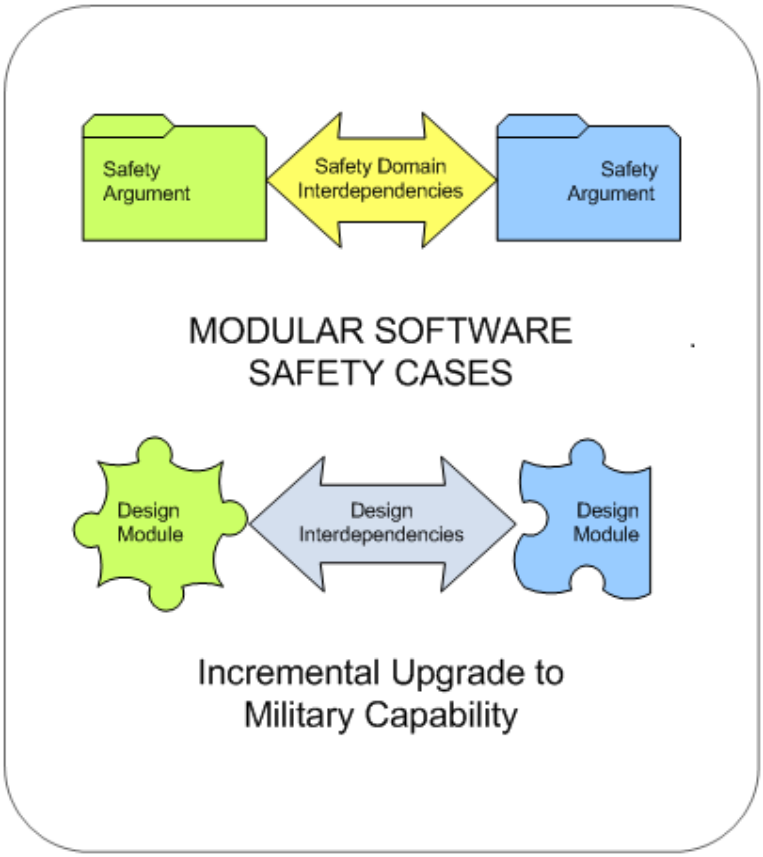
Modular Software Safety Case Process Description

Date: 19-Nov-2012



Contents

Figures

Tables

1 Introduction

1.1 Purpose of The MSSC Process

The purpose of the Modular Software Safety Case (MSSC) Process is:

> ***To provide instruction on the organisation of a Software Safety Case in a beneficial modular arrangement so that it may provide a compelling argument that a body of evidence for the software system sufficiently demonstrates its safety aims.***

Top level introductory material about the process is presented in [101]. Those seeking an executive summary of the process, its applicability and its supporting documentation are recommended to consult that document.

1.2 Purpose and Scope of This Document

This document defines the MSSC Process, which is a method for developing modular Safety Cases which, though developed in the software safety context specifically, is aimed at general applicability. The process is presented in a form suitable for consumption by engineers and practitioners in Systems, Software and Safety domains. The document introduces specific terminology and concepts. Section 2 provides a readable explanation of concepts and their relationships in the MSSC Process.

Applying the process successfully requires some background knowledge of techniques and specific definitions which are documented in the accompanying material listed in section 1.3.

In this document:-

Section 1 contains introductory material, including this subsection.

Section 2 presents relevant concepts.

Section 3 is an overview of all eight process steps.

Sections 4 to 11: each of the eight process steps and any sub-steps are explained in detail, with referrals to Appendix A example material.

Section 12 describes managerial activities that may be carried out in support of the process.

Appendix A: presents a simple example system, its Safety Case Architecture and an incremental change that affects its safety requirements.

1.3 Accompanying Documents and References

Introductory and executive summary material:

[101] MSSC 101 – MSSC Process Overview, Current Issue Applies

Documents that accompany this process description:

[202] MSSC 202 – MSSC Glossary, Current Issue Applies

[203] MSSC 203 – MSSC GSN, Current Issue Applies

[204] MSSC 204 – MSSC Artefacts, Current Issue Applies

2 MSSC Concepts

2.1 Key Concepts

A **Safety Case** presents an argument supporting the top claim that the System is safe to operate in the stated context. The Safety Case describes the system from the safety perspective.

This safety focused view of a system and the corresponding subset of system properties is called the **Safety Domain** in order to distinguish it from the implementation focussed perspective, or **Physical Domain.**

A **Modular Safety Case** is one that is presented as a loosely coupled set of cohesive Safety Case Modules.

The structure of a Modular Safety Case maximises the containment of changes to the SC over the long term system lifecycle. It will tend to reflect the modularity of the system components in the Physical Domain.

A **Safety Case Module** provides one or more public claims supported by evidence, which may be used in support of other Safety Case Modules, ultimately supporting the top claims of the completed Safety Case. A Safety Case Module will set out claims that it depends upon in order to assure its own claims. These are represented as **Goals** *requiring support*.

A **Safety Case Architecture** is an arrangement of Safety Case Modules that collectively support one or more top-level goals.

Block

A grouping of one or more elements from the Physical Domain that is to be the subject of a dedicated Safety Case Module is referred to as a **Block**. A Block defines the scope of a Safety Case Module's 'footprint' in the Physical Domain.

A **Block** may correspond to more than one component within the physical domain.

There are several kinds of Safety Case Module, key examples being:

- **Block Safety Case Module** – This argues that a Block's allocated safety-related requirements are satisfied. Block Safety Case Modules form the foundations of the Safety Case Architecture and provide the base part of the argument that the Block related evidence supports the system's safety-related claims.
- **Integration Safety Case Module** – This provides supporting argument where the required behaviour is the result of combining the behavioural attributes of a group of Blocks.
- **Safety Case Contract Module** – This is used to formalise and argue the satisfaction of dependencies between modules where the relationship between the claims requires some justification. The arguments in a Contract Module justify why the **Claims** providing support satisfy the **Claims** requiring support in a specific instance.

Safety Case modularity provides *separation of concerns* amongst modules by encapsulating their safety arguments, with the **Public Interfaces** of a Safety Case Module defining both safety objectives and required support. This arrangement supports **Change Containment** such that, providing the Public Interfaces remain unchanged, the way in which the **Public Claims** are supported is independent of other Safety Case Modules.

Public Interface

Public Interfaces of Safety Case (SC) Modules are defined in terms of:

- Provided Support:
  - **Claims** that state what the SC module provides, including claims that capture **Guarantees** about the behaviour of the implementation.
  - **Context** including assumptions that state the external factors and conditions that may affect the validity of the arguments within the SC module and is relevant to the support it provides.
- Required Support:
  - **Claims** that need to be supported by other SC modules in the Safety Case including those that capture **Dependencies** from the implementation domain.
  - **Context** including assumptions that state the external factors and conditions that may affect the validity of the arguments within the SC module and is relevant to the support it requires.

Note: This description is independent of the symbols that may be used to represent the public interface graphically, for example using a notation such as GSN.

The declared Public Goals offered by a Safety Case Module will generally be conditional on there being no external influences that may interfere with a Block's ability to guarantee its behaviour or intrinsic properties. This will require that there are **No Unwanted Interactions** between Blocks. This in turn may require that each Block guarantees to be **Well Behaved**, meaning the Block implementation does not break architectural rules. It may also be possible to argue that unwanted interactions are actively prevented. A combination of both approaches may provide the strongest assurance.

The relationships between Safety Case Modules are often derived from corresponding relationships between Blocks in the Physical Domain. Where one Block satisfies the requirements of another Block it may offer a set of **Guarantees** which may derive from behavioural or intrinsic properties. The ability of a Block to provide such Guarantees may be conditional on satisfaction of certain **Dependencies**. An association of Dependencies to a Guarantee *within a Block* is referred to as a **Dependency-Guarantee Relationship** or **DGR**.

The relationship *between* Blocks, where one Block's Dependency is satisfied by another's Guarantee may be captured as a **Dependency-Guarantee Contract** or **DGC** and referred to in a Safety Case Contract Module.

Validity of the DGRs within and DGCs between Blocks

For a Safety Case Module to make a claim assuring a Guarantee provided by a Block it will need to be able to claim that all the Block Dependencies have been satisfied. During MSSC Impact Analysis the association of Guarantees with a specific set of Block Dependencies in a DGR is used and may be relied upon for the correct identification of the impact of a change.

Similarly when Safety Case Modules are integrated to form a complete Safety Case, all inter SC module dependencies need to be satisfied, including the inter-Block dependencies.

For any system, the expectations for its evolution over its service life should be captured in a Lifecycle Plan. Based on this plan, a set of **Change Scenarios** is derived. It is good system design practice to engineer a system in the Physical Domain such that expected changes can be implemented with minimum rework. Similarly the Change Scenarios should guide the development of the Safety Case Architecture so that the impact of any change is minimised.

When a system is changed a Safety Case **Impact Analysis** is undertaken to determine how the Safety Case is affected. This analysis will take as its starting point a set of **Driving Changes** that are made during the development of the system increment that impact the Safety Case. The result of a change will be a new Safety Case comprising unchanged elements from the previous Safety Case and new or modified elements. The MSSC Process

produces a **Change Argument** that justifies why the unchanged Safety Case Modules do not need to be revisited, and why the updated Safety Case continues to encapsulate any necessary safety properties expressed in the previous Safety Case.

The MSSC Process presumes that safety arguments are presented using **Goal Structuring Notation** (**GSN**) with its notions of Goal, Strategy, Context, Assumption, Solution – supplemented by modular GSN extensions, which include such notions as Public Goal, Away Goal, and Module reference. For further information about modular extensions to GSN refer to [203].

Use of MSSC with other notations

Although the MSSC Process has been described using a derivative of Modular GSN, the concepts and approach to generation of modular software Safety Case are transferable to other notations.

2.2 Supporting Concepts

It is recognised that there are different approaches to producing Safety Cases, one example being **Process Focussed**, and another **Product Focussed**, the latter emerging as a preferred approach because it directly addresses the mitigation of hazards.

Argumentation in Process Focussed and Product Focussed Safety Cases

A **Process Focussed Argument** offers claims, supported by evidence, that a system will be safe because it is the result of properly qualified people following adequate processes with sufficient rigour.

A **Product Focussed Argument** is based on the premise that hazard analysis has identified all relevant hazards and specified mitigations for each; then that each mitigation requirement is fulfilled in the final system. The presentation of a Product focussed Safety Case offers claims, supported by evidence, that each Hazard is mitigated and / or that each safety related requirement is implemented. As safety related requirements are allocated through-out a system architecture so may the corresponding claims be distributed over a **Block** based **Safety Case Architecture**.

In addition to the three types of module described in the previous section, the following types may also be appropriate.

- **Software Safety-related Requirements** or **"SSR"**
  The top claim of a Product Focussed argument is that a system can be considered "safe" (in the stated context of usage) if it fulfils all its Safety Related Requirements. An SSR SC Module may be used to argue that all Safety Related Requirements are supported by guaranteed behaviour provided by the Blocks

- **Configuration Data**
  In a system that is configured by data it will be necessary to argue that the configuration data accessed by the system is both valid and correct. This is most easily handled as the subject of a Safety Case Module.

- **System Wide Issues**
  In any system there are inevitably certain aspects of behaviour or performance that cannot be assured by a single Block or argued by a single Block Safety Case Module. Such aspects must be argued at a higher level, often at a system-wide level. End to end latency is an example of this where several Blocks may contribute to latency but none has total responsibility for it.

- **Hardware Safety Case Wrapper**
  The operation of any software system will be dependent on hardware. Where it is necessary to make a safety claim for such a dependence on hardware this can be achieved by providing a wrapper Safety Case Module that supports the claim that the specific hardware is both available and sufficiently dependable.

It is often helpful to both the producer and reviewer of a Safety Case to establish generic forms of argument that can be reused in commonly occurring situations.

Patterns and Templates

- **Argument Patterns**
  An argument pattern is an example of a form of augment that may be tailored and used in different situations. MSSC offers argument patterns for most important elements of a Safety Case to be adopted and adapted as required.
- **Argument Templates**
  An argument template is a fragment of argument created in a form that can be formally *instantiated* by substituting *instantiation parameters* with actual values to reflect the specific usage in a specific situation. This allows a particular argument to be re-applied many times by presenting the template argument once together with some form of tabulation to show each instantiation of it.
  Evidence referenced in solutions at the bottom of the argument may also be Instantiable.
- **DGR templates**
  A template form of DGR, with appropriate instantiations, may also be useful when preparing DGRs if the analysis is similar for a set of Guarantees. The exact form of a DGR may vary according to the subject system and the applicable development processes.

The majority of patterns provided in this document only describe the public interfaces of the safety case modules. This imparts sufficient detail for prospective users to build SC Modules that can be integrated, without constraining the argument approach of different authors.

Any Safety Case is specific both to the particular system and the **context** in which it is used. It is vital in preparing a Safety Case to capture all elements of context that refine, elaborate, restrict or constrain the applicability of the safety claims - where or when they can be relied on. The rigour achieved in capturing context is referred to as **Context Completeness**; it is especially important for systems that are expected to evolve in response to changing requirements or operational scenarios.

When producing a Safety Case by integrating Safety Case Modules, which may have been developed independently, it is then necessary to assure that the context applicable to each module individually is compatible with the context for the overall system; this is referred to as **Context Compatibility.**

It is presumed that a Modular Safety Case will evolve both during the development of the system and during the life of the system as requirements or operational context change. Safety Cases have been classified in standards as **Preliminary**, **Interim**, **Final** or **Operational Safety Cases.** The MSSC Process is applicable to Safety Cases at any stage of the system lifecycle from concept through to disposal. The MSSC Process helps to limit the cost of a change to an existing final or operational Safety Case.

3 Overview of the MSSC Process Steps

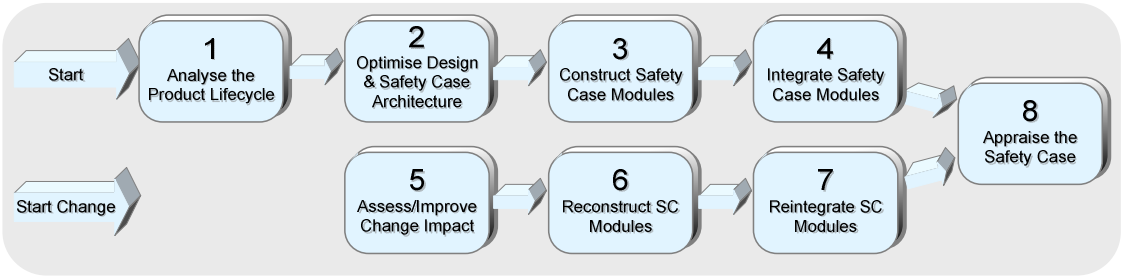


Figure 3-1: Process Steps

The Modular Software Safety Case (MSSC) presents the development and change of modular Safety Cases in eight steps (see Figure 3-1); five are used in initial development of the Safety Case, and four for incorporating a change. The final step in each is common.

Step 1 : Analyse the Product Lifecycle

The first step in the MSSC Process is to define significant future change scenarios expected during the lifetime of the product which will necessitate the Safety Case being re-visited. These change scenarios provide the driving criteria for choosing both design and Safety Case Module boundaries.

Step 2: Optimise Software Design and Safety Case Architecture

An initial safety case architecture is defined which identifies the safety case modules and their interactions. This is then considered alongside the proposed design architecture and the change scenarios from Step 1 to predict the impact of expected changes. Through review of alternative design or Safety Case module options, and iterative evaluation, a selection is made based on optimised resilience to change. Methodology suggestions are provided in the MSSC process.

Step 3: Construct Safety Case Modules

The Safety Case Modules identified in Step 2 are developed in this step. A hazard mitigation argument is formed and mitigation requirements directed to Block SC Modules. The guaranteed behaviour offered by each block in support of these is captured, along with dependencies on other blocks. A Block Safety Case Module is constructed providing argument and evidence for each Block based on the Guarantees and Dependencies. Additional SC Module types may also be constructed to cover Configuration Data, Software System Wide issues.

Step 4: Integrate Safety Case Modules

The Safety Case Modules are integrated so that claims requiring support in one Safety Case Module are linked to claims providing that support in others. This step of the process results in a fully integrated Safety Case.

Step 5: Assess/Improve Change Impact

When a system change is implemented, the impact on the design modules and associated Safety Case Modules is assessed.

Step 6: Reconstruct Safety Case Modules

This step mirrors Step 3, but is undertaken for the new or changed Safety Case Modules only.

Step 7: Reintegrate Safety Case Modules

This step mirrors Steps 4, but is undertaken for the new or changed Safety Case Modules only. A ‘change argument’ is also produced addressing the adequacy of the change impact analysis process and the rationale for not re-visiting unaffected Safety Case Modules.

Step 8: Appraise the Safety Case

The purpose of appraising the completed Modular Safety Case in this step is

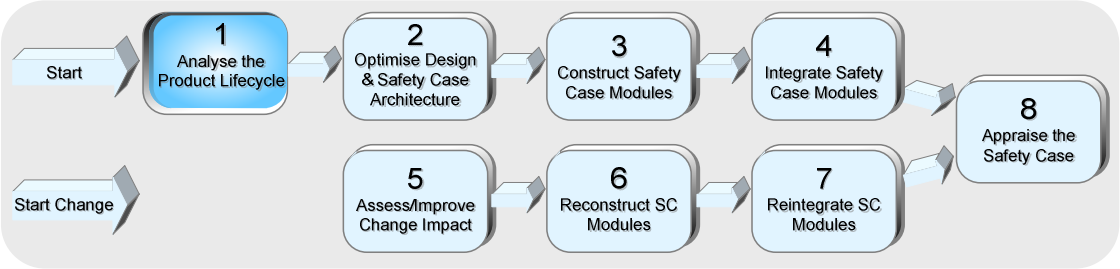
- Local process improvement and
- to evaluate the Safety Case generated using the MSSC Process to ensure that it is potentially capable of meeting the anticipated safety and cost related objectives throughout the expected lifetime.

NOTE: Tailoring of the Process

All process steps are essential; rigour may be adapted.

Each process step should be executed with a level of rigour appropriate to the strength of assurance required. Some elements of process steps may be considered optional.

4 Step 1: Analyse the Product Lifecycle



Objectives	
1. Identify and assess potential sources of change in order to identify change scenarios that may be safety relevant. 2. Characterise the change scenarios and prepare lifecycle plan material that maps out increments of software that will incorporate such changes.	
Inputs	**Outputs**
Product Lifecycle Plan or equivalent material that may identify sources of change arising in the following 1. Platform use 2. Technologies and standards 3. Obsolescence 4. Future capability	1. A predicted software lifecycle plan for the system 2. Set of safety relevant change scenarios for the system, their likelihood, frequency and magnitude. 3. Collation of any assumptions made

Figure 4-1: Analyse the Product Lifecycle

It is assumed here that some form of description of the expected **product lifecycle** will already have been produced as part of the project planning activities. This may take the form of a product lifecycle plan, which indicates the initial configuration of the product at first use, variants for different markets, any planned developments.

Execution of this process step results in the production of a **software lifecycle plan** which is used to identify areas of the software which are known to be, or considered likely to be subject to change through the life of the product.

Production of the **software lifecycle plan** involves first identifying and assessing change scenarios that are likely to arise during the life of the software.

4.1 Identifying and Assessing Change Scenarios

Changes to a system may arise from various sources:

- New or changed functional requirements (from customers, users, regulatory bodies).
- Changes in operational requirements or usage.
- Problem fixes (either outstanding or as yet unknown).
- Technology insertion
- Obsolescence of hardware (in new build and maintenance).

Technical and schedule information relating to the system and the changes likely to affect it in its predicted lifetime should be collected.

Each of these sources of change should be investigated with input from all interested parties, in order to identify the likelihood of changes to areas of system software throughout the lifecycle of the system.

In addition, consideration should be given to potential software modifications arising from software engineering domain concerns.

- Obsolescence of tools required in producing or maintaining the software
- Software technology roadmaps
- Requirement to increase throughput to support enhancements
- Adoption of new software procedures and standards.

The change scenarios identified through the above process should be categorised as either **known** (arising from the schedule of planned updates) or **predicted**.

In addition to those system changes categorised as **known** and **predicted,** there are others that may arise sporadically in unplanned ways from any of the sources listed above. These may be categorised as **unknown.**.

For each change scenario the identification process may capture;

- The Type of change
  - Requirements change (new/modified/removed) as specified upon the Safety Case, and if a mitigation argument is in scope, an associated hazardous condition change (new/modified/removed) that might be due to an expansion of operating conditions or a change of operational mode.
  - Implementation driven change, such as structural change to the system.
- the Grouping of changes (from any mandatory scheduling of the changes, relationship to external events or other changes)

Assessment of the change scenarios identified above involves:

- Identifying which changes may be related to safety.
- Making a determination as to the likelihood, frequency and magnitude of such changes by engaging with all interested parties.

4.2 Preparing and Reviewing the Software Lifecycle Plan

The product lifecycle plan or equivalent description of the expected product lifecycle is used at this stage as the basis for preparing or augmenting a software Lifecycle Plan.

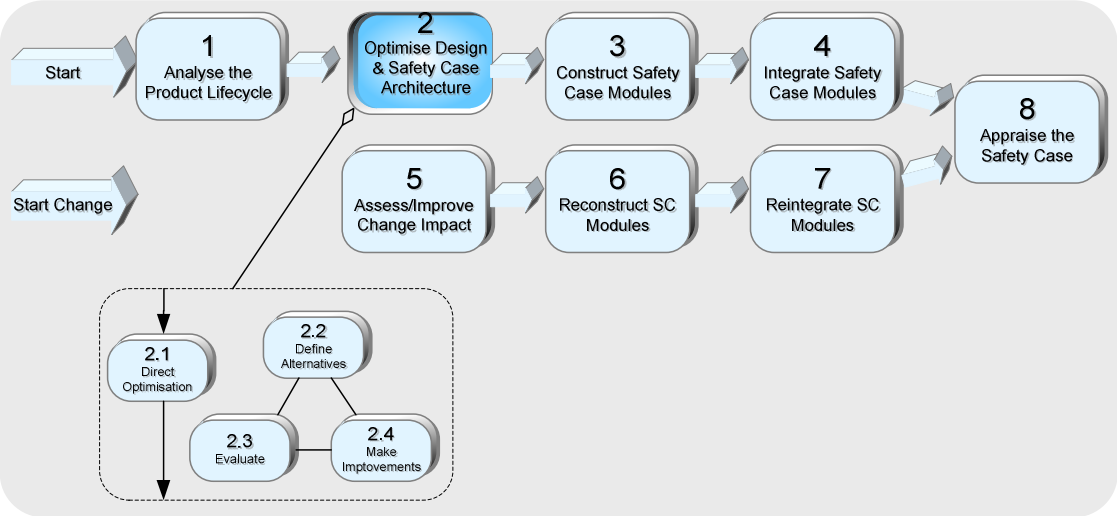
The software Lifecycle Plan is augmented with the known and predicted changes as highlighted by the above identification and assessment process. It should identify which software releases are for development/trials and which are to be certified.

The software lifecycle plan should consider incremental development and “type variants”.

When the MSSC Process is applied. the overheads of a change are expected to be smaller. There is less need to combine changes into larger updates, so there may be more, smaller change scenarios in the Lifecycle Plan

The software lifecycle plan should be reviewed by all interested parties and updated with any new or changed requirements. This plan forms a primary output from Step 1 of the MSSC Process, and includes the set of safety-relevant changes scenarios, and a collation of assumptions made during the stages of this process step.

5 Step 2: Optimise Design and Safety Case Architectures



Objectives	
1. To define the optimal architecture of the Safety Case. 2. To determine an optimised deployment of process. 3. To recommend an optimised evidence specification. 4. To recommend an optimised system architecture.	
Inputs	**Outputs**
1. Evidence specification (if any). 2. System architecture (if any) 3. Safety related and non-safety-related requirements (if any have been identified) 4. The lifecycle plan for the system (see section 4).	1. Safety Case Architecture. 2. System architecture recommendations (if any). 3. Evidence specification recommendations (if any).

Figure 5-1: Optimise Design and Safety Case Architectures

This section describes not only how to build a Safety Case Architecture for a system but also how to architect a system so that it facilitates construction of a modular Safety Case.

The optimisation of the design and Safety Case Architectures is a directed, iterative process that starts with the definition of the design architecture and an initial Safety Case Architecture, based on modularity that exists in system components, evidence or processes, and susceptibility to change. Architectural alternatives are:

- Defined (sub-step 2.2) then
- Evaluated (sub-step 2.3), principally against the considerations of containment of change versus cost and efficiency (see Figure 5-2), and finally
- Refined and Improved (sub-step 2.4) as appropriate.

The process is typically iterated, either by directly performing further evaluation, or defining additional alternatives for evaluation. The process step completes when an optimum balance between the objectives of the design architecture and those of the Safety Case Architecture is achieved.

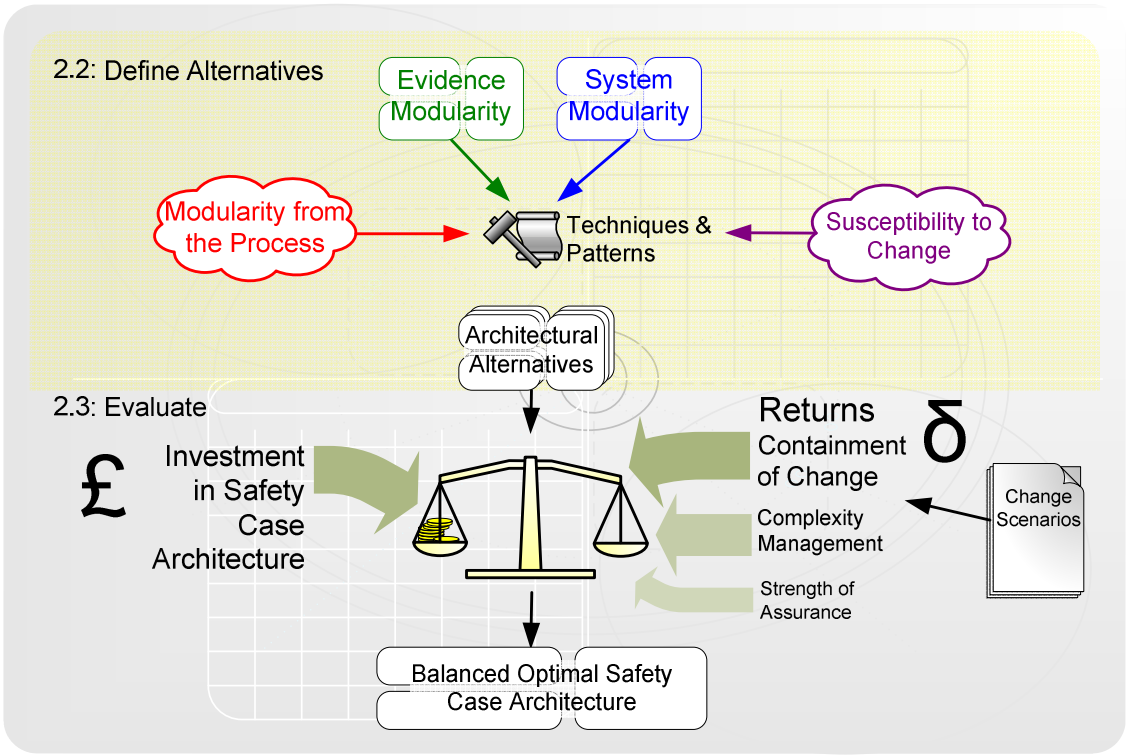


Figure 5-2: Influences on Safety Case Architecture[1]

In the event of a change (or improvement) to the modularity in the system, evidence or process, or any of the evaluation considerations, such as the change scenarios, the alternatives should be re-evaluated.

Improving the Safety Case or design architecture is a matter of examining the architecture definition and evaluation processes to see what changes are feasible and beneficial.

An architect will, through familiarity with the different considerations, develop an ability to quickly focus on a small number of alternatives.

Alternatives should be evaluated until the architect is satisfied that further modifications to them, or modularity in other areas, or different considerations that would significantly improve the evaluation results are no longer possible.

If the system is already in service, the system and evidence modularity may be considered fixed; and this process will not make recommendations for improving the design.

All Safety Case Module boundaries need to be sufficiently complete to support integration, but a Safety Case Module may be identified during this step as "to be split" so that a new dividing boundary may be defined at a later date when it is needed for change containment. This technique may allow some of the up-front effort needed to modularise the Safety Case to be deferred to a later iteration. Care should be taken to ensure that the knowledge required to split the Safety Case Module will be available at that time.

The process for optimisation of the design and Safety Case Architecture is split into 4 major sub-steps:

[1] Step 2.4 is not shown in this diagram, but it can operate upon any part of it, making changes in an effort to achieve better results.

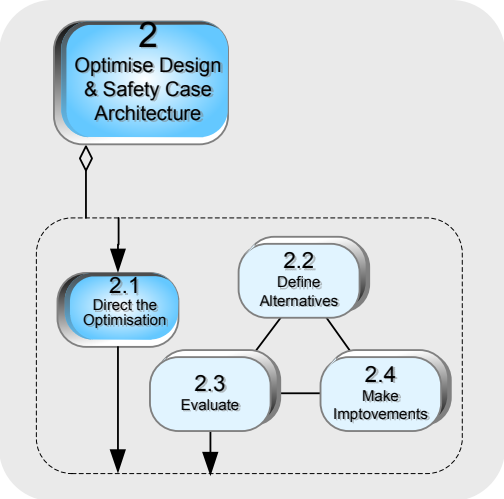
Step 2.1 Direct the Optimisation
Ensures that effort is optimised and significant concerns addressed early (section 5.1);

Step 2.2 Define Alternatives
Identifies alternative architectural elements in both the Design and the Safety Case domain (section 5.2);

Step 2.3 Evaluate
Evaluates each alternative in terms of cost-effectiveness over the life of the platform. (section 5.3);

Step 2.4 Make Improvements
Changes the design or Safety Case Architecture to improve its evaluation results (Section 5.4). This will involve the safety team consulting with other teams on optimally meeting their safety relevant responsibilities (see 5.4).

5.1 Step 2.1 Direct the Optimisation



Objectives	
1. To prioritise improvements to the Safety Case or the design; or to other aspects (evidence, process). 2. To direct the extent of the optimisation to minimise unnecessary effort.	
Inputs	**Outputs**
1. The lifecycle plan. 2. Test Plan 3. [on return from 2.2] Definition of architectural alternatives. 4. [on return from 2.3] Report on whether change is adequately contained by a given alternative. 5. [on return to 2.4] List of changes required to the design or Safety Case Architecture.	1. Direction to the next step, either: • step 2.2: Define alternatives for the design or Safety Case Architecture. • step 2.3: The decision to evaluate the alternatives at a certain level of detail. • step 2.4: An improvement to the Safety Case Architecture, system, change or other aspect. 2. Guidance on the aim or target of next step

Figure 5-3: Direct the Optimisation

This step selects and directs the execution of other sub-steps, ultimately directing the architect to step 2.3 which evaluates the alternatives in detail and demonstrates which alternative(s) provide the design and Safety Case Architecture which is the most cost-effective over the life of the platform.

Optimisation of design and Safety Case Architectures starts with the definition of alternatives for design and Safety Case Architectures for evaluation. The director ensures that the evaluation proceeds at an appropriate level of detail and rigour for the maturity of the inputs by setting targets for each iteration of the other steps. Improvements to the design or Safety Case Architecture are made and re-evaluated until further modifications to them, or modularity in other areas, or different considerations that would significantly improve the evaluation results are no longer beneficial. The director considers what is significant and decides whether another iteration is necessary.

Decisions about directing the progress through steps 2.2, 2.3, 2.4 should consider the following advice:

- **Decision to execute Step 2.2, Define Alternatives**

  This step needs to be visited first and revisited in the event of a change (or improvement) to the modularity in the system, process or evidence.

- **Decision to execute Step 2.3, Evaluate**

  The director task (Step 2.1) should ensure that more obvious improvements are identified early by a preliminary execution of the evaluation in step 2.3 at appropriately low levels of rigour. Only on the final run through will step 2.3 need to achieve the formality and produce the outputs necessary to define the final Safety Case Architecture.

  If there are no improvements currently evident, then step 2.3 is undertaken to evaluate the alternatives. At the discretion of step 2.1, step 2.3 might initially be undertaken “coarsely” to bring out further possible improvements.

  Some alternatives may be related to each other, and it may therefore be prudent to address those alternatives together in step 2.3.

  The evaluation in step 2.3 must be performed again after a change is made to the design or Safety Case Architecture. The final evaluation by step 2.3 needs to be at the required level of rigour to support the safety argument.

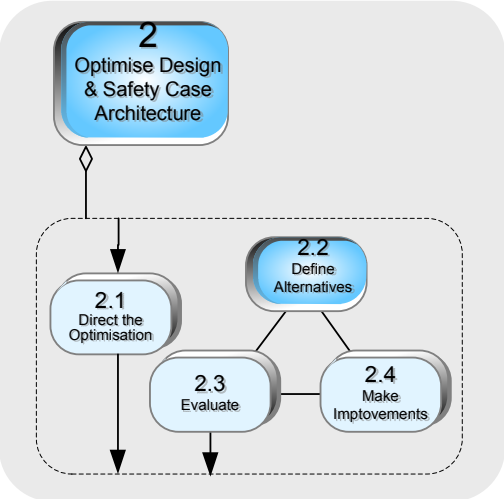
- **Decision to execute Step 2.4, Make Improvements**

  The decision to make improvements should be considered during the execution of the evaluation, where it is shown that any of the evaluation criteria are not adequately addressed by the Safety Case Architecture.

  Improvements may be made to either a non-SC Artefact (design, test plan) or to the Safety Case Architecture.

  Improvements to the Safety Case Architecture may be applied simply to make the argument for system safety simpler or more compelling. Improvements to the system may facilitate the safety argument, or they may even make the system safer.

5.2 Step 2.2: Define Alternatives



Objectives	
1. To define Safety Case Architecture alternatives.	
Inputs	**Outputs**
1. Process & organisational architecture. 2. Evidence specification (if any). 3. System architecture (if any). 4. Lifecycle Plan / information (see Section 4).	Safety Case Architecture alternatives.

Figure 5-4: Define Alternatives

During this sub-step a number of architectural alternatives, based on modularity in system components, evidence or processes, are defined. An architectural alternative may be a choice between options in a particular area of the architectures, or a complete and self-consistent architecture. The Safety Case Architecture may include different types of Safety Case Modules, such as Block Safety Case Modules, Integration Safety Case Modules, Safety Case Contract Modules as described in Section 2. The Safety Case Architecture defines the identity of each of the Safety Case Modules and their inter-relationships.

The process for defining the initial Safety Case architecture can be divided into three main tasks:

- Divide the system into Blocks and form Public Interfaces for the Block Safety Case Modules – All elements of the system are split into blocks to be the subject of Block Safety Case Modules. Each Block Safety Case Module presents argument about the safety-related behaviour of that Block;
- Define additional Safety Case Modules types as necessary – for example:
  Software Safety Requirements SC Module
  Software System Wide SC Module,
  Configuration Data SC Module and
  Safety Case Contract Modules;
- Define Safety Case Integration Modules – these provide the argument about the combined behaviour of interdependent Safety Case Modules.

A pattern for the SC Modules in an initial SC architecture is shown in Figure 5-5.

The three tasks are generally performed in the order presented above, but can be iterated as a particular alternative is refined.

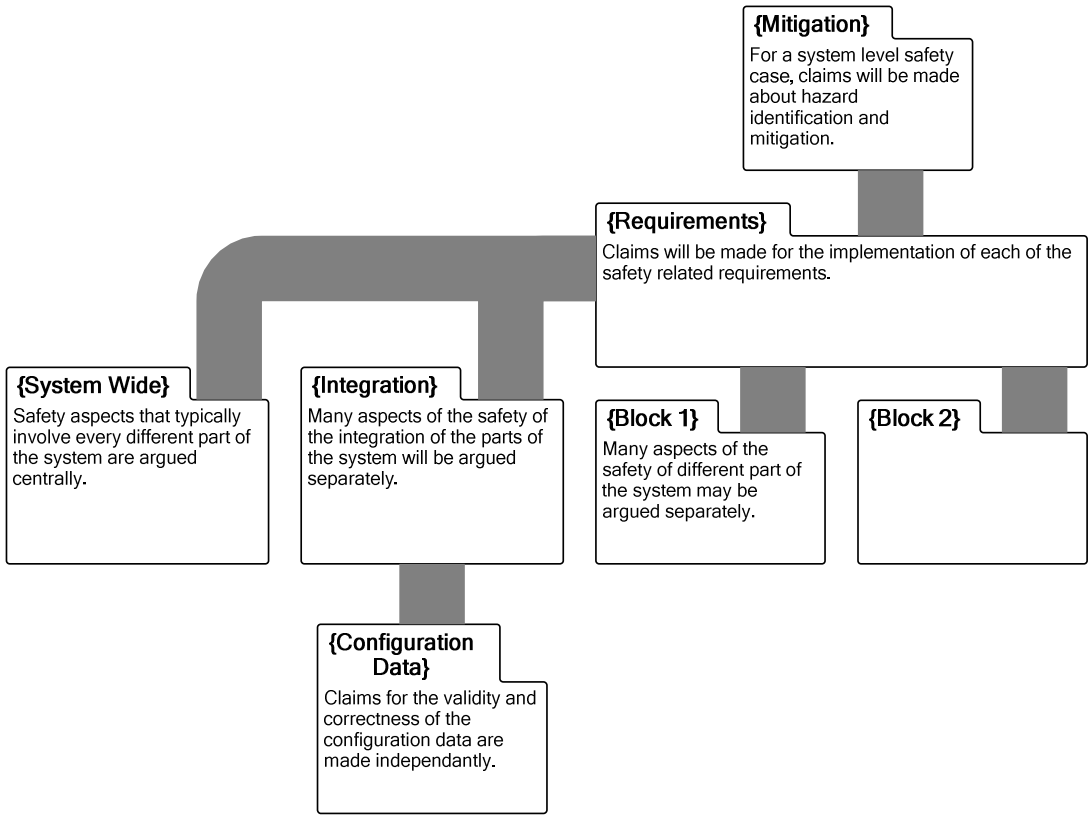


Figure 5-5: Pattern for SC Modules in the Initial SC Architecture

Definition of the Safety Case Architecture is complete when all parts of the design are covered by a block, and each block has a corresponding Safety Case Module, all 'special' Safety Case Modules (e.g. System Wide Issues) are declared, and all 'block' Safety Case Modules are covered by an appropriate Safety Case Integration Safety Case Module. Constraints on the content of each Safety Case Module's public interface will also be detailed.

The architect must also consider to what extent the public interfaces of the Safety Case Module can be defined by the initial architecture. A full definition facilitates later independent Safety Case Module construction, but ultimately the public interface becomes the responsibility of the Safety Case Module author who may also own related IPR. By making an initial definition of the public interfaces the Safety Case Architect establishes what is needed to construct the integration arguments.

5.2.1 Alternatives Influenced by Modularity of Evidence

Ultimately the assurance of any safety-related claim is based upon the evidence that supports it. Evidence may provide assurance to a collection of components, or only some of the functionality of a single component. In practice evidence assuring claims about collections of components is usually less costly to produce than individual components, even if those components are configurable. The more of a system that a piece of evidence relates to, the greater the drive for the Safety Case Module that uses that evidence to address a

similar scope. Modularity in the evidence is likely to be reflected in the modularity of the design process, but may or may not be reflected in the modularity of the design.

5.2.2 Alternatives Influenced by Modularity of the System

This section assumes that the architecture (including the modularity) of the system is expressed in its design, which may have a project specific format.

Figure 5-5 is an example that shows four different components of a system. Modularity in the Safety Case Architecture may align with each component, as the core software is unlikely to be changed, but Comms A and B may be expected to change independently at some point in the lifecycle.

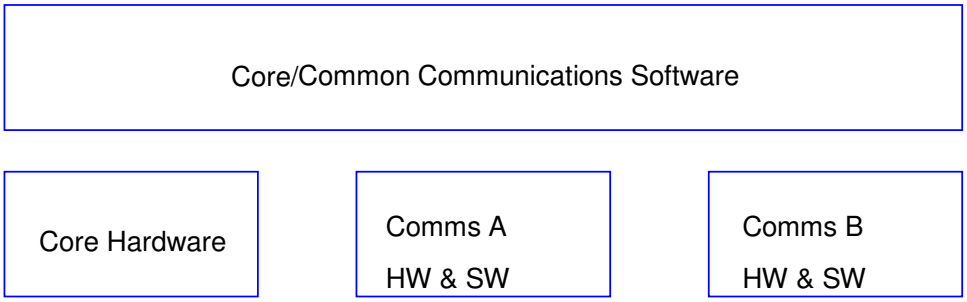


Figure 5-6: Example Configuration of Components

The design information should include:

1. A definition of where safety relevant requirements are implemented in the design. (i.e. where and how the hazards are mitigated).
2. A definition of where legacy sub-systems are incorporated.
3. A definition of where COTS or third party elements are incorporated.

Safety Case architecture boundaries which reflect modularity in the system may also include modularity in the hardware. For example, applications may be split across multiple software partitions. A trade off then needs to be considered as to whether to include the software partitions in a singular Block (which would span multiple partitions) or to place each in its own Block. Where a block spans multiple processor modules any dependencies between processors become an internal matter for the corresponding Safety Case Module, but dependencies on communication mechanisms provided outside the block (such as those furnished by middleware) still need to be captured.

5.2.3 Alternatives Influenced by Process Demarcation

Adverse cost impacts may arise from:

- Different required levels of integrity
- Different responsible, regulatory or legislative organisations (i.e. differing standards)
- Different implementers
- Mitigation Strategies enforcement of process diversity
- The distinction between processes for legacy/asset and new components
- The distinction between processes for COTS and bespoke components

Where parts of a system are developed using different design processes (or standards), it might be useful to deal with those parts of the system in separate Safety Case Modules. In this way, Safety Case modularity may be derived from the system lifecycle processes.

Adequate application of the evaluation criteria (which will be described in Section 5.3.1) helps avert these adverse cost impacts.

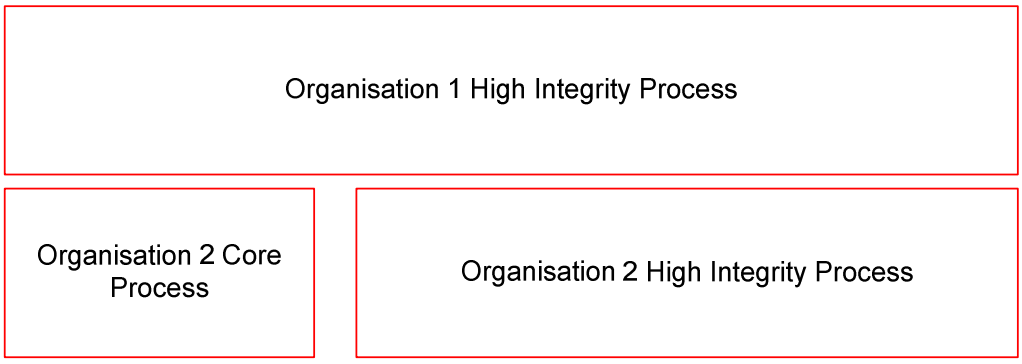


Figure 5-7: Example Application of Different Processes

Figure 5-7 shows how three different processes might relate to a system development; in this case where development is split across two organisations and two levels of integrity (Core and High).

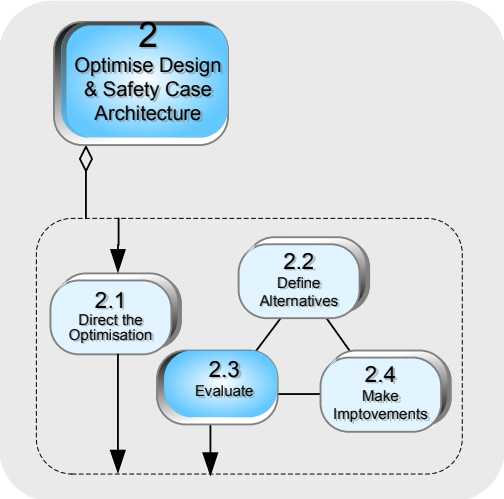
Each process is usually applied to a specific set of components or system modules.

5.2.4 Alternatives Influenced by Organisational Structure

Where different sub-systems are the responsibility of different organisations there is usually a contractual motive to have a well defined interface, this is something which can be mirrored by Safety Case modularity in order to maintain that interface.

Different organisations are also likely to make use of different processes, which may affect modularity as discussed in Section 5.2.3.

5.3 Step 2.3: Evaluate



Objectives	
To predict how well each alternative will perform.	
Inputs	**Outputs**
Safety Case Architecture alternatives. Lifecycle Plan including change scenarios. Design data Resource budget (if any).	1. Evaluation results for each alternative.

Figure 5-8: Evaluate

This process sub-step evaluates the alternatives to determine whether an optimal design vs. Safety Case trade-off has been achieved.

Firstly the evaluation criteria are applied to the candidate SC architecture: Criteria include, as a minimum the following, for which explicit guidance may be found below

- Containment of Change
- Strength of Required Assurance
- Likelihood and Impact of Change
- Cost Effectiveness

This list may be extended to include project-specific criteria.

Secondly problems with the SC Architecture are identified and strategies to overcome them are formed.

The solutions require an optimal Safety Case Architecture is developed by balancing decisions relating to the evaluation criteria that fundamentally trade off against each other. Adverse cost impacts may arise from:

- Poor containment of change (see section 5.3.1)
- Modularity broken by evidence (see section 5.3.2)
- Conflicts with modularity from the process, focally susceptibility to change and strength of required assurance (see section 5.3.3)
- Excessive or unnecessary modularisation (see section 5.3.4)

A report on the assessment of alternatives should raise any perceived shortcomings and propose possible alternative designs and methods for consideration in improving either the software design or the Safety Case Architecture.

This report provides material required to support improvement of the Safety Case Architecture.

5.3.1 Containment of Change

Using each Change Scenario from the lifecycle plan, especially those that are scheduled or are most likely, and the alternatives (from section 5.2) asses which Safety Case Modules would be affected. Describe the nature of the impact on each Safety Case Modules and classify it in terms of both magnitude and frequency.

5.3.2 Strength of Required Assurance

Defining more module boundaries forces more detailed definition of the system, through the definition of the additional boundaries, and in itself increases confidence in it. This can be beneficial. Excessive modularisation can also, however, be detrimental to confidence; it results in a shift of complexity (in the Safety Argument) to the integration, making the integration argument overly complex undermining confidence. Defining more module boundaries will also incur greater cost, which must be balanced against the need for higher assurance.

Determine whether it is reasonable that population of each of the Safety Case Modules will achieve the required level of assurance for the claims at the top level.

5.3.3 Likelihood and Impact of change

Each Block should be further categorised according to the likelihood of change to that block and the impact in terms of the assurance that would have to be applied to that change. This is achieved by analysis of each block according to the change scenarios.

Note that for each planned change scenario, (even if, for example, they only occur once, or are of small magnitude), consideration should be given to the investment/returns trade-off for the introduction of modularity specifically to contain that particular change.

A chart can be drawn showing the blocks categorisation according to susceptibility to change on one axis and a prior categorisation capturing modularity of the process, normally required level of assurance, on the other, as shown in Figure 5-9.

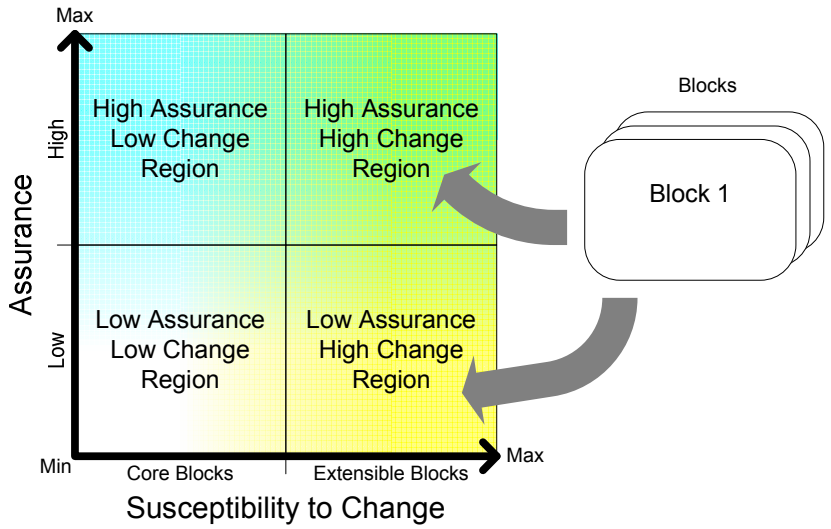


Figure 5-9: Defining Regions for Blocks

Blocks may be categorised as:

- Core, which does not change much, i.e. the high assurance/low change and low assurance/low change regions;
- Extensible which changes a lot, i.e. the high assurance/high change and low assurance/high change regions;
- High assurance, i.e. the high assurance/low change and high assurance/high change regions;
- Low assurance, i.e. the low assurance/low change and low assurance/high change regions.

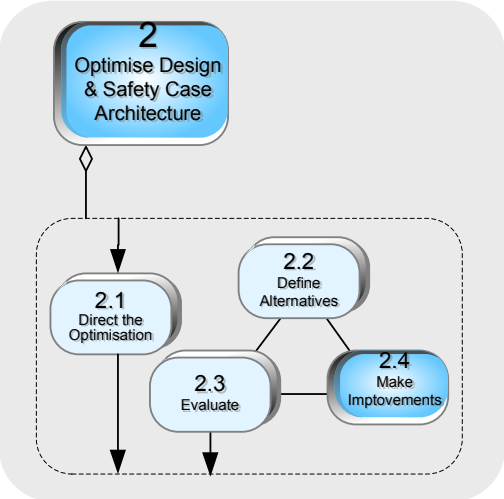
The optimal use of modularity is where large blocks are subject to low change/low assurance and small blocks are subject to high change/high assurance.

5.3.4 Cost Effectiveness

The benefit of each architectural decision needs to be weighed against the cost. While a decision may maximise the containment of change for example, the cost implication of this decision will also increase. A balance must be struck between adequate containment of change, strength of assurance etc. and minimising the cost incurred.

The level of precision of cost estimation may be limited by the level of detail available about the Safety Case proposed and the metrics available. This should not be critical as the objective is to determine a relative estimate between two alternatives, to determine which offers the best value.

5.4 Step 2.4: Make Improvements



Objectives	
To make improvements to the Safety Case, implementation and/or evidence architecture in light of the evaluation made in step 2.3.	
Inputs	**Outputs**
1. All of the inputs and outputs from steps 2.2 and 2.3.	Improved input data for step 1.1 and/or 1.2: Improved process deployment strategy (if any) Improved evidence specification (if any) Improved system architecture (if any) Improved Safety Case Architectures (if any) Improved change scenarios (if any) Improved resource budget (if any)

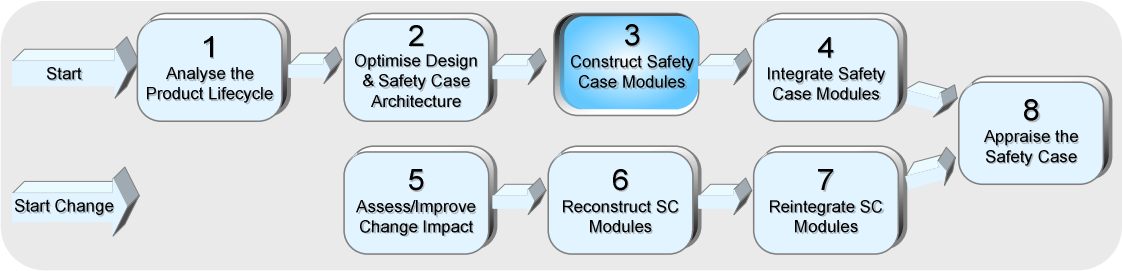
Figure 5-10: Make Improvements

This process sub-step suggests improvements to either the design or the Safety Case Architecture, or both. Those improvements may include changes to:

- The Safety Case Architecture;
- Block definition;
- The design architecture;
- Test plans.

This process sub-step may also identify improvements in the change scenarios generated in Step 1. For example, the evaluation and subsequent improvements may highlight, or simplify, an additional or existing change in the Change Scenarios, thus increasing its likelihood of being implemented.

6 Step 3: Construct Safety Case Modules



Objectives	
1. To construct Safety Case Modules as identified in the Safety Case Architecture.	
Inputs	**Outputs**
1. Allocated safety-related requirements 2. Safety Case Architecture including public interfaces (to the extent made available from Step 2) 3. Engineering artefacts from development lifecycle. 4. Specification of Context, including operating environment and usage 5. Applicable safety standards 6. Pre-existing reusable Safety Case Modules and patterns	1. The requisite Safety Case Modules - Public interface & Context - Body of argument - Link to evidence

Figure 6-1: Construct Safety Case Modules

6.1 Form of a Safety Case

A Safety Case are typically constructed using two forms of argument:

A Product argument
which claims safety is assured by showing conformance to mitigation requirements that were derived from hazard analysis

A Process argument
which claims that safety is assured by following robust development & test processes

The MSSC GSN patterns provided in this document assume a product focused approach to the safety case.

6.1.1 Product argument

A **Product Argument** offers assurance of the correctness of the implementation with respect to safety-related requirements placed on an element of the system.

Figure 6-2 shows the form of a product argument within a Block Safety Case Module.

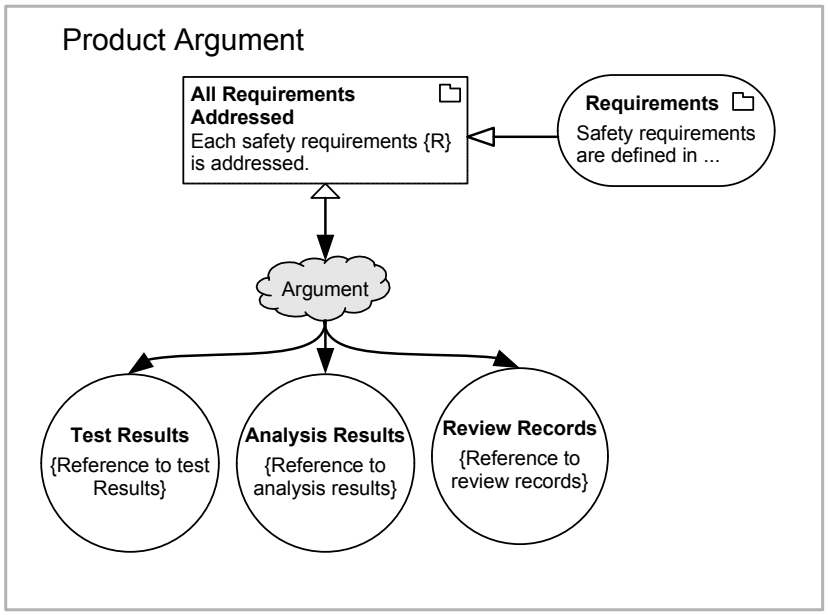


Figure 6-2: General form of a product argument

A modular Product Argument for a system may be formed by constructing and then integrating a set of Safety Case Modules, each of which will support a subset of the safety-related requirements in the final safety case. See section 6.5

A *Product Argument* may be supported by a specific *Process Argument*.

Product argument will always be made for a specific context of usage which will include some limiting statements about the applicability of the product claims.

6.1.2 Process argument

An example of a process argument is shown in Figure 6-3:

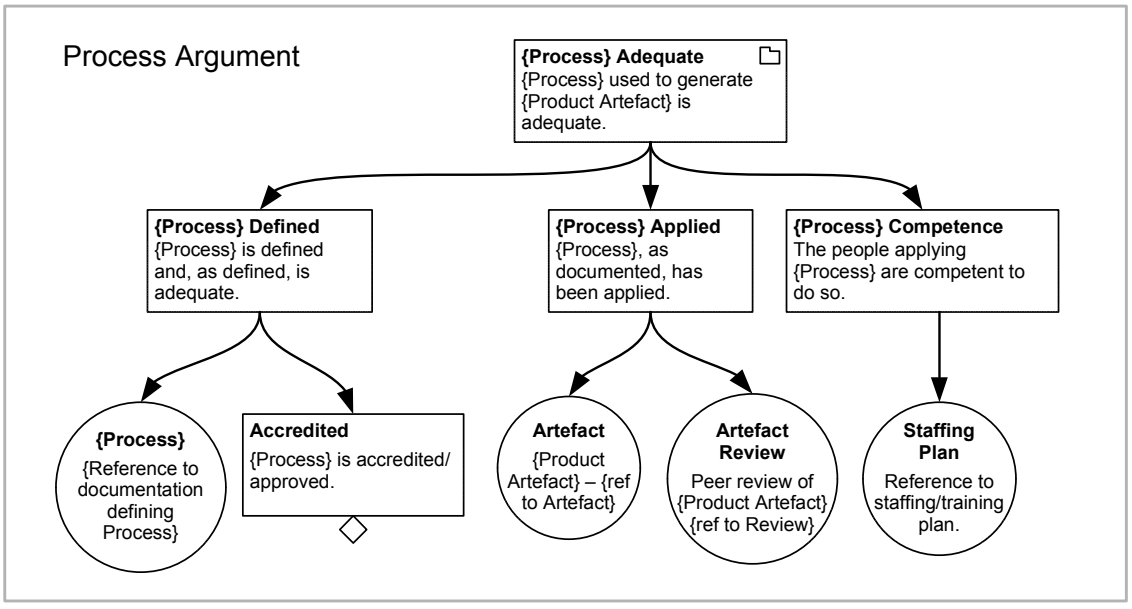


Figure 6-3: Pattern for a process argument

- To assure a process it will normally be presumed that there is a formal capture of the process and that it has been approved or accredited by internal or external authorities sufficient and appropriate to the Context.
- A process argument may identify steps in a process and provide evidence that each step was executed and appropriate review was carried out.
- It is normal practice to identify staff held accountable for the execution and oversight of processes together with evidence of their competence to take such responsibility.

An overarching software development process may apply to an entire system; there could be development processes followed by each supplier of a Block of software or there may be a detailed process specific to a particular step in the system development such as an offline modelling or data processing activity.

A process argument may identify sub-processes where these are distinct and support them with subsidiary process argument.

An important element of process that has to be the subject of a claim (even though it is not part of the MSSC process per se) is hazard analysis. Where potentially hazardous behaviour might be identified during the development process, there may be a need to make claims about the particular aspect of the engineering process that identifies and specifies additional safety relevant requirements or context. The scope of this analysis might be specific to a particular system domain context (e.g. air, sea or land), this should be made clear.

So a Process Argument fragment may be incorporated at the System Wide level, at the level of a Block Safety Case Module or at a particular place within the safety argument, depending on the scope of the process being addressed.

A Process Argument with supporting evidence may also constitute a Safety Case Module in its own right.

Claims made within a Safety Case Module relating to process may be made public so that argument in other Safety Case Modules is able to appeal to them.

6.1.3 Types of Safety Case Module used in construction

There are several types of Safety Case Module that go to make up a complete MSSC Safety Case, each constructed in a specific way:

- Hazard mitigation argument
  which argues that the specified mitigation (safety-related) requirements do in fact mitigate the identified hazards, on the basis of appropriate hazard analysis has yielded a complete set of identifiable "safety related" requirements.
- Software Safety-related Requirements (SSR) Safety Case Module
  which argues that the specified mitigation (safety-related) requirements are all supported (by Block Safety Case Modules) or directed outside the scope of a software Safety Case.
- Block Safety Case Modules
  each of which argues about a Block's guaranteed behaviour that will be used in support of safety-related requirements as propagated by the SSR SC Module
- Configuration Data Safety Case Modules
  which argue that data used in configuration of the system supports the requirements of the configurable elements of the system
- System Wide Safety Case Modules
  which present arguments involving behaviour of several Blocks as a group

A Safety Case may also incorporate argument and evidence to support claims about hardware and this can be formed into a HW Safety Case Module, although the nature of the argument is likely to align with the HW engineering processes.

The construction of these types of Safety Case Module is described in section 6.2 et seq. and the steps to be taken shown in Figure 6-4.

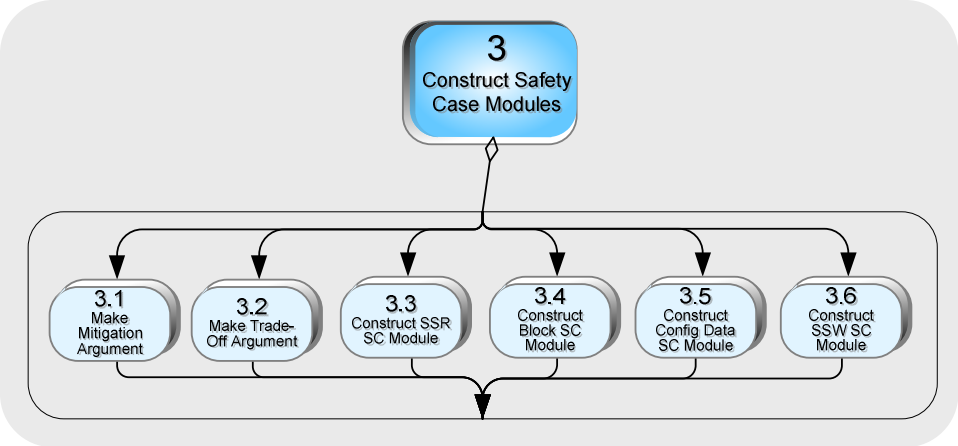


Figure 6-4: Steps in integrating a modular Safety Case

The sequence of the sections does not imply the SC Modules have to be constructed in that order.

There are also types of Safety Case Module used during *integration*, which are described in section 7

6.1.4 Containment

It is permissible in the MSSC derivative of Modular GSN for a Safety Case Module to *contain* other Safety Case Modules.

So for example it would be permissible for a Block Safety Case Module to contain subordinate Block Safety Case Modules, perhaps with associated Process Safety Case Modules.

One reason for doing this is to support integration of two or more Block Safety Case Modules and then present a consolidated view of the integrated set while hiding some of the detail of the individual Safety Case Modules.

Containment might also be used in a situation where Application Blocks are supported on a software platform that is itself composed of an operating system plus middleware. The Application Block Safety Case Modules would interact with a "Software Platform" Safety Case Module without visibility of the embedded operating system Safety Case Module.

Containment may also be useful where an existing Safety Case Module is de-composed (at some later point in the product life cycle, say) but it is desirable to preserve the existing Safety Case Module interactions.

6.1.5 Context Capture

Any module of safety argument is likely to be constructed to be applicable in specific *Context* and will potentially be invalid beyond that Context.

In a safety argument Context means any statement or reference that serves to refine, limit or constrain the applicability or validity of a safety claim that an integrator might need to consider in forming a link between consumer and provider claims.

Contextual information (such as limitations on usage) may reside within engineering artefacts such as requirements, design and implementation documentation and may be invoked by reference - a guarantee is assured "for use in accordance with user documentation"

It is also important that matters arising from hazard analysis that may need to be taken into consideration by an integrator are also captured, e.g. if component level hazard analysis reveals "potentially hazardous behaviour" it should be captured as GSN Context for the *guaranteed behaviour* and qualified with any assumptions made about the target system.

A complete analysis of all possible behaviour may, however, not be practical, so the component level hazard analysis may be limited according to the anticipated domain of usage of the SC module (e.g. the expert identification of the behaviour may have focused on anticipated use of the SC module on a fixed wing aircraft). It can be beneficial to the integrator to capture such anticipated domains of use as an assumption on the *process* claim about adequacy of the analysis. It may benefit integrators in other domains if the domain is clarified by identifying the associated general characteristics.

The need to state Context will arise in different places in different form throughout construction of a safety argument.

Therefore, during construction of the modular Safety Case it is important to ensure that, in addition to the specified behavioural & intrinsic *requirements*, all elements of *Context* associated with Safety Case goals are captured as they arise. A Safety Case Module always *offers* its public claims *in the declared context*.

All elements of Context arising within a Safety Case Module will need to be examined to assess applicability at the boundary (Public Interface) of the Safety Case Module and if relevant, must be propagated to the Public Interface in order for the integrator to be able to assure the links between Safety Case Modules are compatible during integration.

During integration it will be necessary to be able to argue that *Context* attaching to a supporting goal is *compatible* with the *expectations* of the supported goal. (See section 7.4)

For example a Safety Case may be constructed in a form that is only applicable to a specific usage of the system - so that specific usage definition is itself Context; then in declaring a Solution there may be a particular (limited) level of testing applied - and that may represent a constraint that needs to be captured as Context. Both elements of Context would need to be made Public at the Safety Case Module Boundary.

An integrator may then claim only that the system can be considered for the specific usage; or that the evidence for the system can validly be read across to another ("closely equivalent") usage.

Context completeness will need to be assured through the application of appropriate expertise during Safety Case construction. This expertise should cover:

- The problem domain (what the system needs to do);
- The solution domain (what the system actually does);
- The implementation domain (how the system actually works);
- The Safety Case domain (how safety is to be argued)

The decision to propagate particular elements of Context to the Safety Case Module boundary may depend on the scope and strength of assurance required of the claims it makes.

Wherever possible, Context should be captured in a form that is amenable to a check of compatibility between *expected* Context and *offered* Context.

Examples of general types of Context that might be relevant in construction of a Safety Case include;

Overall Safety Case	Configuration control information for the Safety Case and the system it applies to
	The application domain considered as applicable in the development phase
	The certification and standards regime under which it was produced
	Usage restrictions for the system
	Safety Operating procedures for the system covered by the SC
Block Safety Case Modules	Operating procedures for the block including any potential safety related aspects
	Usage restrictions for the block
	The certification and standards regime under which it was produced
Integration Safety Case Modules	Configuration control information for the constituent Safety Case Modules
	Context from the Block Safety Case Modules that is propagated to the integration Safety Case Module claims

Table 6-1 Examples of general types of Context

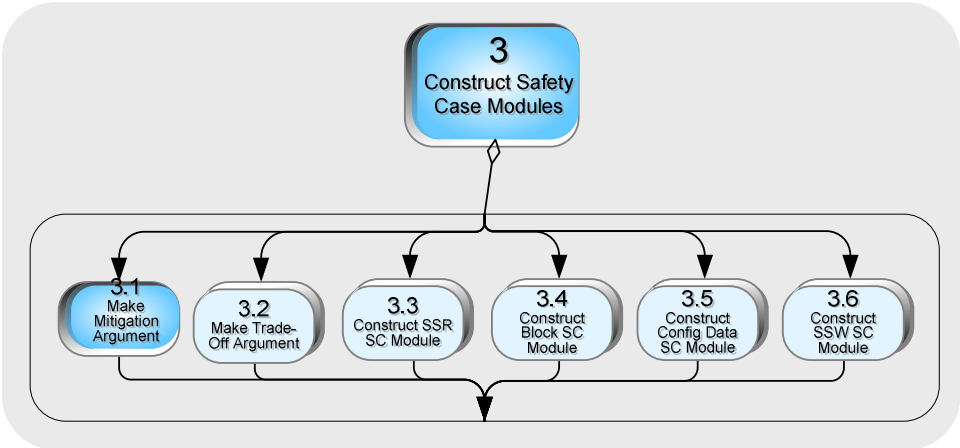
For each type of Safety Case Module, specific types of Context will be safety related. Examples of those specific types of Context that are often relevant include;

Software	Specified / permissible usage of an API (usually by reference to engineering artefacts), including details such as valid sequence of API calls
	Processor (target hw) or virtual machine for which it is assured
	Operating system for which it is assured
	Identity and configuration of specific HW devices with which it must interact
	Scope of the testing carried out, e.g.: textual requirements or use cases input space output space executable software, path, instruction, condition/decision state space
Configuration Data	A reference identifying the particular data
	The specifications against which the data is claimed to be correct
	The process by which the configuration data was generated

Data Resident in or exchanged with the block	The specification of the machine representation of data, including Units, Scaling, Precision/Resolution, Range Persistent or volatile Reference System / Datum (for relative measurements); Location of origin Format of Bits, e.g. 2's complement, endianness , padded bits
	Real-Time Data Accuracy & resolution Latency, Update Rate/Period Validity, validity period
	Data Semantics Subject of measurement, semantically acceptable range Max and Min possible delta between updates Relations with other data input values or timing
	Whether an error is reported in some circumstances instead of the data being provided (if not captured in the guarantee itself)

Table 6-2 Examples of types of Context for specific Safety Case Module types

6.2 Step 3.1: Making a Hazard Mitigation Argument



Objectives	
1. To present an argument, supported by evidence, that the specified mitigation (i.e. safety-related) requirements do in fact mitigate the identified hazards	
Inputs	**Outputs**
1. Hazard Analysis Report 2. Specified mitigation requirements 3. Context of operation / usage	Hazard mitigation Safety argument + evidence

Figure 6-5: Creation of a hazard mitigation argument

A product based Safety Case has to present an argument that a system is safe based on two logical steps:

- there is sufficient assurance that hazards have been identified;
- for each identified hazard there is a sufficient mitigation.

These both originate from the hazard analysis process which is not form part of the MSSC Process as such, but the final Safety Case needs to include the claim and evidence that hazard analysis has been completed as a precursor to presenting the product argument.

So for a complete Safety Case a **mitigation argument** begins with a claim that each hazard is adequately mitigated - by means of requirements placed on the system implementation and operation. A pattern of argument is shown in Figure 6-6.

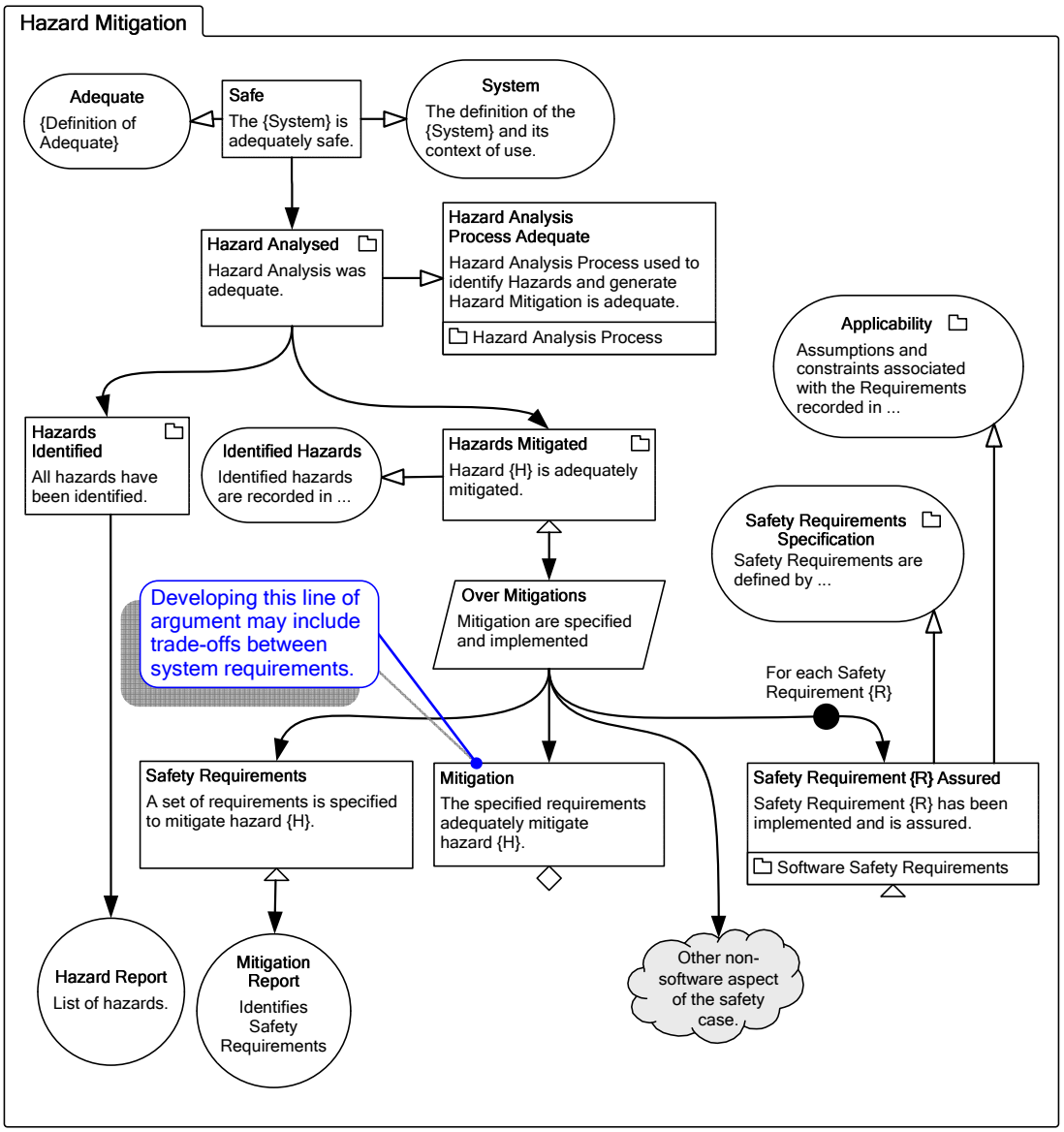


Figure 6-6: Mitigation Argument Pattern

The claim that "All system hazards have been identified and addressed" is supported by the claims:

- that all hazards have been identified

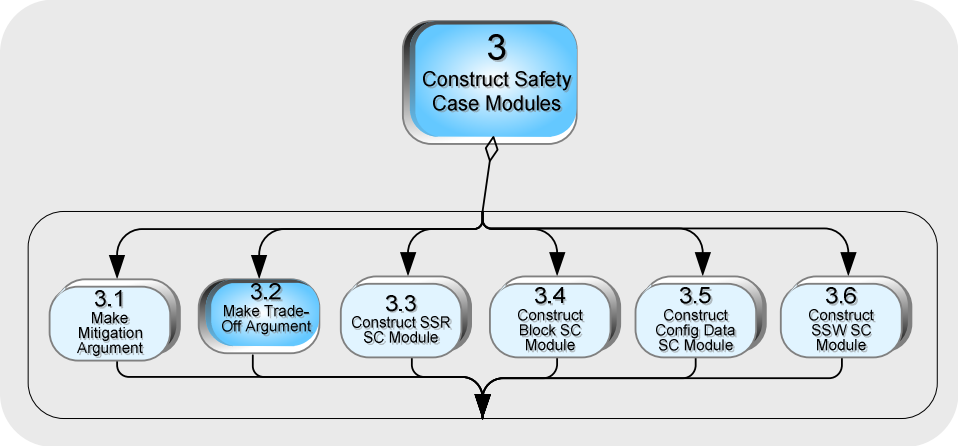
- that a set of mitigation requirements have been specified for each hazard
- that the specified requirements do in fact mitigate each hazard
- that each resulting safety-related requirement is implemented.

The first two claims are not developed further within the MSSC Process but are supported by evidence from Hazard Analysis and other aspects of the safety engineering process. If there is a conflict between the hazard mitigations and other concerns, the second claim may need to be developed by the integrator in a trade-off (see section 6.3).

The third claim is developed by way of the SSR argument and ultimately supported by Block Safety Case Module goals.

The mitigation argument may not need to be a Safety Case Module in its own right. It could be incorporated as part of the SSR Safety Case Module - see next section; or it might form part of an argument at a higher (system) level.

6.3 Step 3.2: Making a Trade-off argument



A **trade-off argument** may be needed to indicate that an implementation decision was the result of a trade-off between two conflicting imperatives (such as safety concerns vs security concerns).

It may be necessary to form an argument that justifies the priority assigned to one mitigation over another, and the mitigation (or justification) of any compromise.

This argument is closely related to the *mitigation argument* discussed in section 6.2, and should be made prior to (and separate from) construction of the Safety Case Modules that will pick up the claims relating to implementation of safety-related requirements

The structure of a trade-off argument is not prescribed but an example is shown in Figure 6-7.

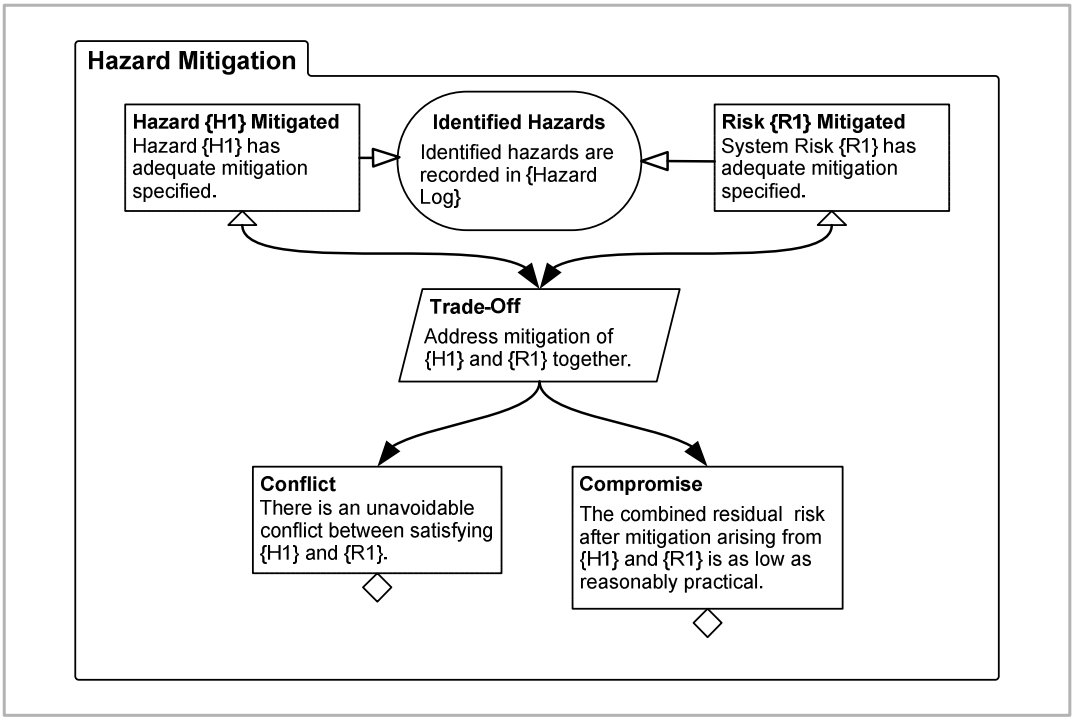
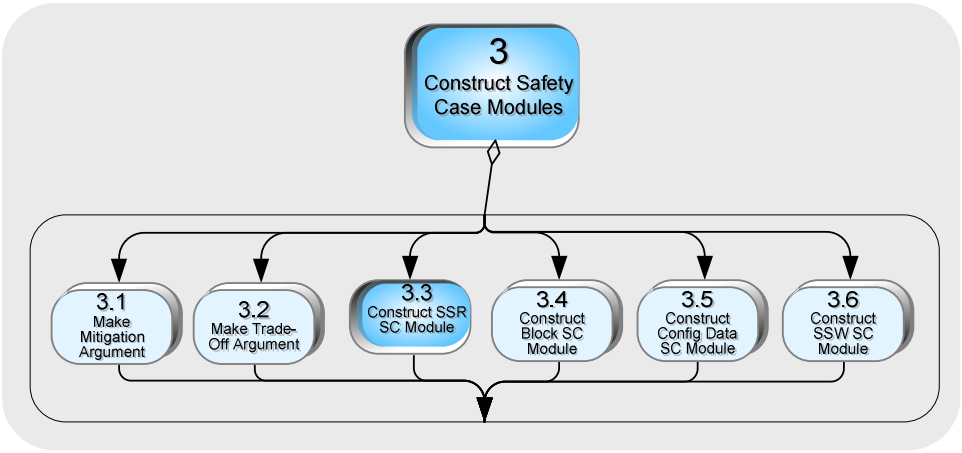


Figure 6-7: Example Trade-Off Argument Pattern

6.4 Step 3.3: Constructing an SSR Safety Case Module



Objectives	
1. To present an argument, supported by evidence, that all identified safety-related requirements are supported by the set of Safety Case Modules identified in the SCA	
Inputs	**Outputs**
1. The SCA, including public interfaces 2. The set of safety-related requirements that must be satisfied	1. SSR Safety argument + evidence

Figure 6-8: Creation of a SSR Safety Case Module

The purpose of the Software Safety-related Requirements (SSR) Safety Case Module is to offer the Safety Case integrator the top level claims that all software safety related requirements have been met. These safety-related software requirements will have been derived from the hazard analysis as mitigation requirements allocated to software.

The SSR Safety Case Module has to provide assurance that each is satisfied by way of Public Goals offered by one or more Block Safety Case Modules.

There are three main Public Goals that may be offered by the SSR Safety Case Module:

- **All hazards are adequately mitigated** (see section 6.2)
  This may be included in the SSR Safety Case Module if a dedicated mitigation Safety Case Modules is not used
- **Trade-offs between conflicting requirements have been resolved.**
  This is to justify any assigning of precedence where requirements may conflict. (see section 6.3)
  This claim may be presented in the SSR Safety Case Module or with the hazard mitigation claim in a separate Safety Case Module.
- **Each safety-related requirement is assured.**
  This is the top claim in the assurance of functional and non-functional software safety related requirements, to be satisfied at SC integration by claims about blocks in Block Safety Case Modules, possibly referring to the specification and context of use as *GSN Context*.

The first two aspects have already been described. The third is in effect forming a contract between the *mitigation argument* and the Block Safety Case Modules that will assure implementation of the safety-related requirements. A form of argument for this aspect is shown in Figure 6-9

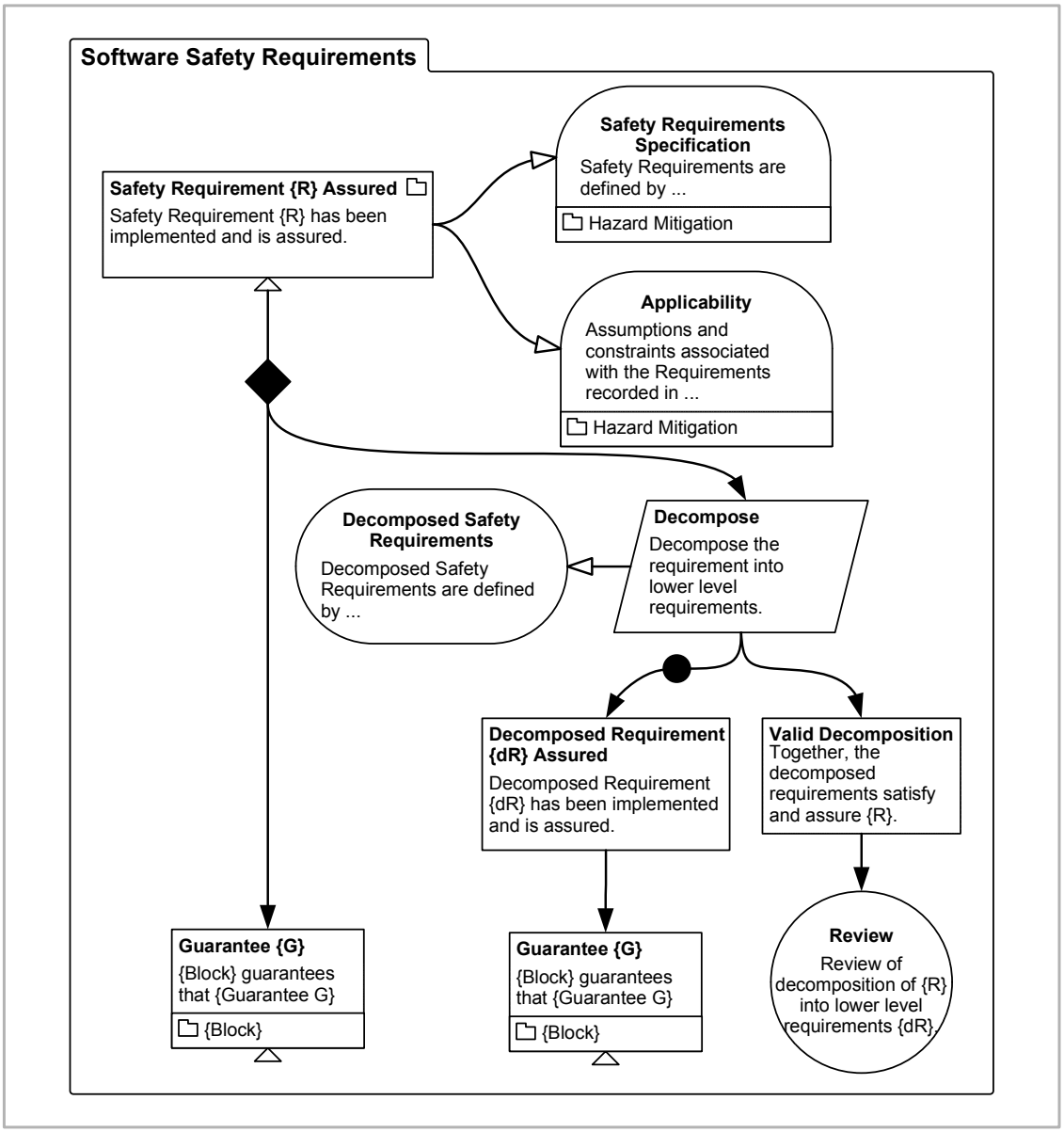
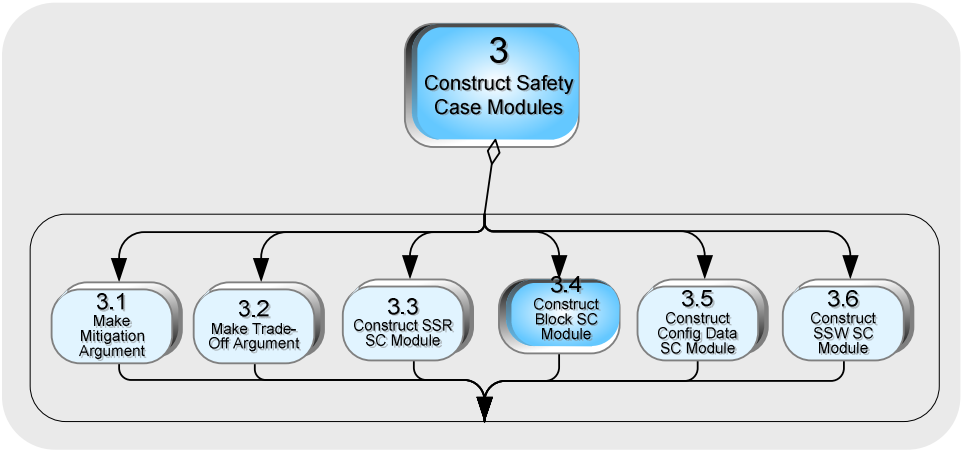


Figure 6-9: Form of argument for SSR

Detailed construction of this SSR Safety Case Module will depend on the nature of the system and the form of the SR Requirements, but several principles apply:

- All SR Requirements must be addressed
  There must be direct traceability between the output from hazard analysis to the top claim in this SSR Safety Case Module (*Safety Requirement {R} Assured*).
- Each SSR identified at the top claim must be supported by logical argument to one or more Block Safety Case Modules that offer appropriate claims to address the SSR, (e.g. as a claim that *Guarantee {G}* is supported),
  There may well be SSRs that map to more than one SC Block.
- SSR requirements may be decomposed within this SSR Safety Case Module if necessary in order to distribute responsibility to Block Safety Case Modules (as in *Decomposed Requirement {dR} Assured*). This may need to be supported by justifying claims about the decomposition where appropriate.

6.5 Step 3.4: Constructing a Block Safety Case Module



Objectives	
1. To provide a structured argument, supported by a body of evidence, that provides a compelling, comprehensive and valid case that the implementation of a SW Block that will, when integrated into the SC, satisfy the requirements for potential safety related behaviour and properties' (for each Block as identified in the SCA). a. To capture and make public all claims that the Safety Case Module offers as support for other Safety Case Modules b. To capture and make public all claims that the Safety Case Module requires of other Safety Case Modules for support of its public claims. c. To capture and collate all necessary public Context that may define, constrain or otherwise elaborate on the applicability of the Block for its intended usage. d. To provide argument and evidence to support claims that the Block satisfies implicit requirements to initialize correctly and to be "well behaved"	
Inputs	**Outputs**
1. Safety-related requirements [anticipated] to be satisfied by the Block. 2. Public interface for the Safety Case Module. 3. Engineering artefacts from development lifecycle. 4. Specification of the operating environment. 5. Specification of the intended operation. 6. Applicable safety standards?	Block Safety Case Modules, comprising - Public interface, including public Context - Elaborated argument - Linkage to evidence - Relevant instantiations of generic artefacts

Figure 6-10: Population of Safety Case Module content

Block safety case modules argue about the safety of part of the system. The part of the system they argue about is called a Block, but note that Blocks are not necessarily individual components or design modules. What Blocks contain is determined in step 2 by the safety case and system architects.

The core claims of the Block SC Module argument are that its guaranteed function and behaviour are assured. These claims relate directly to Dependencies and Guarantees.

The claims should be worded as generally as practical if the intent is to allow re-use of the safety case module.

A simple illustration of the relation of System and Safety Case Domain is shown in Figure 6-11.

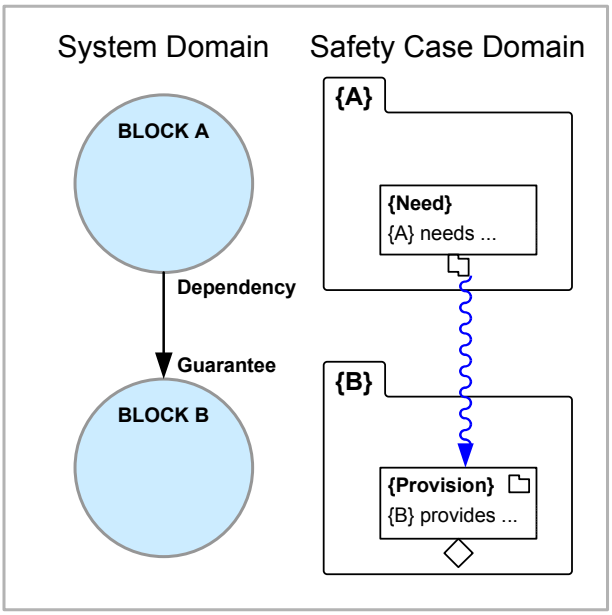


Figure 6-11 Illustration of System domain vs Safety Case domain

6.5.1 General form of a Block Safety Case Module

The purpose of a Block Safety Case Module is to support claims that the Block satisfies all its functional and non-functional requirements.

Figure 6-12 shows the overall form of a Block Safety Case argument.

The argument generally offers three public claims:

Assurance of the Block's *guaranteed behaviour*
which itself requires assurance that it will not be prevented by unwanted interactions

Assurance of the Block's initialisation
which may require assurance that any initialisation routine will be invoked at the appropriate point in the system's initialisation sequence

Assurance that the Block is well behaved
which means it will conform to some defined set of standards, rules, conventions such that it will not interfere with other Blocks' ability to assure their own guaranteed behaviour

A Block Safety Case Module will present its public claims subject to a collation of Public Context which may constrain or limit the validity of the goals. This should include context from any component level hazard analysis for this Block.

The Public Goals needing support may include:

Service {s}
which means the Block requires access to certain services provided by other Blocks or maybe another layer or group of Blocks within the overall SW system.

Goal: Data item {d} is provided via {c}
which indicates that for this guarantee, the Block is dependent on input data {d} to be provided by another Block via communication channel {c}

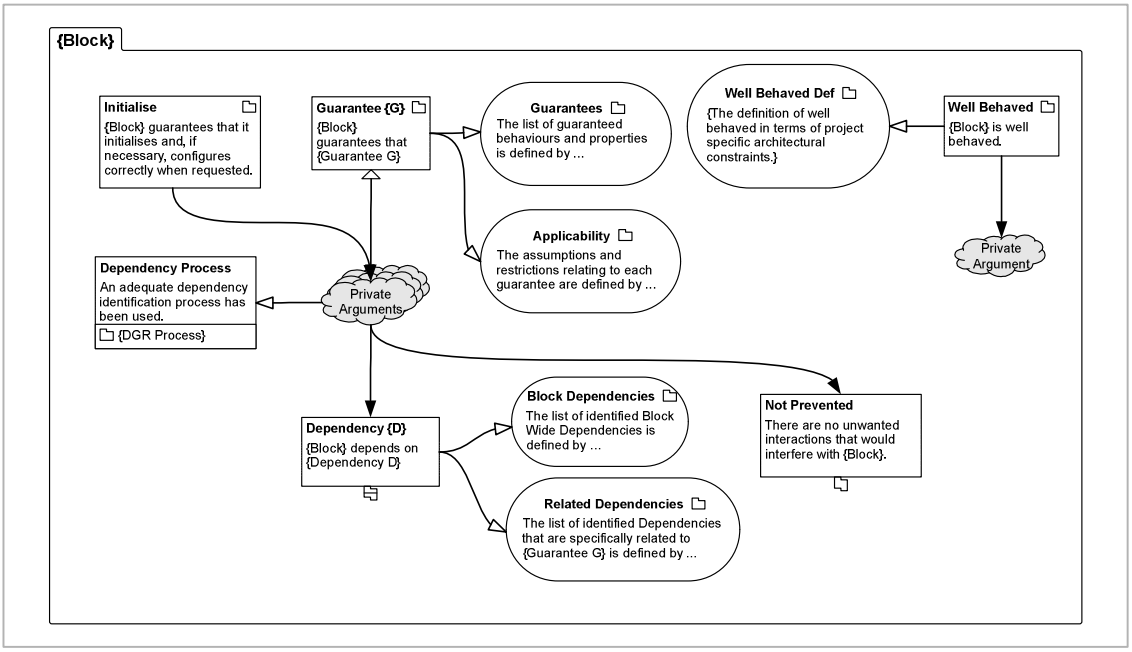


Figure 6-12: General Pattern for a Block Safety Case Module

Note that the pattern for a Block Safety Case Module can be tailored for usage in a particular system; it may still be *generic* in that a *template argument* is presented (once) and *instantiated* for each Block.

Patterns for each of the 3 top public claims in Figure 6-12 are found in later sub-sections.

6.5.2 Forming the DGRs for a Block

Building a modular software Safety Case requires argument about the “guaranteed” safety related functions and behaviours of the identified *blocks* of software and the interactions between them.

A principal *guarantee* offered by a *block* would ideally be unconditional:

> *“I guarantee to send this message (intact, within some predetermined time... etc.)”*

In reality a *guarantee* is likely to be in some way conditional on certain things beyond the control of the software *block* in question - it is *dependent* on things that must be guaranteed by other elements of the system.

> *"I can only make my guarantee provided someone else can guarantee that the requisite data path has been configured"*

The relationship between a *guarantee* and its *dependencies*, referred to as a *Dependency-Guarantee Relationship* or *DGR* is represented in Figure 6-13.

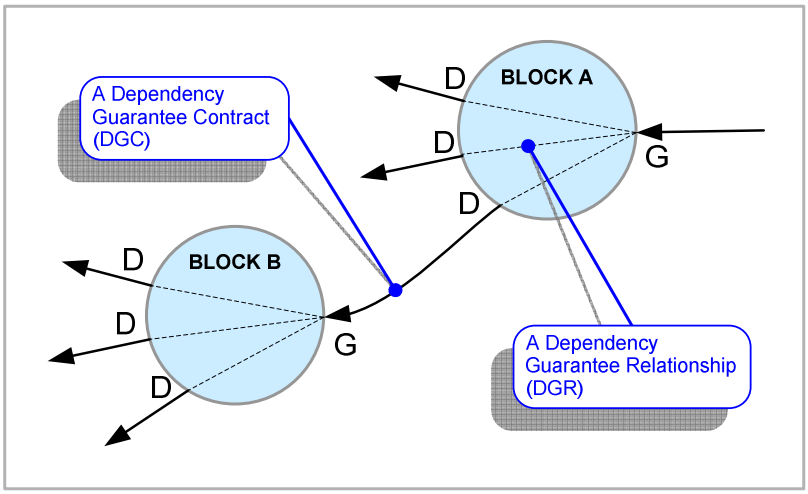


Figure 6-13: The relationship between *guarantees* and *dependencies*

The starting point in forming up the necessary set of DGRs for a given Block is to identify the required set of Guarantees for the Block.

If a Block corresponds to a single software component or unit of design then this is a straightforward analysis of the externally visible safety related requirements for that component and transposing each into the form of a Guarantee.

If a Block is made up of multiple software components or unit of design then this analysis may be performed at the block boundary or it may initially be related to those individual components and then the resulting Guarantees collated into the set that apply at the Block boundary.

In wording the Guarantees, it is important to consider the situations in which the Guarantee may fail and word the Guarantee to cover this: "I guarantee to produce this output - OR - return an error code".

It is also essential to consider necessary reference information, constraining assumptions and other information that needs to be associated with the Guarantee and ultimately propagated to the Safety Case claim that relates to the Guarantee.

It is then necessary to consider what the Block implementation may depend on if the Guarantee is to be upheld - the Dependencies, including preconditions or post conditions (such as the need for failure handling) that need to be reflected back to the consumer of the Guarantee.

It is possible that requirements analysis reveals Guarantees or Dependencies that are not "visible" at the Block boundary. These do not need to appear in a DGR.

6.5.3 The link from DGR to MSSC

DGRs *describe* safety-related aspects of an implementation; a Safety Case is concerned with presenting argument *about* the implementation.

DGRs form part of the foundation on which a Safety Case is built and the relationship between DGRs and the Blocks is shown in Figure 6-13.

On the one hand, DGRs describe properties & behaviour of an *implementation*, although it is not necessary to cover all properties and behaviour, only what is relevant to the safety argument.
"The (safety related) requirements state that the system (software) must do this... I hereby guarantee that it does (and there is evidence)"

On the other hand, SC arguments deal in “claims” (or “goals” in GSN) “*If the system is to be safe it must behave like this... I hereby claim that it does ... by pointing*

the reader to an appropriate (guaranteed) aspect of system behaviour / performance that is supported by appropriate evidence"

In the final safety case all safety requirements identified must be supported. Prior to integration the Block supplier may use his expert judgement and experience to anticipate the potential safety related behaviour and properties that will be required. The latter approach is particular relevant to COTs suppliers and re-usable sub-systems.

6.5.4 DGRs as context

A DGR is supporting information for the safety argument. A Safety Case Module can make a claim about a guarantee and support it with evidence and claims assured in other Safety Case Modules (including about the corresponding dependencies).

In the GSN form the DGRs are introduced as *context* to the argument.

The claim is made in Safety Case terminology; by invoking the DGR as context it points the reviewer to the technical detail as to how the claim is substantiated in the implementation.

Figure 6-14 below shows an example of how an argument fragment that needs to refer out to another Safety Case Module for support invokes a DGR as context to provide detail of the dependency that has to be satisfied within the implementation.

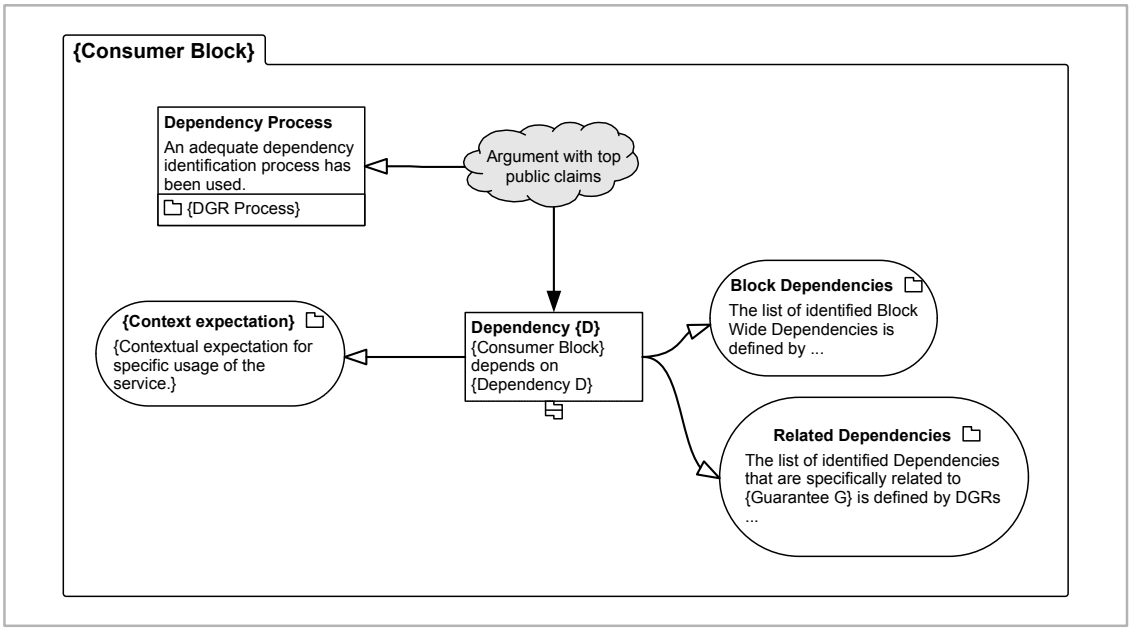


Figure 6-14 Consumer Module A invoking DGRs as context

Correspondingly the Safety Case Module that makes claims about provision of the service may invoke a DGR for the service as supporting context, see Figure 6-15 below

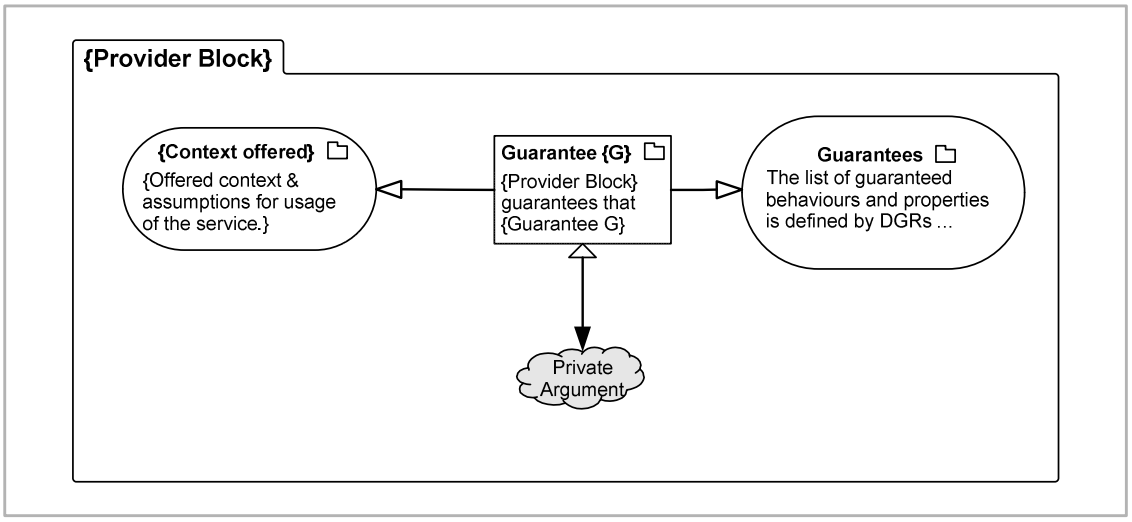


Figure 6-15 Provider Module B invoking DGRs as context

In Figure 6-14 the argument needs to be supported by a claim assuring a guaranteed sub-service in another Safety Case Module. So the fragment of argument corresponds to a DGR in the implementation domain.

Note that it is implicit for this view that a DGR as context has been propagated to the boundary (public goal)

It is important to note that an assurance of a guaranteed property or behaviour given "in the context of a DGR" implies that the DGR dependencies will be satisfied (by Safety Case Contract or equivalent in the integrated final SC), and this includes any precondition and failure handling dependencies that reflect back on the consumer.

Referring to the dependencies and guarantees in the DGR as context makes them part of the public interface of the Safety Case Module. It is not necessarily the case that the relationship between DGRs be considered part of the public interface. This information may be unavailable (for example in COTS software) or it may contain significant intellectual property that a supplier would benefit from containing.

6.5.5 Assurance of Guaranteed Behaviour

Figure 6-16 shows a pattern of argument for the "Guaranteed Behaviour" aspect of a Block Safety Case Module.

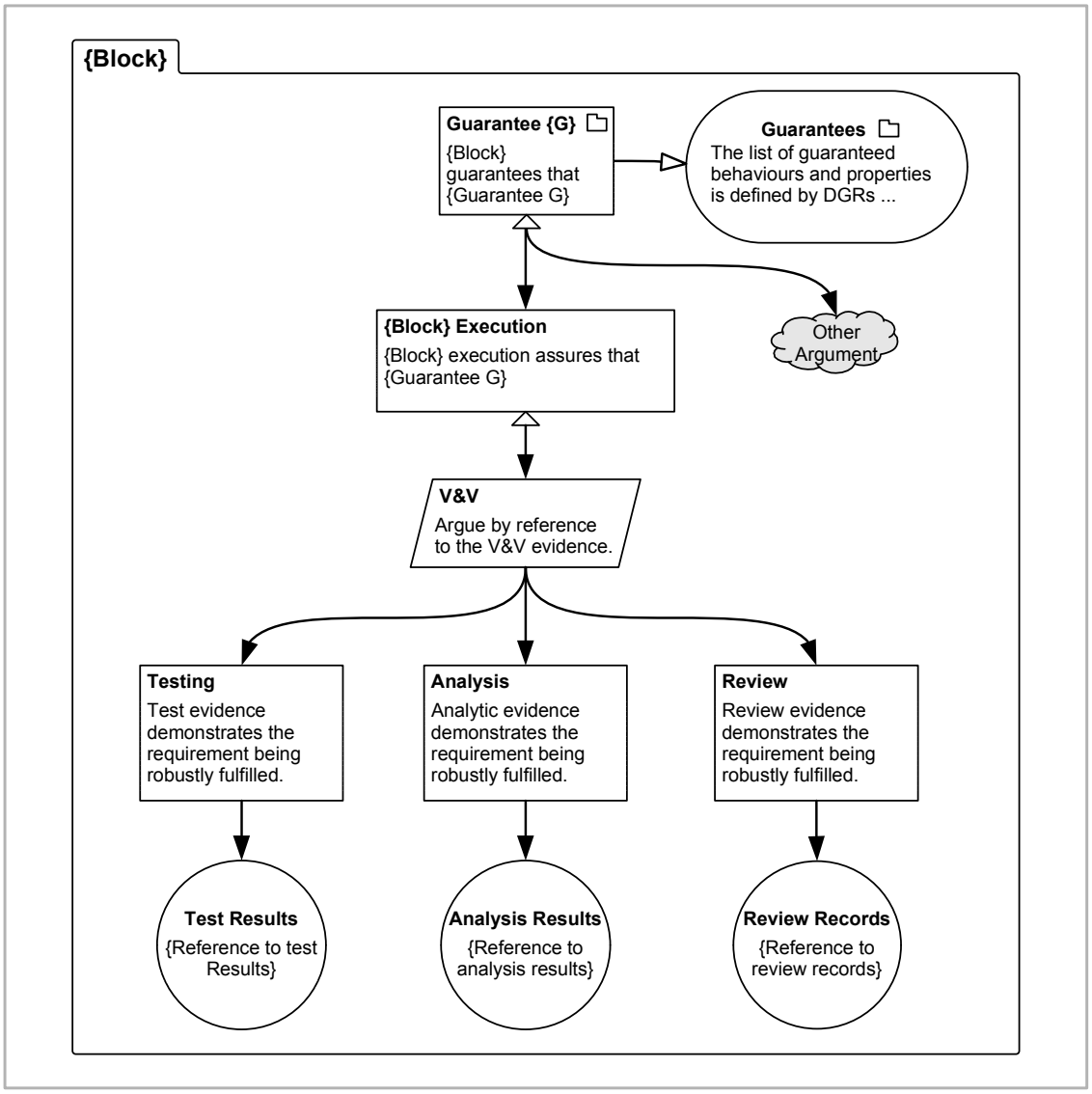


Figure 6-16: Pattern for Guaranteed Behaviour

This pattern offers support for a top claim that refers to the set of DGRs for the Block.

The Public Goal offered:

> **{Block A} guarantees that {Guarantee G}**
> which means the particular functional or non-functional requirement placed on the Block is supported to the requisite level of assurance, for the stated Context and is supported by appropriate V&V evidence.
>
> **Context will include:**
> A collation of limitations or constraints in usage or dependence on this guarantee (When does it apply or not apply)
> The DGR that captured the specific Guarantee with is dependencies
> A definition of the term "assurance" in this case

6.5.6 Block Initialisation

Figure 6-17 shows a pattern of argument for the "Block Initialisation" aspect of a Block Safety Case Module.

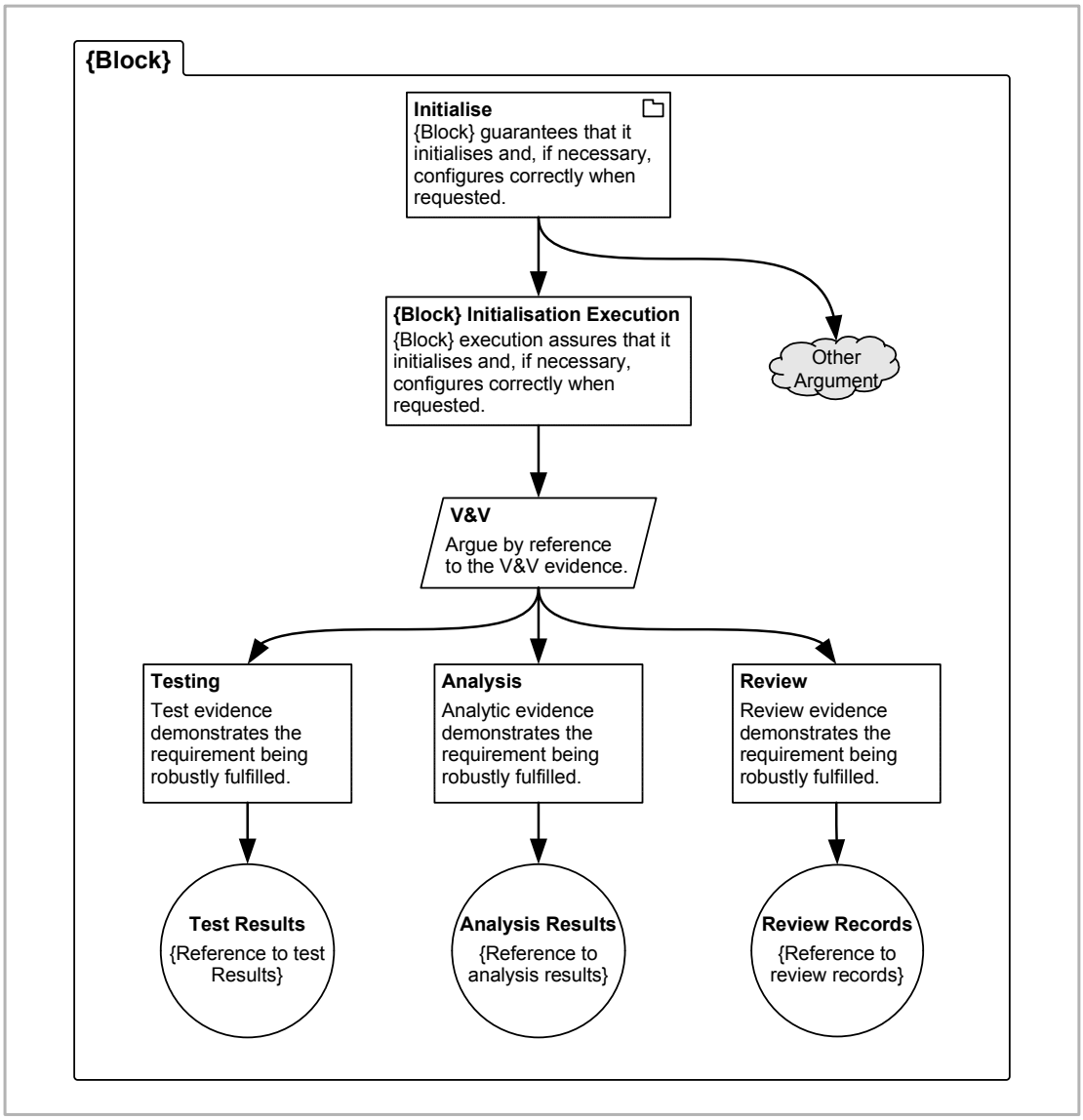


Figure 6-17: Pattern for Block Initialisation

The Public Goal offered is:

> **Block will initialise correctly when requested**
> which means that any initialisation that is required before the Block can support its Guarantees will be carried out in response to a call from that part of the system responsible for managing the initialisation sequence.
>
> **Context will include:**
> Assumptions concerning invoking of initialisation routines

The Public Goals needing support may include:

> **Goal: Service {s} is provided**
> where initialisation may depend on service routines to perform aspects of the Block's initialisation.

6.5.7 Block Well Behaved

Figure 6-18 shows a pattern of argument for the "Well Behaved" aspect of a Block Safety Case Module.

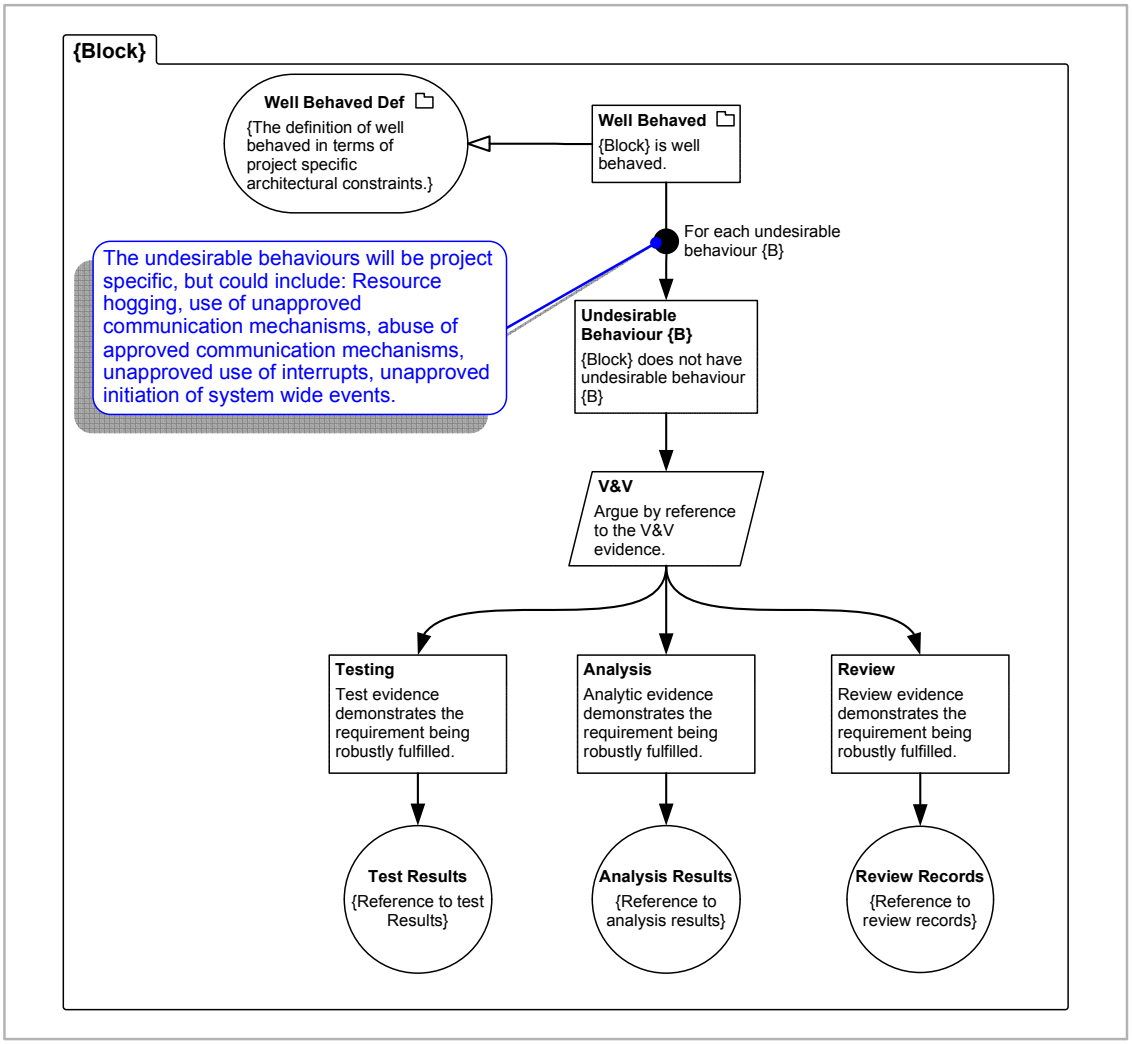
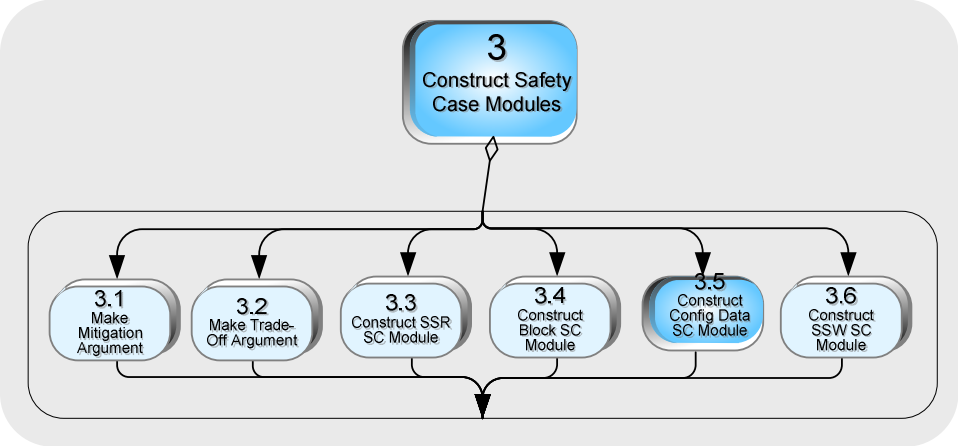


Figure 6-18: Pattern for "Well behaved" argument

The Public Goal offered is:

> **{Block} is well behaved**
> which means the Block will conform to the specified behavioural constraints, such as limits on resource usage, proper usage of authorised communications mechanisms, no attempt to use unauthorised interactions.
>
> **Context will include:**
> A definition of what "well behaved" is taken to mean - for the particular programme / deployment.

6.6 Step 3.5: Constructing a Configuration Data Safety Case Module



Objectives	
1. To present an argument supported by evidence that configuration data used in the configuration of a system is valid 2. To present an argument supported by evidence that configuration data used in the configuration of a system is correct 3. (optional) To present an argument supported by evidence that the configuration data was prepared by following adequate process.	
Inputs	**Outputs**
1. Definition of configurable aspects of the system design 2. System design, including instantiation of configurable aspects 3. Prescribed definition of the format for configuration data	2. A Configuration Data Safety Case Module - Public interface, including public Context - Argument body - supporting evidence of validity & correctness - (optional) argument that the data was prepared by following an adequate process

Figure 6-19: Configuration Data Safety Case Module

The purpose of a Configuration Data Safety Case Module is to provide assurance that, where the safety-related behavioural or intrinsic requirements placed on a Block of software depend on some aspect of the system's configuration, the configuration data made available at configuration time is both valid and correct.

Configuration Data might be considered as a Block, albeit data rather than code. The nature of the Safety Case Argument, however, will be rather different and therefore warrants separate treatment.

Configuration Data may comprise many elements from several sources to be provided to several components within a software system.

Configuration Data may affect the behaviour of the components or the system as a whole but it should be emphasised that this Configuration Data Safety Case Module is not concerned with claims about *correct behaviour of the system* given the specific data; that is the concern of SW Blocks whose behaviour is *configurable*.

Similarly, claims that the system is in fact *configured correctly* (i.e. "in accordance with the provided configuration data") is the concern of an Integration Safety Case Module.

The purpose of the Configuration Data Safety Case Module is simply to claim the data is valid and correct.

Figure 6-20 shows the general form of a Configuration Data Safety Case Module.

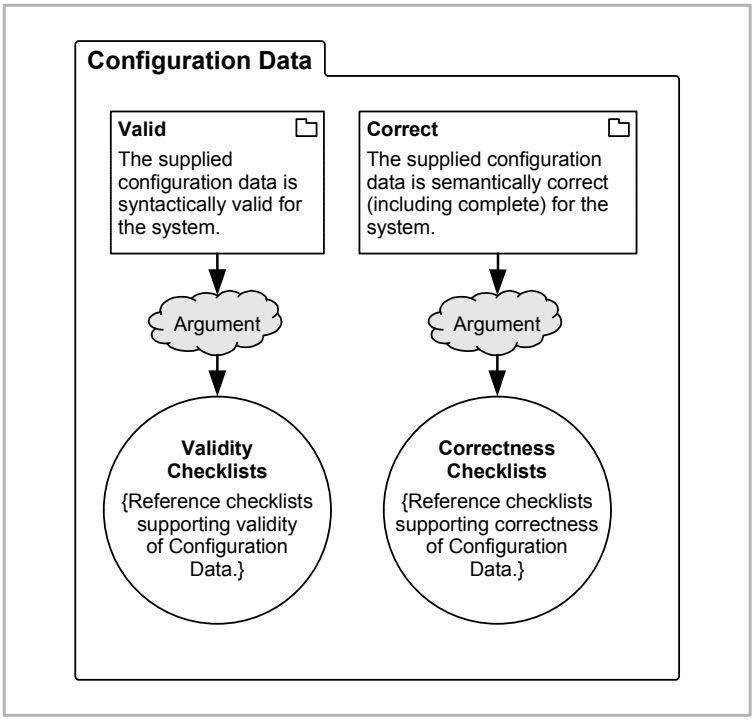


Figure 6-20: General form of a Configuration Data Safety Case Module

6.6.1 Configuration Data valid

In this context, **Valid** is defined to mean the configuration data set is complete, consistent, achievable, and conforms to the framework defined for usage by the relevant configuration software. Note that "complete" in this context means the use of the data would not fail because of missing data. If an aspect of configuration has simply been omitted the data set may be valid but also be "not correct".

The evidence will be completion of an appropriate set of validity checks that are made on the data provided to the software system against a set of validity rules.

The Public Goal offered:

> **Configuration Data is valid**
> which means the configuration data presented to the system will be complete, consistent and well formed
>
> **Context** will include:
> Definition of "valid".
> Reference to the system to which the Configuration data applies
> Reference to evidence (completed validation checklist)

6.6.2 Configuration Data correct

Correct is defined to mean "reflects the designer's intent" in that when correctly applied it will create a system configuration that completely conforms to the system designers intent. To be correct, a data set must contain *all* aspects that have to be configured. The evidence will be completion of an appropriate set of correctness checks that are made on the data provided to the software system against the system design data.

The Public Goal offered:

> **Configuration Data is correct**
> which means the configuration data presented to the system will be a complete

and true representation of the system designers intent (against system level design data)

Context will include:
Definition of "correct"
Reference to the system to which the Configuration data applies
Reference to evidence (completed correctness checklist)

6.6.3 Configuration Data preparation process

There may need to be a supporting *process argument* supporting a claim:

Adequate process followed in preparation of Configuration Data

This may be elaborated to:

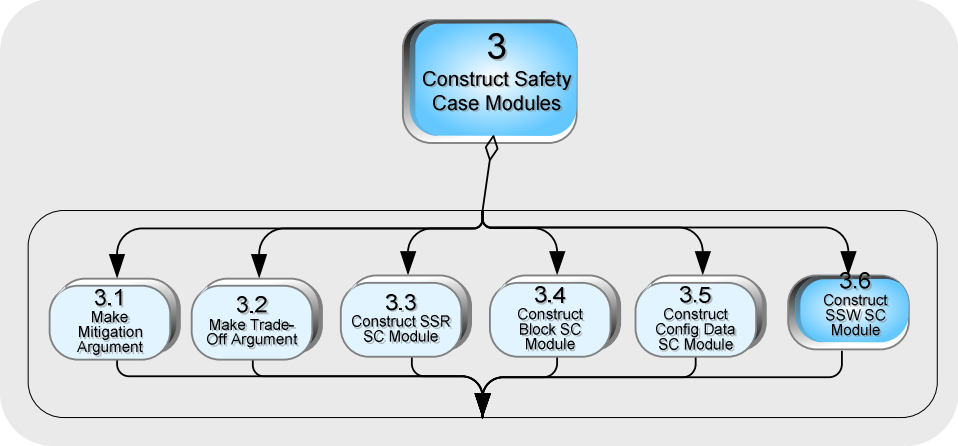
Configuration Data Generation Process: supports the claim that the process defined for the "generation" stage of configuration data development is adequate for the context.

Configuration Data Validation Process: supports the claim that the process defined for the "validation" stage of configuration data development is adequate for the context.

Configuration Data Verification Process: supports the claim that the process defined for the "verification" stage of configuration data development is adequate for the context.

Configuration Data Loading Process: supports the claim that the process defined for making the configuration data available to the configuring software is adequate for the context.

6.7 Step 3.6: Constructing an SSW Safety Case Module



Objectives	
1. To present an argument supported by evidence that any "System Wide" requirements are assured	
Inputs	**Outputs**
1. SCA for the system 2. System Wide requirements 3. Public interfaces for constituent Block Safety Case Modules	1. A System Wide Safety Case Module - Public interface, including public Context - Body of argument - Evidence

Table 6-3: Software System Wide Safety Case Module

Where the argument relating to a particular safety related requirement cannot easily be made by any individual Block in isolation it will be necessary to make that argument in a "system wide" context.

This means that support for the particular requirement will be a public claim within an SSW Safety Case Module rather than a Block Safety Case Module.

The argument may be elaborated down to a complete Solution within the SSW Safety Case Module; or it may be decomposed and some aspects passed down (as claims to be supported) to one or more Block Safety Case Modules. Which of these two approaches is taken will depend on the nature of the system-wide issue and the implementation & testing strategy adopted by the system developers.

For example a system may have an end-to-end maximum latency requirement such as "the system must respond to an input signal by triggering the appropriate output *within 10 ms*. and it may be that the sequence of operations between detection of the input and triggering the output involves several stages of processing by several Blocks.

In that case the goal of satisfying that latency requirement is best made as a "System Wide" claim. Having taken responsibility for that claim, the integrator has two options.

It may be that the claim can be decomposed into constituent parts and an allocation assigned to each Block (e.g. so that each can be required to complete its part of the operation in some portion of the 10 ms).

Or the claim may be supported more directly (within this SSW Safety Case Module) by reference to system level test (e.g. to an actual end-to-end timing measurement).

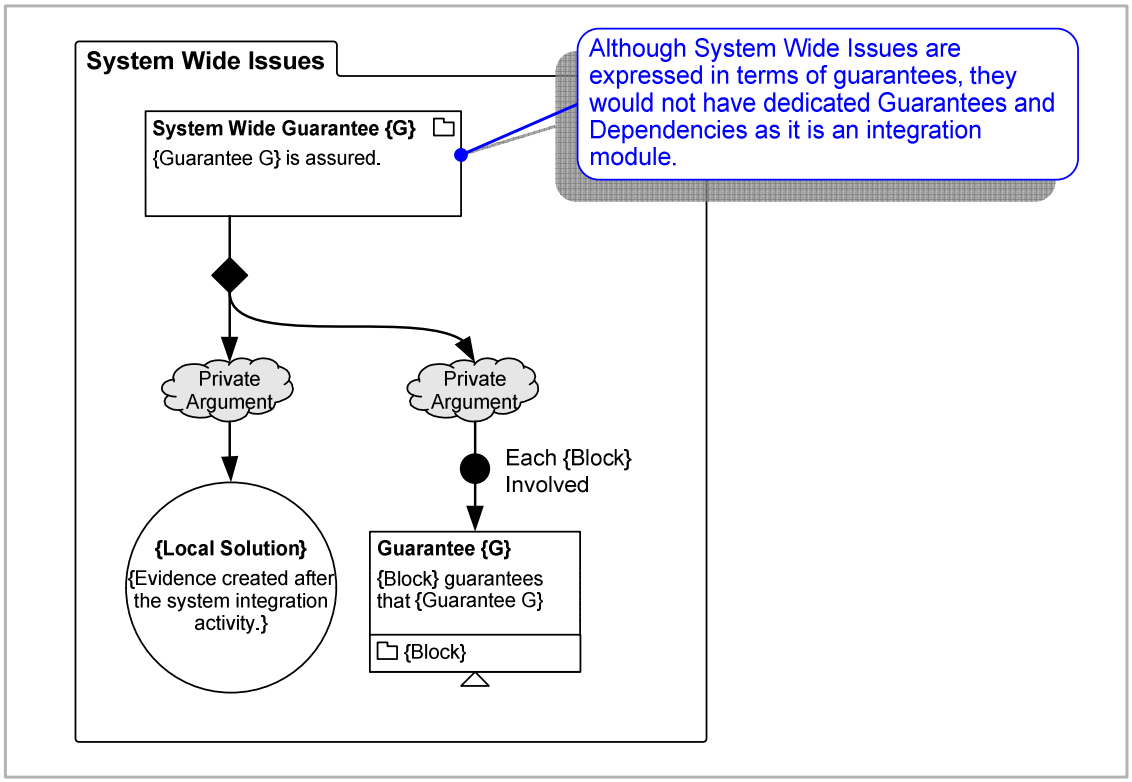


Figure 6-21: General form of a SSW Safety Case Module

Some aspects of system wide behaviour that might be handled in a SSW Safety Case Module are described below and in each case there may be the possibility of closing down the argument to a Solution at the SSW level or it may be necessary to pass down requirements to Block CSC Modules.

This is by no means an exhaustive list.

6.7.1 System Wide Aspect: Latency

The time data takes to pass from *provider* to *consumer* (data latency) may be the subject of a critical system requirement.

This latency may be made up of contributions by several components on the communication path. Each component (both hardware and software) could be identified in an argument that a latency requirement is met.

Or the requirement may be assured by actual measurement of the end-to-end timing without reference to individual contributing components.

6.7.2 System Wide Aspect: Shared Resource - Time

There will be an implicit system level requirement that there is sufficient resource to execute all required software operations; each operation will require an amount of processor time in order to complete. Thus all software components have an implicit dependency on the system integrator to assure that sufficient processing capacity is made available at the right time.

This system requirement may relate to a *cyclic* style scheduler where processes are scheduled in strict order within a fixed frame (cycle); or it may relate to a multi-processing style (priority pre-emptive) scheduler where individual processes may have time related (responsiveness) requirements independent of the others.

Again the argument has to be instigated at system level. It may conclude at system level by showing that all software components are successfully completed in the required time intervals.

Or it may be that each component is given an allocation and evidence provided to establish it does not break its allocation.

6.7.3 System Wide Aspect: Shared Resource - Memory

There will be an implicit system level requirement that there is sufficient resource to execute all required software operations; each operation will require a certain amount of memory on a processor in order to complete. Thus all software components have an implicit dependency on the system integrator to assure that sufficient memory is made available.

This is a further example where the argument has to be instigated at system level. It may conclude at system level or be decomposed to Block level.

6.7.4 System Wide Aspect: Shared Resource - Bandwidth

Any configured data path will have a natural limit to the quantity of data that can be transferred in a given time. This bandwidth is a limiting resource that has to be shared among users of the data path. If the data path is complex (involving multiple software and hardware components) the allocation of bandwidth to users may require detailed analysis and verification; each user may be making demands on different combinations of components.

Each Block, in making its guarantee of data transfers will need to be assured that sufficient bandwidth is made available for each critical transfer. Again, this argument has to be instigated at system level. It may conclude at system level or be decomposed to Block level.

6.7.5 System Wide Issue: Modes and Reconfiguration

Safety properties required of a system may depend on the operational mode.

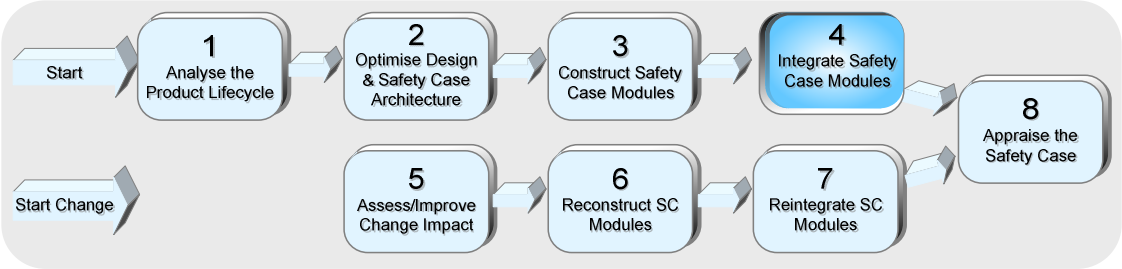
Software modules (particularly "SW Infrastructure" modules) may be 'aware' of modes but may have no understanding of what is 'safe' or 'unsafe' in any particular mode; the requirement will generally permeate multiple application Blocks.

The consequence of this is that an argument about safety of behaviour in the various modes may have to be made at *system* level.

Properties of software modules that relate to safety will need to be defined for each mode and necessary mode related assurances required of each software module - such as responding to mode-switch commands; performing or refraining from specific operations in any given mode, etc.

The *system integrator* takes responsibility for assuring the system is in the right mode at the right time; that it satisfies its mode-specific safety requirements; and is mode-switched or reconfigured as required. This may require safety argument instigated at system level. It may conclude at system level or be decomposed to Block level.

7 Step 4: Integrate Safety Case Modules



Objectives	
1. To bring together constituent Safety Case Modules to form an integrated Safety Case that meats its safety objectives.	
Inputs	**Outputs**
1. Allocated safety requirements Safety Case Architecture including Safety Case Module public interfaces 7. Specification of Context, including operating environment and usage Applicable safety standards Populated Safety Case Modules	1. Integrated Modular Safety Case

Figure 7-1: Integrate Safety Case Modules

This section describes what integration of Safety Case Modules into a complete Safety Case involves and how it is achieved.

A fully integrated product focused Safety Case will normally be based on the premise that all safety related requirements have been allocated to a set of supporting Safety Case Modules. Each of these Safety Case Modules may in turn appeal to other Safety Case Modules for support forming a complete top down argument, potentially with many threads each leading to a solution supported by evidence.

The subject of the Safety Case is the *implementation* of a system. Artefacts from the *development lifecycle* may be called up as Context or supporting evidence but the complete argument & evidence set is focused on assuring the safety of the delivered system.

The steps in achieving this are shown in Figure 7-2

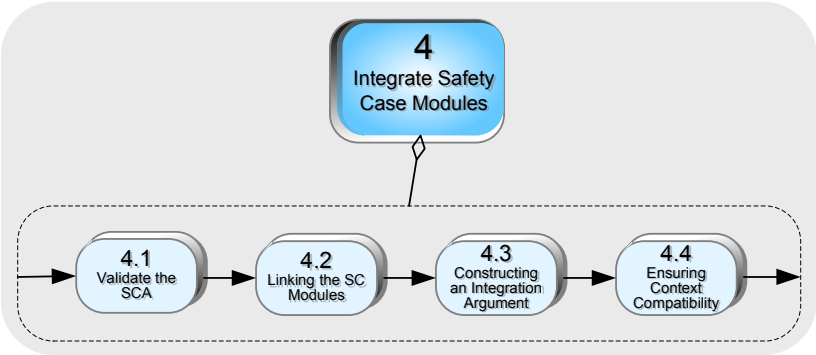


Figure 7-2: Steps in integrating a modular Safety Case

The Safety Case Architecture identifies the set of Safety Case Modules with their *public interfaces* and shows how Safety Case Modules relate to each other.

The relationships between Safety Case Modules will normally have been outlined in the SCA during Step 2; however it is likely that further refinement of the Safety Case Modules

relationships and specifically their Public Interfaces will need refining when it comes to integration. The first step in integration will therefore be to validate the SCA. See section 7.2

Refer to Figure 13-11 for an example Safety Case Architecture diagram.

Integration of a set of Safety Case Modules to form a complete Safety Case may simply require the Safety Case Modules to be linked using *Public-A*way goal referencing. See section 7.2

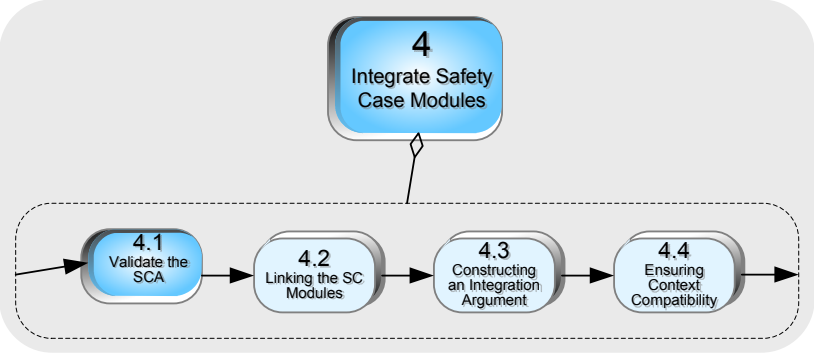
It may be that an integrator choses to form Safety Case Contracts between Safety Case Modules; see section 7.2.1.

SC Contracts between Safety Case Modules may be related to the implementation by way of Dependency-Guarantee Contracts (DGCs); see section 7.2.4.

There may be aspects of the complete Safety Case that involve additional "integration argument", see section 7.3.

Finally, it is essential to ensure that each interaction between Safety Case Modules is valid, given the sum total of Context, see section 7.4.

7.1 Step 4.1: Validating the SCA



The Safety Case Architecture for a system will identify all the constituent Block Safety Case Modules and Configuration Data Safety Case Modules, each with its Public Interface declaring the Goals that they provide and those they need to be supported, together with the collated Context at the Safety Case Module boundary.

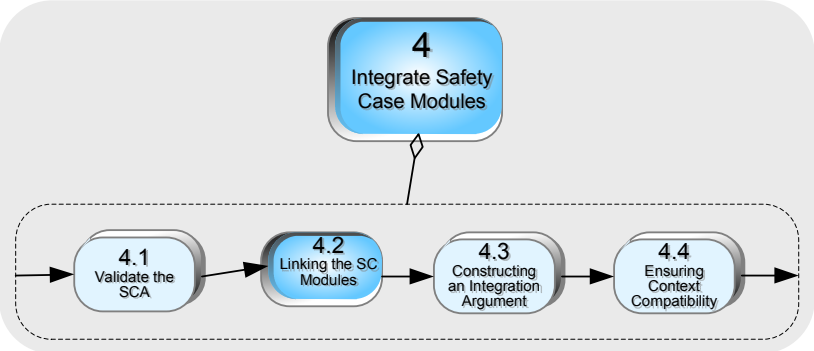
It also identifies other types of Safety Case Module that may be required, such as SSR, SSW.

The SCA captures the inter-relationships between Safety Case Modules and these resolve to linkages between *consumer* and *provider* goals. Every *consumer* goal, wherever it occurs, must be supported by an equivalent *provider* goal from some other Safety Case Module. A safety argument cannot be considered complete until all these inter-relationships are satisfied down to Solutions, leaving only the *top claim(s)* presented as the goal(s) that satisfy the overall safety objective(s).

Firstly an integrator will need to confirm that all constituent Safety Case Modules are delivered in accordance with the SCA.

Then, working from the top goal(s), it is necessary to trace to each *consumer* goal and ensure it is in fact satisfied by a valid *provider* goal and that the representation of the Safety Case (in GSN or other representation) captures all the relevant linkages.

7.2 Step 4.2: Linking Safety Case Modules



The integration of Safety Case Modules involves identifying how each *consumer Goal* (*requiring support* from another Safety Case Module) is to be supported by a *provider G*oal in another Safety Case Module, and forming the requisite linkage.

A **Public Goal** is a goal declared within a Safety Case Module that is accessible from another Safety Case Module. It will either be a *provider* to (offering support to another) or *consumer* of *(*requiring support from another) the other Safety Case Module.

An **Away Goal** reference is a reference made within one Safety Case Module that links to a Public Goal in another Safety Case Module. The *Public Goal* it is linked to may then take on and elaborate the supporting argument (as *provider*) or it may be appealing (as *consumer*) to the *Away Goal* to take on and support the argument.

Linkage between Safety Case Modules is formed using an *Away Goal* in one module that refers to a P*ublic Goal* in another module, as shown in Figure 7-3.

In this diagram the way line between an Away Goal reference and its referenced Public Goal is not strictly a "solved by" link as the two "goals" are actually the same goal; however for the purpose of the logical flow of the argument it can be seen as such.

Note that the Safety Case Module publishing a *Public Goal* "owns" that goal; an *Away Goal* is only a reference and takes its definition from the referenced *Public Goal*. It is an *integrator's* responsibility to ensure that an Away Goal reference is correct and matches the wording of the Public Goal it refers to; this may be assisted by a (argumentation or GSN) tool.

It is also the *integrator's* responsibility to offer any necessary *justification* as to why it can be claimed the *provider* satisfies the *consumer*.

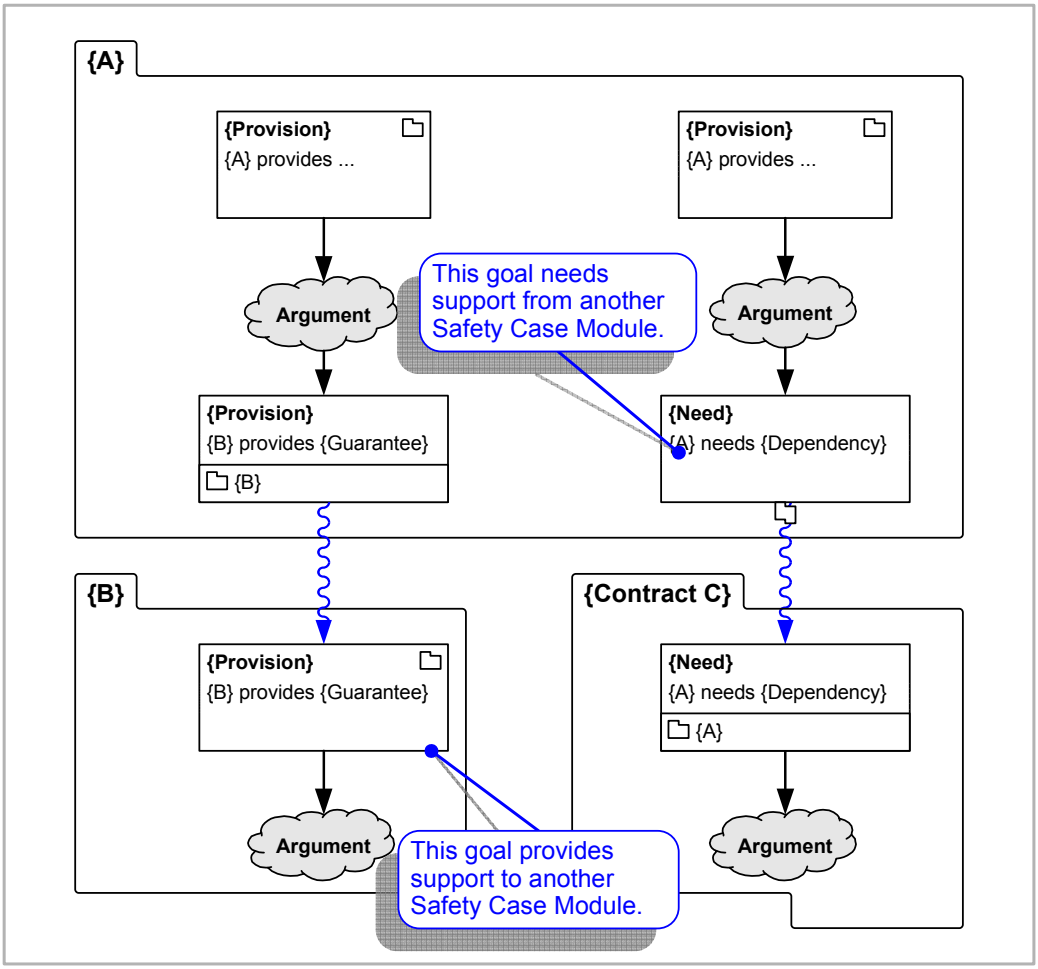


Figure 7-3 Linkage between a public goal in one module to an away goal in another

7.2.1 Using Safety Case Contract Modules

There are reasons why it may be desirable to interpose an actual Safety Case Module between a *consumer* Goal and a *provider Goal*:

- It may be desirable to de-couple the two so that neither has knowledge of the location or identity of the other - only the SC integrator needs that information.
- It may be that one is not available when the other is being constructed and a reference cannot be made directly.
- It may be necessary to introduce *justifying argument* to support a claim that the *provider Goal* does in fact satisfy the *consumer Goal*.

This may be achieved by constructing a *Safety Case Contract Module* between them as shown in Figure 7-4.

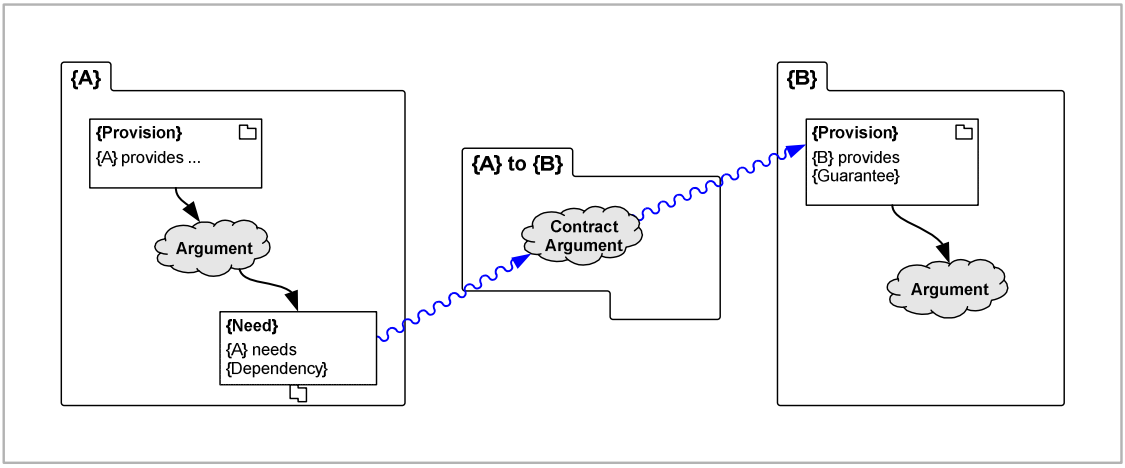


Figure 7-4: Example of Safety Case Module Integration by Contract

A Safety Case Contract Module is a module of argument that provides support to one Safety Case Module by linking it to support provided by another Safety Case Module.

A Safety Case Contract Module allows a *consumer* Safety Case Module that needs support to declare a *goal* that is part of the public interface that needs to be supported from elsewhere; and a *provider* Safety Case Module to declare a *public goal* that it can offer in support independently. It decouples the direct link between *consuming* and *providing* Safety Case Modules so that neither needs to identify the other directly. In theory the identity of the *provided* goal can be mapped to a *consumer* goal with a different name (assuming the goal is in effect the same).

Because it is a Safety Case Module in its own right, it also provides the opportunity to incorporate justifying argument (why the integrator believes the *provider* can satisfy the *consumer* in the particular Context.

Where a similar form of Safety Case Contract argument is used repeatedly for many instances of linkage between Safety Case Modules it may be convenient to present the argument in *generic* form and then provide an instantiation table to elaborate each instance.

For example in the argument in Figure 7-4 Safety Case Module {A} has a {need} which is ultimately satisfied by a {provision} in Safety Case Module {B} and that could represent a generic argument for a claim that a particular set of services are provided. The integrator could then create an instantiation table such as is shown in Table 7-1 to show contracted provision of those services.

{need}	Context	{provision}	Context
Send a message	128 characters	SendMessage	256 characters
Receive a message	128 characters	RecMessage	128 characters

Table 7-1: an example of an instantiation table for a generic Safety Case Contract

7.2.2 Forming DGCs between Blocks

Matching interdependencies between software components in the implementation is an integrator's task; similarly, matching one Block's *dependencies* with another Block's *guarantees* in the DGR domain is an integration task.

In Figure 7-5 two DGRs are shown and the link between them indicates how a *dependency* in one is satisfied by a *guarantee* in another; this is referred to as a *Dependency-Guarantee Contract* or *DGC*

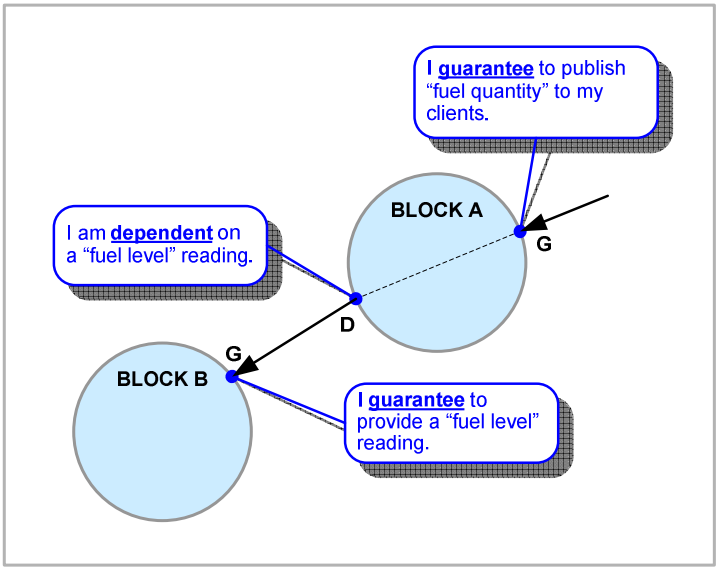


Figure 7-5: A Simple D-G Contract between Blocks

Block B is providing a guarantee to support Block A's dependency

The *block* that is offering up the *guarantee* (Block B) is referred to as the *provider*; the *block* that has the *dependency* being satisfied by the guarantee (Block A) is referred to as the *consumer*. Note that a *block* will normally be both a *provider* of *guarantees* and a *consumer* via its (various) *dependencies.*

Each link from a D to a G link is a DGC and apart from the supplier-provider aspect there will be other information that the integrator requires or has to provide to make the DGC complete. For example a DGC for a data interdependency may need to include and compare the data format and units; the integrator may need to specify details of the communication channel.

DGCs may be captured in generic form and tabulated for each instantiation.

A simple representation of a DGC is shown in Table 7-2

Dependency – Guarantee Contract			**<Block Name>.<DGC Name>**
Consumer Dependency	**Integrator**	✓	**Provider Guarantee**
<Block A Name>.<DGR Name>.<Dependency Number>	**has SC Contract with**		**<Block B Name>.<DGR Name>.Guarantee**
<*Concise Definition* as in the DGR>	is supported by		<*Concise Definition* as in the DGR>
<other *Definitive Context* as in the DGR>	is consistent with		<other *Definitive Context* as in the DGR>
…	is consistent with		…

Table 7-2: Example of a simple DGC

A DGC captures the coupling of a provider's guarantee to a consumer's dependency:

Note that three views are represented in a DGC:

- A **Consumer**
  declares a *dependency* - a need for a specific guarantee to be satisfied - with specific (expected) context.
- A **Provider**
  offers a *guarantee* to be used if the specific (offered) context is compatible
- An **Integrator**
  has the responsibility for matching an appropriate (compatible) guarantee to each declared dependency.
  In general this "matching" will result in the integrator adding certain linking information such as a mechanism, assignment of resource or other.

There may be multiple *instantiations* of a DGC capturing similar interactions between Blocks, for example where a particular Block requires several data items that are to be provided by another Block. This can be captured as multiple instantiations of a DGC in a DGC *instantiation* table such as Table 7-3.

Dependency – Guarantee Contract		**<Block Name>.<DGC Name>**	
Consumer Dependency	**Integrator**	✓	**Provider Guarantee**
<Block A Name>.<DGR Name>.<Data1> Dependency	**has SC Contract with**		**<Block B Name>.<DGR Name>.Guarantee of <Data2> Provision**
Block A <data1> needed	is supported by		Block B <data2> provided
<data1 units>	is consistent with		<data2 units>
Northern hemisphere only	is consistent with		North of the equator
…	is consistent with		…

Table 7-3: Example of a simple instantiation table

7.2.3 Construction of a Safety Case Contract Module

Figure 7-6 shows an example of the essential content of a Safety Case Contract Module.

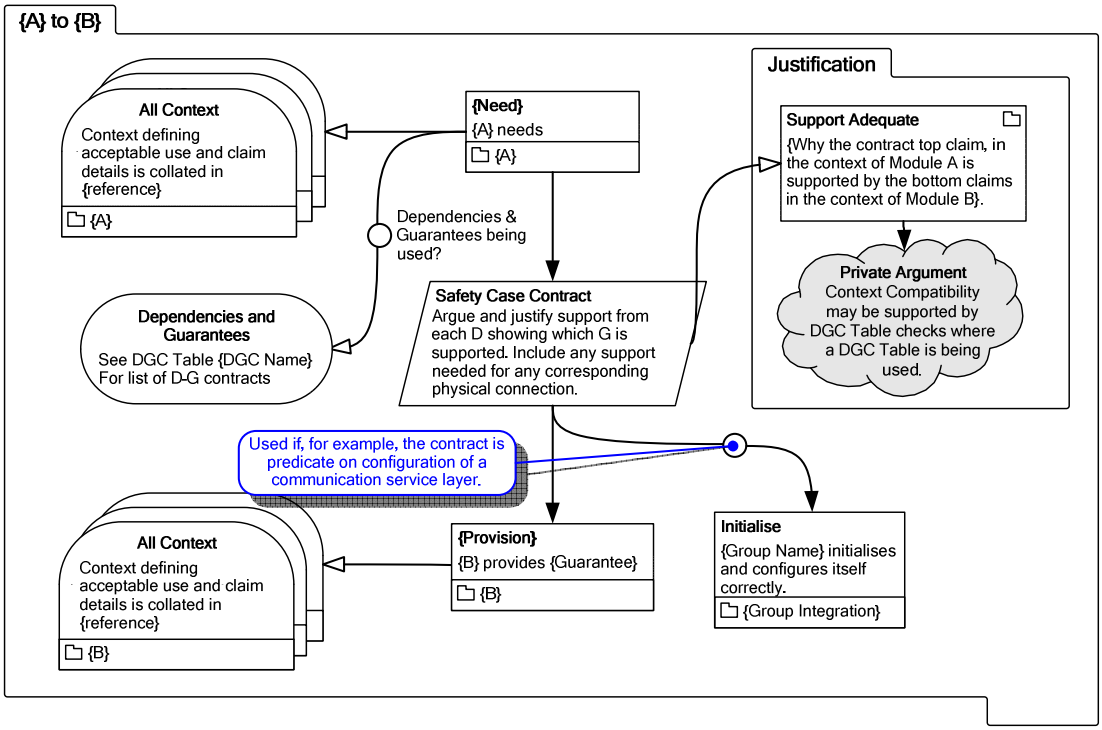


Figure 7-6: Example Content of a Safety Case Contract Module

The top *Away Goal* exists to support a *Public Goal* in a *consumer* Safety Case Module that requires support.

The bottom *Away Goal* exists to propagate that need to a *Public Goal* in the *provider* Safety Case Module where the argument will be developed.

The *Context* bubbles attached to the top and bottom goals refer to the necessary information about the *consumer* and *provider* Safety Case Modules that might need to be considered in establishing whether the required and provided goals are in fact compatible.

The *justification* (which is shown *contained* in a dedicated *Safety Case Module*) satisfies the *strategy* that sits between the top and bottom goals to justify why the integrator believes the *provided* goal adequately supports the *consumer, given* all Context including the *level of assurance* expected.

7.2.4 Relationship between DGCs and Safety Case Contracts

***NB**: The term Safety Case Contract is used to describe the linkage between a consumer goal in one Safety Case Module and a provider goal in another. This is formed in a derivative of Modular GSN for which there are two acceptable representations using public and away goals directly or more typically via a Safety Case Contract Module.*

A DGC captures the required link between a dependency declared in one DGR and a satisfying guarantee provided by another. [Such a link is only relevant or meaningful if it spans a boundary between one *Block* and another.]

In a modular SC argument composed of Safety Case Modules that argue about particular Blocks in the implementation domain any linking between Dependencies and Guarantees across *block* boundaries will have a correspondence with links across SC boundaries. A *consumer* Block Safety Case Module will contain a claim that a Dependency is supported and the *provider* Block Safety Case Module will provide a claim that offers that support. Such

linkage has to be formed between the two Safety Case Modules – either directly or via a Safety Case Contract.

A DGC will be introduced as *context* to a Safety Case Contract (or other inter-SC Module link). This is shown in Figure 7-7.

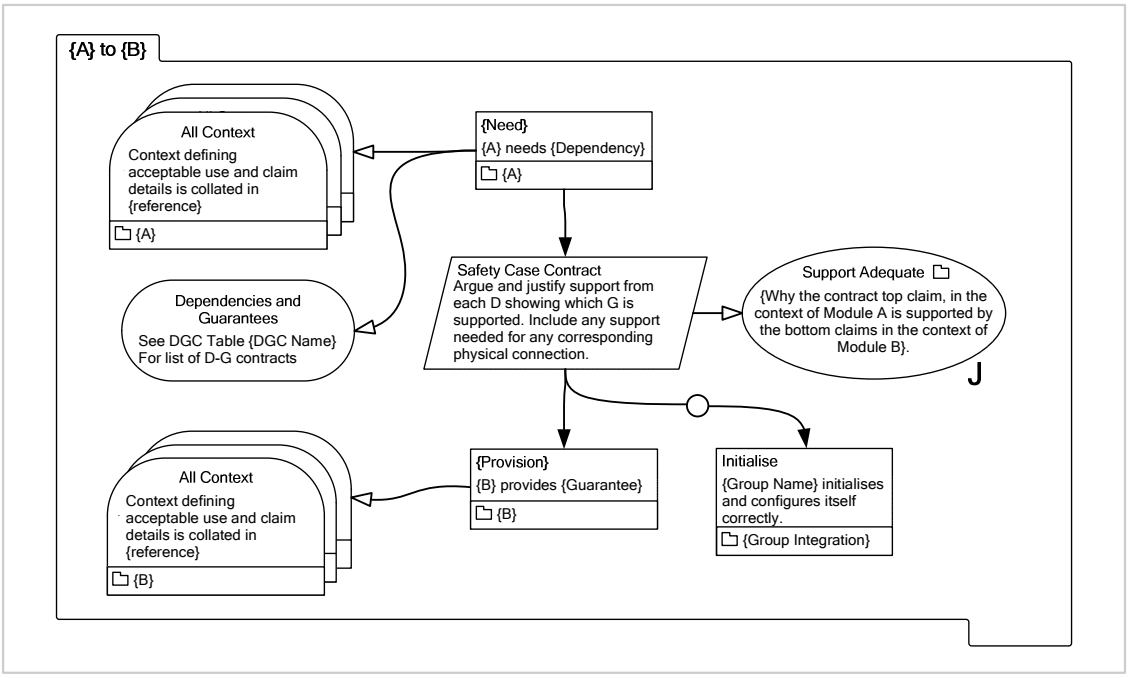
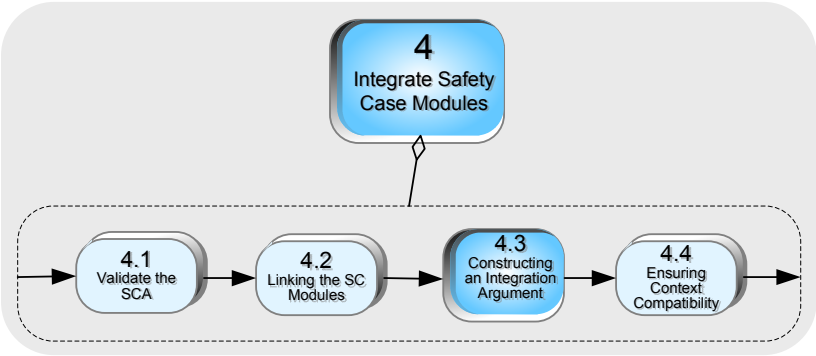


Figure 7-7: Example of using DGCs as Context

Note that Safety Case Module interactions may not all relate to DGCs; a goal claiming provision of a data item is sufficiently assured may derive from the DGC relationship; a claim that "appropriate processes have been followed" may not.

7.3 Step 4.3: Constructing an Integration Argument



7.3.1 The need for an Integration Argument

There are situations where the integration of Safety Case Modules requires additional argument:

- Group Support
  To provide assurance of certain behaviours or properties that are needed by several individual Blocks. The need to assure "no unwanted interactions" between blocks is an

example as it requires a central claim to be made that may in turn require support from each Block Safety Case Module.

- Republishing Claims
  To propagate claims from individual Block Safety Case Modules to outside the group.
- Accessing Claims
  To provide support to the integrated set of Safety Case Modules in the form of shared services to be provided by some other layer or group of Blocks
  e.g. when Block-to-Block interactions within an application layer are supported by a set of communications services in a service layer.
- Group Responsibilities
  To make claims about the set of Safety Case Modules *as a whole* to support SSR if they cannot be made by any individual Safety Case Module; end-to-end timing across a set of functions could be an example and is typically covered in a "Software System Wide" Safety Case Module.

Two types of module are implied here: an **Integration Safety Case Module**, which deals with the first three bullets and a (**Software**) **System Wide Safety Case Module**. Their inter-relationships are illustrated in Figure 7-8.

Blocks A & B may interact (eg, if one calls functions in another) and both their SC Modules will normally require support from an Integration Safety Case Module (eg for access to shared services).

System wide issues may require support from Block Safety Case Modules to contribute to a system wide argument (e.g. to constrain consumption of a shared resource).

It may be that the Integration argument requires support from a "system wide" argument (e.g. when assuring non-interference across a system a claim that all blocks are well behaved may be needed).

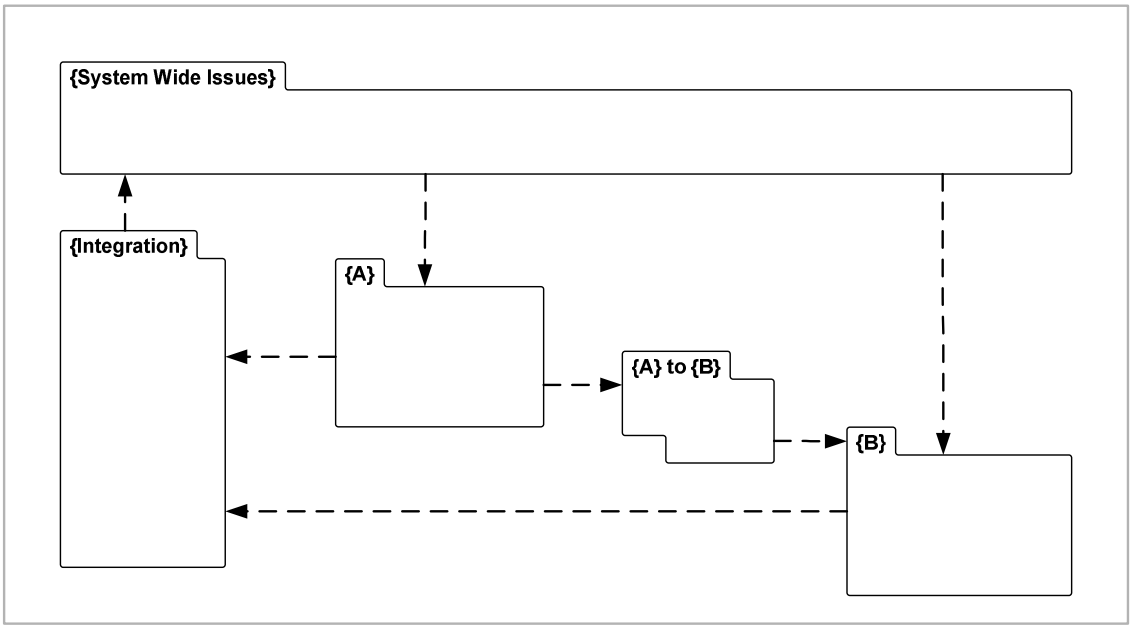


Figure 7-8: Safety Case Modules that may be involved in the integration of Block Safety Case Modules

7.3.2 Integration Safety Case Modules

An Integration Safety Case Module is formed in the same way as any other Safety Case Module, offering Public Goals to *provide* support and declaring *consumer* Goals to be supported elsewhere.

The purpose of an Integration Safety Case Module is to take collective responsibility for a group of Block Safety Case Modules and offer assurance that they can depend on such things as resource allocation, partitioning, interaction mechanisms and other common services that are provided by the environment in which the Blocks themselves operate.

Any grouping of Blocks can be handled as the subject of an "integration Safety Case Module"; a layered architecture is a specific grouping in which, for example, an "application layer" comprising a set of application Blocks is integrated and supported on a provided SW platform layer that provides "Infrastructure" services and system management functions.

A pattern for an Integration Safety Case Module is shown in Figure 7-9, Figure 7-10 and Figure 7-11.

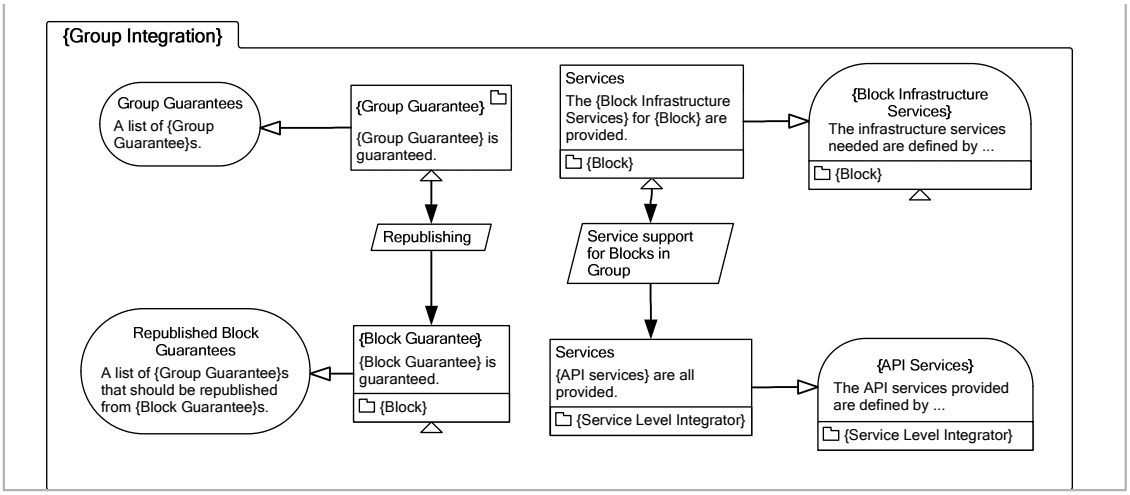


Figure 7-9: Pattern for an Integration Safety Case Module Part 1

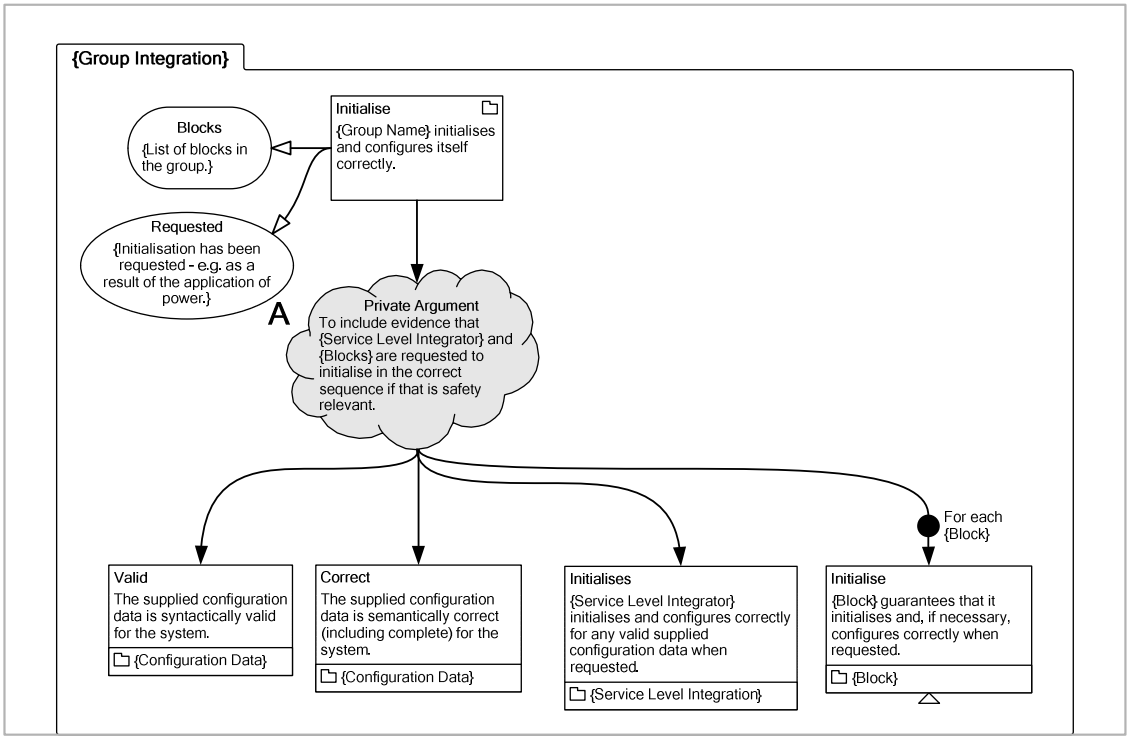


Figure 7-10: Pattern for an Integration Safety Case Module Part 2

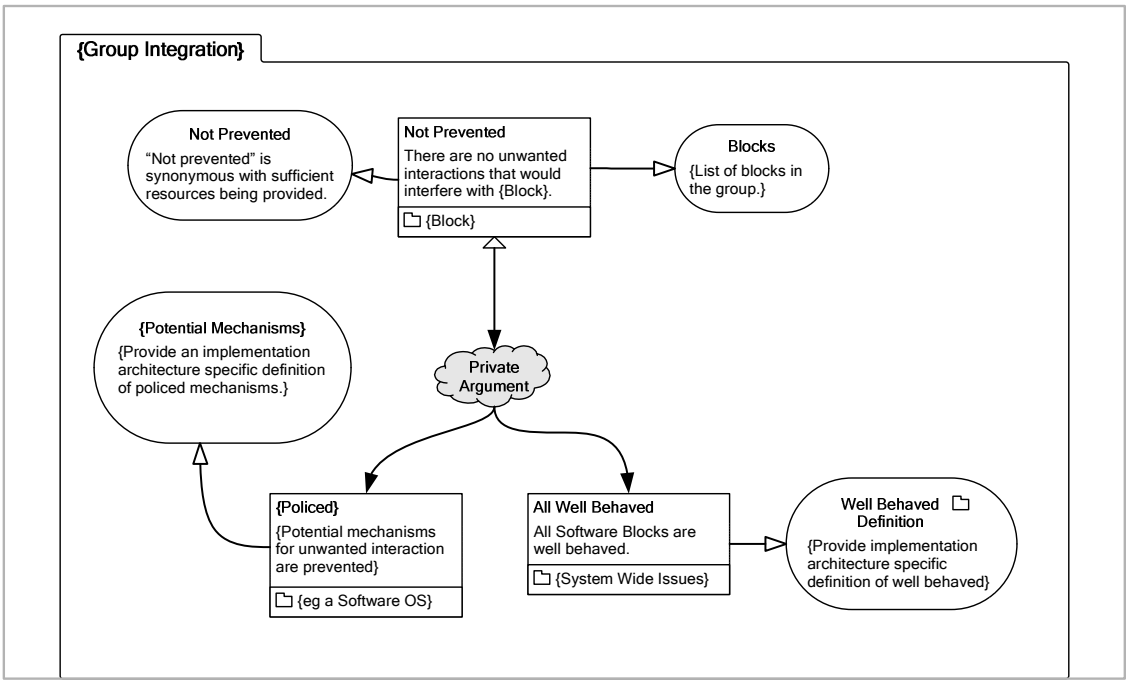


Figure 7-11: Pattern for an Integration Safety Case Module Part 3

In this pattern an Integration Safety Case Module is offering Public Goals that an application Block Safety Case Module (or another Integration Safety Case Module) can appeal to for:

Infrastructure Service {s} is assured
which provides a claim (on behalf of other groups or layers) to support those services that other layers or groups of SW Blocks will provide.

Correct initialisation is assured
which allows Blocks to place responsibility on the integrator to offer assurance that initialisation will be achieved.

Correct configuration is assured
which allows Blocks to place responsibility on the integrator to offer assurance that configuration of relevant aspects of the system will be properly configured (using valid & correct configuration data).

{Block} is not prevented
which provides each Block with assurance that it will not be prevented from fulfilling its (safety related) obligations by unwanted interactions.
This may be addressed by arguing that unwanted interactions are actively *prevented* or that requiring each Block to offer assurance that it will be *well behaved* - i.e. will not attempt to interact with other Blocks by unauthorised means. The two approaches are complementary; both may apply.

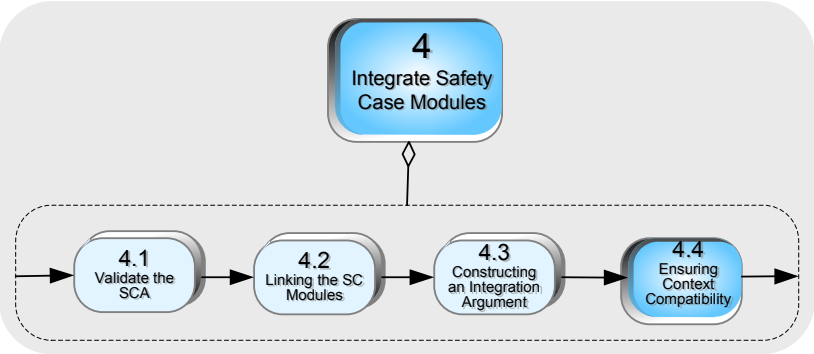
Group / Layer Public {Guarantee} is assured
which makes available a declared set of Guarantees beyond the scope of this group or layer.
[A Group or Layer means "whatever collection of Blocks are the subject of this Integration Safety Case Module]

Generally the integration Safety Case Module will need support from other Safety Case Modules:

- Assurance that any configuration data is both valid and correct
  which should be provided for by a "Configuration Data" Safety Case Module

- Assurance that required communication & other "API" services are provided by other identified layer or group.
  which should be provided by the integration argument from a providing group / layer
- Assurance of memory allocation & protection
  which may be supported by claims in a SSW Safety Case Module
- Assurance of temporal partitioning or scheduling
  which may be supported by claims in a SSW Safety Case Module
- Assurance that Blocks are "well behaved"
  to be provided by each Block Safety Case Module.

7.4 Step 4.4: Ensuring Context Compatibility



Justifying the linking of a *provider* goal to support a *consumer* goal firstly involves confirming that the provided goal does semantically match the consumer's goal, in other words that the provider offers at least what the consumer expects.

The validity of an offered goal is defined and constrained by its quoted *Context*. This offered Context must not conflict with the expected Context, otherwise the provided goal cannot be said to support to consumer's need *in the Context*.

An essential second part to the justification, therefore, is to **ensure that the validity of the provided goal is compatible with the consumer's expectation - ensuring *Context compatibility***.

So, during Safety Case integration, links between Safety Case Modules must only be made where they are contextually compatible. To ensure this, every safety relevant element of Context for all the claims being linked needs to be scrutinised. Any incompatibilities between the linked claims will invalidate the integration of the Safety Case Modules.

It is presumed that in constructing Safety Case Modules relevant Context has been fully captured as described in section 6.1.5.

Scrutiny of the links needs to be performed by a relevant expert similar to the Context capture earlier in the process.

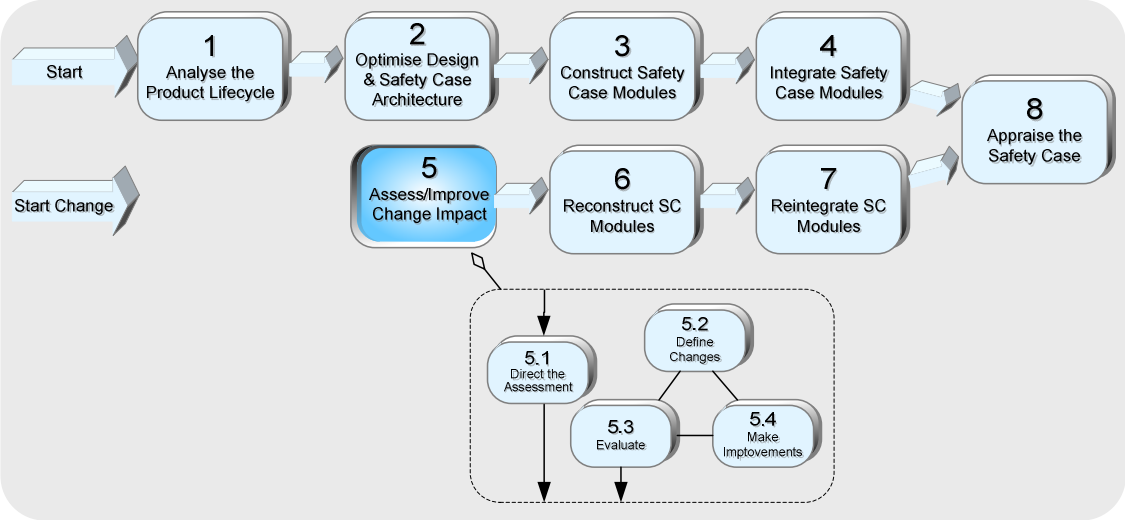
Expert judgment is required to ensure that like-for-like elements of Context are compared between the two goals and are compatible. If two items of Context are defined in such a way that they cannot be directly compared, their definition may have to be revisited.

If elements of Context exist on only one side of the link then some further analysis may be needed to ensure compatibility.

It should be noted, particularly when migrating from one application domain to another, that Context declared for a SC Module might be limited in scope by the supplier's view of the expected usage.

For discussion of the types of *Context,* its capture, and the assurance of *completeness* of the captured Context see section 6.1.5.

8 Step 5: Assess/Improve Change Impact



Objectives	
1. To identify what changes are necessary to the Safety Case. 2. To make beneficial modifications to the Safety Case Architecture. 3. To identify beneficial modifications to the system or system lifecycle plan etc.	
Inputs	**Outputs**
1. Previously certified modular Safety Case. 2. Defined system changes. 3. The lifecycle plan. 4. [optional] Report(s) on step 2, the original optimisation of the Safety Case Architecture.	1. List of necessary changes to Safety Case elements, including any architectural changes. 2. (optionally) improved system or system lifecycle plan or other aspect.

Figure 8-1: Assess/Improve Change Impact

MSSC Impact analysis is a process by which the changes to a Safety Case are identified during an incremental change.

In parallel, it may also emerge that modifications to the organisation of the Safety Case by the Safety Case Architecture could improve change containment for this and/or future iterations. The design team, users or customers may participate in this step and it may lead them to improvements to the system, the system lifecycle plan, the test plan or other elements from the development lifecycle.

The process for impact assessment is split into 4 major sub-steps which are shown in Figure 8-2. A variant of Figure 8-2 is also used to assist navigation through this section 6.1.4

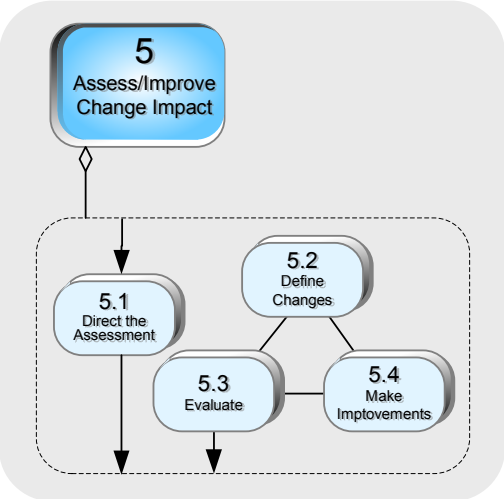


Figure 8-2: Impact Assessment Process Overview

Step 5.1 Direct the Assessment
Ensures that effort is optimised and significant concerns addressed early (see 8.1).

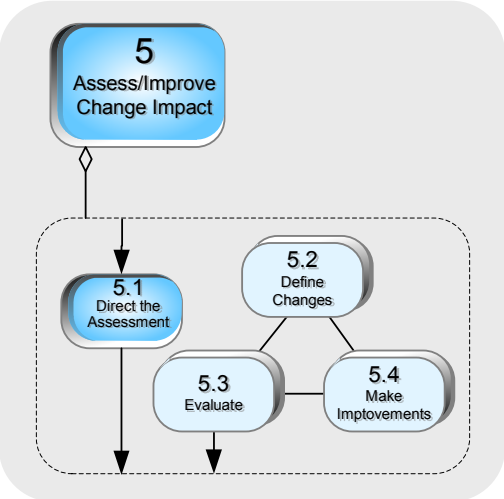
Step 5.2 Define Changes
Identifies the driving changes to be evaluated (see 8.2).
Determines how the Safety Case must be updated and provides evidence that assures the determination is correct (see 8.2).

Step 5.3 Evaluate Impacts
Identifies improvements to the Safety Case Architecture to optimise the containment of the impact of current and planned changes by Safety Case Module boundaries (see 8.3).

Step 5.4 Make Improvements
Makes the improvements proposed in step 5.3 (see 8.4) without undermining the change containment of the Safety Case Modules.

The example in Appendix A shows how change to a system (Example Figures: Figure 13-10 EspressoMat Change Impact Analysis to Figure 13-12 EspressoMat Mk II Supporting Change Argument) can result in incremental change to a Safety Case. Example Figures Figure 13-5 EspressoMat Mk I Safety Case Architecture and Figure 13-11 EspressoMat Mk II Safety Case Architecture shows the Safety Case before and after modification.

8.1 Step 5.1: Direct the Assessment



Objectives	
1. To prioritise improvements to the Safety Case Architecture, the design or to other aspects (evidence, process). 2. To direct the extent of the impact assessment to minimise re-work when improvements to the SC Architecture are made (under step 5.4).	
Inputs	**Outputs**
1. Previously certified modular Safety Case. 2. Defined System Changes 3. The lifecycle plan. 4. [optional] Report(s) on step 2, the original optimisation of the Safety Case Architecture. 5. List of changes to Safety Case elements [as updated on return from 5.2] . 6. [on return from 5.3] List of improvements to Safety Case Architecture or other artefacts.	1. The next step, either: • step 5.2: Define all changes to the Safety Case • step 5.3: Evaluate the impact to see if it may be improved. • step 5.4: Make an improvement to the Safety Case Architecture, system, change or other aspect. 2. The aim or target of next step

Figure 8-3: Direct the Assessment

The sub-steps 5.2, 5.3 and 5.4 may be executed in different sequences and to different extents or levels of refinement each time. In this sub-step the director determines which sub-step it is optimal to undertake next and the level of rigour, detail or scope that should be covered and sets this as the aim or target of the next step.

This step may identify priority or simple additional changes to the Safety Case Architecture or Design that might further reduce the impact of the changes to the system. Changes to the Safety Case Architecture impact the organisation of the Safety Case. They do not directly affect safety, although they may allow it to be demonstrated more efficiently.

To make those additional changes it directs the analyser to step 5.4, which performs the change and records it to support the change argument as necessary.

The impact assessment should close with a review of the recorded changes/impacts on the Safety Case. It should be checked for consistency and correctness at an appropriate level of rigour, particularly if a GSN mark-up representation is not used.

Some of the changes considered may not be to Safety Case artefacts, for example they may be to the system design or test strategy. Some of the effort vs. benefit trade-offs may not balance until many increments later in the product lifecycle so the consequence of any improvement on the product evolution should be considered.

The following inputs may assist expert judgement in this part of the process:

1. The architecture of the existing Safety Case
2. The lifecycle plan.
3. Any report(s) or rationales on the original step 2, the optimisation of the Safety Case Architecture, design and other influential aspects.
4. The modified outputs from the previous execution of step 2 to determine how to improve the Safety Case Architecture.
5. The results of revisiting step 2 from scratch as if it had not been done before. A well-chosen modularisation of the Safety Case Architecture should normally make it possible to follow point 4 (above), but it could be the case that a change situation arises which makes a complete reworking of the SCA the most effective way forward.

The unfinished results of the impact analysis in step 5.2 (when available) may also provide guidance. For example if the impact is going to be too large, it is these results that will show it:

6. [on return from 5.2] The list of changes to Safety Case elements.
7. [on return from 5.3] List of improvements to Safety Case Architecture or other artefacts.

The expert analyst is not constrained by these inputs, and may consider other potential improvements; further guidance is given in section 8.1.3.

Step 5.1 allows expert judgement to be used to set aims/targets for the next step (5.2, 5.3 or 5.4).

8.1.1 Decisions to Execute Step 5.2, Define Changes

The director of this task should ensure that more obvious improvements are identified and made early because the evaluation needs to be reworked when improvements are made. A preliminary execution of the impact definition in step 5.2 may be made at appropriately low levels of rigour. Only on the final run through will step 5.2 need to achieve the formality and produce the outputs necessary to support the change argument described in section 10 (process step 7).

For example, if on a preliminary, coarse-grained analysis under step 5.2, it is observed that when a new block is added then it is obvious that a corresponding Safety Case Module should be added to the Safety Case, then step 5.4 should be executed next to implement that.

Step 5.2 needs to be undertaken first and revisited if the system change is altered. If the system change is an improvement output from step 5.4, it will still need to be fed back in to step 5.2.

The director may dictate a preliminary pass through step 5.2 focused on specific requirements changes before the others because their impact may be more efficiently assessed.

It is often easier to trace an impact down from the requirements at the top of the Safety Case because of the direction of the references between Safety Case Modules in a typical Safety Case Architecture.

Improvements should be looked for during the execution of the impact definition in step 5.2, whenever there is difficulty containing the impact, as well as when the Safety Case Impact is evaluated.

8.1.2 Decisions to Execute Step 5.3, Evaluate Safety Case Impact

Expert analysts should take the opportunity in step 5.3 to think about whether there may be any other changes that could improve a re-evaluation of the Safety Case Architecture as described in section 5 (process step 2). They should trade-off the investment of effort against the benefits of the change.

The director may identify, as a priority, changes to the inputs to step 2 since it was executed, and direct them to be fed into step 5.3.

Improvements may be made to either a non Safety Case Artefact or to the Safety Case Architecture.

Separating any areas of higher assurance that may be introduced by any of the changes into their own blocks can potentially result in considerable saving of effort. If evaluation under step 5.3 determines this to be beneficial, it should be implemented under step 5.4 as soon as possible because further evaluation under step 5.3 could be invalidated by it.

When no beneficial changes are apparent the detailed definition of the changes (step 5.2) should be completed. Step 5.2 is always the last step undertaken.

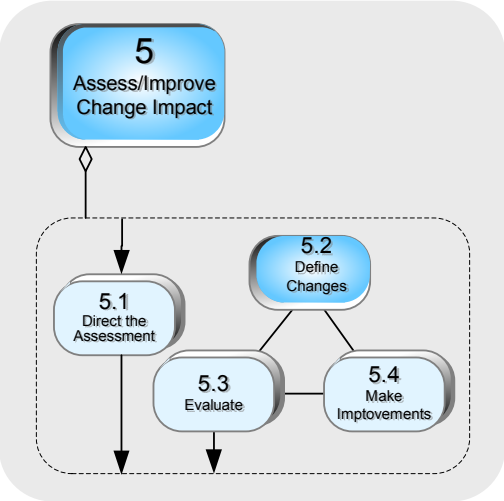
8.1.3 Decisions to Execute Step 5.4, Make Improvements

During the mutual optimisation of design and Safety Case Architectures (step 2 of the MSSC Process), it may have been decided that some Safety Case Module boundaries would be declared, but only be defined during a later increment. This step should consider deferred Safety Case Module boundary definition work. If the Safety Case Module boundary would assist with the containment of change during this increment the deferred work should be undertaken.

Improvements to the Safety Case may just make the argument for system safety simpler or more compelling. Improvements to the system may facilitate the safety argument, or they may even make the system safer.

The impact identification in step 5.2 **must** be performed again after a change is made to the Safety Case Architecture or any other relevant artefact by step 5.4. The final evaluation by step 5.2 needs to be at the required level of rigour to support the change argument, which is a level of rigour consistent with that required of any of the changed parts of the Safety Case. For efficiency, prior executions of the impact analysis may be modified at the analyst's discretion.

8.2 Step 5.2: Define Changes



Objectives	
1. To identify a list of driving changes to the system or artefacts from its development lifecycle. 2. To assess the extent of non-architectural changes necessary to the Safety Case.	
Inputs	**Outputs**
1. System development and verification information needs to be available to the design team for their reference. 2. Previously certified modular Safety Case. 3. List of driving changes. 4. [optional] previously set aims or targets from step 5.1 5. [optional] and prior list of affected Safety Case elements.	1. (updated) List of Driving Changes which impact the Safety Case. 2. List of affected Safety Case elements. 3. Report on not arguably safe "supported by" links or Safety Case contracts between Safety Case Modules.

Figure 8-4: Define Changes

During this activity participants from the design team identify the changes that are to be made to the system and determine whether they are Driving Changes that impact on the Safety Case. (The safety team participate in clarifying the changes to the system and provide knowledge of MSSC Process.)

The resulting list of driving changes should include those that manifest through:

- Standards
- Requirements
- Design
- other artefacts from the development and verification lifecycle.

The list should reflect all aspects of the system that are involved in change.

The impact of changes on the Safety Case in particular will then be evaluated. Engineers applying this step consider each identified driving change and navigate to related elements within the safety argument to determine all further impacts.

When one change is made it may necessitate a subsequent change in a related artefact. This is referred to as change propagation. The correctness of the impact evaluation within the Safety Case depends upon the Safety Case being sufficiently complete. A critical aspect of this is the capture of context for public claims and away goals that has been performed during Safety Case construction.

Change propagation is illustrated in Figure 8-5. In this illustration each alteration to a development lifecycle artefact and each consequent change to a Safety Case relevant artefact is given a unique number that may be used to identify the specific details of the change. (The numbers don't have any significance other than being unique for each change.)

The driving changes identified are numbered 1, 2, 3, 4, 5 and 16.

Changes made that affect one Safety Case Module are shown propagating out to a Safety Case Contract Module where they are met at the Safety Case Module Boundary/Public Interface (numbered 14) by equivalent corresponding changes (numbered 9) that have been made to the requirements Safety Case Module. It is here the change impact finishes.

Any architectural changes to the SC are made under step 5.4 as they are a means to reduce the impact of the other changes on the Safety Case.

Changes to claims or other safety argument elements in a Safety Case Module must not undermine the modularity of the SCA (upon which change containment depends). A changed or additional claim, for example, should not duplicate all or part of a claim already made in a different Safety Case Module. Evidentiary references within one Safety Case Module should likewise remain distinct from those made in other Safety Case Modules.

If the claim or evidence is sufficiently cohesive with the scope of the Safety Case Module defined by the SCA, an expert can be confident that the ability of the Safety Case Module to contain change is unaffected. Less commonly some analysis/search of possible areas of overlap may be required to support expert judgment.

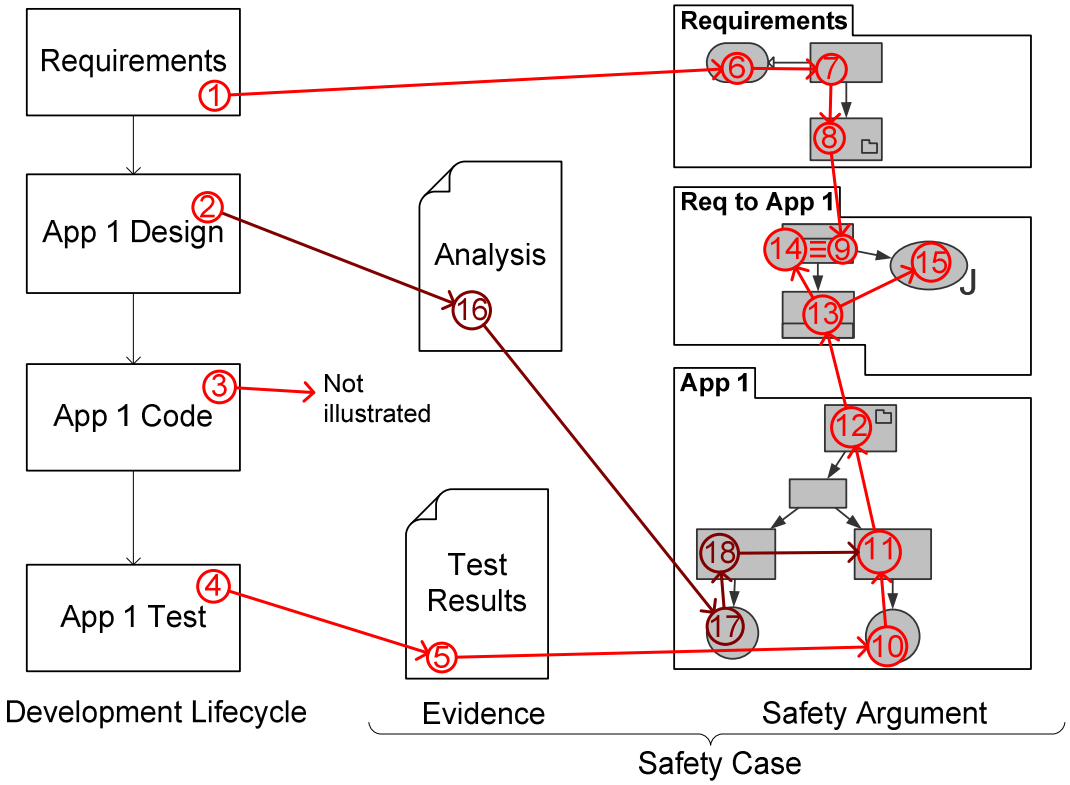


Figure 8-5: Illustrating the Safety Case Impact Analysis

There are a variety of analyses that may be used as evidence by the safety argument, and each one may be produced according to a specific process (in Figure 8-5 change 16 is to such an analysis of the design of App 1). The impact analyst should identify the inputs for all safety analysis processes and consider whether they have changed to determine if the output of the analysis is impacted.

In particular the safety analyses used by any System Wide Issues Safety Case Module need to be considered.

Step 5.2 is further broken down in steps 5.2.1 to 5.2.4. See Figure 8-6 for a summary diagram. The impact evaluation process clearly identifies all the parts of the Safety Case that have been altered, both the direct impacts on each Safety Case Module and any consequent effects of them.

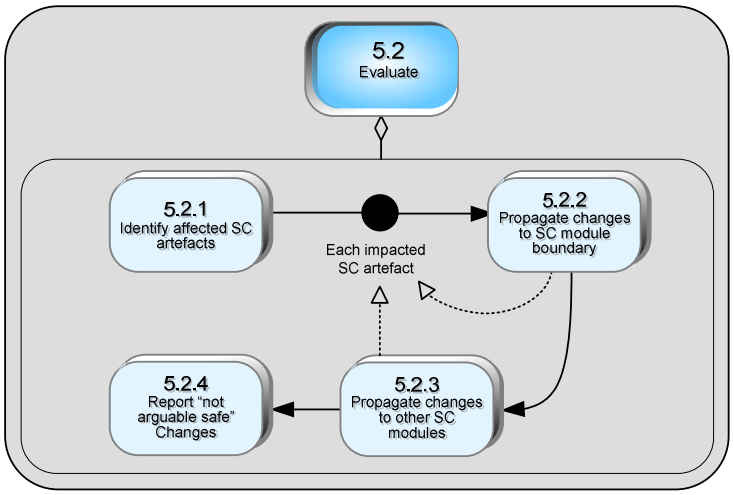


Figure 8-6: Process breakdown of "Evaluate" During Impact Analysis

In Figure 8-6 the dotted lines indicate the possibility of changes being added to an overall list of everything that has changed. The consequent impact of all these changes will also need to be assessed.

An example of the execution of impact analysis is given in Appendix A: section 13.6.

8.2.1 Step 5.2.1: Identify Affected Safety Case Elements

A relationship has been established between the design and Safety Case architecture. This is now used, given a Driving Change to the system or any development lifecycle artefact such as the design, to identify which Safety Case Modules to search for its initial Safety Case impact.

In the example in Figure 8-5 this results in the identification of changes numbered 2, 9 and 14.

8.2.2 Step 5.2.2: Propagate Changes to the Safety Case Module Boundary

The impact of the changes identified in step 5.2.1 is propagated through the argument so that all consequent changes to the boundary or Public Interface of the Safety Case Module are identified.

A change to a claim or the context associated with it may have a consequent effect on any claims it supports, is supported by, or any 'sibling' claims that contribute support to the same 'parent' claims.

The analysis in this step ends when:

1. All consequent changes within the Safety Case Module have been identified AND
2. All changes at the Safety Case Module Boundary (Public Interface) have been passed to step 5.2.3 AND
3. Any impacts that identify inadequacies in artefacts from the development and verification lifecycle of the system have been passed to step 5.2.4

In the example in Figure 8-5 this results in the identification of changes numbered 3,10, 11 and 15.

8.2.3 Step 5.2.3: Propagate Changes to other Safety Case Modules

In this step the changes that impact the Safety Case Module boundary are collated, and their use by other Safety Case Modules is identified. The Safety Case Architecture will indicate which Safety Case Modules are potential users of any Public Goals.

If the impact analysis has already identified a corresponding change for all users of a changed Safety Case Module Public Interface then there is no need to propagate the change further.

If there are some references to a changed Safety Case Module Public Interface that are not already changed as the result of a driving system change this indicates an impact on a Safety Case Module that the design team may not yet have identified. This should be dealt with under Step 5.3.

In the example in Figure 8-5 this results in the collation of changes 9 and 14, and the identification of the Safety Case Contract module named “Req to App 1”.

The changed Safety Case Modules are visited by step 5.2.2 to propagate the change across them.

8.2.4 Step 5.2.4: Report “Not Arguable Safe” Changes

If this step is reached from step 5.2.3 it indicates that there is either an evidentiary or an actual safety issue with the proposed changes to the system.

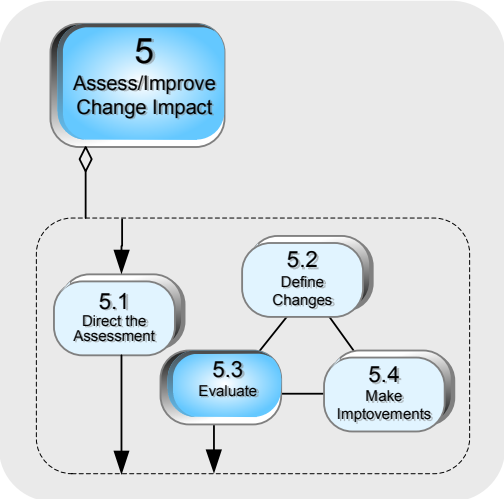
This step is necessary when the impact analysis chain requires a change to the evidence or context which is the responsibility of the development or verification team. For example, if the impact chain 2, 16, 17, 18 were not identified in Figure 8-5, then analysis of change number 11 would have identified change 18 and consequently change 17. change 17 being evidentiary, it would need to be reported outside of the SC impact analysis. The identification of changes 16 and 2 by those responsible for these artefacts should follow.

The other change that may be reported is one to a claim or any of its associated context at a Safety Case Module boundary. It is at this boundary that an impact chain coming from the other Safety Case Module is expected to meet it. If this does not happen something is wrong or missing from the list of Driving Changes. For example, in Figure 8-5 change 14 and Change 9 are equivalent, but identified by different impacts chains. At this point tracing the impact into the new Safety Case Module until it reaches evidence or context may be useful to clarify some details of the other missing impact chain.

A report should collate the details of the change, all impact on the evidence that were not driven by the development or verification changes, along with the expert rationale for why they are necessary. This rationale should at least in part describe the impact chain that has led to identification of the change.

A solution needs to be agreed with the responsible authority or team, and the impact analysis then needs to be updated with this additional modification in place.

8.3 Step 5.3: Evaluate Impact on Safety Case



Objectives	
1. To identify whether changing Safety Case Modules would provide benefit. 2. To identify beneficial safety relevant improvements to artefacts which are not the sole responsibility of the MSSC Process, e.g. to the design or requirements.	
Inputs	**Outputs**
1. Previously certified modular Safety Case. 2. List of affected Safety Case elements. 3. List of change scenarios. 4. Relevant Non MSSC artefacts. 5. [optional] previously set aims or targets from step 5.1.	1. List of proposed improvements to the Safety Case Architecture 2. [optional] Recommended or agreed Improvements to Non MSSC artefacts.

Figure 8-7: Evaluate

Having evaluated the change impact for the current Safety Case Architecture in step 5.2, in this step the material produced so far by the impact analysis is examined and expert judgement is used to determine if there may be any beneficial refinements of:

- Any non MSSC artefacts including the system design;
  The MSSC Process does not describe how to modify non MSSC artefacts such as the system design or the test plan. The Safety Case team need to consult with those responsible for a non MSSC artefact about its improvement. Improvements made should be recorded as the change argument may need to claim that the safety argument has not been undermined by them.
- The Safety Case Architecture (including any Safety Case Module Public Interfaces);
  Changes to the SCA include the removal or addition of a Safety Case Module. There may also be a need to rearrange Safety Case Module boundaries and to alter the Dependencies or Guarantees associated with one or more Safety Case Modules about a Block.

A change is beneficial if it improves the results of the evaluation of the Safety Case Architecture as described in step 2.3 or the impact of changes for this iteration from step 5.2. Step 2.3 may need to be revisited with all inputs up-to-date and considering the specific change scenario in progress.

Reasons for rearranging Safety Case Module Boundaries

` Possible reasons for rearranging Safety Case Module Boundaries may include:

- Component change
- Higher or Lower integrity introduced
- New supplier introduced for part of a Block previously developed by a single supplier
- Completion of boundaries within the architecture that was deferred
- Improved change containment for a change scenario previously unanticipated

Architectural refinements may alter the location of the boundaries between Safety Case Modules. In this sense "location" refers to the mapping of Safety Case Modules to the system elements which are the subject of their argument.

Architectural changes should be recorded on the list of Safety Case changes along with those changes from step 5.3. The change argument will subsequently make claims about these listed changes so that confidence can be established that they do not undermine system safety requirements.

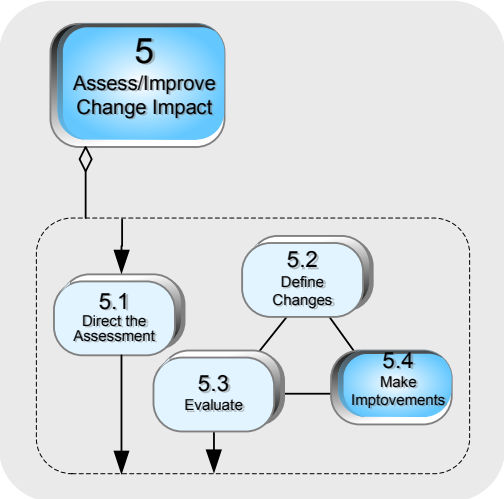
The factors that influence the Safety Case Architecture are described in the process for modular Safety Cases, Step 2: Optimise Design and Safety Case Architectures in section 5. The same influences apply in the context of an incremental change (modularity of the system, modularity of evidence and modularity due to application of different processes, etc). What is different is that:

a) the influences are applied as a result of changes only, not for a new Safety Case
b) the changes made on account of this influence must be captured to support the change argument.

The analyst should:

- Review and update the change scenarios (see section 4.1) for the system.
- Propose candidate architectural refinements and assess these against the change scenarios.
- Select and agree which architectural refinements will bring sufficient benefits.
- Ensure that for each desired architectural change the necessary changes to arguments and to the system to maintain non-interference claims about Safety Case Module boundaries have been made.
  The non-interference argument is the basis for containment of change. It is essential that the argument for non-interference of behaviour on both sides of a Safety Case boundary be strong enough to support containment of change. In practice if there is a basis for this argument that is common to many Safety Case Module boundaries, such as physical separation, that may form the basis of the non-interference argument for the new boundary.

8.4 Step 5.4: Make Improvements



Objectives	
1. Modify the Safety Case Architecture to add, remove, modify, combine, split, grow or shrink Safety Case Modules as directed. 2. To make beneficial safety relevant improvements to artefacts which are not the sole responsibility of the MSSC Process, e.g. to the design or requirements.	
Inputs	**Outputs**
1. List of proposed changes to the Safety Case. 2. List of proposed improvements to the Safety Case Architecture 3. [optional] Recommended or agreed Improvements to Non MSSC artefacts. List of change scenarios. 4. Relevant Non MSSC artefacts. 5. [optional] previously set aims or targets from step 5.1.	1. List of improvements to the Safety Case Architecture 2. List of all proposed changes to SC updated with and for consistency with the SCA improvements. 3. [optional] Updated list of change scenarios.

Figure 8-8: Make Improvements

The SCA provides the basis for containing the impact of changes to the Safety Case. Although change containment may be improved by modifying the SCA, it may also be undermined. For this reason any changes to the Safety Case Architecture need to be made with demonstrable due care and attention. The Change Argument constructed in step 7 will be responsible for assuring readers of the updated Safety Case of this.

The rest of Step 5.4 exemplifies a relatively formal approach to this, one that should be tailored to an appropriate level of rigour for the Safety Case being changed.

Some types of SCA change that it is possible to undertake safely are identified as well defined operations. These are “atomic” in that they can be combined together to achieve more significant changes.

Changes should be made to the Safety Case Architecture and generate any new Safety Case artefacts in accordance with modular Safety Case process in the sub sections 8.4.1 and 8.4.2 below. Confidence in the results of the Impact Analysis can be maintained by conducting changes as a sequence of well-defined atomic operations. These operations are:

1. The deletion of an existing Safety Case Module
2. The addition of a new Safety Case Module
3. Splitting an existing Safety Case Module into two or more Safety Case Modules (which introduces new Safety Case Module boundaries without altering the existing ones). This is described further in section 8.4.1.

4. Combining/Joining two Safety Case Modules into one (which archives or conceals a previously developed Safety Case boundary). This is described further in section 8.4.2.

Thus, moving a Safety Case Module boundary can be understood as a sequence of splitting a Safety Case Module and then combining one of the resultant Safety Case Modules with another.

If changes have been made to the Safety Case Architecture the "list of proposed changes" output from the impact analysis and subsequently used by the change argument pattern as evidence, must be updated.

- Firstly the architectural changes should be identified at the top of the list, each classed as one of the five types of change listed above.
- Secondly the non-architectural elements impacted may now be different or be in different Safety Case Modules, so step 5.2 needs to be revisited.

8.4.1 Combining/Joining Two Blocks

Motives for joining two Block related Safety Case Module

Even though a Safety Case Module boundary may have been identified as no longer needed to support the planned lifecycle, consideration should be given to the following points before deciding to remove it:

- Removal of the Safety Case Module boundary will take additional effort;
- Whether leaving the Safety Case Module boundary in place requires any additional effort, such as maintenance of associated DGRs or evidence;
- The presence of the Safety Case Module boundary can only improve the ability of the SC to contain change;
- Effort may have been invested in the original definition of the SC boundary, and the benefits of it will be lost if it is removed.

To join two block related Safety Case Modules, the interface between them is removed from both Safety Case Modules, and the other parts of their SC boundaries are combined.

Care should be taken not to remove a dependence on a third party to support the interface between Safety Case Modules, e.g. on a communication service. That will still be needed both in the implementation and must be claimed by the new Safety Case Module.

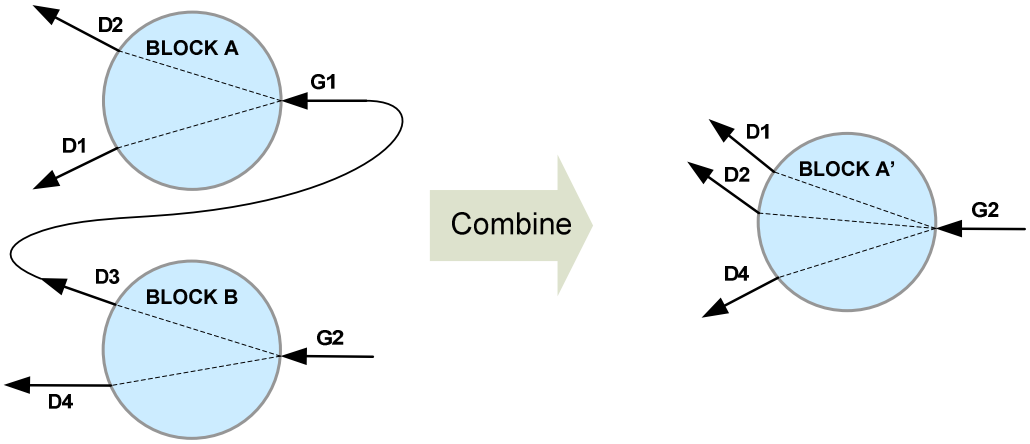


Figure 8-9 Combining Blocks and Combining a DGR

8.4.2 Splitting a Block

Motives for splitting a Block related Safety Case Module

` One motive for splitting a block related Safety Case Module is to provide stronger assurance that changes that impact one DGR do not impact any of the others. Strengthening this non-interference argument can strengthen the change argument if it was previously its weakest aspect.

Another example of splitting a Safety Case Module is, if part of a block previously argued to be safe by a single Safety Case Module has been made more safety relevant, it may be possible to split it off into its own Safety Case Module so that it can be argued to be safe at a higher level of integrity.

A more obvious example is that a new physical module has been added, in which case it will need to be suitably argued about in a (new or existing) Safety Case Module.

If the Safety Case Module is a Block related Safety Case Module based on Dependencies and Guarantees or on DGRs, then there are two ways it can be split.

Firstly there can simply be two groups of Dependencies and Guarantees or DGRs, as depicted in Figure 8-10. The isolation of the functionality associated with each new group must be established, and the system may need to be changed to achieve this. For example, a software application might be split into two applications, and they might be run in different partitions policed by an operating system. If the argument for isolation of the functionality is not as strong as that between other Safety Case Modules the Safety Case may be undermined.

Secondly the block related Safety Case Module can be split into one responsible for its guarantees and its higher-level functionality, and one responsible for requiring its dependencies and its lower level functionality. This requires that an interface between them is established in which a new set of guarantees are provisioned to support a new set of dependencies. Again, the system should allow the two new product modules to be isolated from each other to the same extent as for product corresponding to existing Safety Case Modules.

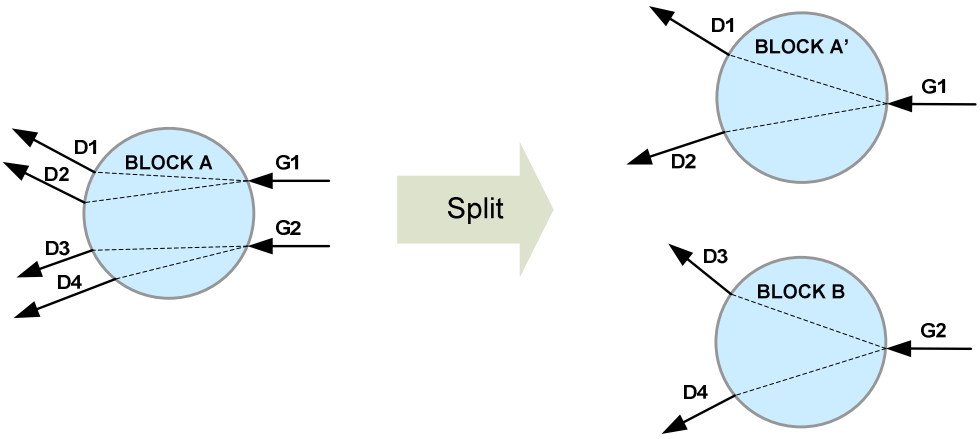
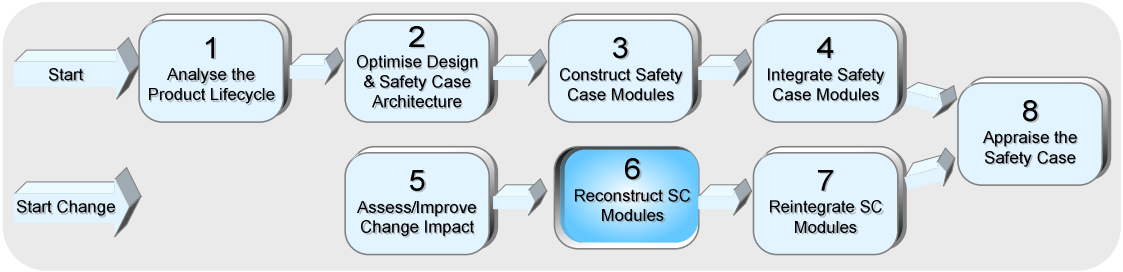


Figure 8-10 Example Splitting a Block and Reallocating DGRs Intact

Evidentiary support should also be split between the two Safety Case Modules.

9 Step 6: Reconstruct Safety Case Modules



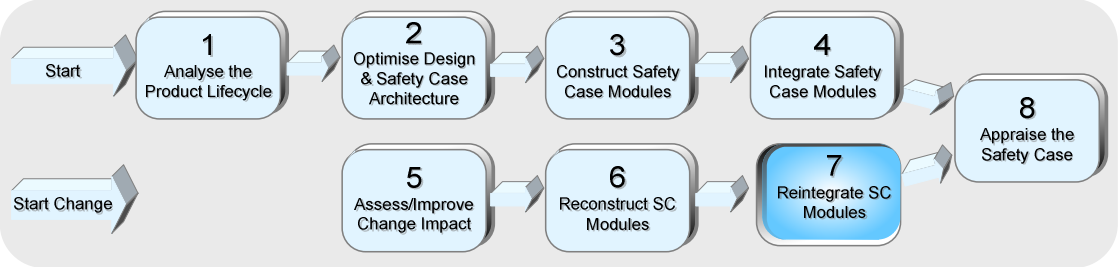
Objectives	
1. To produce the changed Safety Case	
Inputs	**Outputs**
1. Integrated Safety Case Modules from a prior iteration 2. List of proposed Safety Case artefact changes from impact analysis	1. Changed Safety Case 2. Supporting Change Argument for this increment

Figure 9-1 Reconstruct Safety Case Modules

Reconstruction of Safety Case Modules is a process of adapting existing Block Safety Case Modules, or creating new Block Safety Case Modules, to take account of changes made to the associated blocks.

The reconstruction is effectively a re-run of the construction step (see section 6- step 3) but with the scope of the work limited to dealing with the Safety Case Modules affected by changes (determined by section 8 - step 5) rather than all Safety Case Modules.

10 Step 7: Reintegrate Safety Case Modules



Objectives	
1. To bring together reconstructed Safety Case Modules to reintegrate an existing integrated Safety Case; the result should provide an acceptable argument, which, when supported by a body of evidence, demonstrates that the system satisfies its safety requirements.	
Inputs	**Outputs**
1. Allocated safety requirements 2. Safety Case Architecture 3. Specification of the operating context. 4. Applicable safety standards 5. Populated/Reconstructed Safety Case Modules 6. Integrated Safety Case from a prior iteration	1. Integrated Modular Safety Case

Figure 10-1 Reintegrate Safety Case Modules

This step will complete the bringing together of the new, incremented, Safety Case and provides a supporting Change Argument that justifies the limited re-evaluation of the new Safety Case.

10.1 Reintegration

Reintegration of Safety Case Modules is a process of adapting the existing Safety Case Module integration to take account of changes in the interfaces among those Safety Case Modules which have been changed during the Safety Case Module reconstruction step (section 9).

The reintegration is effectively a re-run of the integration step (see section 7 Step 4: Integrate Safety Case Modules) but with the scope of the work limited to dealing with incremental changes to the integration resulting from Safety Case Module changes, rather than the previous integration as a whole.

10.2 Change Argument

The Change Argument claims that the Safety Case argument has been adequately updated to reflect the new system. This is a supporting claim about the new Safety Case argument, not about the safety of the system to provide any reviewer with an understanding of why an incremental update to the Safety Case was considered appropriate. It is an argument about argument; see Figure 10-2 which shows how the Change Argument might feature in the overall certification context. The original System Safety Case and original System Certificate are shown as they set the starting context for the increment; however, they are not subject to re-assessment. The new System Safety Case provides the necessary supporting case for the new certificate; if it were re-assessed on its own, in full, it must provide an adequate case

for the new system to be certified. The Change Argument provides the supporting case for the new System Safety Case having not been re-assessed in full.

The Change Argument will be seen by the Safety Case reviewers, and it is crucial that it be a well presented and adequate argument. As with most arguments there are varying levels of rigour that may be applied to it, and they should be commensurate with the applicable parts of the Safety Case. If they are not, lack of confidence in the application of changes may undermine the Safety Case reviewers' confidence in the final Safety Case.

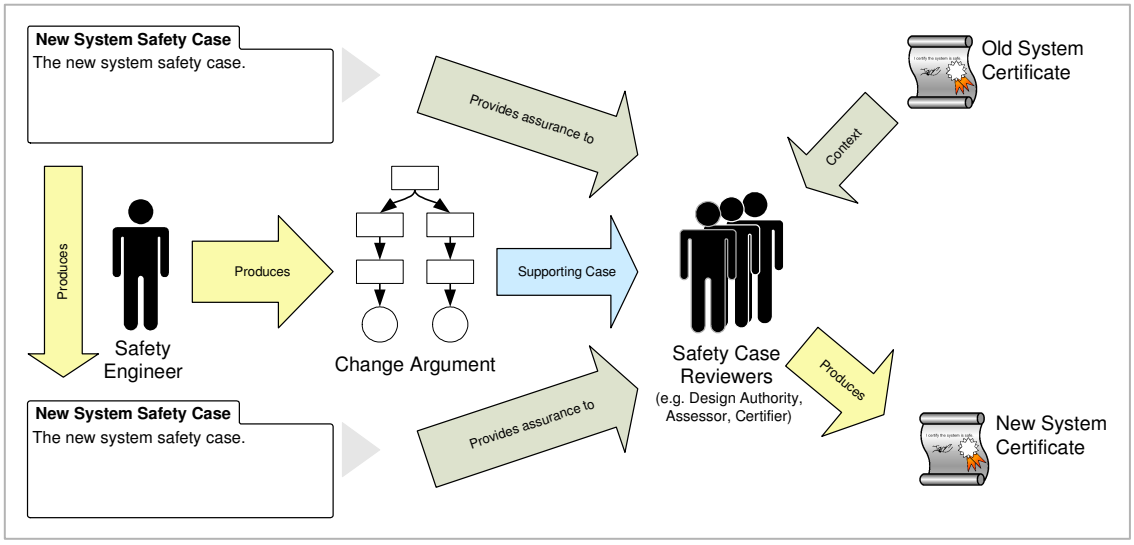


Figure 10-2: Contribution of MSSC artefacts to Increment Certification

The detail of the Change Argument is likely to be determined by the extent and nature of the change made, and the degree of assurance expected; however it should address two principal matters:

- The reasoning supporting the case for not revisiting those Safety Case Modules not re-assessed in the new Safety Case – this is likely to depend upon claims that the Impact Analysis (see section 8 Step 5: Assess/Improve Change Impact) is complete, and that there is adequate Configuration Control to ensure that changes to the system are managed and identifiable;
- The reasoning for believing that the changes introduced to the system will not introduce unexpected errors, or lead to omissions in the new Safety Case.

This argument pattern is illustrated in Figure 10-3.

In forming the Change Argument the developer must have a reason for believing the two principal claims and is expected to set out these reasons in a structured way to provide a compelling case for others to believe the same. Where appropriate the developer should consider what evidence there is to support claims within the argument.

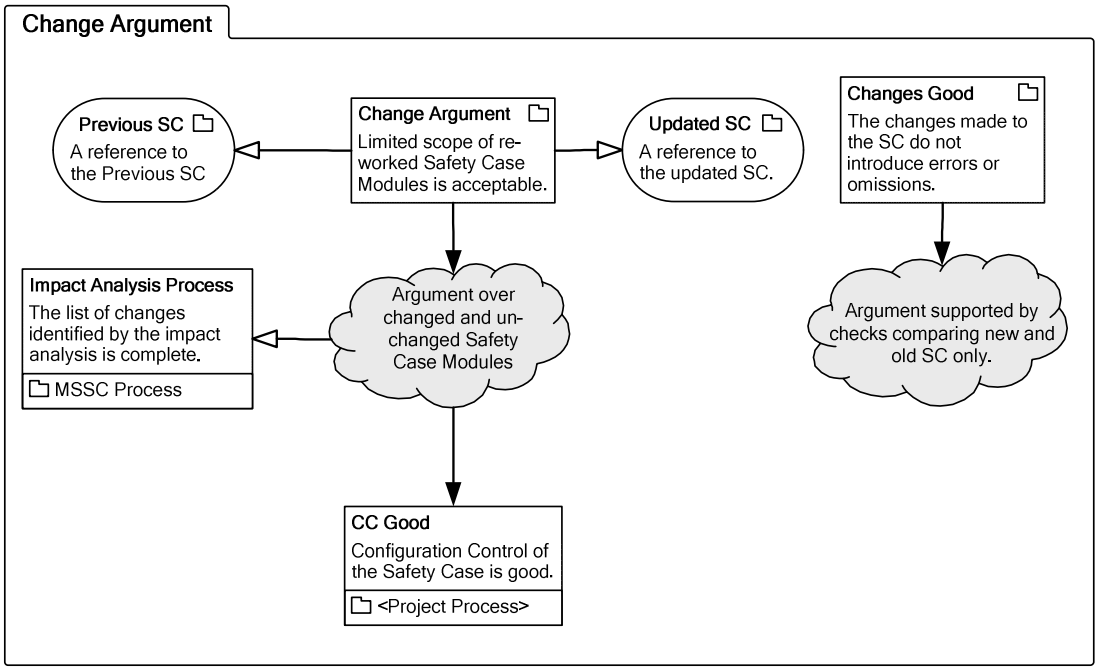
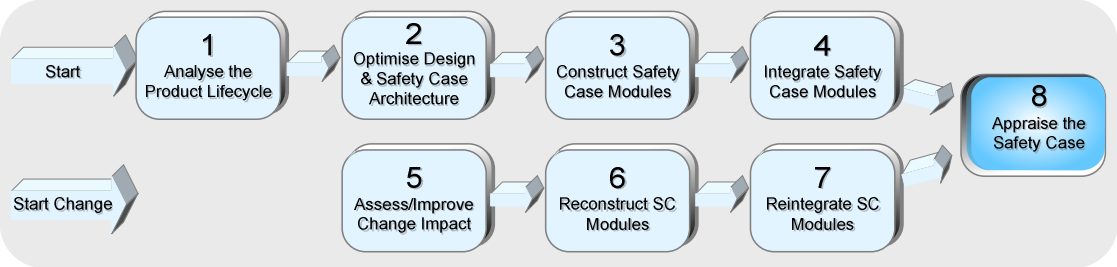


Figure 10-3 Change Argument Pattern

11 Step 8: Appraise the Safety Case



Objectives	
1. Gateway decision on adequacy of Modular Safety Case. 2. Ensure the Safety Case Architecture remains aligned with the product lifecycle projections. 3. Improve the efficacy of preparing the Modular Safety Case.	
Inputs	**Outputs**
1. The Modular Safety Case. 2. Product Lifecycle Plan from section 4 Step 1: Analyse the Product Lifecycle	1. Report on the adequacy of the Modular Safety Case. 2. Updated product Lifecycle Plan. 3. Report on the efficacy of the Modular Safety Case and the process of producing it. 4. Recommendations for process improvement.

Figure 11-1 : Appraise the Safety Case

The appraisal of the Safety Case consists of two threads to consider how well the Modular Safety Case:

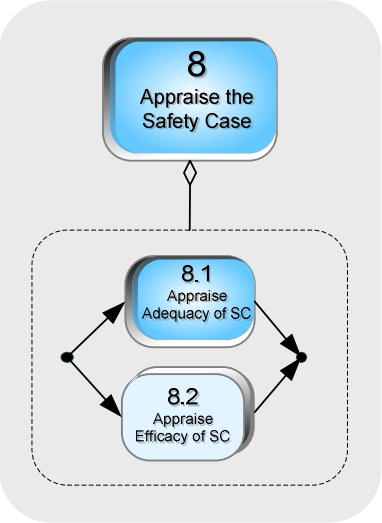
- Works as a Safety Case; and
- Remains aligned with the programme objectives which led to a modular approach being adopted in the first place.

These threads should be conducted and reported independently as their outcomes have different significances and may lead to different responses from programme management; but the overall appraisal of the Safety Case is not considered complete until both threads have been completed.

This appraisal step is intended to be internal to the product developer, it is not the intention of this process step to define what an ISA ought to do as part of their audit – this should be defined as part of the ISA's plans.

The level of detail and rigour of the appraisal will be programme specific, being determined by the criticality of the Safety Case and the likely adverse programme effects if the Modular Safety Case is not precisely aligned with the current understanding of the future programme needs. The level of detail and rigour might also be determined by the size and scope of change when an incremental update to the Modular Safety Case is being appraised.

11.1 Step 8.1: Appraise Adequacy of Safety Case



Objectives	
1. Gateway decision on adequacy of Modular Safety Case.	
Inputs	**Outputs**
1. The Modular Safety Case.	1. Report on the adequacy of the Modular Safety Case.

Figure 11-2 : Appraise Adequacy of Safety Case

Appraising the adequacy of the Modular Safety Case provides the developer with the opportunity to step back and consider whether the Modular Safety Case they have produced is adequate as a Safety Case. The appraisal should typically involve Independent Technical Authorities who have special responsibilities for safety or who will be ultimately signing off the system.

> *Note: The independence being recommended here only needs to provide a sufficient degree of separation from the development team that the developers can gain confidence that the Safety Case is credible to a third party. This step does not represent the formal involvement of an ISA.*

The appraisal is aiming to respond to the question:

> "*Is this a structured argument, supported by a body of evidence, that provides a compelling, comprehensible and valid case that the system is safe for its stated application and intended environments?*"

The Safety Case generated as a result of the previous steps of MSSC should have met the objectives of providing a 'compelling, comprehensive and valid structured argument. It is not the role of the generic MSSC process to provide criteria for evaluating whether all aspects of a specific Safety Case are sufficiently compelling, however, the following provides suggestions for the appraisal process, focussing on specific facets of the modular presentation of the Safety Case.

This appraisal depends upon expert evaluation as it requires judgements to be made about the completeness of the Safety Case, the relevant contexts of different parts of the Safety Case, the strength of reasoning in the arguments used, and the relevance of the supporting evidence.

The appraisal should be conducted (or overseen) by an Independent Technical Authority who has:

- Sufficient product domain knowledge that they are aware of the general hazards associated with such systems and so can reasonably judge the completeness and relevance of the Safety Case both with respect to technical matters and the normal expectations for certification of such products;
- Sufficient familiarity with Safety Cases that they can reasonably judge the strength of reasoning and coverage of the arguments used, the acceptability of different types of evidence in supporting such arguments, and the adequacy of the overall Safety Case in satisfying the levels of assurance being sought;
- Sufficient awareness of the concepts of Modular Safety Cases, and the MSSC Process, that they can make appropriate judgements about the modularisation and integration aspects – balancing the increased complexity of comprehension against the benefits being sought through the use of a Modular Safety Case.

Using their skills and knowledge the appraiser(s) should determine whether the Safety Case is credible and appropriate for the intended use of the system and would be robust against possible challenges. The appraisal should be conducted in a methodical manner and should consider:

- **Safety Case Modules** – Whether the provided and required claims, together with their supporting context (including DGRs), have been expressed clearly, precisely and in a manner appropriate to the level of assurance associated with the claim. Whether the arguments and evidence supporting the provided claims are compelling, comprehensible and valid (assuming that the required claims are satisfied);

- **Integration Safety Case Modules** – Whether the expression of the contract between Safety Case Modules, including the context compatibility, has been expressed clearly, precisely and in a manner appropriate to the level of assurance associated with the claims depending on the contract. Whether the arguments (and evidence – if applicable) presented in Integration Safety Case Modules and Safety Case Contract Modules are compelling, comprehensible and valid;

- **System-Wide Issues** – Whether the expression of claims, together with their context, for system-wide issues have been expressed clearly, precisely and in a manner appropriate to the level of assurance associated with the claim. Whether the arguments and evidence supporting the claims are compelling, comprehensible and valid;

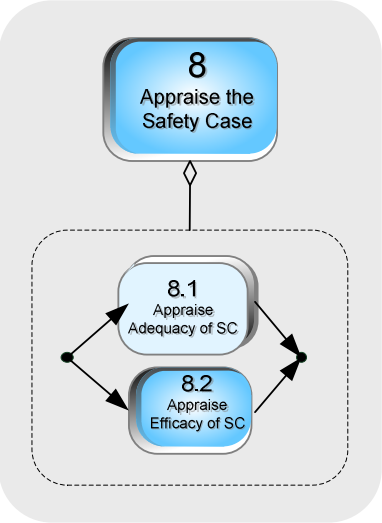
- **Integration Completeness** – Whether all the required claims of every Safety Case Module, needed to support the provided claims used in the Safety Case Argument, are satisfied via an Integration Safety Case Module;

- **Addressing of the System Level Safety Requirements** – Whether each Safety Requirement defined at the system level has a traceable link to a Safety Case claim that makes a compelling, comprehensible and valid argument that the requirement is satisfied.

Note: The list above presents a bottom-up view of the Modular Safety Case structures, but the appraisal could be undertaken top-down, middle-out or in any methodical order that suits the appraiser.

The findings of the appraisal, which should include a clear statement as to whether (in the view of the appraiser) the Safety Case meets the criteria necessary for an adequate Safety Case, should be recorded in a report on the Adequacy of the Modular Safety Case. This report should be used by programme management to make a ‘decision’ on proceeding with the next stage of development/system delivery, or instigating remedial action.

11.2 Step 8.2: Appraise Efficacy of Modular Safety Case



Objectives	
1. Ensure the Safety Case Architecture remains aligned with the product lifecycle projections. 2. Improve the efficacy of preparing the Modular Safety Case.	
Inputs	**Outputs**
1. The Modular Safety Case. 2. Product Lifecycle Plan from Step 1: Analyse the Product Lifecycle.	1. Updated product Lifecycle Plan. 2. Report on the efficacy of the Modular Safety Case and the process of producing it. 3. Recommendations for process improvement.

Figure 11-3 : Appraise Efficacy of Safety Case

Appraising the efficacy of the Modular Safety Case provides the developer with the opportunity to review whether the Modular Safety Case remains aligned with the anticipated product lifecycle and whether, as a result of experience gained in preparing the current Modular Safety Case, there are opportunities to improve the processes used in subsequent development. Appraising the efficacy of the Safety Case should address the following:

- **Review the current Product Lifecycle Plan** – The existing product Lifecycle Plan should be reviewed against current knowledge about the product and its anticipated future development. Any modifications to the change scenarios should be incorporated into an update of the product Lifecycle Plan. The review should use the methods described in section Product Lifecycle Plan from section 4 Step 1: Analyse the Product Lifecycle to identify change scenarios and assess their likelihood.

  As a review activity, the extent to which the potential change scenarios are investigated might be guided by a risk assessment based on knowledge of events and product changes that have occurred since the last review.

- **Evaluate Safety Case Architecture against anticipated Change Scenarios** – The Safety Case Architecture should be re-evaluated against the change scenarios identified in the product Lifecycle Plan using the methods described in section 5.3 Step 2.3: Evaluate. This evaluation should be compared to previous evaluations to determine if a divergence is arising between the Safety Case Architecture and the expectations of the product Lifecycle Plan.
- **Assess Safety Case for Emerging or Unexpected Complexity** – The Safety Case should be assessed to identify cases where it appears to be becoming more complex than was initially anticipated. This might be indicated by ad-hoc or unplanned argument claims, unplanned activities to provide additional supporting evidence, or unexpected remedial actions arising from review or audit activities.
- **Assess Change Containment Effectiveness** – Where a change has been incorporated into the product and an Incremental Safety Case has been produced, the impact analysis for the increment should be reviewed against the original anticipations of the product Lifecycle Plan for the relevant change scenario. This review should assess how well the modularisation met its containment objectives and identify any trending in the incremental change argument that would indicate that the Safety Case argument is becoming too complex, less robust, or less cost-effective.
- **Report Efficacy Findings** – The results of the evaluations, and comparison with earlier evaluations, should be recorded in a report on the efficacy of the Modular Safety Case. This report should be used by project management to consider:
    - A revision of the Safety Case Architecture to better match the product Lifecycle Plan and the emerging;
    - A revision of development practices to better address the emerging Safety Case issues;
    - A revision of the Design to better match the product Lifecycle Plan;
    - Updating the project risks register to record the potential adverse effects that would arise if the change scenario(s) in the product Lifecycle Plan (for which the Modular Safety Case now seems less suited to) materialise.

The appraisal may also identify other issues pertinent to the efficient development and maintenance of the Safety Case and so might provide useful input into process management – see section 12.

12 MSSC Managerial Processes

General management of the development of a modular Safety Case using the MSSC Process should follow the normal practices of an organisation. For any Safety Case it is advisable to engage with all interested parties (e.g. Independent Assessors, Certification Authorities) at the earliest opportunity in order to maximise the likelihood of a successful outcome. A modular Safety Case might be unfamiliar to some, so early engagement of all interested parties is highly recommended when using the MSSC process.

The preceding sections have laid out all of the steps in the MSSC Process, and concluded with an Appraisal step. That step is a good example of a suitable point for managerial and technical teams to come together to reflect on the application of the process.

12.1 Planning, Tracking and Measuring the Process

There are several aspects of the MSSC Process that provide opportunities to aid management activities such as tracking progress, estimation of task size and measuring productivity.

The process steps provide a natural structure for a project plan. Plans should take account of the fact that sub-steps and some top level steps may need to be applied iteratively.

The effort required to complete a process step will depend on the complexity of the software, the number of safety related requirements, and hence the complexity of the software Safety Case.

Complexity of software may be measured by conventional means. Software Safety Case complexity may be approximated by sizing the work products:

- The number of Safety Case Modules developed;
- Number of guarantees, dependencies and Safety Case Contracts that must be formed between them;

Alongside

- A count of safety related requirements;
- The number of change scenarios identified.

Such measures provide a means for evaluating whether the developed software Safety Case matches the profile of the programme as determined at the time of decision to adopt the MSSC Process.

A programme's managers will wish to measure performance through the life of the product:

- Comparing actual changes, sizes and rates against anticipated change scenario;
- Recording the number of MSSC work products affected by different classes of change as a measure of change containment;
- Recording effort required per work product affected.

A useful body of knowledge should result, which should inform future estimates and support improved decision making on safety module sizing, or decisions regarding the reworking of existing Safety Cases, perhaps to better realise the benefits of The MSSC Process for future iterations of a programme.

12.2 Continuous Process Improvement

Metrics and performance data (such as end-of-development-cycle ‘lessons learnt’ reports) should be reviewed to identify inefficiencies or problematic areas in the development of the Modular Safety Case.

Possible remedies for identified problems should be investigated, including:

- Use of new or alternative techniques for implementing particular steps in the Safety Case process;
- Automation of activities;
- Revision of development practices to provide better supporting data for the Safety Case;
- Revision of the Design or Safety Case Architecture to eliminate problem areas.

13 Appendices

Appendix A: Illustrative Safety Case - The EspressoMat Hot-Drink Vending Machine

The EspressoMat hot-drink vending machine is introduced to illustrate aspects of the MSSC Process. The EspressoMat uses software to control its functions; this software is implemented on top of a simple operating system (EspressOS) which supports various facilities including a partitioning system that allows software functions to be rigorously separated from each other (except for communications implemented using the EspressOS services). Safety for the EspressoMat is concerned with the risk of the user being scalded by hot water. This is addressed by a Guard Door on the dispensing area and an interlock that ensures the hot water is turned off if it is open.

As this is to be used as an example the software is somewhat over-engineered so that aspects of the MSSC Process can be illustrated – rather than be representative of a good design for a vending machine.

For the purposes of introducing modularity the functions of interfacing with the Guard Door sensor, providing the interlock logic, and interfacing with the hot water control valve are separated and treated as distinct modules in the design and Safety Case.

To extend the example an update to the EspressoMat (Mk II) is also introduced later on. This adds a Drip Tray and a risk associated with the machine dispensing hot water if the Drip tray is not present or is full. This leads to a new module to provide the interface to the Drip Tray sensor and a change to the interlock logic to account for the new conditions controlling the isolation of the hot water. This update is used to illustrate how the MSSC Process supports an Incremental Change to the Safety Case.

13.1 Lifecycle Plan

In designing the EspressoMat the potential changes it might face during its life were considered to determine the parameters for a Design Architecture/ Safety Case Architecture trade-off analysis. These are shown in Table 13-1 EspressoMat Change Scenarios.

Change	Likelihood	Details
Processor upgrade: faster processor.	Low, Infrequent. Assessed as low probability as utilisation of current processor will be very low, providing adequate scope for software growth.	Impact limited to change of clock speed and shorter execution times.
Processor upgrade: obsolescence – remain with same processor family.	Medium, Infrequent.	Retain object code compatibility, but need to re-verify existing code on new hardware coupled with change of clock speed and shorter execution times.
New hot drink vending options.	High, Frequent	Software changes to add new selection options and different vending sequencing.
Support installation sites with limited plumbing facilities.	High, Infrequent Assessed as high to reflect current marketing/ business exploitation plans.	Addition of a removable drip tray to handle spillage waste water.
Cold Drink vending option.	Medium, Infrequent Assessed as medium as a balance between marketing/ business exploitation plans and complexity/ cost of change.	Addition of an independent cold water system. Significant impact on software to support additional vending options and control the cold water system.

Table 13-1 EspressoMat Change Scenarios

The resulting software Design Architecture for the EspressoMat (Mk I) is shown in Figure 13-1 & Figure 13-2.

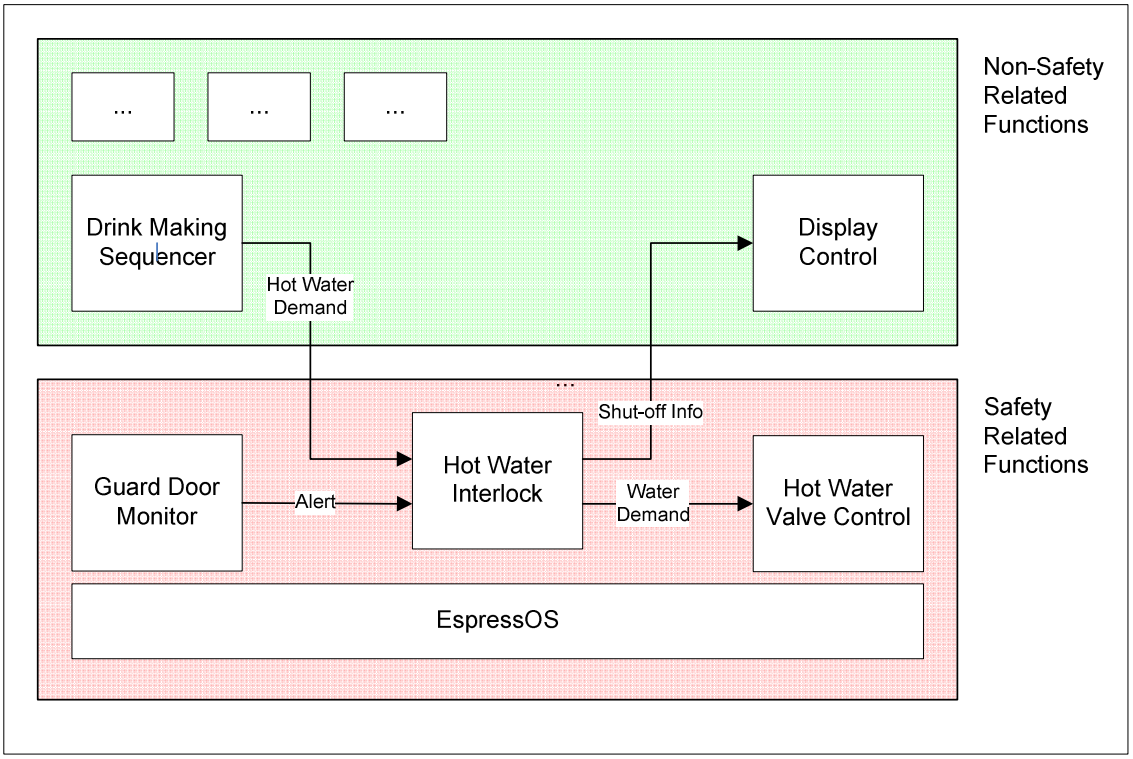


Figure 13-1 EspressoMat Mk I Software Architecture

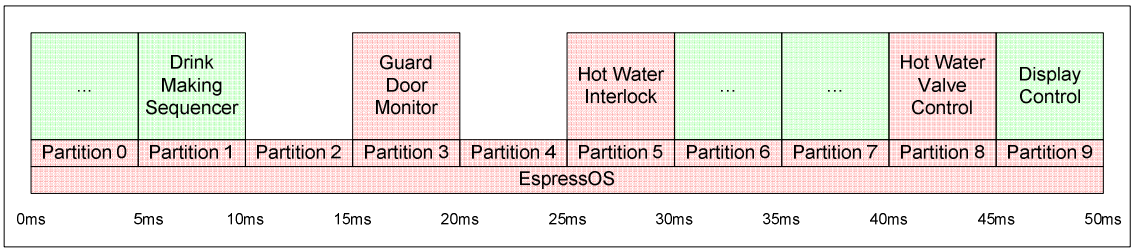


Figure 13-2 EspressoMat Mk I Partition Schedule

13.2 Mk I Safety Case Architecture

The Safety Case sets out the argument and evidence that the EspressoMat is adequately safe to be put into service; part of this Safety Case is the Software Safety Case and this sets out the argument and evidence that the Safety Requirements allocated to the software are adequately shown to be satisfied. This top level argument is shown in Figure 13-3.

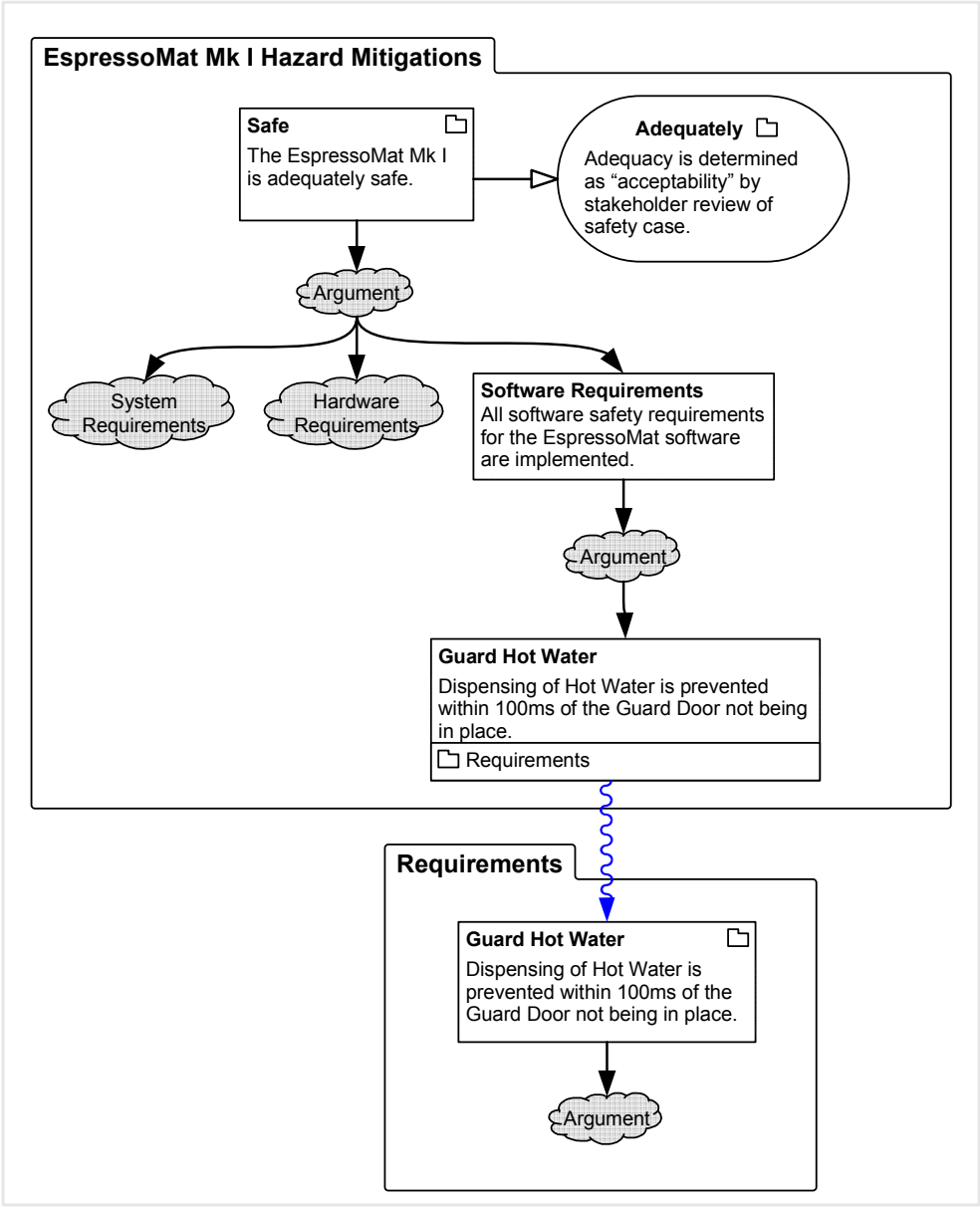


Figure 13-3 EspressoMat Mk I Top Level Safety Goal

The rest of the Software Safety Case Architecture derived for the system is shown in outline in Figure 13-4; this shows the connections between the different Safety Case Modules. Figure 13-5 fills in some of the detail of this architecture by showing the goals providing and requiring support in each Safety Case Module, and where these goals are used and developed (which define the interconnections between the Safety Case Modules). The main functional applications are connected via Contract Safety Case Modules (indicated by the small black contract module icons).

> *Note: The Safety Case structure presented addresses the EspressoMat in its main operating mode, any issues of its safety during initialisation are not addressed. Similarly, detailed argument relating to the underlying EspressOS software has been omitted and the focus kept on the main functional applications in the EspressoMat software. As a further simplification some of the supporting claims (such as Initialise/Configures) are only shown in the "Hot Water Interlock" Block Safety Case Module.*

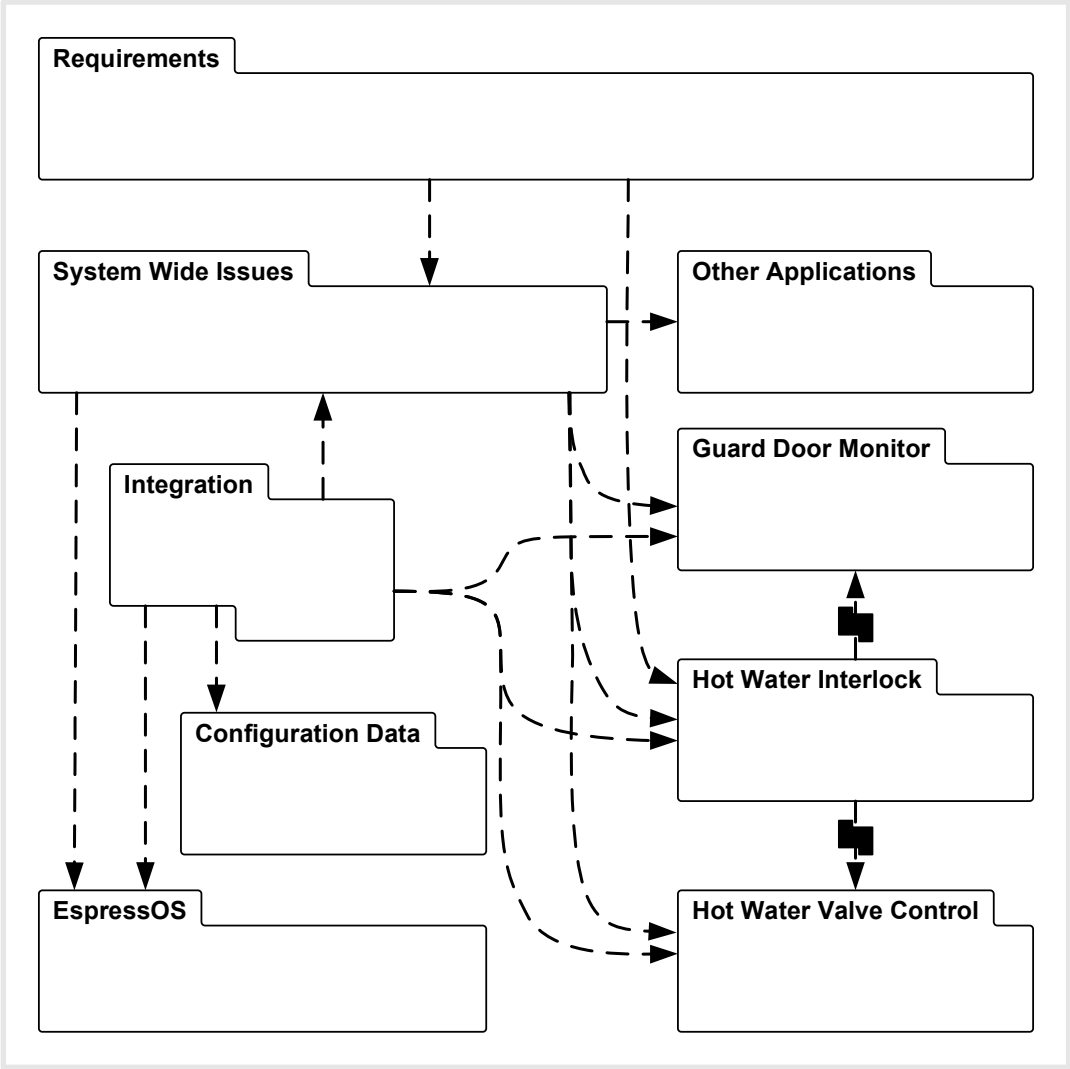


Figure 13-4 EspressoMat (Mk I) Safety Case Architecture Summary

In Figure 13-5 the “Requirements” module develops the top level safety requirement, separating it into a functional claim (which is supported by the guarantee of the “Hot Water Interlock” module) and a performance (latency) claim (addressed by the “System Wide Issues” module). The argument shown here, in “Requirements”, covers the normal operational mode of the EspressoMat, but this still depends upon the system having been properly initialised – this is supported by the “Initialised” goal from the “Integration” module.

The “Hot Water Interlock” module provides a guarantee (“Hot Water Off”) that is suited to supporting the principal safety function, but this guarantee depends upon guarantees from other parts of the system. Two of these are supported by other functional applications (“Guard Door Monitor” and “Hot Water Valve Control”) which are developed independently from “Hot Water Interlock”. To preserve this independence in the Modular Safety Case Contract Safety Case Modules (indicated by the small black icons mimicking the GSN contract module shape) are used to link the goals requiring support in “Hot Water Interlock” with goals providing support in “Guard Door Monitor” and “Hot Water Valve Control”.

“Hot Water Interlock” also relies upon the inter-application communication services (which are provided by EspressOS in the EspressoMat), this dependency is identified by the “Correct Services” goal requiring support. The integration of such dependencies (from the

application modules) with the guarantees of the “EspressOS” module are made in the “Integration” module; this is shown as a pattern argument (starting with the goal “Correct Services” from “{SW Block}”).

Similarly the “Not Prevented” goal requiring support in each Block Safety Case Module, is addressed in the “Integration” module; this in turn relies on a claim that all the software in the system (including other applications which are not otherwise involved in safety) is well-behaved. This claim is provided by the “System Wide Issues” module which in turn depends upon each Block Safety Case module providing a guarantee that it is well behaved.

The application modules also provide other guarantees which support different aspects of the Safety Case argument; the “Response” guarantee provides performance assurances, and these are used in the “System Wide Issues” module to support a claim that the performance (latency) part of the safety requirement is satisfied.

The final parts of the Block Safety Case Module relate to the correct initialisation and configuration of each software component. The operational guarantee (e.g. “Hot Water Off” in the “Hot Water Interlock” module) is dependent on the software for that block having been correctly initialised/configured. In the EspressoMat this happens during the power-up sequence and that aspect of the system is not presented here. However, any such initialisation/configuration is likely to rely on the software component performing its own initialisation/configuration when requested to by the system; this is supported by the “Initialise”/”Configures” guarantee of the Block Safety Case Module (this is only shown in the “Hot Water Interlock” module and without any development of the argument).

For the EspressoMat the initialisation/configuration is controlled by configuration data tables, which need to be valid and correct. This guarantee is provided by the “Configuration Data” module and this is used to support the “Initialised” argument in the “Integration” module.

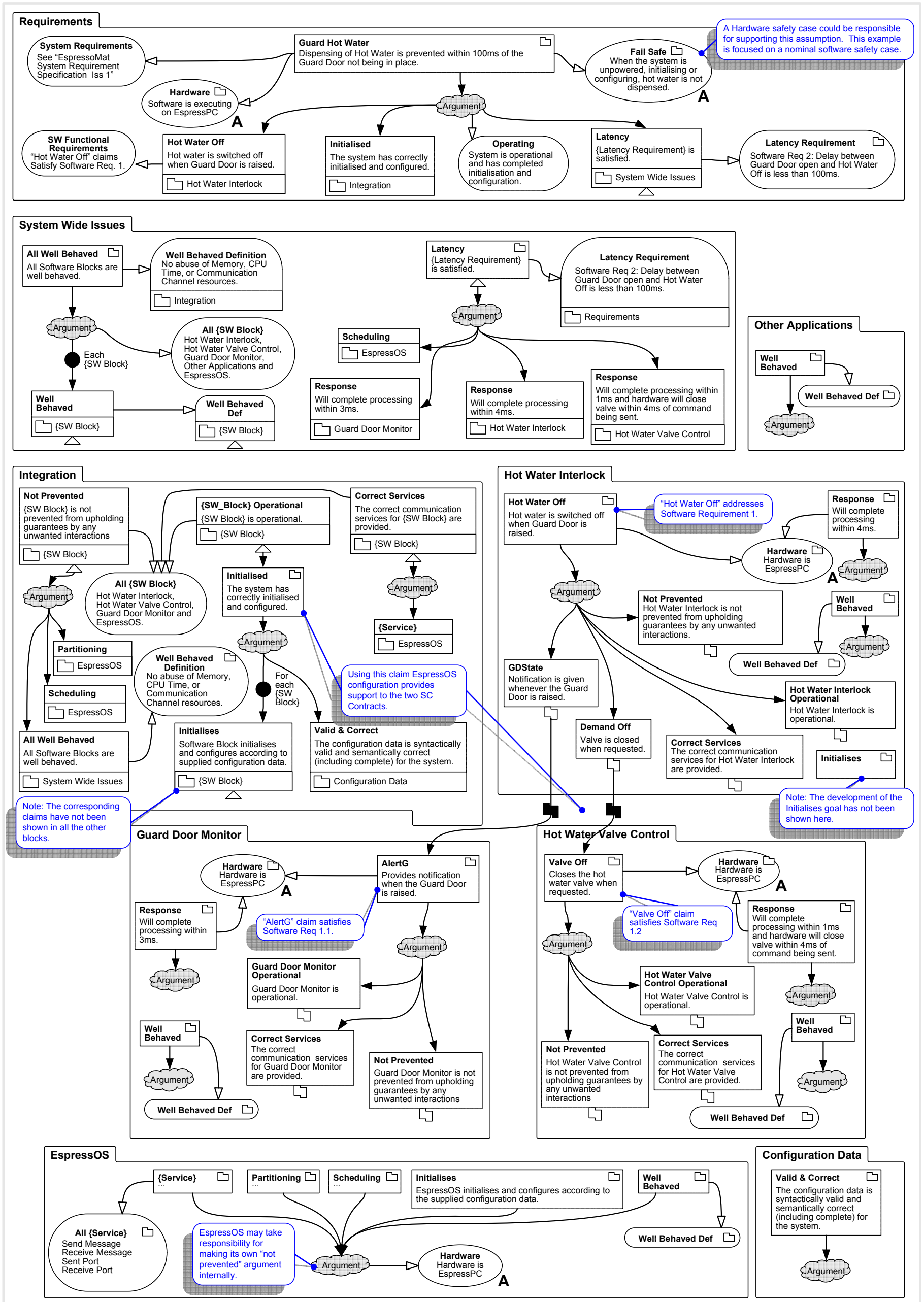


Figure 13-5 EspressoMat Mk I Safety Case Architecture

13.3 Dependency Guarantee Relationships

As an alternative to what is shown in the Safety Case Architecture diagram above (explicit dependency references) the supported claims (goals) of a Block Safety Case Module may be expressed using Dependency Guarantee Relationships (DGRs). This would be presented as a guarantee claim pattern with the actual guarantees and the dependencies shown as context, see Figure 13-6. For the EspressoMat there are no block wide dependencies, all the dependencies are captured explicitly under the "Related Dependencies" context. The DGRs for the '*AlertG*' claim of '*Guard Door Monitor*', and the '*Hot Water Off*' claim of '*Hot Water Interlock*' are shown in Table 13-2 & Table 13-3. These are used to support the integration argument (see section 0).

> *Note: The supporting 'Not Prevented' goal does not appear as a dependency; this is a goal that is to be addressed directly during integration (rather than by reference to a supporting goal from another application module). The 'Not Prevented' goal would appear elsewhere in the Block Safety Case Module argument – but is not shown here.*

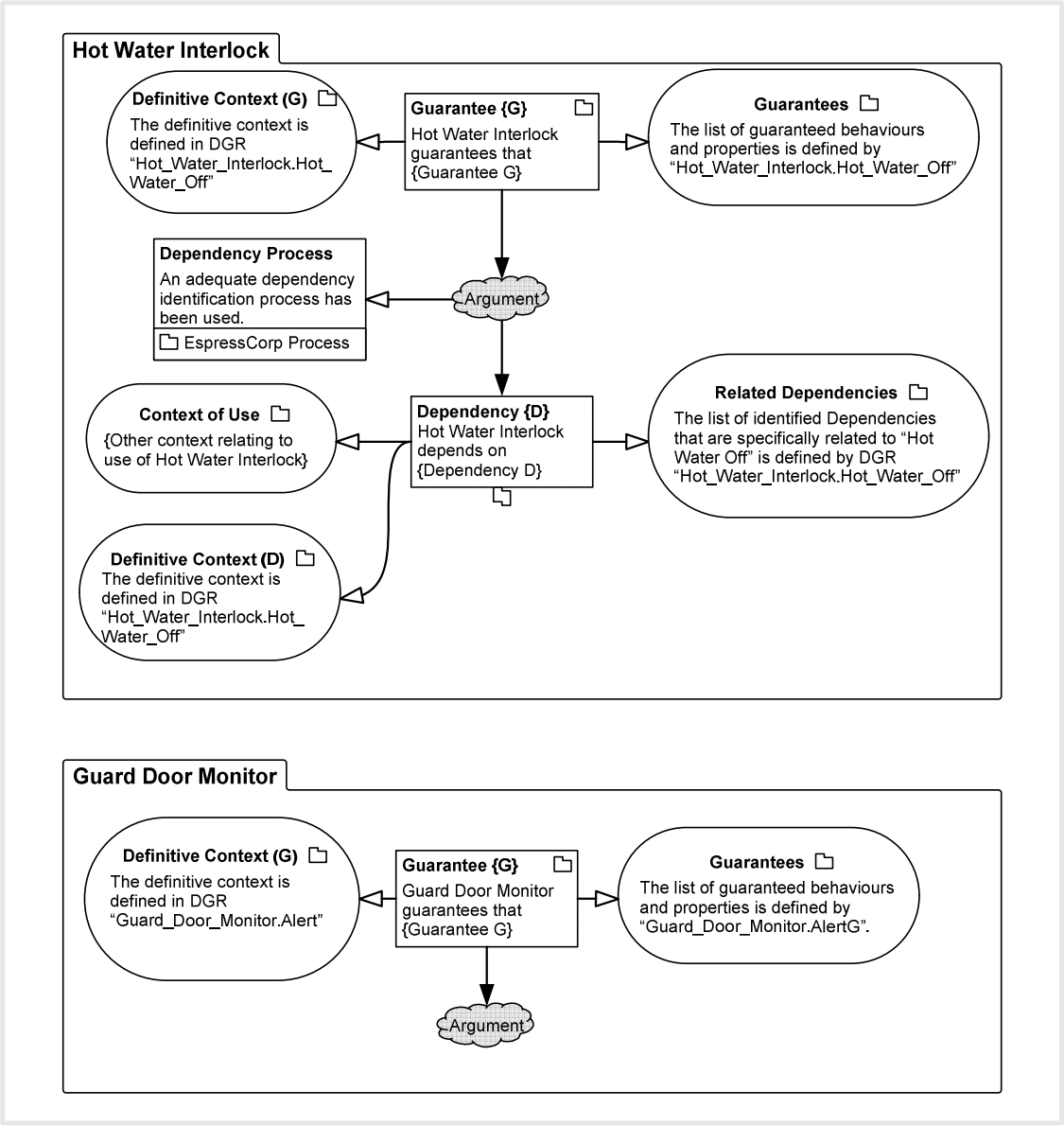


Figure 13-6 DGRs as Context

Dependency – Guarantee Relationship	Hot_Water_Interlock.Hot_Water_Off

Guarantee

Concise Definition	Definitive Context	Incidental Notes	Traceability
Hot water is switched off when Guard Door is raised.	*Hot Water* control is on channel 102. *Water Off* message format defined by Ref {Interface Specification}.		Req 1

Related Dependencies

N°.	Concise Definition	Definitive Context	Incidental Notes	Traceability
1	Guard Door notification message is received on channel 100.	The notification message format is defined by the EspressoMat {Interface Specification}.		
2	*Hot water valve is closed when request is sent on channel 102.*	The notification message format is defined by the EspressoMat {Interface Specification}.		
3	*A Receive_Message* service is available.	*channel* set to 100. *Receive_Message* behaviour is as defined in {API Specification}		
4	A *Send_Message* service is available.	*channel* set to 102. *Send_Message* behaviour is as defined in {API Specification}		
5	*Hot_Water_Interlock* is correctly configured.	Is executing and has completed configuration.		

Table 13-2 DGR Hot Water Interlock.Hot Water Off

Dependency – Guarantee Relationship	Guard_Door_Monitor.AlertG		
Guarantee			
Concise Definition	**Definitive Context**	**Incidental Notes**	**Traceability**
Provides notification when the Guard Door is raised.	*Alert* is sent on channel 100. *Alert* message format is defined by EspressoMat {Interface Specification}.		Req 1.2
Related Dependencies			
N°.	**Concise Definition**	**Definitive Context**	**Incidental Notes**	**Traceability**
1	Guard Door status is received via port 50.	Guard Door status message format defined by EspressoMat {Interface Specification reference}.		
2	*A Receive_Port* service is available.	*port* set to 50. Receive_Port behaviour is as defined in {API Specification}		
3	A *Send_Message* service is available.	*channel* set to 100. *Send_Message* behaviour is as defined in {API Specification}		
4	*Guard Door Monitor* is correctly configured.	Is executing and has completed configuration.		

Table 13-3 DGR Guard Door Monitor.AlertG

13.4 Dependency Guarantee Contract

DGRs are used during the integration of Block Safety Case Modules to provide the details of the goal *requiring* support and the goal *providing* support. The integration is performed within a Contract Safety Case Module which uses a DGC to set out the matching between the DGRs of the goals involved. Figure 13-7 shows the Contract Safety Case Module argument pattern for the integration of the *Hot Water Interlock* and *Guard Door Monitor* Safety Case Modules; this references the DGC *Hot Water Interlock*. Table 13-4 shows the part of the DGC table concerned with the functional aspects of the *Hot Water Off* goal where the dependency is supported by the *AlertG* goal.

> *Note: In this example the concept of virtual communication channels being mapped to physical communication channels has not been developed; instead a single channel identification is used throughout – it might be presumed that this is a restriction of EspressOS.*

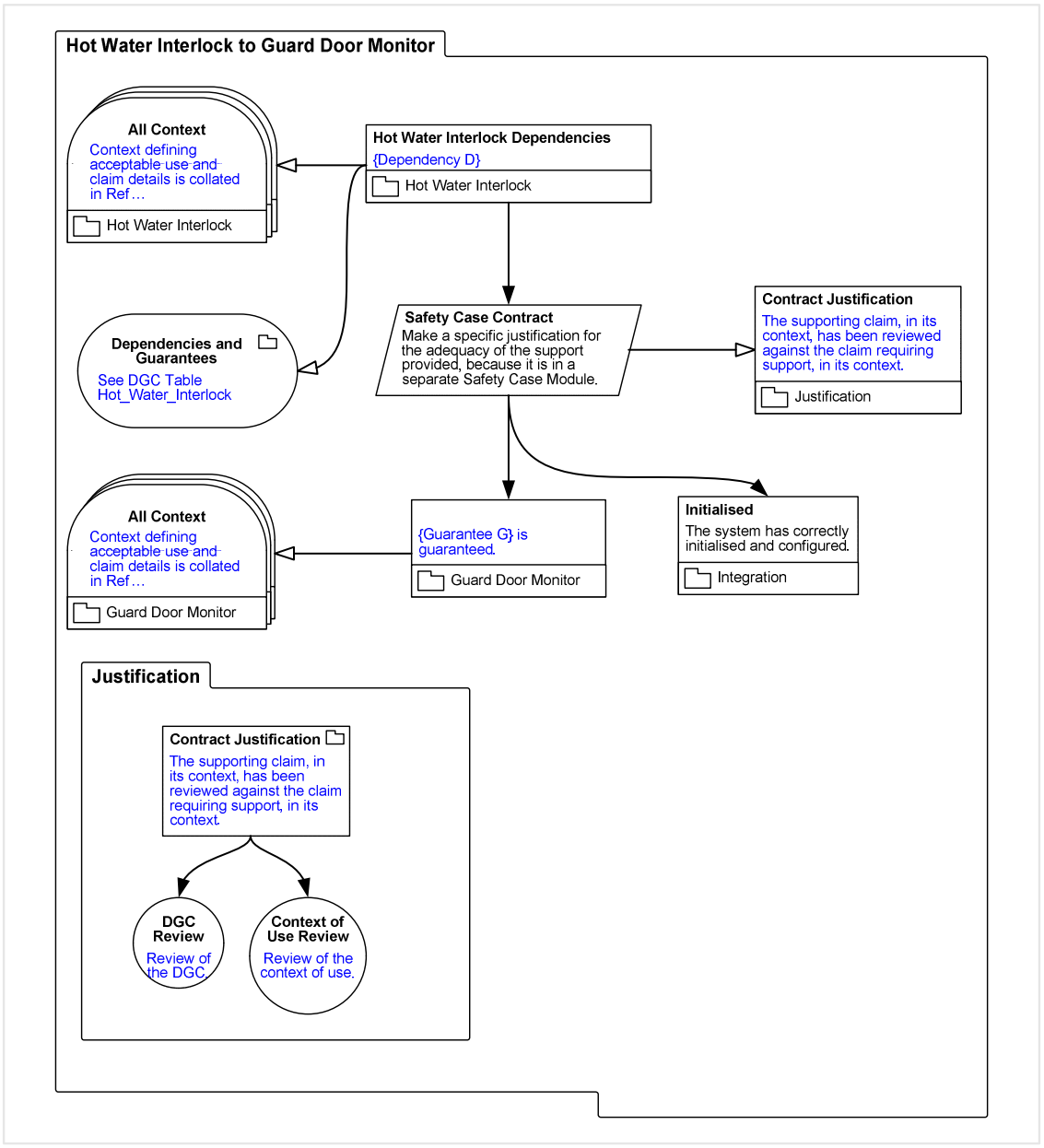


Figure 13-7 Integration Argument Pattern for Hot Water Interlock

Dependency – Guarantee Contract		Hot_Water_Interlock	
Consumer Dependency	**Integrator**		**Provider Guarantee**
Hot_Water_Interlock.Hot_Water_Off.1	**has SC Contract with**	✓	**Guard_Door_Monitor.AlertG**
Guard Door notification message is received ….	is supported by		Provides notification when the Guard Door is raised.
… on channel 100.	is consistent with [In the system configuration data this connection uses physical channel 100]		*Alert* is sent on channel 100.
The notification message format is defined by the EspressoMat {Interface Specification reference}	is consistent with		*Alert* message format is defined by EspressoMat {Interface Specification reference}.
…	…		…

Table 13-4 DGC for Hot_Water_Interlock.Hot_Water_Off

13.5 Update to Create EspressoMat Mk II

To examine how a change, or increment, would be incorporated into the Safety Case the Mk II EspressoMat is introduced. This addresses one of the Change Scenarios shown in Table 13-1:

> "*Support installation sites with limited plumbing facilities.*"

This introduces a new component to the design, a removable Drip Tray, and new hazards concerned with operating the machine when the Drip Tray is full or removed.

The revised design is shown in Figure 13-8 and Figure 13-9; the changed elements are shaded. Note that although the Display Control function is also being altered, this is not safety related so has no direct impact on the Safety Case; there would be an indirect impact through supporting System Wide Issues (such as supporting the '*All Well Behaved*' claim).

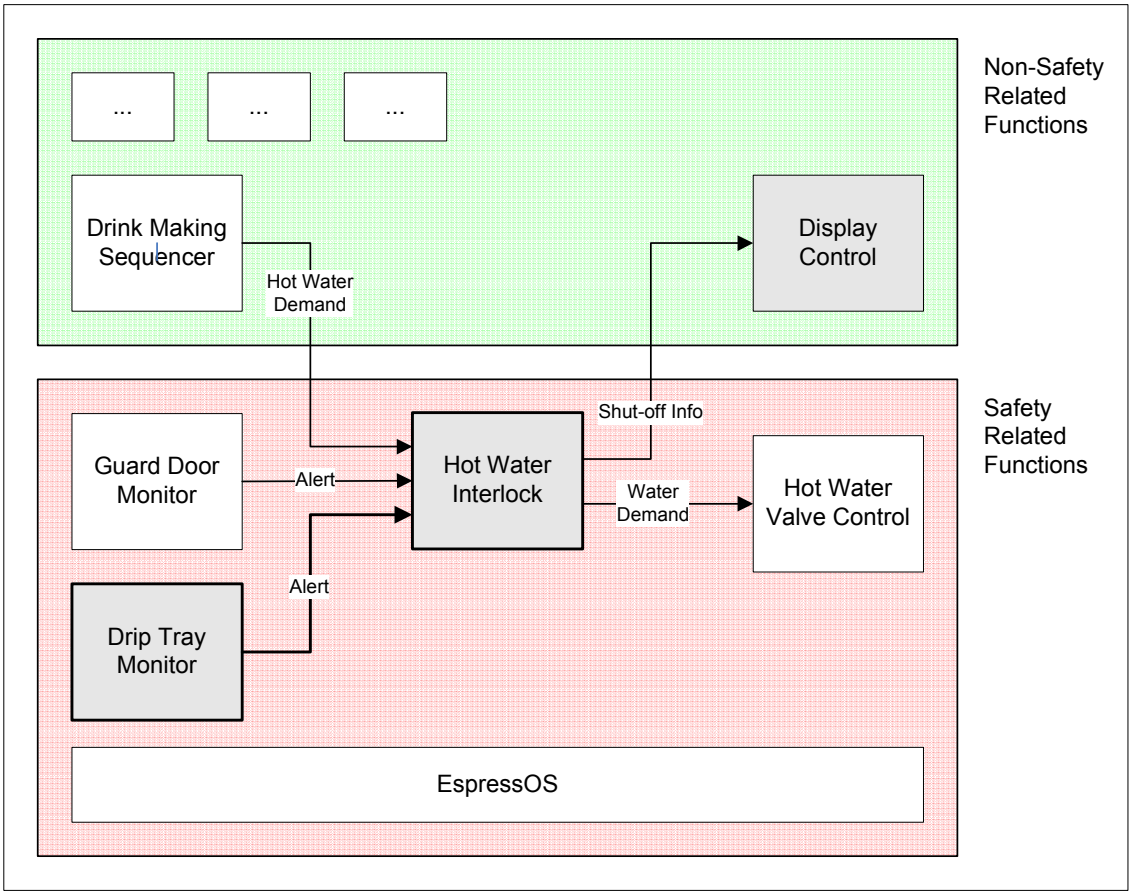


Figure 13-8 EspressoMat Mk II Software Architecture

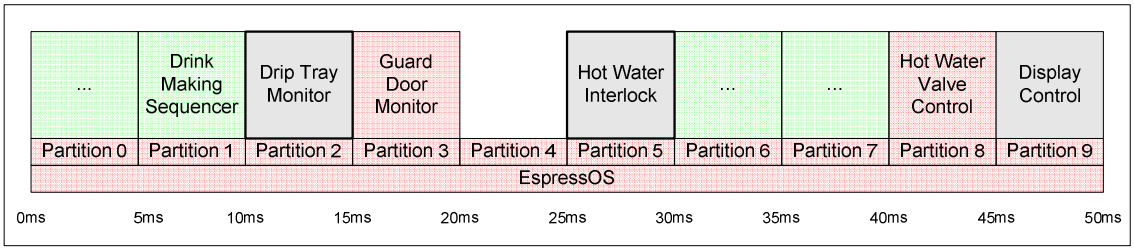


Figure 13-9 EspressoMat Mk II Partition Schedule

13.6 Safety Case Impact Analysis

Using the EspressoMat Mk I Safety Case as the starting point the Change Impact Analysis identifies the areas of the Safety Case that need to be revisited. Table 13-5 shows part of the impact analysis concerned with:

- Introducing the new requirement,
- The new argument elements expected in Hot Water Interlock to account for the Drip Tray, and
- The re-visiting of the previously existing integration Contract Safety Case Modules for Hot Water Interlock as a result of it being up changed.

Figure 13-10 shows more of the impact analysis; the Safety Case structure represented in GSN has been used to guide the analysis by showing which other parts of the Safety Case are connected to changed elements. These connected elements can then be assessed to determine whether they are affected or not (adding to the tabulated analysis).

Note: Tracing of the code and supporting evidence changes are not shown in this figure as the example has not developed those parts of the Safety Case in

sufficient detail. Also, the impact of the changes to Other Applications (which would require their 'Well Behaved' claim to be re-analysed) is not shown here.

Ref	SCM Affected	SC Element Affected	Claim Affected	Explanation	Impact Ref
1	-	-	-	New Requirement - "Dispensing Hot Water prevented within 100ms of Drip Tray becoming full or removed"	2
2	Requirements	Context	System Requirements	Requirements document updated.	3
3	Requirements	Claim	Guard Hot Water	Requirement revised	4
4	Requirements	…	…	…	
…	…	…	…	…	8
8	Requirements	Claim	Hot Water Off	Functional claim requiring support changed	9
9	Hot Water Interlock	Claim	Hot Water Off	Extend claim to cover Drip Tray becoming full or being removed.	…
…	…	…	…	…	…
15	Hot Water Interlock	Claim	*New*	Include a functional claim requiring support to capture dependencies on Drip Tray Monitor.	16
16	*New*	-	-	New Application Integration Safety Case Module to handle integration of Hot Water Interlock with new Application Module for Drip Tray	-
…	…	…	…	…	…
45	…	Contract	…	Change to Hot Water Interlock SCM requires integration contracts to be re-visited.	46, 47
46	Hot Water Interlock to Guard Door Monitor	Contract	-	Re-integrate new issue of Hot Water Interlock SCM; no changes at this interface.	-
47	Hot Water Interlock to Hot Water Valve Control	Contract	-	Re-integrate new issue of Hot Water Interlock SCM; no changes at this interface.	-
…	…	…	…	…	…

Table 13-5 Change Impact Analysis

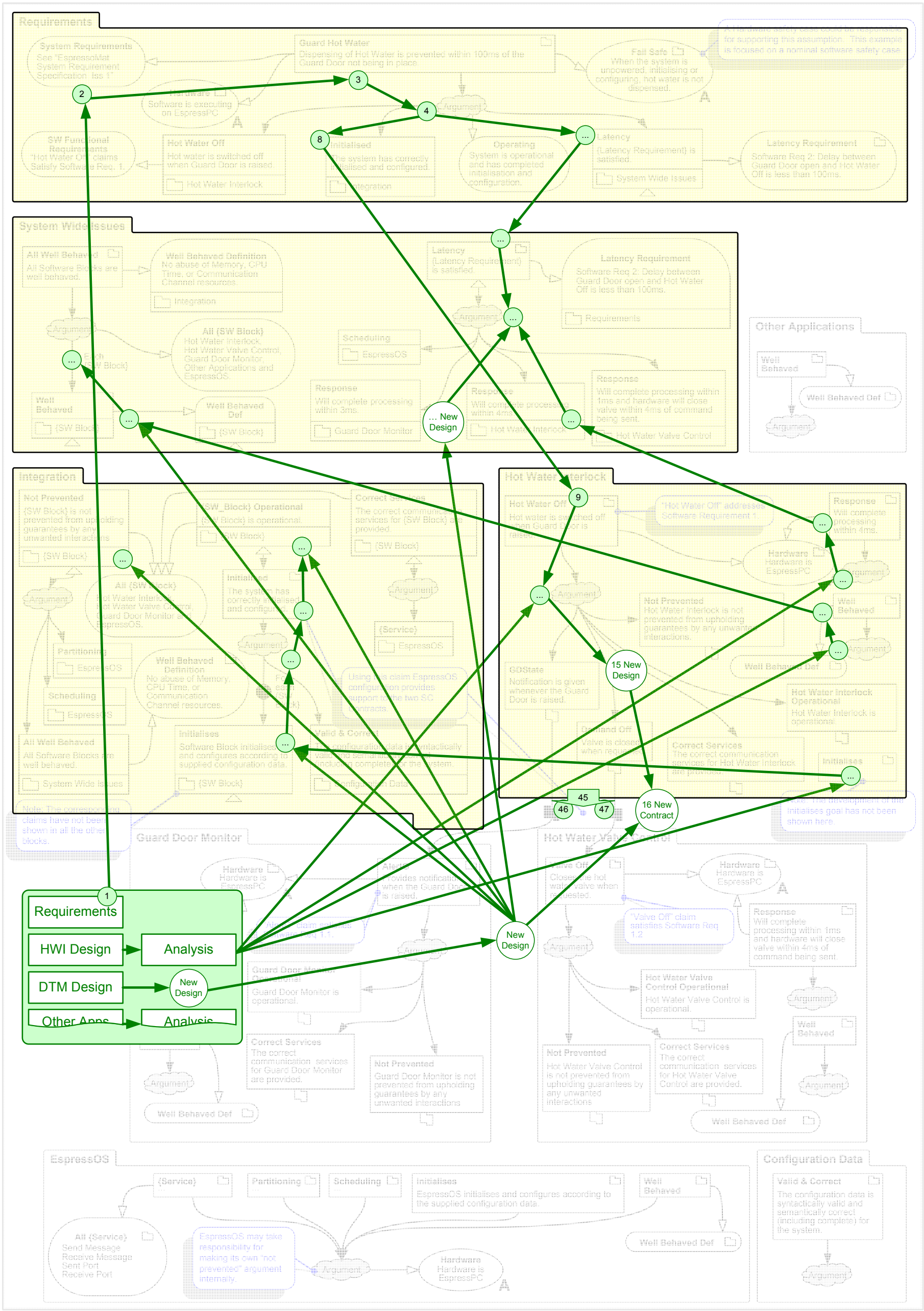


Figure 13-10 EspressoMat Change Impact Analysis

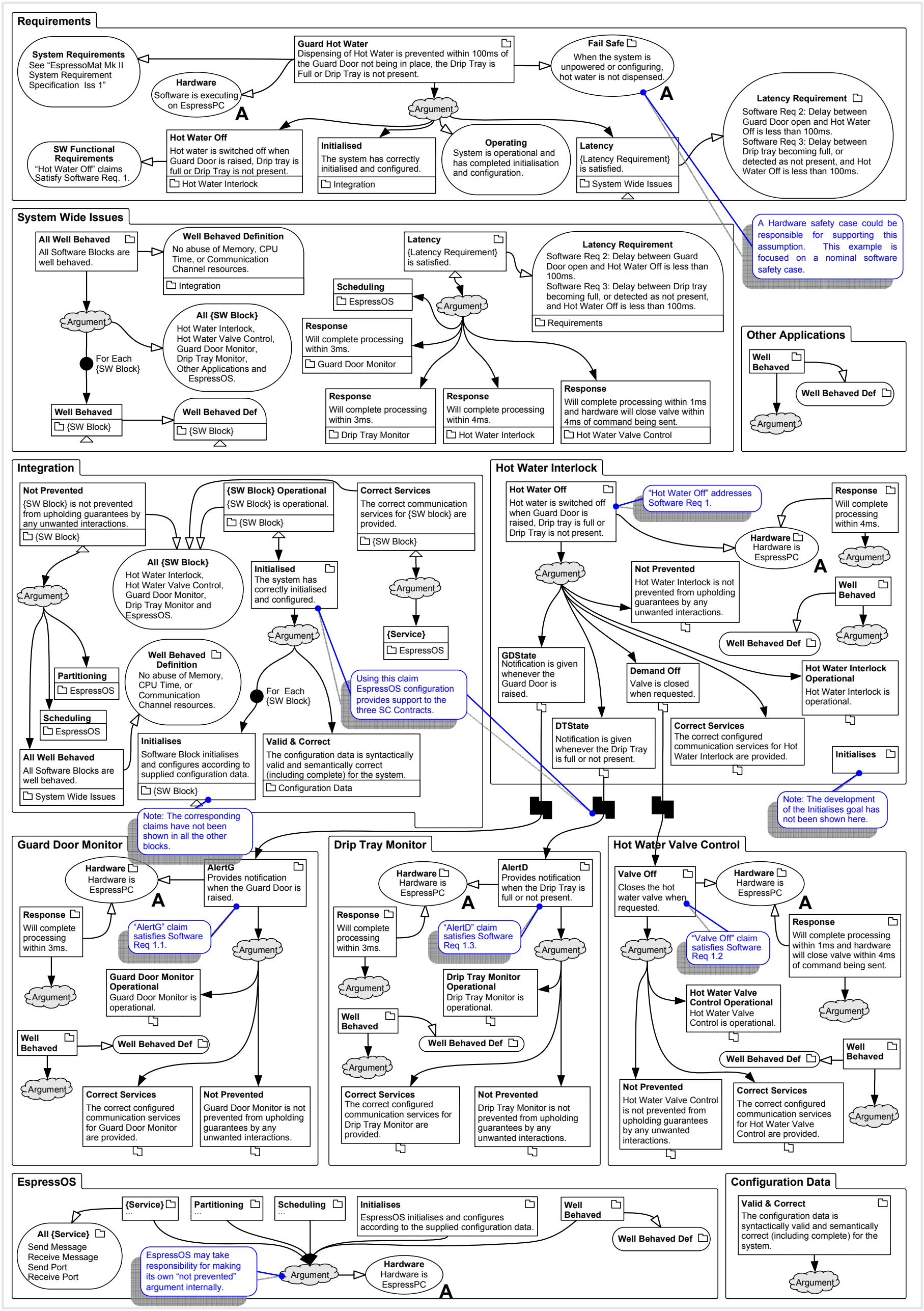


Figure 13-11 EspressoMat Mk II Safety Case Architecture

13.7 Mk II Safety Case Architecture

The updated detailed Safety Case Architecture for EspressoMat Mk II is shown in Figure 13-11. This shows the revised Requirements, Hot Water Interlock, Integration and System Wide Issues Safety Case Modules together with the new Drip Tray Monitor Safety Case Module.

13.8 Change Argument

Figure 13-12 shows the Change Argument used to support the incremental update to the Safety Case for EspressoMat Mk II; it shows the reasoning used by EspressCorp to convince themselves of the two principal claims.

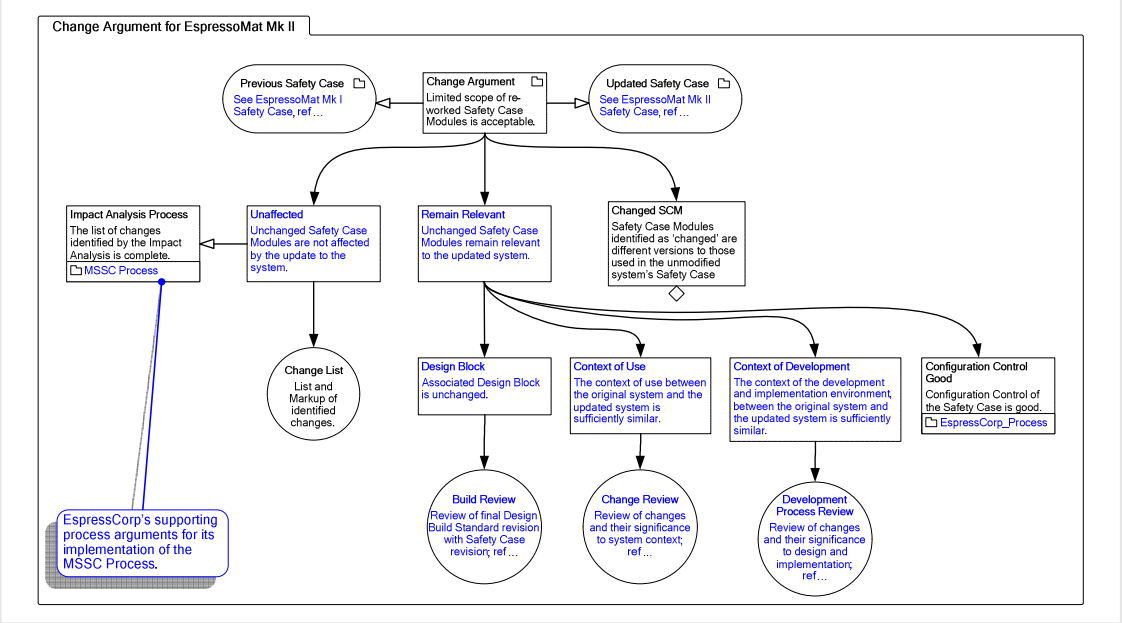


Figure 13-12 EspressoMat Mk II Supporting Change Argument