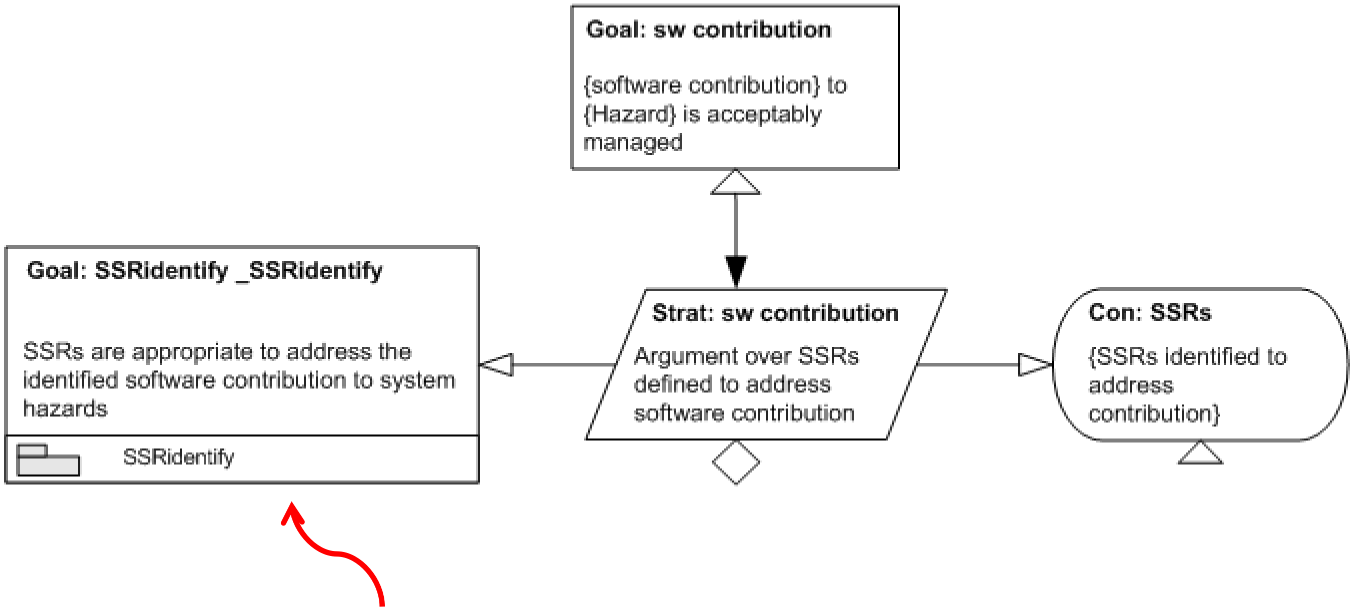
Even though still treating sw as 'black-box' it can be hard to tease out

Principle 1

We address each contribution through defined Software Safety Requirements (SSRs)



Principle 1



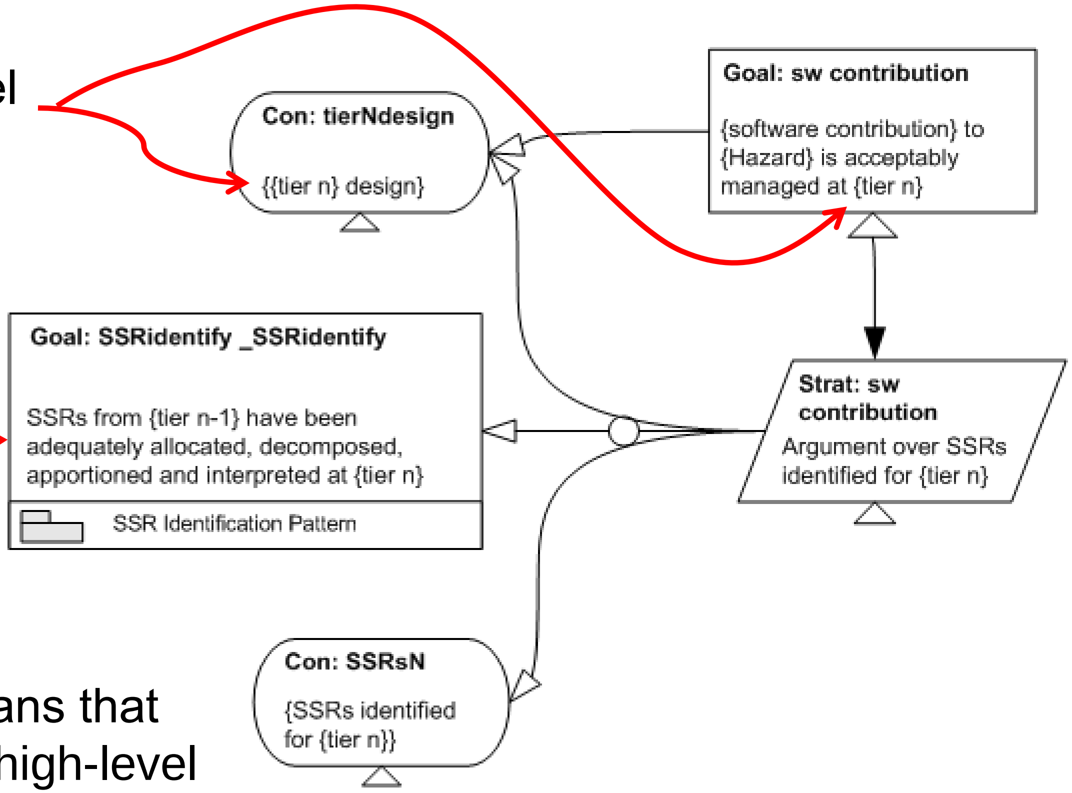
Need to be able to argue that the SSRs you've defined are appropriate to manage the contribution to the hazard

Note that these requirements are at the software – system boundary

Principle 2

We need to be able to show that the SSRs are correct not just at the top level, but at each level of software design decomposition

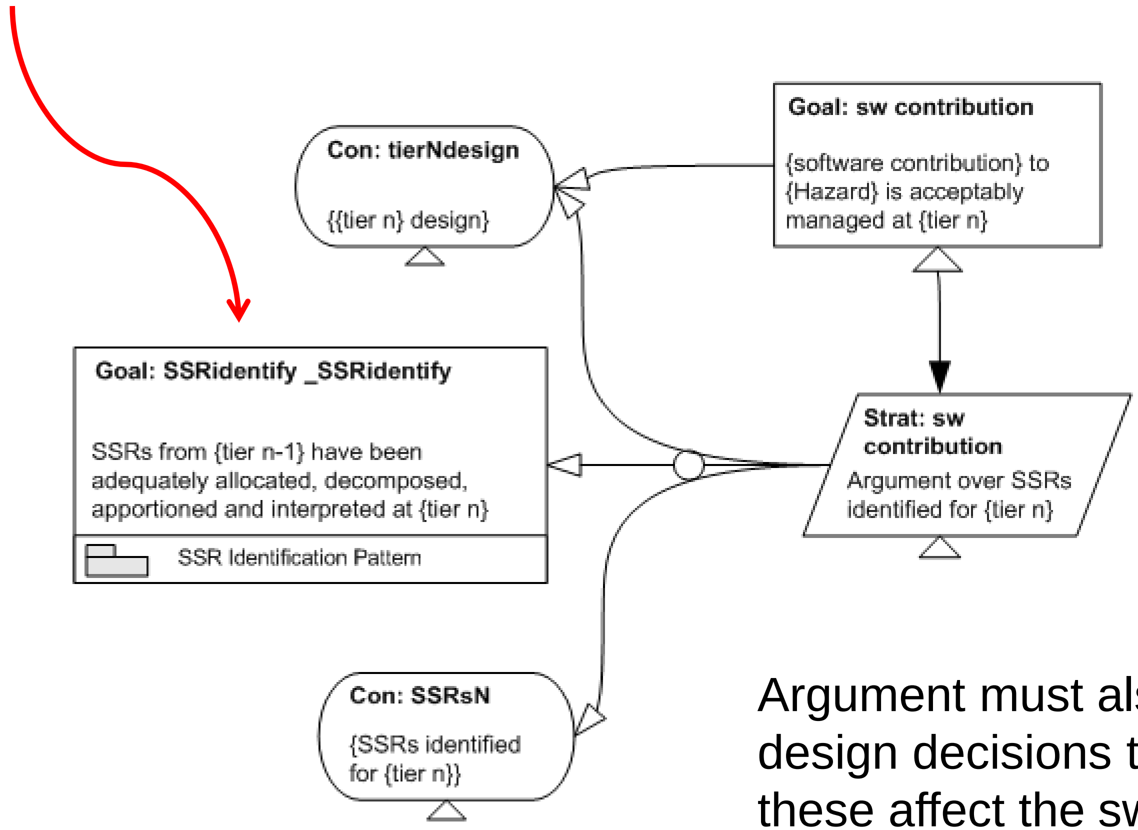
A ‘tier’ is one level of design decomposition



‘Adequately’ means that the intent of the high-level SSRs has been maintained

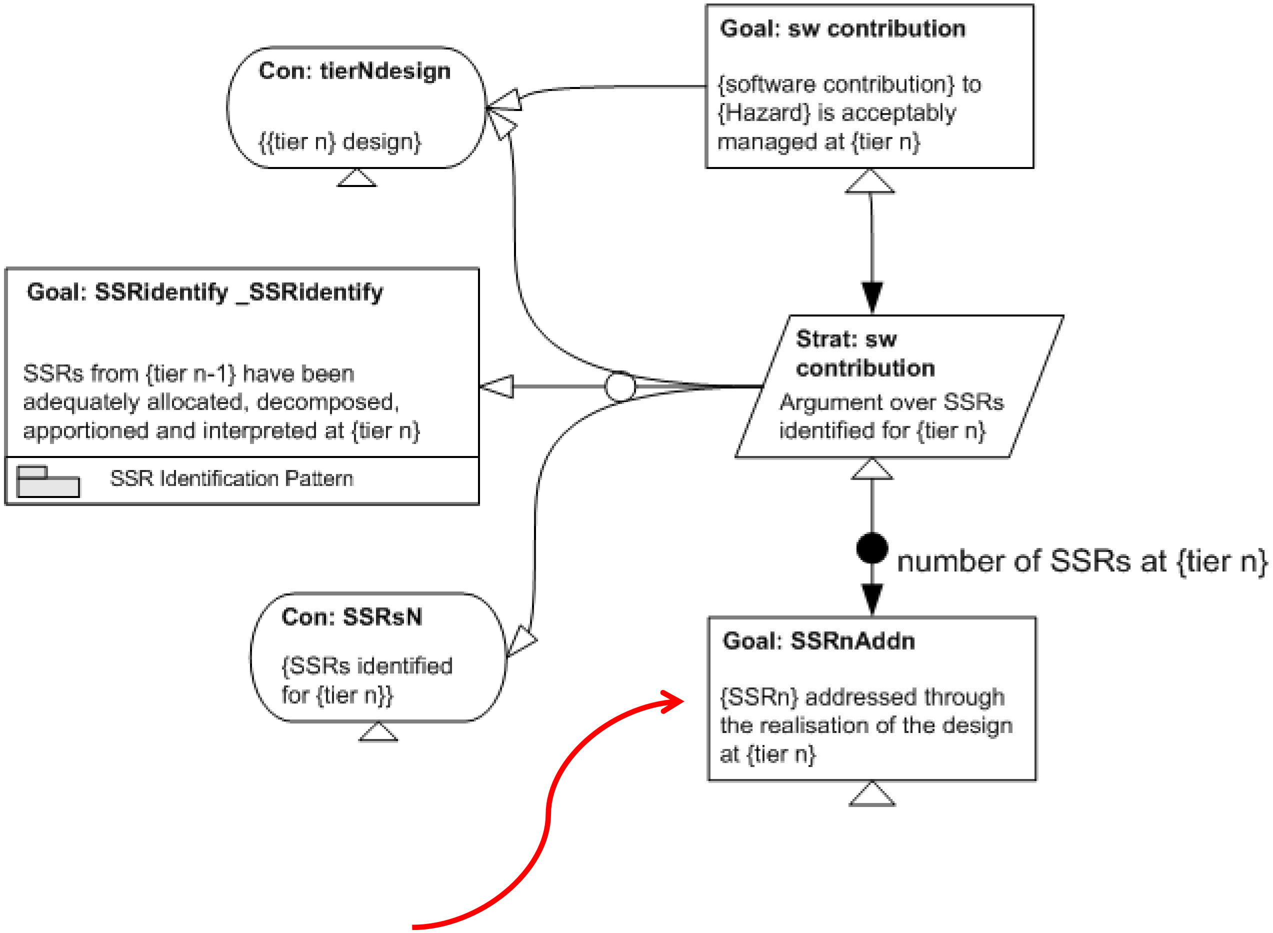
Principle 2

This is more than just a traceability argument – must demonstrate that the behaviour is equivalent – more akin to what some people call "rich traceability"



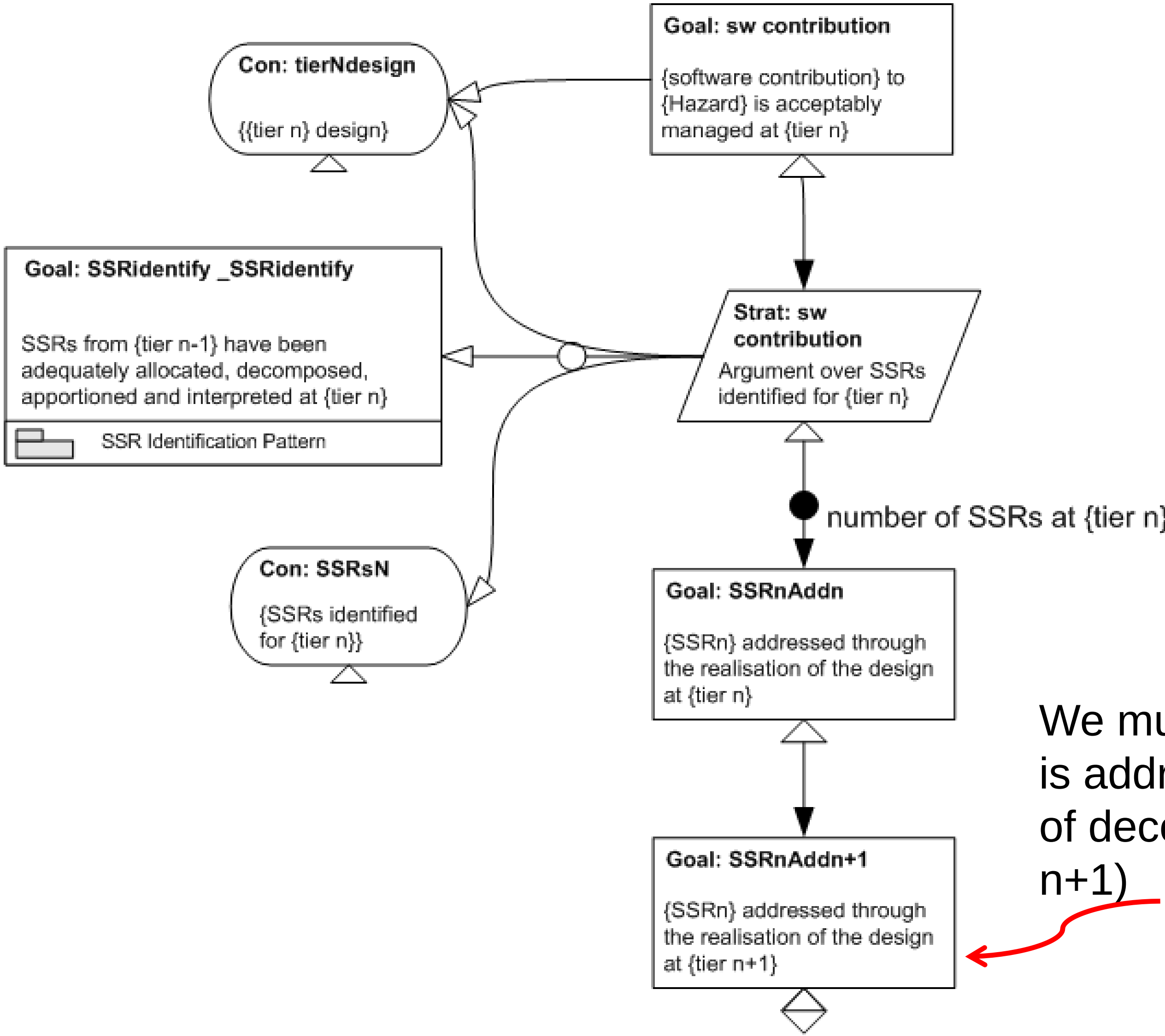
Argument must also consider the design decisions themselves – do these affect the sw ability to meet SSRs?

Principle 2



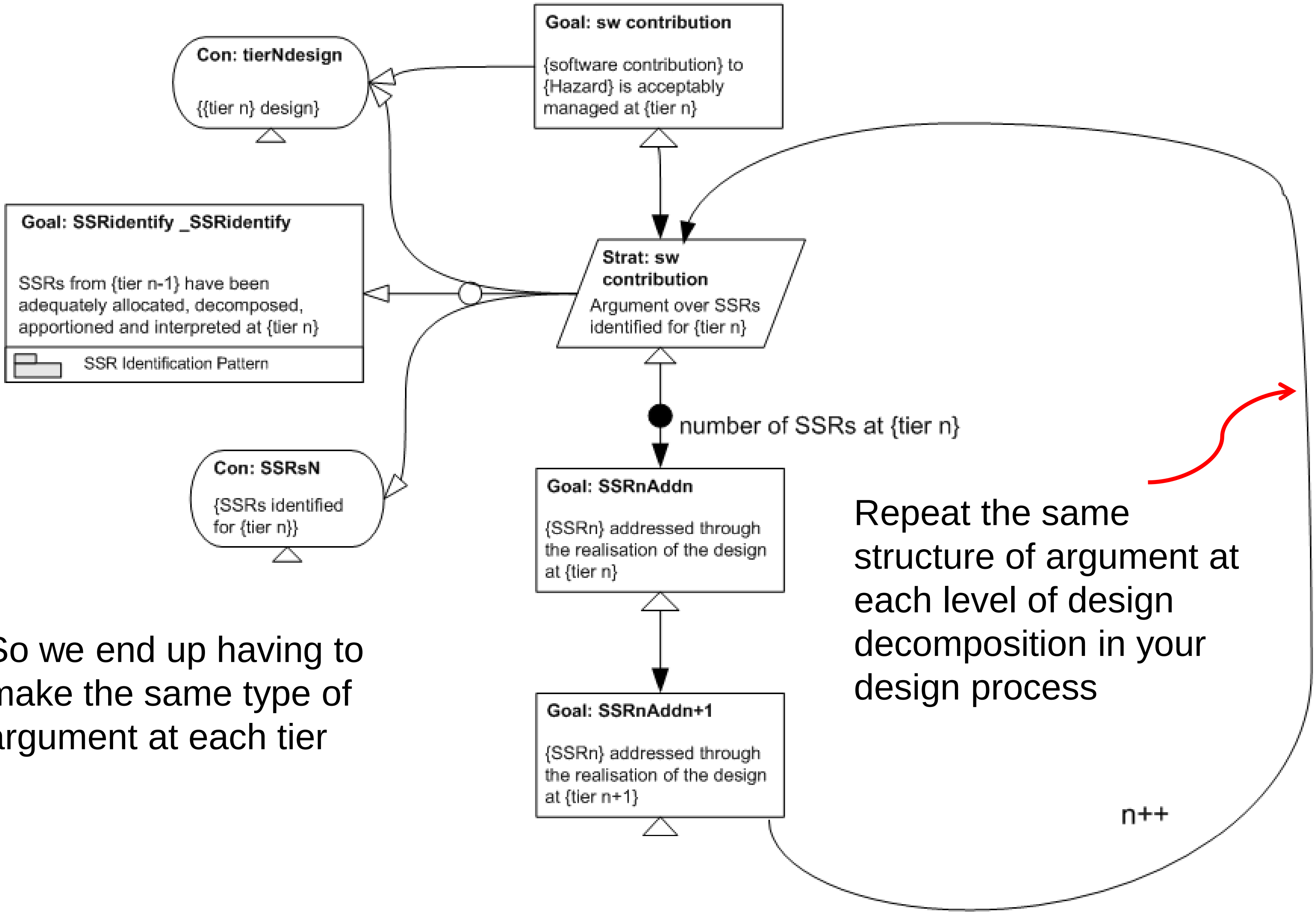
We need to argue for each SSR at each software design tier

Principle 2



We must show that the SSR is addressed at the next level of decomposition as well (tier n+1)

Principle 2

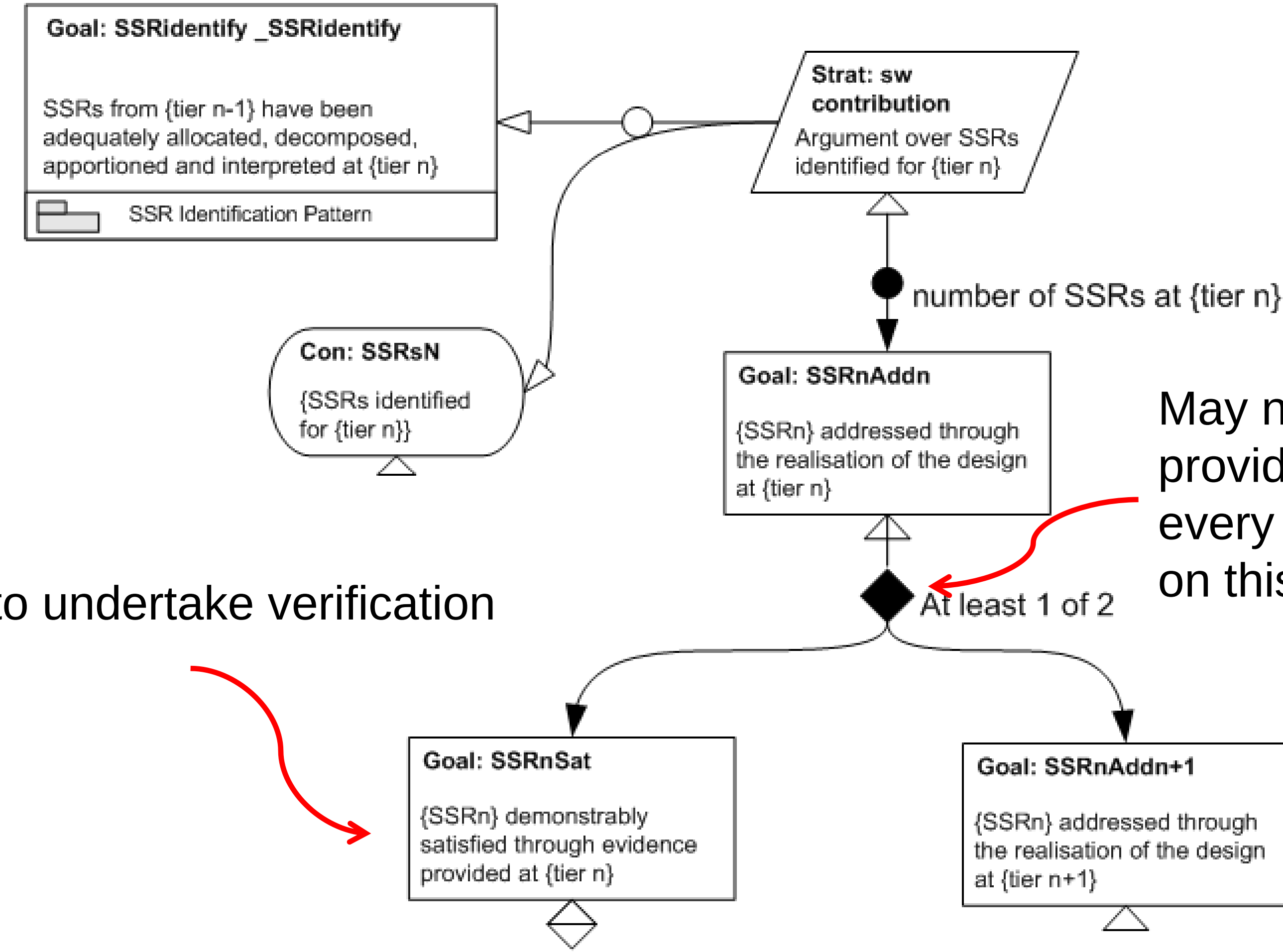


Repeat the same structure of argument at each level of design decomposition in your design process

So we end up having to make the same type of argument at each tier

Principle 3

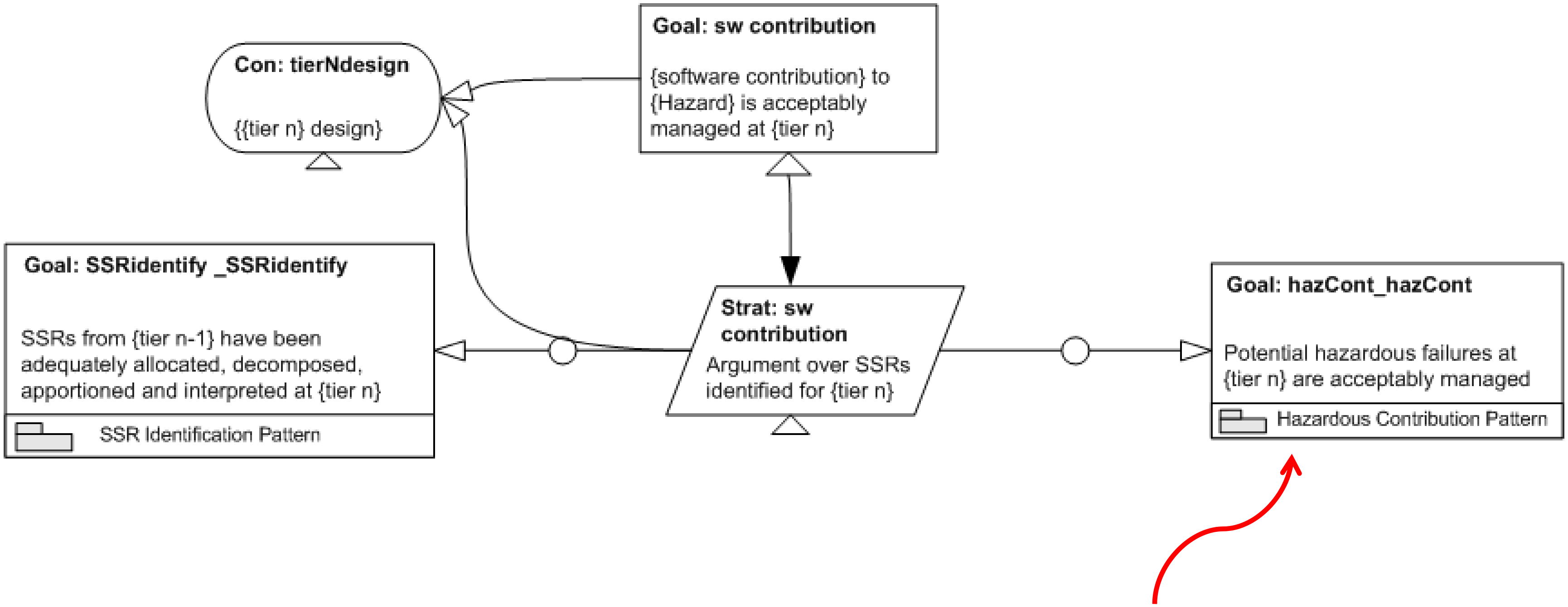
We need to demonstrate the SSRs are satisfied



May not always provide evidence at every level – more on this later

Potential to undertake verification at any tier

Principle 4

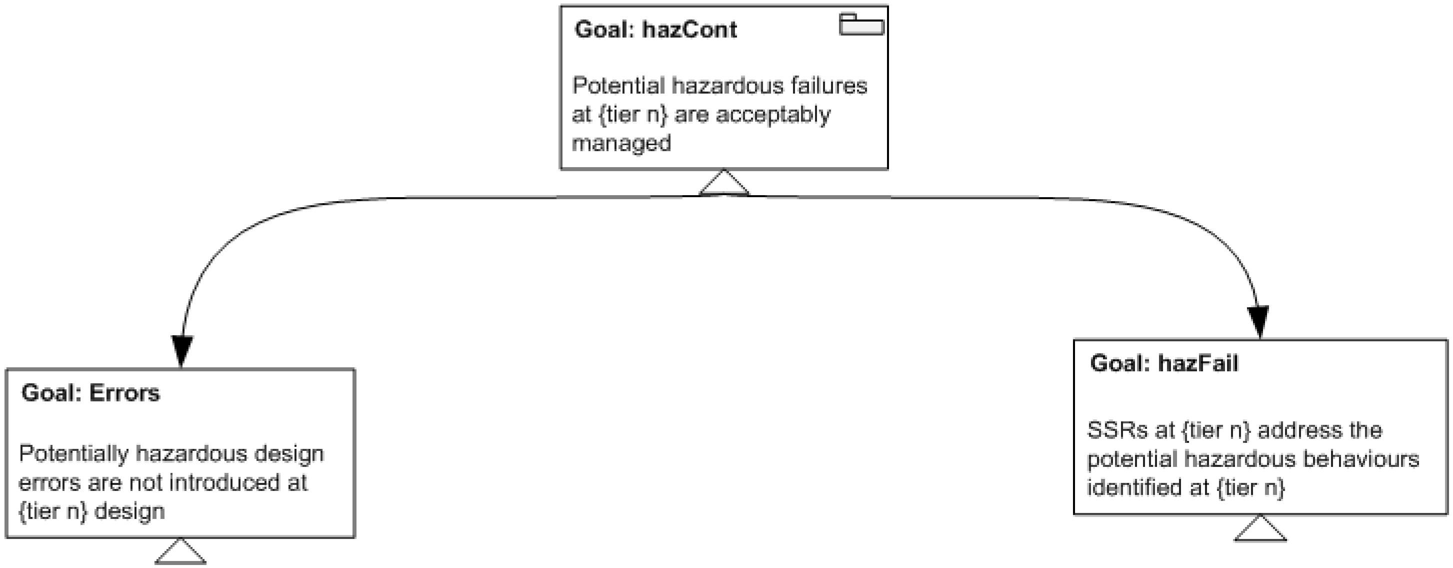


We need to be able to argue that we are managing hazardous behaviour at each level of design

How might we argue this?

Principle 4

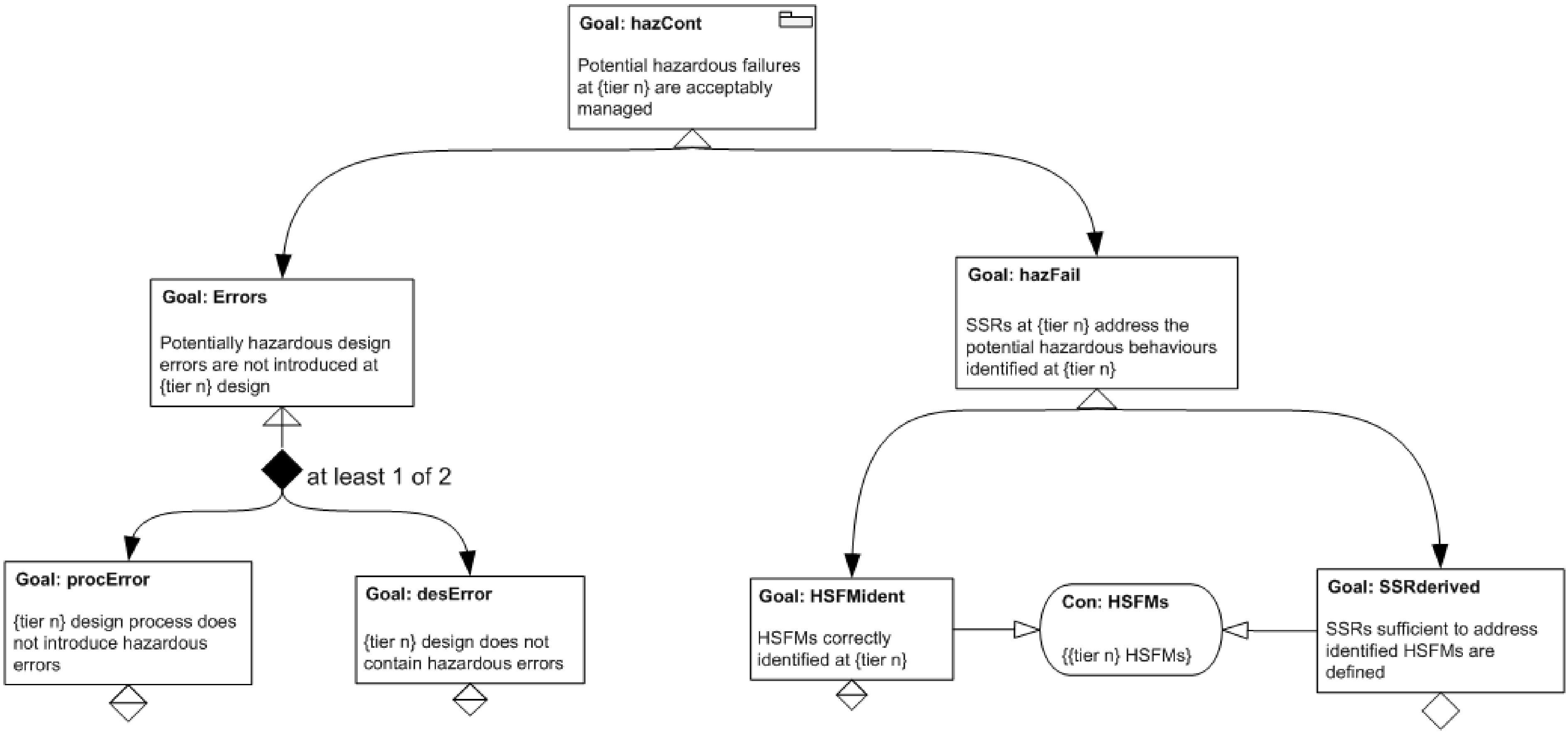
Argument should consider two things



Systematic errors introduced at this step in the design process

Unanticipated behaviours and interactions arising from the software design decisions at this tier (mitigated through additional SSRs)

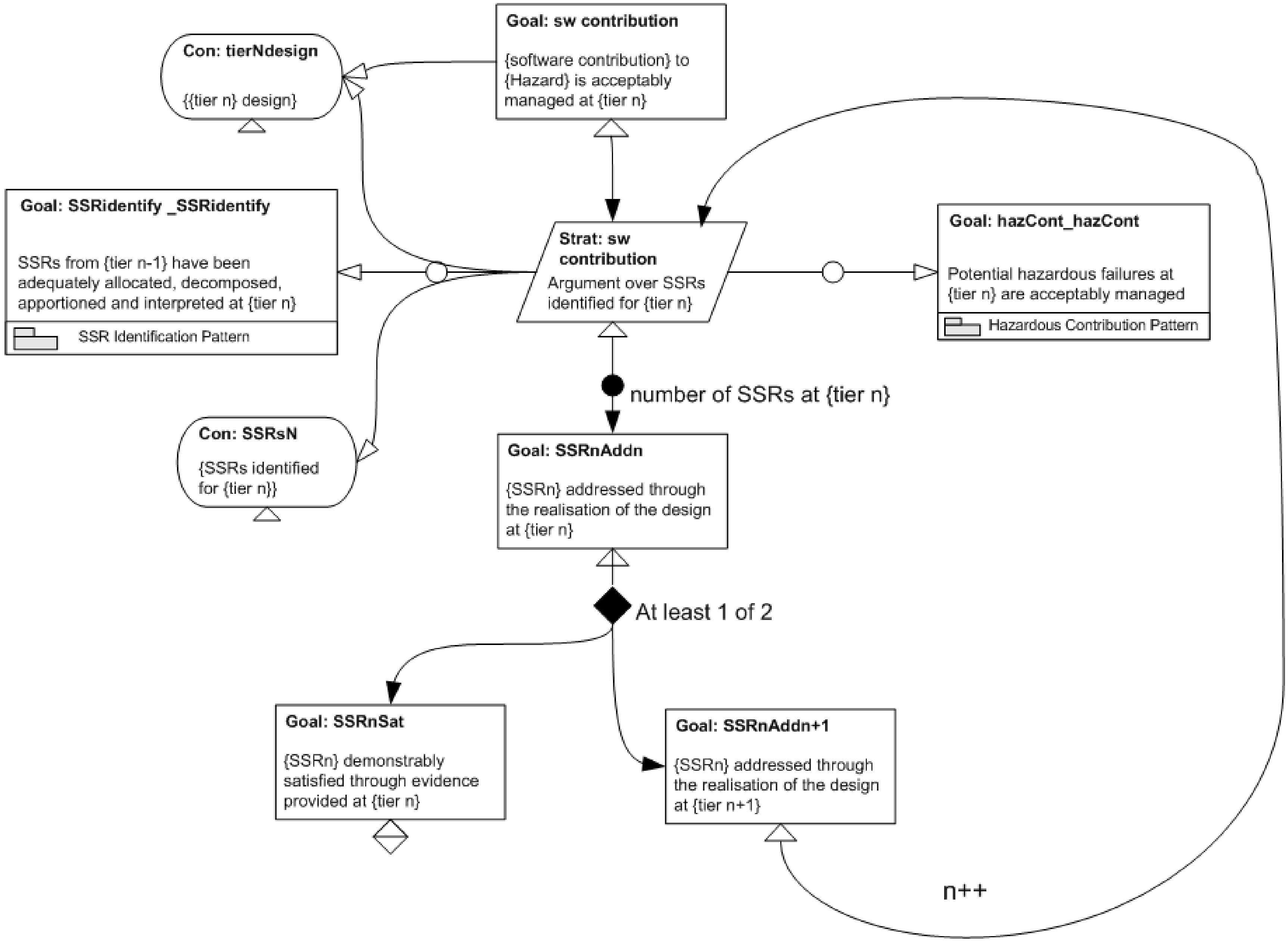
Principle 4



Control the development process but also check your design!

Mitigation through design or requirements

Bringing 4 Principles Together



Principle 4+1

- Must be able to demonstrate in the software safety argument that:
  - ***confidence*** with which the principles have been addressed is commensurate to the contribution to system risk
- This requires the provision of a ***confidence argument***
  - Confidence argument documents reasons for having confidence in the main (software) safety argument
- Confidence is ultimately established through the provision of evidence to support claims made in safety argument
  - Evidence required for *all* of the principles (not just satisfaction)