

ASSURANCE OF CLAIMS AND EVIDENCE FOR AVIATION SYSTEMS

Squadron Leader D.W. Reinhardt

Royal Australian Air Force
Deputy Senior Design Engineer – Avionics C-130H/J
Building 108, RAAF Richmond, NSW 2755, Australia

derek.reinhardt@defence.gov.au

Professor J.A. McDermid OBE FREng

Head of Department of Computer Science
University of York
United Kingdom

john.mcdermid@cs.york.ac.uk

Abstract

The failure circumstances of complex aviation systems involving technologies such as software are dominated by systematic faults. However, systematic faults are often poorly resolved by the coupling of software assurance with traditional system safety methodologies. This paper examines an alternative approach to the assurance of software against systematic faults in aviation systems.

Earlier work in this body of research (refer to [ReM10]) proposed an assurance framework that provides a direct measure of the extent of a system's fault tolerance against systematic faults and failures. Central to the assurance framework was the concept of an Architectural Safety Assurance Level (ASAL) which was a product measure of the number of systematic failures a system is resilient against in a given context. The ASAL framework can be used to infer a system's architectural suitability for use in the presence of aircraft level failure conditions of differing severities.

However, the extent of discussion in the earlier research has so far only addressed the architectural effects of layered detection and handling mechanisms on bounding the uncertainty of systematic faults. The earlier research did not address which specific detection and handling mechanisms are most appropriate in each context, and how claims to that effect might be assured. Furthermore, the framework hasn't yet set any benchmarks for the provision of evidence in this regard.

Furthering the earlier research, this paper proposes a Claims Safety Assurance Level (CSAL) and Evidence Safety Assurance Level (ESAL) concept that is compatible with the ASAL concept. The core idea behind claims assurance is to ensure that any assurance levels used for articulating claims assurance in the context of the ASAL have a specific product safety focus (i.e. each and every assurance level has a product meaning, not just a top-down or bottom up process interpretation). For evidence assurance, the core idea introduces the concept of 'tolerability of limitations'. The 'tolerability of limitations' is intended to be a product

behavioural measure of the 'tolerability' in the provision of suitable evidence, while explicitly taking into account any limitations / shortfalls in the provision of evidence. The 'tolerability of limitations' also takes into account any known product shortfalls/limitations. The intent of evidence assurance is to provide a framework that is explicit with respect to the 'tolerability of limitations' of evidence with respect to safety.

Keywords: Architecture, Assurance, Aviation Systems, Fault Tolerance, Safety, Software Assurance, Software Safety, Safety Critical.

1 Introduction

In complex aviation systems involving technologies (e.g. software) whose faults are dominated by a class of faults referred to as systematic faults, there are substantial challenges to providing assurance that these faults do not lead to unacceptable failure conditions.

Most safety and assurance standards for software systems in the aviation domain deal with systematic faults through the identification and allocation of software safety requirements and through the specification of software integrity levels or design assurance levels. However there are numerous limitations (refer to [JTM07], [McD07], [McK06], [NTS06], [Wea03]) with the mechanisation of the current assurance frameworks that limit their effectiveness at providing robust assurance that systematic faults do not lead to unacceptable aircraft failure conditions. This is primarily because of limitations in the treatment of requirements validity of the system and software, as well as in the direct provision of evidence that behaviours of the system and software are acceptable with respect to safety. The current standards are also limited in their ability to determine the impact on the aviation system of shortfalls against the criteria of the standards.

This paper extends earlier work in this body of research (refer to [ReM10]) that proposed an assurance framework that provides a direct measure of the extent of a system's fault tolerance against systematic faults and failures. Central to the assurance framework was the concept of an Architectural Safety Assurance Level (ASAL) which was a product measure of the number of systematic failures a system is resilient against in a given context. The ASAL framework proposed the use of layered absence or detection/handling

mechanisms at the Software, LRU and System levels for treating sources of systematic faults, as is evident in the implementation of numerous actual aviation systems. The ASAL framework can be used to infer a system's architectural suitability for use in the presence of aircraft level failure conditions of differing severities.

This paper extends the earlier research by examining an approach to defining assurance levels for claims and evidence associated with layers of absence or detection/handling mechanisms defined in the ASAL concept. The framework proposed in this paper is intended to be compatible with existing assurance standards' approaches, even if the existing standard were not to explicitly adopt the claims and evidence assurance framework proposed in this paper.

2 Background

In order to provide an explicit basis for establishing the CSAL and ESAL concepts described in this paper, the approach chosen in this research was to define a set of key principles for assurance levels in assurance standards that is consistent with the themes developed in the ASAL concept.

To derive these key principles for assurance levels in assurance standards, it is necessary to understand the role of assurance standards, the limitations in the effectiveness of the current assurance frameworks with respect to claims and evidence in aviation systems, and how the sufficiency of claims and evidence might be measured. The following subsections elaborate on these topics, and identify key principles of assurance frameworks that assurance standards should uphold.

2.1 Roles of assurance standards

Understanding the roles of assurance standards is vital to defining frameworks within assurance standards that are compatible with those roles. The following subsections, describe several factors in relation to the role of assurance standards.

2.1.1 Standardisation of acceptable practice

One of the most important roles for standards is to standardise acceptable practice. The word 'standard' in general English language definition can imply the following:

- "anything taken by general consent as a basis of comparison"
- "a level of quality which is regarded as normal, adequate, or acceptable"

The key points here are the basis of comparison, usually expressed by a set of criteria, against a measure of acceptability (i.e. the passmark). Therefore, applying these key points to the concept of an assurance standard should lead to requirements on the assurance standard for providing a basis of comparison between a resulting product and its assurance evidence, and the desired outcomes of the standard – in the case of an assurance standard the goal might be

safety. For example posing the question rhetorically, what are the structured set of properties of the product and its evidence that permits a conclusion to be directly established that the behaviours are appropriate with respect to safety?

Unfortunately many of the frameworks underpinning assurance standards that exist today confuse premises for conclusions/outcomes and thus prescribe a basis of comparison focussed around the means of development or assessment, rather than about the suitability of the behaviours with respect to safety. The authors of this paper in general raise no objection to the many valid premises (all reflecting acceptable practices) that underpin these standards, but the inference that they lead to the right conclusions is usually implicit, if not missing altogether. There are also limited instances where it is questionable that some standard's premises even link to an appropriate conclusion. While, this is certainly a limitation to existing frameworks, the developers of these frameworks were not entirely at fault for this circumstance.

When acceptable practice is established based on premises (things practitioners become very familiar with through practical experience), then the acceptable practice will focus on the means (e.g. what test method should I use, how should I write my requirements, etc). This is acceptable where premises lead directly to conclusions. However for assurance standards involving technologies, whose failures are dominated by systematic failures, rarely does a premise lead directly to a conclusion. Furthermore, the technologies involved, plethora of techniques and methods, architectural options and implementation possibilities, all lead to numerous approaches to any one design problem. This creates a challenging conundrum. Should the assurance standard define the preferred combination of the means (or those premises with which practitioners are most familiar), or should the assurance standard focus on how the premises link to conclusions? The authors reason that the focus should be on the latter, rather than the former.

2.1.2 Benchmarks for contractual or regulatory compliance

In addition to the standardisation of acceptable practice, a related role of standards which is wholly applicable to assurance standards is providing consistent benchmarks for contractual or regulatory compliance. While the concept of benchmarks is relatively straightforward, which benchmarks are suitable, and how best to articulate them, is substantially a more challenging question to the question of benchmarks itself.

Often regulators will publish documents that specify 'one means but not the only means'. This is an approach to documenting a set of premises established from the means (with which it is hoped the industry will use or employ), and providing a worked example, acceptable to the regulator, of how these premises satisfy the conclusions/outcomes of the assurance standard in question.

These benchmarks per se, are very important for documenting practical understanding of the regulator's interpretation of the application of the outcomes/conclusions specified in an assurance standard.

For contractual compliance, however, such guidance documents, are much less frequently available, particularly where the role of the regulator may be divorced from the contractual compliance substantiation. In these cases it is very important that the assurance standard is self standing. Therefore, a key factor for any assurance standard is ensuring that the means of satisfaction be made explicit, without overly constraining the means (or set of premises) with respect to the conclusions/objectives of the standard.

2.1.3 Compliance assurance and managing risk

Inevitably the compliance assurance programs of regulators will eventually find non-compliances with respect to the impact of any standard. The important thing is that the meaning of these non-compliances can be established. For product standards, the meaning is usually fairly straightforward – i.e. the product is unsafe because it doesn't have a particular property. However, for assurance standards, which have historically had large process associations, the impact of the non-compliance might be less certain.

Therefore, it is very important that the definition of objectives and outcomes in an assurance standard be explicit with respect to their product meaning, so the safety impact of any non-compliance can be determined. This provides the regulator with a much better basis for managing the tolerability of any risk associated with non-compliance, rather than being uncertain as to the specific risk. There are also benefits to this approach if shortfalls are learned about retrospectively, and the regulator is faced with reassessing risk and promulgating interim risk treatments until the non-compliance can be properly resolved.

2.2 Suitability of claims and sufficiency of evidence

[ReM10] identified that one of the limitations with existing software assurance standards is with the direct provision of evidence that the behaviours of the system and software are acceptable with respect to safety. This limitation exists because:

- the assurance levels used in existing standards don't have any inherent product meaning; and
- the objectives (where used) are all expressed as outcomes of the software lifecycle, rather than in terms of their contribution to assuring behaviours of the software product with respect to safety.

Therefore it is important that any assurance framework used for claims ensures that the relevance of the claims to the assurance of behaviours of the software with respect to safety remains explicit. The assurance framework should also be explicit in how much (and of what strength) of evidence is necessary to make the claims compelling and bound the uncertainty of the claim being violated.

2.3 Key principles of assurance level definitions

Based on the discussion in sections 2.1 and 2.2, key principles of assurance level definitions are established as follows:

- assurance levels should inherently have a product meaning – i.e. they should be a measure of some physical property of the product and its behaviours, and non-satisfaction of the assurance level should directly infer a product behavioural difference;
- assurance levels should focus on outcomes rather than activities – i.e. they should not concern themselves with specific techniques or methods, but instead set objective benchmarks for properties of the product that should be established;
- the assurance framework should make explicit the relevance of the claims underpinning the assurance level definition;
- the assurance level framework should include a mechanism for inferring the relationship between any given technique and method, and the outcomes or objectives they satisfy by ensuring that the factors/properties underpinning each objective are explicit; and
- the assurance framework should be goal setting in terms of outcomes and objectives of the framework, and only as prescriptive in premises as necessary to ensure explicit benchmarking for compliance with respect to product related behaviours of the software.

The frameworks proposed in this paper are intended to satisfy these principles.

2.4 Relationship to architectural assurances

The ASAL concept, discussed in [ReM10], uses layers of absence and detection/handling mechanisms to provide assurance that systematic faults do not lead to unacceptable failure conditions. Within the scope of this paper, each of the absence or detection and handling mechanisms shall be considered a 'constraint' on the behaviour of the system and its software. Therefore, irrespective of the specific absence or detection/handling mechanism being considered, the claims made with respect to assuring the 'constraint' can be considered from a common basis. This also permits the claims and evidence for each 'constraint' to be developed to inherently argue that the specific absence or detection and handling mechanisms (i.e. the 'constraint') is appropriate in the given system wide context.

For example, for an omission failure mode whose primary absence mechanism is "All feasible control paths through CSF include a unique output statement" (refer to [Wea03]); this behaviour would be annotated as a 'constraint' in the framework defined by this paper.

For each 'constraint' it is necessary to establish the degree to which the 'constraint' should be assured, and also what the degree of assurance actually means in a product sense.

Assurance of the claims and evidence with respect to the ‘constraint’ is the topic of this paper.

2.4.1 Non interference of constraints

Since the assurance approach relies on layers of absence and detection / handling mechanisms to provide assurance that systematic faults do not lead to unacceptable failure conditions, it is important that the non-interference of these mechanisms is addressed, so the reader can understand the relationships to the concepts outlined in this paper.

[Wea03] addressed this by including elements of the argument that address failures of other software components which could lead to the specific software failure mode. Further leveraging off the ‘constraint’ approach described above, the framework described in this paper would also consider the mechanisms that provide this non-interference as ‘constraints’. For example, such ‘constraints’ should address non-interference of both intended and unintended coupling paths between software components.

Defining ‘constraints’ for intended coupling paths to show that these do not lead to a violation of the initiating constraint will usually involve addressing all intended coupling paths such as control and data flows, intentionally shared resources, etc. Each coupling path then inherently implies a potential software failure mode that requires treatment by an absence or detection/handling mechanism, and thus a ‘constraint’.

Defining ‘constraints’ for unintended coupling paths to show that these also do not lead to a violation of the initiating constraint will usually involve addressing all feasible spatial and temporal coupling paths. For example ‘constraints’ can be defined that use containment and/or mediation mechanisms for spatial interference paths. Such ‘constraints’ might include such mechanisms as the application of protected modes, virtual machines, memory management units, data wrappers, cache management, and software instruction run time evaluation. ‘Constraints’ for mediation mechanisms of temporal interference paths might include execution time monitors, and real time software scheduling mechanisms (earliest deadline first, rate monotonic, cyclic executive with interrupts, etc).

Ultimately each of these containment and or mediation mechanisms can in turn be dealt with as a ‘constraint’ under this framework, for assuring those mechanisms. In many cases, it will be most convenient to identifying logical pieces of software to group together as ‘constraints’ for the purposes of providing a collective absence mechanism (i.e. collective behaviours are acceptable with respect to safety), with detection and handling mechanisms provided for those software failure modes that are distinctly resolvable for treatment at the software level, or at higher levels of abstraction, such as at the LRU or system level.

3 Claims Safety Assurance Level

This paper proposes a framework for assuring ‘constraints’ based around the concept of a Claims Safety Assurance Level

(CSAL). The CSAL provides direct quantification of the extent to which the ‘constraint’ (refer to Section 2.4) is assured based on a taxonomy of factors that might transpire to violate the constraint. These factors include intended and unintended behaviours, the degree to which the behaviours are systematically accounted for, the certainty in sources of faults that might violate the constraint, etc. The degree to which these sources of violations are considered underpins the definition or category of assurance. The CSAL levels are presented in Table 1 (over page), along with their definition and relationship to applicable claims.

The reader of this paper is encouraged to examine the content of Table 1 before reading the following sub-sections, which provide further elaboration on the motivation for the context and structure of Table 1.

3.1 Systematically accounting for the intended and unintended behaviours

The core idea behind claims assurance is providing a quantification of the degree to which both the intended and unintended behaviours of the ‘constraint’ are accounted for with respect to safety. The extent to which there is systematised coverage of potential sources of violations is a key determiner in the level of assurance provided. The degree to which assurance is achieved is distinguished by the amount of uncertainty remaining that could lead to violation of the constraint. Potential sources of violations of the ‘constraint’ might include faults with the ‘constraint’ that would lead to its intended behaviours not being satisfied, circumstances that would lead to the ‘constraint’ being invalid, or other behaviours of the software or system that might interfere with the ‘constraint’.

The uncertainty differences between the CSAL levels are expressed as follows:

- the remaining uncertainty would unlikely lead to a violation of the constraint under any circumstances (CSAL 3)
- the remaining uncertainty would only lead to a violation of the constraint under unexpected circumstances (CSAL 2)
- the remaining uncertainty could lead to a violation of the constraint, but this would not be expected under normal operating conditions that would exercise the constraint (CSAL 1)

Therefore, the substantiation of claims with respect to the ‘constraint’, and ultimately the provision of evidence (covered later in this paper) are structured to bound the uncertainty that the ‘constraint’ could be violated. Refer to [ReM10] for a discussion on how layers of reliable mechanisms contribute to bounding uncertainty. For CSALs, the same concepts apply, but rather than layers of absence or detection and handling mechanisms for ASALs, instead there are layers of examination of potential source of confirmation or violation of the constraint.

Claims Safety Assurance Level (CSAL)	Category	Definition	Guiding principles for the substantiation of claims and provision of evidence with respect to satisfaction of attributes of software lifecycle products
			Claims are to address suitability of:
CSAL 4 (not used)	Absolute Assurance	<u>Intended</u> and <u>unintended</u> behaviours of the absence or detection and handling constraint are absolutely assured with respect to safety, such that there is <u>no uncertainty</u> in behaviour	Not practicable (or affordable) to make this argument compelling – Near Absolute Assurance provides sufficient control of the uncertainty
CSAL 3	Near Absolute Assurance	All reasonably practical and effective steps have been taken to <u>systematically account</u> for the <u>intended</u> and <u>unintended</u> behaviours of the absence or detection and handling constraint with respect to safety, such that the remaining uncertainty would unlikely lead to a violation of the constraint under any circumstances	<ul style="list-style-type: none"> Specified behaviours with respect to the constraint Refined behaviours with respect to the constraint Implementation behaviours with respect to the constraint Introduced or generated behaviours (e.g. from translation or code generation toolsets) that may violate the constraint Target Computer behaviours that may violate the constraint Conditions or behaviours external to the constraint, but internal to the system, that may violate the constraint Conditions or behaviours external to the system that may violate the constraint
CSAL 2	Nominal Assurance	Steps have been taken to <u>systematically account</u> for the <u>intended functional behaviours</u> of the absence or detection and handling constraint with respect to safety, such that the remaining uncertainty would only lead to a violation of the constraint under unexpected circumstances	<ul style="list-style-type: none"> Specified behaviours with respect to the constraint Refined behaviours with respect to constraint Implementation behaviours with respect to constraint Target Computer behaviours with respect to the constraint Conditions or behaviours external to the constraint, but internal to the system, that may violate the constraint
CSAL 1	Limited Assurance	Claims broadly account for the <u>intended functional behaviours</u> of the absence or detection and handling constraint with respect to safety, such that the remaining uncertainty could lead to a violation of the constraint, but this would not be expected under normal operating conditions that would exercise the constraint	<ul style="list-style-type: none"> Specified behaviours with respect to the constraint Implementation behaviours with respect to the constraint
CSAL 0	No Assurance	No evidence exists to assure the absence or detection and handling constraint with respect to safety	No evidence

Table 1: Claims Safety Assurance Level (CSAL)

3.2 Guiding principles for the substantiation of claims

Table 1 also specifies guiding principles for the substantiation of claims and provision of evidence with respect to the satisfaction of attributes of software lifecycle products (described further in Section 6). The intention of this column is to capture the top level relationships between the potential layers and sources of uncertainty and the associated conditions under which this uncertainty may be a potential source of confirmation or violation to the constraint. The guiding principles for the substantiation of the claims with respect to the constraint describe a set of claims that provides a level of certainty consistent with the qualitative CSAL level definition.

Subsections 3.2.1 through 3.2.7 summarise each of these claim topics.

3.2.1 Specified behaviours with respect to constraint

At the highest level, a set of behaviours must be specified for the ‘constraint’ that captures the intended function of the ‘constraint’ in the system, taking into account whether the ‘constraint’ is applied at the software level, LRU level or system level (see ASAL definition). These behaviours will be a function of the circumstances under which the ‘constraint’ is necessary to achieve safety (e.g. initiating circumstances), and the resultant behaviour to ensure a safe system response to the initiating circumstances (e.g. the effects). Internally, these behaviours should be consistent with each other, and the behaviours should resist violation from external factors.

3.2.2 Refined behaviours with respect to the constraint

As the specified behaviours at Section 3.2.1 are generally specified at a level of abstraction commensurate with their role at providing the ‘constraint’ at the necessary absence or detection/handling layer (because at that level it is possible to directly reason about system safety, and is unbiased by implementation specifics) it is generally necessary for these behaviours to be refined. The context to the refinement is the levels of abstraction at which important technology specific architectural properties and relationships can be accurately specified. Hence the concept of refined behaviours is introduced to account for the additional behavioural refinements and the suitability of presenting evidence at this level.

Like the specified behaviours, these refined behaviours will be a function of the circumstances under which the ‘constraint’ is necessary to achieve safety, and the resultant behaviour to ensure a safe response. However the context of the refined behaviours is the level of abstraction at which they are expressed, and no longer the ‘constraint’ layer level. For example, software architectural behaviours and their relationship to the ‘constraint’ are usually best reasoned about as refined behaviours. Refined behaviours should be logically

equivalent to the specified behaviours, be consistent with each other, and resist violation from external factors.

3.2.3 Implementation behaviours with respect to the constraint

As the behaviours specified at both the top ‘constraint’ level and refined levels are usually unable to address the specifics of the implementing language (e.g. source code) and the development environment, the implementation behaviours must be accounted for with respect to the ‘constraint’.

Like the refined behaviours, the implementation behaviours will be a function of the circumstances under which the ‘constraint’ is necessary to achieve safety, and the resultant behaviour to ensure a safe response. However the context of the implementation behaviours is the implementation language and the chosen design. For example, software language properties, including language constructs, vulnerabilities, and their relationship to the ‘constraint’ are usually best reasoned about as sources of violation to implementation behaviours. Implementation behaviours should be logically equivalent to the specified and refined behaviours, be consistent with each other, and resist violation from external factors.

3.2.4 Introduced or generated behaviours (e.g. from translation or code generation toolsets) that may violate the constraint

As a translation is inevitably required to produce executable object code from the source code, it is necessary to account for any additional behaviours introduced by the translation and whether these additional behaviours might violate the constraint. For example the behaviours of the run time machine of various language implementations are one source of potential violations that would require reasoning about with respect to the constraint. Introduced or generated behaviours should be logically equivalent to the specified, refined and implementation behaviours, be consistent with each other, and resist violation from external factors.

3.2.5 Target Computer behaviours that may violate the constraint

The unavoidable execution of the implementation (i.e. executable object code) on the target computer exposes the implementation (source code) level behaviours of the ‘constraint’ to a potential set of violators caused by the target computer. For example, the target computer might have initialisation properties, memory management or arithmetic handling behaviours that are incompatible with the intended behaviour of the ‘constraint’. The result might be the introduction of unintended implementation behaviours caused by the target computer, or target computer behaviours that are invalid with respect to the constraint. Target computer behaviours should be logically equivalent to the specified, refined and implementation behaviours, be consistent with each other, and resist violation from external factors.

3.2.6 Conditions or behaviours external to the constraint, but internal to the system, that may violate the constraint

The consideration of conditions or behaviours external to the constraint, but internal to the system, is intended to ensure that all credible sources of violation of the constraint external to the system are considered, such that the remaining uncertainty would only lead to a violation of the constraint under unexpected circumstances. The notion here is to eliminate the factors that might violate the context of any of the previous sets of the claims that would be caused by incompatibility between behaviours of the system components. For example, interrelated constraints that have unacceptable interference might be a source of violation of the ‘constraint’.

3.2.7 Conditions or behaviours external to the system that may violate the constraint

The consideration of conditions or behaviours external to the system is intended to ensure that all credible sources of violation of the constraint external to the system are considered, such that the remaining uncertainty would unlikely lead to a violation of the constraint under any circumstances. The notion here is to eliminate the factors that might violate the context of any of the previous sets of the claims. For example, the system operating environment (including failure environment) might be a source of violation of the ‘constraint’.

4 Relationship between ASALs and CSALs

The ASAL concept, discussed in [ReM10], uses layers of absence or detection/handling mechanisms to provide assurance that systematic faults do not lead to unacceptable failure conditions. The ASAL quantifies the number of systematic faults that are necessary for the failure condition of a given severity to be realised.

Each ‘constraint’ as it is defined in this paper will be associated with a specific layer of absence and detection/handling mechanisms in the context of the system or software architecture. Therefore, the degree of claims assurance, as expressed by the CSAL is related to the role of the constraint in the architecture, as expressed by the ASAL. Implicitly, CSAL is also related to the severity of failures associated with the system through the ASAL definition.

Table 2 describes the relationship between ASAL and CSALs. The basic principle behind the definition is that the CSAL is commensurate with the ASAL, noting that the ASAL is already defined in terms of the severity of failures of the system. Therefore, the stronger the architectural necessity for the system to resolve systematic faults, the stronger the motivation for claims assurance and evidence. While one might argue that claims assurance might also be used to provide additional strength for one layer of mechanism to mitigate the need for one or more requisite layers, this is not the intent of this approach. The architectural benefits of resolving faults at differing layers of abstraction and the impact of this on bounding uncertainty are an important facet of this framework, which should not be overridden by the properties of claims assurance.

ASAL	1 st Absence/Detection and Handling Mechanism		2 nd Detection/Handling Mechanism		3 rd Detection/Handling Mechanism		Additional Detection/Handling Mechanisms	
							Potentially Interfere ¹	Can't Interfere ²
ASAL3	Software Level	CSAL3	Partitioned Software Level [#] or LRU Level [*]	CSAL3	LRU Level [*] or System Level	CSAL3	CSAL2 ^{\$}	CSAL0
ASAL2	Software Level	CSAL2	Partitioned Software Level [#] or LRU Level or System Level	CSAL2	Not Required		CSAL2 ^{\$}	CSAL0
ASAL1	Software Level OR LRU Level OR System Level	CSAL1	Not Required				CSAL1	CSAL0
1 Potentially interfere with subsequent detection and handling 2 Can't Interfere with subsequent detection and handling # must be independent of the initiating failure and the 1st Absence / Detection and Handling mechanism (i.e. through a partitioning mechanism) * must be independent of the preceding detection/handling mechanism \$ additional mechanisms behaviour must be assured to reason that it won't interfere with the main mechanisms								

Table 2: ASAL to CSAL relationships

Recognising that some systems might include additional layers of absence or detection/handling mechanisms (over and beyond the requisite layers); Table 2 also defines the CSAL associated with additional layers. The key factor in specifying the CSAL for addition layers is the extent to which the additional layer might potentially interfere with the required layers.

Careful consideration is required when assigning layers of detection/handling mechanism as either the primary mechanisms, or the additional mechanisms. Depending on the layer's role in the architectural hierarchy of fault detection/handling, some mechanisms might be more suitable defined as primary layers (and subject to non-reduced claims assurance) rather than additional layers due to their potential for interference.

5 Linking Claims Assurance to Evidence

Section 3.2 introduced the guiding principles for the substantiation of claims. Specifically, the following were described:

- Specified behaviours with respect to the constraint
- Refined behaviours with respect to the constraint
- Implementation behaviours with respect to the constraint
- Introduced or generated behaviours (e.g. from translation or code generation toolsets) that may violate the constraint
- Target Computer behaviours that may violate the constraint
- Conditions or behaviours external to the constraint, but internal to the system, that may violate the constraint
- Conditions or behaviours external to the system that may violate the constraint

Each of these claim sets, in conjunction with the CSAL benchmark for intended behaviours, unintended behaviours and uncertainty of the 'constraint', sets the expectations for the provision of evidence with respect to the 'constraint'.

This section further examines how the requirements for the provision of evidence with respect to the 'constraint' might be established in this framework based on these guiding principles.

5.1 Associating Attributes of Software Lifecycle Products with Claims

If the typical software lifecycle products are examined (noting that at this point it doesn't matter if the lifecycle model is waterfall, spiral, etc. because holistically the individual lifecycle products are similar), it is evident that the typical lifecycle products include:

- requirements
- refined and design requirements (perhaps at various levels)

- source code (in one or more languages)
- executable object code / binary code
- verification results of executable object code with respect to:
 - source code
 - refined and design requirements
 - requirements
- validation results of:
 - executable object code
 - source code
 - refined and design requirements
 - requirements

Each of these products is generally self standing in its physical definition. For example requirements are usually described in a Software Requirements Specification (SRS) or similar artefact, refined requirements are often captured in subordinate SRS parts, or in the Software Design Description (SDD), Interface Design Document (IDD), etc.

Therefore, it is possible to specify attributes of each lifecycle product (i.e. a set of outcomes associated with the lifecycle product) that have direct relevance to the claims proposed in Section 3.2. Each of these attributes will also have a general alignment (and commensurate fidelity) to specific evidence types, which can be exploited in providing evidence assurance.

This paper proposes that attributes based on outcomes/results of software lifecycle products (rather than the techniques or methods that produced the results) can be defined as a basis of an assurance framework.

6 Attributes of Software Lifecycle Products

Annex A lists each of the necessary attributes, details the impact of the attribute not being satisfied, and specifies the relationship to the CSAL. The following paragraphs elaborate on the key principles and reasoning used in establishing the attributes and expressing their relationship to the CSAL.

6.1.1 Coverage of properties relevant to assuring the constraint

Each of the attributes was determined by ensuring that each potential source of violation with respect to specified requirements, refined requirements, implemented requirements, target computer, executable object code, etc, in the context of the specific lifecycle product, had a set of attributes that provides coverage of:

- requirements validity
- requirements satisfaction
- requirements traceability

The observant reader will note that many of the attributes share a striking resemblance to the objectives of RTCA/DO-178B. There is some truth in this observation, as many of the attributes have been specified consistently with established DO-178B terminology, however the key differences are as follows:

- in this framework they are referred to as attributes of software lifecycle products – which differs in interpretation from DO-178B objectives;
- each of the attributes in this framework is with respect to the ‘constraint’ being considered – in DO-178B the objectives relates to the entirety of the software;
- each of the attributes is organised in a set with respect to a software lifecycle product (i.e. with respect to evidence) – in DO-178B the objectives are organised around software lifecycle phases and integral processes;
- each of the attributes is focussed on behaviours of the software with respect to the constraint – DO-178B has objectives related entirely to process, such as the planning objectives; and
- additional attributes have been developed to address behavioural interferences (or non-interference as should be the goal) between constraints.

6.1.2 Discriminating tolerability

For each attribute to CSAL relationship expressed at Annex A, a measure of the tolerability of not satisfying the attribute is presented. Three levels of tolerability are used: Intolerable, Constrained and Tolerable. These levels are elaborated further in Section 7.

The tolerability of each attribute with respect to the CSAL is inferred by:

- application of the guiding principles specified in Table 1 to each criteria,
- the impact of not satisfying the attribute, as listed at Annex A, and
- consideration of accounting for intended behaviours, unintended behaviours and uncertainty in behaviours.

The expression of tolerability can be broadly partitioned into two categories, those that are specified in a binary fashion (e.g. Intolerable – Tolerable) and those that are specified in a graduated fashion (Intolerable – Constrained – Tolerable). The binary specification is used when assurance of an attribute is clearly either achieved or not, and that any graduation in assurance has little meaning with respect to the constraint. Attributes related to requirements traceability are a good example of a binary attribute, as there is little purpose to establishing traceability to products for which no further properties are going to be established (for lower levels of assurance). For attributes where the graduation is less explicit, the graduated specification is used, permitting the developer to express arguments about the ‘tolerability of

limitations’ with respect of the assurance of the specific attribute.

6.2 The inherent argument in claims and attributes

Inherently Annex A provides a template (pattern) software safety case for each ‘constraint’ based on the extent to which potential violations of the ‘constraint’ could be tolerated. When coupled with the tolerability of limitations approach described in Section 7, this framework provides explicitness in the top and bottom level claims of a software safety case, along with the inherent relationships of how these claims combine.

The advantage of this approach is that there is inherent consistency in software safety cases without unduly limiting or constraining the software products. It also ensure the emphasis is on the software products, and their evidence, without burdening the developer with the difficulty of architecting holistically unique software safety cases for each development.

7 Tolerability of Limitations

For each attribute to CSAL relationship in Annex A, a measure of the tolerability of limitations to not satisfying the attribute is presented. Tolerability is expressed as either Intolerable, Constrained or Tolerable, which are described by the ESAL concept.

[Wea03] describes two properties between related claims and evidence types:

- Relevance (directness and coverage)
- Trustworthiness

At an evidence level, relevance (directness and coverage) can be argued distinctly from trustworthiness. This is because, relevance is to do with the strength of the result of a technique or methods with respect to an associated attribute (and ultimately the claim being made) of the software lifecycle product, whereas trustworthiness is the extent to which the results of the evidence are correct. This distinction is reflected in the ESAL concept.

7.1 ESAL concept

This paper proposes a framework that includes the concept of an Evidence Safety Assurance Level (ESAL). The ESAL provides a direct quantification of the tolerability of limitations to assuring the applicable attribute of the software lifecycle product. The ESAL serves two functions. The first is to set benchmarks for the importance of specific attributes in assuring the specific ‘constraint’, as disclosed at Annex A. The second is to provide qualitative argument prescriptions (i.e. benchmarks for argument construction based on evidence claims) for the:

- relevance of evidence (and the combination of methods or techniques used to present evidence) with respect to the

attribute of the software lifecycle product in the context of the constraint,

- trustworthiness of the evidence (i.e. to what extent can the results of the evidence be tolerated to be incorrect?), and

- the outcome of the evidence (i.e. what the evidence actually shows?) to ensure that the presence of counter evidence is appropriately treated.

Three ESALS are proposed as presented in Table 3.

Tolerability of Limitations to Assuring Attribute	Relevance of Evidence	Trustworthiness of Evidence	Results of Evidence
Intolerable (ESAL3) – <i>limitations in evidence would be intolerable</i>	<u>No</u> limitations to the collective relevance of the method or methods' with respect to the attribute Limitations of each method or technique are systematically identified and treated by the application of complementary methods and techniques.	<u>No</u> limitations to the evidence's trustworthiness with respect to the attribute. Limitations of the trustworthiness of evidence are systematically identified and treated by the application of appropriate competencies, reviews and inspections, and independence.	The results of the method or methods provides evidence of satisfying the attribute AND there is <u>no</u> counter evidence or potential source (uncertainty) of counter evidence to satisfying the attribute
Constrained (ESAL2) – <i>limitations in evidence would be tolerable provided those limitations are constrained with respect to relevance, trustworthiness and results</i>	<u>Constrained</u> limitations to the method/s relevance with respect to the attribute Limitations of each method or technique are systematically identified and treated where practicable by the application of complementary methods and techniques. Non-treatment of a limitation should not introduce uncertainty grossly disproportionate to the limitation such that it would likely lead to a violation of the constraint	<u>Constrained</u> limitations to the evidence's trustworthiness with respect to the attribute. Limitations of the trustworthiness of evidence are systematically identified and where practicable treated by the application of appropriate competencies, reviews and inspections, and independence. Non-treatment of a limitation should not introduce uncertainty grossly disproportionate to the limitation such that it would like lead to a violation of the constraint	Results of the method or methods provides evidence of satisfying the attribute AND counter evidence to satisfying the attribute is <u>limited</u> such that it would not likely lead to violation of the constraint Uncertainty is constrained such that counter evidence is unlikely.
Tolerable (ESAL1) – <i>limitations in evidence would be tolerable</i>	Notable limitations to the method or method's relevance with respect to the attribute. Limitations of each method or technique may not be systematically identified and treated where practicable by the application of complementary methods and techniques.	<u>Notable</u> limitations to the evidence's trustworthiness with respect to the attribute.	Results of the method or methods may provide evidence of non-satisfaction of the attribute and/or violation of the constraint OR counter evidence indicates possible violation of the constraint OR uncertainty may be substantial

Table 3: ESAL Definitions

7.2 Relevance of Evidence

[Wea03] states that relevance (including both directness and coverage) is the “extent to which an item of evidence directly fulfils or entails the requirement for evidence.” Thus the relevance of the evidence (and the combination of methods or techniques used to present the evidence) with respect to the attribute of the software lifecycle product in the context of the constraint, is based on identifying and treating any limitations to the directness and coverage of evidence with respect to the attribute.

As almost all methods and techniques have limitations to the directness and extensiveness of the evidence produced by the method or technique, the framework is explicit in identifying and addressing these limitations. These limitations exist because almost all methods and techniques are defined based on a model of the problem they are intended to solve, and almost invariably, this model has limitations. An example of this is the application of formal methods to proving behaviours about software. Formal methods are very powerful at showing the correctness and internal consistency of a formally defined model, but to make the models manageable, associated behaviours (e.g. target computer

behaviours) are almost always simplified, or even left out. For this reason, formal models are often used in a way which is complementary to testing on the target computer. Likewise there are limitations to testing, such as the impost to exhaustively test all combinations of input and output data, or states for problems that suffer state explosion, and thus complementary approaches (such as formal methods and static code analysis) are necessary to overcome the limitations of testing.

Therefore the ESAL proposes an approach based on being explicit in the treatment of the limitations of each method or technique. This is achieved by ensuring the assurance framework systematically identifies and treats the limitation by the application of complementary methods and techniques. Depending on the tolerability of limitations in the evidence, the extent to which these limitations are treated varies. The framework defines three general tolerability categories (ESAL 1 through 3), which are based on the extent to which:

- the limitations of each method or technique are systematically identified and treated where practicable by the application of complementary methods and techniques; and
- non-treatment of a limitation should not introduce uncertainty disproportionate to the limitation such that it would likely lead to a violation of the constraint.

The higher the ESAL, the less any limitations are tolerable. The key measure that separates the tolerability is the extent to which a limitation might introduce uncertainty in the applicability of the results.

7.3 Trustworthiness of Evidence

[Wea03] states that trustworthiness is the “perceived ability to rely on the character, ability, strength or truth of the

evidence.” Thus the trustworthiness of evidence is characterised by the extent to which the results of the evidence be tolerated to be incorrect.

Unfortunately, compared with the relevance of evidence and the results of evidence, the trustworthiness of evidence is highly subjective and derivative of human involvement in the production of evidence. Therefore it is much more difficult to develop an approach that parallels the approach for ‘relevance of evidence’ and that reasons about the limitations of human involvement in developing evidence, in reviews and inspections, and the impact of independence being systematically identified and treated. This is because the limitations might vary significantly depending on the specific people involved throughout – something that is exceptionally difficult to use as any basis of comparison with benchmarks, even within competency frameworks.

Therefore, trustworthiness of evidence may be an aspect of the framework that benefits from an increased level of prescription over other parts of the framework.

Tables 4 and 5 presents an example approach as to how the regulator might set benchmarks for measures of trustworthiness. This approach has been derived from an analysis of evidence trends from real world systems, such as those described in [ReM10]. The approach is to intended to set benchmarks that take into account the variability of human involvement and thus avoid the need to systematically model the resultant limitations of human involvement, which are exceptionally hard to model. Conceptually, this approach is not different to what current assurance standards prescribe.

Trustworthiness	Developer Competency (Minimum)	Reviews and Inspections (Minimum)			Mechanistic Independence	Conceptual Independence
		Approach	Competency	Independence		
ESAL3 – Intolerable	Expert	Systematic Inspection OR Criteria Review%	Expert	Organisational OR Intellectual*	None OR Applied (Expert, Organisational)*	None OR Applied (Expert, Organisational)*
ESAL 2 - Constrained	Practitioner	Criteria Review OR Adhoc Review#	Expert OR Practitioner#	Peer	None OR Applied (Expert, Intellectual)#	None OR Applied (Expert, Intellectual)#
ESAL 1 – Tolerable	Supervised Practitioner	Adhoc Review	Practitioner	None	None	None
ESAL 0 – No Assurance	No more than Supervised Practitioner	None	N/A	None	None	None

Table 4: Trustworthiness

% - Conceptual Independence de-obligates the requirement for the review and inspection to be a Systematic Inspection (which inherently contains conceptual independence)

* - Organisational Independence of Mechanistic Independence or Conceptual Independence de-obligates the requirement for the review and inspection to have Organisational Independence (as organisational independence is achieved mechanistically or conceptually).

- Intellectual Independence of Mechanistic Independence or Conceptual Independence de-obligates the requirement for the review and inspection to have Intellectual Independence (as intellectual independence is achieved mechanistically or conceptually).

Developer Competency – Expert, Practitioner, Supervised Practitioner

Reviews and Inspections – Systematic Inspection, Criteria Review, Adhoc Review

Competency – Expert, Practitioner, Supervised Practitioner

Independence – Organisational Independence, Intellectual Independence, Peer Independence, None

Mechanistic Independence – Applied, None

Conceptual Independence – Applied, None

Note – organisational independence assumes intellectual independence

No independent approach (review and inspection, mechanistic, or conceptual) is ever applied by a lesser competency.

Table 5: Trustworthiness table notes and definitions

7.4 Results of Evidence

In addition to the relevance and trustworthiness of the evidence, the final property of evidence is the results of the evidence itself. The results of the evidence are important for several reasons, as follows:

- evidence may provide positive evidence of the behaviour of the software being appropriate with respect to the constraint and the safety of the system,
- evidence may provide direct counter evidence of a behaviour of the software that would violate the constraint with respect to safety; or
- evidence may disclose uncertainty based counter evidence (bounded or unbounded) which may raise questions with respect to the relevance and trustworthiness arguments.

Incorporating the results of evidence into the framework also avoids a common misconception that an absence of evidence infers evidence of absence of faults in a system.

Two types of counter evidence have been identified above – direct counter evidence or uncertainty based counter evidence. For systems with severe failure modes, uncertainty based counter evidence is equally as limiting as direct counter evidence, as either are not positive evidence of appropriate behaviours. For systems with less severe failure modes, reasoning about uncertainty (through tolerability of limitations in attributes being satisfied) is built into the framework. Therefore, the focus changes to ensuring the uncertainty would not likely lead to a presence of a violation to within some level of confidence.

8 Relationship to the Assurance Deficit

The ‘tolerability of limitations’ approach proposed in this paper has similarities to the Assurance Deficit concept also being proposed by the University of York, albeit developed independently. The ‘tolerability of limitations’ approach has several key advantages approach over the Assurance Deficit approach. Namely the ‘tolerability of limitations’ approach

addresses several key limitations of the Assurance Deficit concept (as it is currently described). These are as follows:

- ‘tolerability of limitations’ sets benchmarks for where an assurance deficit would be tolerable or intolerable – and is explicit in the rationale behind the tolerability;
- ‘tolerability of limitations’ provides fidelity of assurance claims and attributes of software products at a level that is sufficiently low to provide a clear taxonomy of software evidence; and
- ‘tolerability of limitations’ clearly distinguishes between attributes with binary tolerability constraints (predominantly traceability, configuration consistency, and development of requirements for behaviours) and those where there is greater potential for justified tolerability in satisfaction.

As the ‘tolerability of limitations’ and Assurance Deficit concepts are being developed independently, further examination of the strengths and weaknesses of these approaches is an ongoing part of this research. The authors note that the Assurance Deficit approach is intended to be completely general (i.e. independent of domain or application), whereas this framework proposed in this paper has the benefit of focusing more narrowly on avionics systems with well-defined architectural approaches, which gives the basis for the ASAL foundation for this work.

9 Application to software with limited design disclosure

One important property of an assurance framework is that it must be cognisant that the context to its application may not always be for a new development. Frequently, an existing software system may be examined for employment in a new context or application. Therefore, an additional property of an assurance framework is seamless application to these circumstances.

Unlike assurance frameworks that apply to the whole piece of software (e.g. DO-178B), the framework described in this paper targets those behaviours that relate to the specific

architectural ‘constraints’ and the mechanisms in the software that might assist with assuring provision of the ‘constraints’ behaviours under specified circumstances. Therefore, a major advantage of this assurance framework is that complete design disclosure is probably not required in some instances. Instead, targeted questions can be posed to the developer (who has not agreed to relinquish the intellectual property rights, or has federal government arms restrictions in the case of military developments) to build a case against the attributes of this framework. These targeted questions will almost always relate to fault detection and handling, rather than capability, which improves the circumstances in the military aviation context.

For agencies that elect to retrospectively generate safety evidence, such as through analysis of source code, or reverse engineering of binary code, this framework is also advantageous. It permits those important behaviours for safety to be examined as primary focus, rather than requiring comprehensive re-engineering of a totality of evidence. The framework will help these agencies determine when the uncertainty becomes sufficiently bounded, and thus provides of means of determining when and if it is possible to bound the uncertainty, and if these limitations would be tolerable.

10 Summary

This paper has extended earlier research (refer to [ReM10]) that targets limitations with the current standards’ frameworks with respect to assuring against systematic faults and failures. [ReM10] proposed an assurance framework that provides a direct measure of the extent of a system’s fault tolerance against systematic faults and failures. Central to the assurance framework was the concept of an Architectural Safety Assurance Level (ASAL) which was a product measure of the number of systematic failures a system is resilient against in a given context.

Furthering the earlier research, this paper proposes a Claims Safety Assurance Level (CSAL) and Evidence Safety Assurance Level (ESAL) concept that is compatible with the ASAL concept. The core idea behind claims assurance is to ensure that any assurance levels used for articulating claims assurance in the context of the ASAL have a specific product safety focus (i.e. each and every assurance level has a product meaning, not just a top-down or bottom up process interpretation). For evidence assurance, the core idea is to provide a framework that is explicit in a product sense of the ‘tolerability of limitations’ in satisfying the objectives

articulated in the framework. The ‘tolerability of limitations’ is intended to be a product behavioural measure of the ‘tolerability’ of either known product shortfalls/limitations, or limitations/shortfalls in the provision of suitable evidence.

Further evaluation of the concepts and approaches described by this paper and [ReM10] are being undertaken as part of this body of research.

11 References

The following documents, papers and publications are referenced throughout this paper. A number of these documents are not available in the public domain for propriety or confidentiality reasons. Readers wishing to seek further information should direct their queries to the author of this paper, or the relevant standards body.

- [DO178B] RTCA Inc., “RTCA/DO-178B: Software Considerations in Airborne Systems and Equipment Certification”, Washington D.C.: RTCA Inc., 1992.
- [JTM07] D. Jackson, M. Thomas, L. Millet, Editors, “Software for Dependable Systems: Sufficient Evidence?”, Committee of Certifiably Dependable Software Systems, National Research Council, National Academy of Sciences, USA, 2007.
- [McD01] J.A. McDermid, “Software Safety: Where’s the Evidence?”, Department of Computer Science, University of York, 2001.
- [McK06] J. McDermid, T. Kelly, “Software in Safety Critical Systems: Achievement and Prediction”, Nuclear Future, Volume 03, No. 03, 2006.
- [NTS06] National Transportation Safety Board, “Safety Report on the Treatment of Safety-Critical Systems in Transport Airplanes”, Safety Report NTSB/SR-06/02, Washington, D.C., USA, 2006.
- [ReM10] D.W. Reinhardt, J.A. McDermid, “Assuring Against Systematic Faults Using Architecture and Fault Tolerance in Aviation Systems”, Department of Computer Science, University of York, to be presented (paper accepted) at the Improving Systems and Software Engineering Conference 23 – 26 August 2010.
- [Wea03] R.A. Weaver, “The Safety of Software – Constructing and Assuring Arguments”, Department of Computer Science, University of York, 2003.

ANNEX A - ATTRIBUTES OF SOFTWARE LIFECYCLE PRODUCTS

Specified Constraint Level Requirements

(specified at the level of the architectural constraint, and at the level at which requirements are allocated to software)

Attribute	Impact of NOT Satisfying	CSAL3	CSAL2	CSAL1	CSAL0
Developed and Defined	Specified Constraint Level Requirements for constraint {constraint} do not exist – therefore there is no basis for the relevant behaviour existing in the software	Intolerable	Intolerable	Intolerable	Tolerable
Valid – Accurate / Consistent / Complete	Specified Constraint Level Requirements for constraint {constraint} exists but the specification of the constraint is invalid – therefore, there is potential for other lifecycle products or translations to refine or implement the behaviour erroneously	Intolerable	Constrained	Constrained	Tolerable
Unambiguous / Precise	Specified Constraint Level Requirements for constraint {constraint} exists but the specification of the behaviour is ambiguous and/or imprecise – therefore, there is potential for other lifecycle products or translations to misinterpret the constraint	Intolerable	Constrained	Constrained	Tolerable
Satisfiable / Verifiable	Specified Constraint Level Requirements for constraint {constraint} exists but the behaviour cannot be verified (analytically or empirically) – therefore verification evidence for the constraint will not exist or be irrelevant	Intolerable	Constrained	Constrained	Tolerable
Compatible with Target Computer	Specified Constraint Level Requirements for the constraint {constraint} exists, but the constraint is not compatible with the target computer – therefore, the specification of the constraint is unsatisfiable and additional behaviours that violate the constraint may be initiated from the target computer	Intolerable	Constrained	Constrained	Tolerable
Traceable to Lower Level Requirements (Refined Requirements or Low Level Requirements)	Specified Constraint Level Requirements for the constraint {constraint} exists, but there is no traceability to a lower level refinement of the behaviour – therefore, there is no traceable basis for the refinement of the relevant Specified Constraint Level Requirements existing in the software design or code	Intolerable	Intolerable	Tolerable	Tolerable
Inadequacies in Specified Constraint Level Requirements are identified and resolved	Compliance, robustness, traceability and verification may identify inadequacies in Specified Constraint Level Requirements – therefore the behaviours implemented by the software may not be consistent with the constraint	Intolerable	Intolerable	Constrained	Tolerable

Refined Abstract Level Requirements (optional in totality)

(refined from Specified Constraint Level Requirements, while still being abstract from Low Level Requirements, and used to provide a means making claims from evidence that cannot be produced directly against Specified Constraint Level Requirements or Low Level Requirements)

Attribute	Impact of NOT Satisfying	CSAL3	CSAL2	CSAL1	CSAL0
Developed and Defined	Refined Abstract Level Requirement for constraint {constraint} does not exist – therefore there is no basis for the relevant behaviour existing in the software	Intolerable	Intolerable	Tolerable	Tolerable
Traceable to Higher Level Requirements (Specified Constraint Level Requirements or high level Refined Abstract Level Requirements)	Refined Abstract Level Requirements exist, but there is no traceability to the higher level Requirements associated with the constraint {constraint} – therefore, the behaviours specified by this Refined Abstract Level Requirements may not be consistent with the constraint	Intolerable	Intolerable	Tolerable	Tolerable
Valid – Accurate, Consistent, Complete	Refined Abstract Level Requirement for constraint {constraint} exists but the specification of the constraint is invalid – therefore, there is potential for other lifecycle products or translations to refine or implement the behaviour erroneously	Intolerable	Constrained	Tolerable	Tolerable
Satisfiable / Verifiable	Refined Abstract Level Requirement for constraint {constraint} exists but the behaviour cannot be verified (analytically or empirically) – therefore verification evidence for the constraint will not exist or be invalid	Intolerable	Constrained	Tolerable	Tolerable
Unambiguous / Precise	Refined Abstract Level Requirements for constraint {constraint} exists but the specification of the behaviour is ambiguous and/or imprecise – therefore, there is potential for other lifecycle products or translations to misinterpret the constraint	Intolerable	Constrained	Tolerable	Tolerable
Compatible with Target Computer	Refined Abstract Level Requirement for the constraint {constraint} exists, but the constraint is not compatible with the target computer – therefore, the specification of the constraint is unsatisfiable and additional behaviours that violate the constraint may be initiated from the target computer	Intolerable	Constrained	Tolerable	Tolerable
Traceable to Lower Level Requirements (lower level Refined Abstract Level Requirements or Low Level Requirements)	Refined Abstract Level Requirement for the constraint {constraint} exists, but there is no traceability to a lower level refinement of the behaviour – therefore, there is no basis for the refinement of the relevant Abstract Level Requirement existing in the software design	Intolerable	Intolerable	Tolerable	Tolerable
Compliant with Higher Level Requirements (Specified Constraint Level Requirements or high level Refined Abstract Level Requirements)	Refined Abstract Level Requirements exist for the constraint {constraint}, but this abstraction of requirements are not compliant with the Higher Level Requirements – therefore, the behaviours specified by the Refined Abstract Level Requirements are not consistent with the constraint	Intolerable	Constrained	Tolerable	Tolerable

Attribute	Impact of NOT Satisfying	CSAL3	CSAL2	CSAL1	CSAL0
Robust with Higher Level Requirements (Specified Constraint Level Requirements or high level Refined Abstract Level Requirements)	Refined Abstract Level Requirements exist for the constraint {constraint}, but this abstraction of requirements is not robust with the Higher Level Requirements – therefore, the behaviours specified by the Refined Abstract Level Requirements may not be resilient to sources of faults that might violate the constraint	Intolerable	Constrained	Tolerable	Tolerable
Inadequacies in Refined Abstract Level Requirements are identified and resolved	Compliance, robustness, traceability and verification may identify inadequacies in Refined Abstract Level Requirements – therefore the behaviours implemented by the software may not be consistent with the constraint	Intolerable	Constrained	Tolerable	Tolerable

Low Level Requirements

(specified at a level that no additional refinement is required to develop source code and that all behaviours of the source code are described by the Low Level Requirements)

Attribute	Impact of NOT Satisfying	CSAL3	CSAL2	CSAL1	CSAL0
Developed and Defined	Low Level Requirements for constraint {constraint} do not exist – therefore there is no basis for relevant behaviour existing in software	Intolerable	Intolerable	Tolerable	Tolerable
Traceable to Higher Level Requirements (Specified Constraint Level or Refined Abstract Level)	Low Level Requirements exist, but there is no traceability to Higher Level Requirements associated with {constraint} – therefore, behaviours specified by Low Level Requirements may not be consistent with the constraint	Intolerable	Intolerable	Tolerable	Tolerable
Valid – Accurate / Consistent / Complete	Low Level Requirements for constraint {constraint} exist but the specification of the constraint is invalid – therefore, there is potential for other lifecycle products or translations to refine or implement the behaviour erroneously	Intolerable	Constrained	Tolerable	Tolerable
Satisfiable / Verifiable	Low Level Requirements for constraint {constraint} exist but the behaviour cannot be verified (analytically or empirically) – therefore verification evidence for the refinement of the constraint will not exist or be invalid	Intolerable	Constrained	Tolerable	Tolerable
Unambiguous / Precise	Low Level Requirements for constraint {constraint} exist but the specification of the behaviour is ambiguous or imprecise – therefore, there is potential for other lifecycle products or translations to misinterpret the constraint	Intolerable	Constrained	Tolerable	Tolerable
Compatible with Target Computer	Low Level Requirements for the constraint {constraint} exist, but the constraint is not compatible with the target computer – therefore, therefore, the specification of the constraint is unsatisfiable and additional behaviours that violate the constraint may be initiated from the target computer	Intolerable	Constrained	Tolerable	Tolerable
Traceable to Source Code	Low Level Requirements for the constraint {constraint} exist, but there is no traceability to an implementation level refinement of the behaviour – therefore, there is no basis for the refinement of the Low Level Requirements existing in the software source code	Intolerable	Intolerable	Tolerable	Tolerable
Compliant with Higher Level Requirements (Specific Constraint Level Requirements or Refined Abstract Level Requirements)	Low Level Requirement exist for the constraint {constraint}, but the low level requirements are not compliant with the Higher Level Requirements – therefore, the behaviours specified by the Low Level Requirements are not consistent with the constraint	Intolerable	Constrained	Tolerable	Tolerable
Robust with Higher Level Requirements (Specific Constraint Level Requirements or Refined Abstract Level Requirements)	Low Level Requirements exist for the constraint {constraint}, but are not robust with the Higher Level Requirements – therefore, therefore, the behaviours specified by the Low Level Requirements may not be resilient to sources of faults that might violate the constraint	Intolerable	Constrained	Tolerable	Tolerable
Inadequacies in Low Level Requirements are identified and resolved	Compliance, robustness, traceability and verification may identify inadequacies in Low Level Requirements – therefore the behaviours implemented by the software may not be consistent with the constraint	Intolerable	Constrained	Tolerable	Tolerable

Software Source Code

(compiler or assembler readable code)

Attribute	Impact of NOT Satisfying	CSAL3	CSAL2	CSAL1	CSAL0
Developed and Defined	Source Code for constraint {constraint} does not exist – therefore no basis for the relevant behaviour existing in the software	Intolerable	Intolerable	Intolerable	Tolerable
Traceable to Low Level Requirements	Source Codes exists, but there is no traceability to the Low Level Requirements associated with the constraint {constraint} – therefore, the behaviours implemented by the Source Code may not be consistent with the constraint	Intolerable	Intolerable	Tolerable	Tolerable
Valid – Accurate / Consistent / Complete	Source Code for the constraint {constraint} exist but the implementation of the constraint is incorrect – therefore, the executable object code will contain an erroneous behaviour	Intolerable	Intolerable	Constrained	Tolerable
Satisfiable / Verifiable	Source Code for constraint {constraint} exists but the behaviour cannot be verified (analytically or empirically) – therefore verification evidence for the implementation of the constraint will not exist or be invalid	Intolerable	Intolerable	Constrained	Tolerable
Unambiguous / Precise	Source Code for constraint {constraint} exists but the implementation of the behaviour is ambiguous or imprecise – therefore, there is potential for implementation of other software components or compiler/linker translations to misinterpret the constraint and introduce vulnerabilities that violate the constraint	Intolerable	Intolerable	Constrained	Tolerable
Compatible with Target Computer	Source Code for the constraint {constraint} exists, but the constraint is not compatible with the target computer – therefore, the implementation of the constraint is invalid and additional behaviours that violate the constraint may be initiated from the target computer	Intolerable	Constrained	Constrained	Tolerable
Traceable to Executable Object Code	Source Code for the constraint {constraint} exists, but there is no traceability to a target computer level refinement of the behaviour in object code – therefore, there is no basis for the complete refinement of the relevant Source Code existing in the Executable Object Code	Intolerable	Constrained	Tolerable	Tolerable
Compliant with Low Level Requirements	Source Code exists for the constraint {constraint}, but the Source Code is not compliant with the Low Level Requirements – therefore, the behaviours implemented by the Source Code are not consistent with the constraint	Intolerable	Constrained	Tolerable	Tolerable
Robust with Low Level Requirements	Source Code exist for the constraint {constraint}, but are not robust with the Higher Level Requirements – therefore, the behaviours implemented by the software may not be resilient to sources of faults that might violate the constraint	Intolerable	Constrained	Constrained	Tolerable
Inadequacies in Source Code are identified and resolved	Compliance, robustness, traceability and verification may identify inadequacies in Source Code – therefore the behaviours implemented by the software may not be consistent with the constraint	Intolerable	Intolerable	Constrained	Tolerable

Executable Object Code

(target computer readable binary code)

Attribute	Impact of NOT Satisfying	CSAL3	CSAL2	CSAL1	CSAL0
Produced	Executable Object Code for constraint {constraint} does not exist – therefore no basis for the refinement of the relevant behaviours of the constraint existing in the software	Intolerable	Intolerable	Intolerable	Tolerable
Integrated onto Target Computer	Executable Object Code for constraint {constraint} exists, but it didn't integrate/load onto the target computer – therefore, there is no basis for the refinement of the relevant behaviours of the constraint existing in the software on the target computer	Intolerable	Intolerable	Intolerable	Tolerable
Compatible with Target Computer	Executable Object Code for the constraint {constraint} exists, but the constraint is not compatible with the target computer – therefore, the implementation of the constraint is invalid and additional behaviours that violate the constraint may be initiated from the target computer	Intolerable	Intolerable	Constrained	Tolerable
Traceable to Source Code	Executable Object Code exists, but there is no traceability to the Source Code associated with the constraint {constraint} – therefore, the behaviours implemented by the Executable Object Code may not be consistent with the constraint	Intolerable	Constrained	Tolerable	Tolerable
Compliant with Specified Constraint Level Requirements	Executable Object Code exists for the constraint {constraint}, but the Object Code is not compliant with the Specified Constraint Level Requirements – therefore, the behaviours implemented by the software are not consistent with the constraint	Intolerable	Intolerable	Constrained	Tolerable
Robust with Specified Constraint Level Requirements	Executable Object Code exists for the constraint {constraint}, but the Object Code is not robust with the Specified Constraint Level Requirements – therefore, the behaviours implemented by the software may not be resilient to sources of faults that might violate the constraint	Intolerable	Intolerable	Constrained	Tolerable
Verification Coverage of Specified Constraint Level Requirements	Executable Object Code exists for the constraint {constraint}, but the Object Code is not verified against all applicable Specified Constraint Level Requirements – therefore, the behaviours implemented by the software may not be consistent with the constraint	Intolerable	Intolerable	Intolerable	Tolerable
Compliant with Refined Abstract Level Requirements	Executable Object Code exists for the constraint {constraint}, but the Object Code is not compliant with the Refined Abstract Level Requirements – therefore, the behaviours implemented by the software are not consistent with the constraint	Intolerable	Constrained	Tolerable	Tolerable
Robust with Refined Abstract Level Requirements	Executable Object Code exists for the constraint {constraint}, but the Object Code is not robust with the Refined Abstract Level Requirements – therefore, the behaviours implemented by the software may not be resilient to sources of faults that might violate the constraint	Intolerable	Constrained	Tolerable	Tolerable

Attribute	Impact of NOT Satisfying	CSAL3	CSAL2	CSAL1	CSAL0
Verification Coverage of Refined Abstract Level Requirements	Executable Object Code exists for the constraint {constraint}, but the Object Code is not verified against all applicable Refined Abstract Level Requirements – therefore, the behaviours implemented by the software may not be consistent with the constraint	Intolerable	Constrained	Tolerable	Tolerable
Compliant with Low Level Requirements	Executable Object Code exists for the constraint {constraint}, but the Object Code is not compliant with the Low Level Requirements – therefore, the behaviours implemented by the software are not consistent with the constraint	Intolerable	Constrained	Tolerable	Tolerable
Robust with Low Level Requirements	Executable Object Code exists for the constraint {constraint}, but the Object Code is not robust with the Low Level Requirements – therefore, the behaviours implemented by the software may not be resilient to sources of faults that might violate the constraint	Intolerable	Constrained	Tolerable	Tolerable
Verification Coverage of Low Level Requirements	Executable Object Code exists for the constraint {constraint}, but the Object Code is not verified against all applicable Low Level Requirements – therefore, the behaviours implemented by the software may not be consistent with the constraint	Intolerable	Constrained	Tolerable	Tolerable
Compliant with Source Code	Executable Object Code exists for the constraint {constraint}, but the Object Code is not compliant with the Source Code – therefore, the behaviours implemented by the software are not consistent with the constraint	Intolerable	Constrained	Tolerable	Tolerable
Robust with Source Code	Executable Object Code exists for the constraint {constraint}, but the Object Code is not robust with the Source Code – therefore, the behaviours implemented by the software may not be resilient to sources of faults that might violate the constraint	Intolerable	Constrained	Tolerable	Tolerable
Verification Coverage of Source Code Structure	Source Code exists for the constraint {constraint}, but the verification has not exercised all behaviours of the Source Code relevant to the constraint – therefore, there may be additional behaviours of the source code which violate the constraint	Intolerable	Constrained	Tolerable	Tolerable
Verification Coverage of Executable Object Code Structure	Executable Object Code exists for the constraint {constraint}, but the verification has not exercised all behaviours of the Executable Object Code relevant to the constraint – therefore, there may be additional behaviours of the Executable Object Code which violate the constraint	Intolerable	Constrained	Tolerable	Tolerable
Inadequacies in Executable Object Code are identified and resolved	Compliance, robustness, traceability and verification may identify inadequacies in Executable Object Code – therefore the behaviours implemented by the software may not be consistent or complete with the constraint	Intolerable	Constrained	Tolerable	Tolerable