Certification Authorities Software Team (CAST)

Position Paper
CAST-24

RELIANCE ON DEVELOPMENT ASSURANCE ALONE WHEN PERFORMING A COMPLEX AND FULL-TIME CRITICAL FUNCTION

COMPLETED March 2006
(Rev 2)

NOTE: This position paper has been coordinated among the software specialists of certification authorities from North America, South America, and Europe. However, it does not constitute official policy or guidance from any of the authorities. This document is provided for educational and informational purposes only and should be discussed with the appropriate certification authority when considering for actual projects.
Reliance on Development Assurance Alone when Performing a Complex and Full-Time Critical Function

1.0 Introduction

1.1 Background

Today, many aircraft-level functions are implemented using diverse and redundant system architectures and capabilities as mitigation techniques. These approaches contribute to an acceptable level of safety at the aircraft level. Experience shows the value of these safety assurance techniques.

New technologies being proposed for use in aircraft systems present even greater challenges and more complexity, and can introduce new sources of development errors and, thus, undesirable or unintended effects. At the same time, because of the increased complexity and integration of aircraft functions, it is generally not practical (and may not even be feasible) to develop a finite test suite for complex airborne systems which can conclusively demonstrate the absence of development errors in these systems.

Since the potential for the existence of these errors is generally not quantifiable and suitable numerical methods for characterizing them are not available, other qualitative and architectural means are used to establish that airborne systems can satisfy safety objectives to an acceptable level.

- Development “process” assurance can establish a level of confidence that the system development has been accomplished in a sufficiently disciplined, rigorous manner to limit the likelihood of development errors that could impact aircraft safety. This includes reliance on development assurance methods such as SAE ARP 4754 and 4761 for system safety assessment and system development assurance, RTCA DO-178B for software development assurance, RTCA DO-254 for complex electronic hardware design assurance, and other industry standards and other internal company standards for airborne systems, software and hardware.

- Architectural means can limit the consequences of development errors and system component/hardware failures, and their likelihood to impact aircraft safety.

Note: The concepts in this paper may also be applied to complex electronic hardware but are focused primarily on software.

1.2 Paper Purpose

It is recognized today that in designing aircraft systems, manufacturers should prevent any single failure that leads to a catastrophic failure condition (JAR/FAR 25.1309 (extremely improbable); AMJ/AC 25.1309-1A). The fail-safe concept
and techniques are discussed in the AMJ/AC 25.1309-1A to support this approach. [Single failures leading to a catastrophic event are prevented (occurrence extremely improbable) by the FAR/JAR, as well as multiple failures (25.1309 (d)(2))]

However, when the failure is caused by a development error in the system, particularly in software or complex electronic hardware, the guidance materials are not clear on the applicability of fail-safe concept and techniques. Thus, the applicant and system designers need to consider the potential effect of such errors in the aircraft-level safety assessment, in order to ensure that their proposed system design and implementation of complex, safety-related systems can be demonstrated to have achieved an acceptable level of safety. The purpose of this paper is to highlight that development assurance alone is not necessarily sufficient to establish an acceptable level of safety for complex and full-time critical functions implemented in software or complex hardware. The paper presents rationale for the use of mitigation means in the system development to prevent either software or complex electronic hardware development errors from becoming a common point of failure that could lead to an unacceptable safety event (accident or incident).

Current regulations require a set of safety techniques to mitigate the development error’s risk to an acceptable level (e.g., occurrence of catastrophic event extremely improbable, occurrence of major event improbable, etc.). As the regulations and policy are not sufficiently explicit, this paper explains how the fail-safe concept and design techniques can be interpreted when addressing software-related and complex electronic hardware-related development errors. Finally, it should be noted that this paper is informational only as it does not provide clear criteria to help the engineering judgement on this matter. As such, no clear solutions are suggested. Moreover, this paper does not promote any particular concept.

1.3 Paper Overview

- General safety techniques commonly applied by the international aviation community are presented in Section 2. This section summarizes state-of-the-practice knowledge of aircraft systems and equipment suppliers, aircraft manufacturers, and certification authorities.
- Section 3 discusses specific regulatory materials supporting the safety approaches currently used. These are used as a way to illustrate how the general safety techniques are used to satisfy the regulations and how they are related to software and complex electronic hardware assurance.
- In Section 4, some best practices are discussed to show how these concerns have been addressed.
• Specific guidelines for certification authorities are presented in Section 5.
• Section 6 provides a brief conclusion to this paper.
• Section 7 summarizes the references used throughout the paper.
• Appendix A discusses diversity as one acceptable mitigation technique against a common point of failure resulting from common mode analysis.

1.5 Definitions

• **Diversity**: Design diversity is a defense against “common mode” or “common cause” development errors in safety critical systems. It is a system design concept that attempts to reduce the possibility that the failure stemming from a common development error in one functional failure path will result in another functional failure path. This is accomplished by designing a functional failure path to be sufficiently different to minimize the likelihood that the error will manifest itself in another functional failure path implementing the system function and, then, allow an unacceptable failure event.

• **Development error**: A mistake in the development process resulting from incorrect method or incorrect application of methods or knowledge (ED-79/ARP4754 [7]).

• **Fault**: A manifestation of an error in software. A fault, if it occurs, may cause a failure. (ED12B/DO-178B) [Note that a fault can also occur as a result of a hardware error or hardware failure.]

• **Failure**: The inability of a system or system component to perform a required function within specified limits. A failure may be produced when a fault is encountered. (ED12B/DO-178B) [Note that another definition of a hardware failure is when the hardware “breaks” or wears out]

• **Full-time critical function**: Function whose failure can lead directly to a Catastrophic event if not mitigated in a safe and timely manner.

• **Functional Failure Path**: The specific set of interdependent items that could cause particular anomalous behavior in the system that implements the function.

• **Redundant**: Multiple, independent means incorporated in a system to accomplish a given function (see ED-79/ARP4754 for redundant architecture principles).

2.0 Safety Concepts

2.1 Fail-Safe Design Concept
The fail-safe design concept and techniques (such as AMJ/AC 25.1309-1A) are used to ensure that, if any single element in a system or sub-system fails in any single flight, such single failure should not prevent continued safe flight and landing, or significantly reduce the capability of the airplane or the ability of the

*NOTE*: This position paper has been coordinated among the software specialists of certification authorities from North America, South America, and Europe. However, it does not constitute official policy or guidance from any of the authorities. This document is provided for educational and informational purposes only and should be discussed with the appropriate certification authority when considering for actual projects.
crew to cope with the resulting failure condition. [Fail-safe design should also be used to prevent or mitigate the effects of multiple failures and undetected failures (25.1309 (d) (2)).] Thus, the application of the fail-safe design concept enables minimal occurrence and/or effects of failures, and provides protection against catastrophic failure conditions.

In a fail-safe system of a federated aircraft architecture, the failure of a single element should be detected, and the system should provide fault tolerant response (e.g., reversion to a “healthy channel”, switch to a backup function, switch to a degraded mode if provided) that ensures continued functionality of the system and its robustness (or notification to the flight crew of degraded system performance or functionality). In order to achieve a fail-safe software-based system, independent, redundant software functions should exist within the system. Redundant software may enable continued functionality by switching to a healthy or backup channel (i.e., computer and channel unaffected by the failure) or may allow continued limited functionality in a degraded mode when a single component fails. Redundancy also contributes to avoiding the failure of the complete system, when a single failure happens and sometimes with multiple failures.

The fail-safe design concept is required by civil aviation regulations. It has implications on the design architecture choices and implies certain architectural techniques used for risk mitigation. Traditionally, the application of the fail-safe design concept results in a fault tolerant system that is based on fault detection capabilities and on the level of independence of the redundant channels. Safety techniques, including fault tolerance and fault detection, are defined in the next section.

2.2 Four Basic Safety Techniques

Four basic safety techniques are recognized by technical experts as significantly contributing to the overall enhancement of safety [1]. The recognized basic techniques are fault tolerance (fault accommodation\(^1\)), fault detection, fault removal, and fault avoidance. These four techniques are related and generally require a combination to be effective. Each of the four techniques is discussed below.

**Fault tolerance** is a fail-safe safety technique applied to system/software design to enhance its robustness in the presence of faults and to allow the system to continue to function in the presence of faults. Examples of fault tolerance practices are defensive programming, fault isolation or containment

\(^1\) Fault tolerance may also be referred to as fault accommodation.

**NOTE:** This position paper has been coordinated among the software specialists of certification authorities from North America, South America, and Europe. However, it does not constitute official policy or guidance from any of the authorities. This document is provided for educational and informational purposes only and should be discussed with the appropriate certification authority when considering for actual projects.
(independence), redundancy, other fail-safe design concepts and techniques, hardware interrupt, dissimilarity, and recovery blocks.

ED-12B/DO-178B [5] and ED-80/DO-254 [6] are development assurance processes that require robustness features be implemented to mitigate abnormal and unexpected data, and to provide failure and error detection.

ED-79/ARP4754 [7] considers different design architectures and protection mechanisms, and provides guidance on determining appropriate system development assurance levels, based on different system architectures and the use of independence, monitoring, dissimilarity and other design features in a proposed architecture.

The fault tolerance principle should be applied for the implementation of critical and complex systems for which it is necessary to mitigate the effect of system failures, hardware failures and development errors to ensure continued safe flight in the presence of failures, faults and the manifestation of errors.

**Fault detection** is a safety technique used to detect faults and trigger an appropriate response. Examples of fault detection include use of built-in-test, comparators, system monitors, safety monitors, and loop back tests. Examples of appropriate responses include switching to a fault-free channel (parallel or backup), isolating the effects of the failed component, ignoring the output of the faulty channel, or switching to reduced (degraded) system mode.

Fault detection is closely linked to fault tolerance. Actually, a system is fault tolerant, when it can detect errors and trigger appropriate system behavior. Fault detection is effective when detection mechanisms are sufficiently independent and dissimilar from the system being monitored; thus, independent and dissimilar implementation is a way to demonstrate detection mechanism efficiency. Independence between the different channels is often used to justify fault tolerance and detection capabilities and safety margins. Fault detection by comparing the outputs of identical channels is also frequently used, but the main flaw of this architecture is that it may not mitigate common development errors.

**Fault removal** is a safety technique used during design to remove faults. Examples of fault removal include error detection and correction functions, built-in test, verification and validation through inspections, reviews, tests, model-checking, static analysis, etc.

Fault removal relies on development assurance process- methods. ED-12B/DO-178B is development assurance-oriented, and strives to achieve the removal of errors during development by using well-defined verification reviews, analyses and testing. Validation and verification enable detection and removal of errors in
the specification, design and coding, and implementation of the software, including integration with the target computer and hardware. Fault removal encompasses a set of techniques that remove the errors that can contribute to faults during the software development in order to decrease the number of errors when the product is used in service. ED-80/DO-254 uses a similar design assurance approach for complex electronic hardware.

**Fault avoidance** is a safety technique used during development to avoid errors that can contribute to system faults. Examples of fault avoidance techniques include selecting an appropriate language subset, defensive programming, minimizing and partitioning safety critical code, minimizing errors in design, and use of an appropriate life cycle methods and techniques.

ED-12B/DO-178B recommends use of requirements, design and coding standards, which is another way to contribute to fault avoidance. Partitioning, protection, safety monitoring, robustness of design and test, and minimization of safety critical parts are recommended by ED-12B/DO-178B as ways to implement fault avoidance.

### 2.3 Summary of Safety Concepts

When safety critical and complex systems are developed, use of the four basic safety techniques above are considered necessary to the relevance of the overall safety rationale. The overall safety of the system can be considered to have reached the highest level, when the four basic techniques are applied in a system. A design without one of the four basic techniques is missing an important part of the safety rationale. Therefore, safety-related systems should be designed applying these four techniques and should apply the fail-safe design concept and techniques.

### 3.0 Regulatory Guidance

The regulations, guidance materials, and industry standards (e.g., FAR/JAR XX.1309, AMJ/AC 25.1309-1A, NPA25F281, ED-79/ARP4754, ED-12B/DO-178B, and ED-80/DO-254) support the fail-safe design concept, the four basic safety techniques, and the use of mitigation means, especially for components subject to fatigue and wear. For development and associated components not subject to fatigue and wear (like software or complex electronic hardware), no absolute techniques apply. It is difficult to find guidance in the following AC/AMJ 25.1309 fail-safe design techniques, as summarized below, when a full-time and complex critical function is implemented using software or complex electronic hardware:

1) designed integrity and quality,
2) redundancy or back-up systems

**NOTE**: This position paper has been coordinated among the software specialists of certification authorities from North America, South America, and Europe. However, it does not constitute official policy or guidance from any of the authorities. This document is provided for educational and informational purposes only and should be discussed with the appropriate certification authority when considering for actual projects.
3) isolation,
4) proven reliability,
5) failure warning or indication
6) flight crew procedures
7) checkability,
8) Designed Failure Effect Limits,
9) designed failure path,
10) margins or factors of safety
11) error-tolerance.
AC/AMJ 25.1309-1A states that the use of only one of these principles or techniques is seldom adequate. The combination of two or more of these is usually needed to provide a fail-safe design. When software or complex electronic hardware components are used alone to implement a complex and full-time safety-related aircraft function, applying these fail-safe techniques relies on a good development assurance process.

4.0 Some Best Practices

4.1 Flight Control Systems
All certified flight control system architectures are designed considering the potential effect of systematic software or complex electronic hardware error and then mitigating them. These architectural forms are assessed by certification authorities, considering the resulting risk mitigation level in accordance with safety techniques recommended by the regulation (safety techniques: fault tolerance, fault detection, fault removal and fault avoidance). There is no quantitative means to assess the acceptable level of safety regarding the mitigation for systematic error; therefore, the acceptance is based upon engineering judgment and common understanding of best practices between certification authorities and applicants.

Below is a summary of different examples for mitigation of system failures that were found acceptable in actual flight control systems.

4.1.1 First example for flight control system
In order to provide fault tolerance for implementing complex and critical requirements, a diverse architectural form was selected at the system level for the flight control system.

The overall system was composed of two computer types: one was a simplified version of the other (simplified control laws). Each computer was divided into monitor and control channels (i.e., a dual monitored channel architecture for fault detection).
Different software implementations were proposed to ensure independence from common development errors of the redundant channels (i.e., N-version programming between dual monitored channels and between the two computer types used in the system) and different hardware were chosen for the two computer types.

Therefore, in addition to development assurance, diverse and redundant architectures were used to provide robust system behavior and to protect the aircraft from unknown development errors. The architecture used degraded modes and robustness to protect against failure. Distinct system and software implementation were proposed to mitigate some common development errors and were seen as an additional means of assurance to enhance safety (i.e., development assurance alone was not deemed sufficient by aircraft manufacturer).

4.1.2 Second example for flight control computers
In this second example, redundant and diverse architectures were proposed at system level for a flight control system in order to mitigate common development errors. Two different computer types were implemented within the system architecture. Diversity between the two computer types was applied both for the software and the hardware. Diverse software due to different compilers and hardware (processors) were used for the primary computers. So, in this case, the applicant relied strongly on the software specification and design requirement validation but did not fully trust the low-level of the software implementation (i.e., compiler translations, microprocessors embedded logic).

Hardware and software development errors were therefore mitigated to a certain level. So, development assurance alone was not deemed sufficient by applicant.

4.1.3 Comparison between first and second approaches
In the first approach, the flight control system architecture provided robustness features to mitigate software and hardware systematic failure risk (i.e., N-version programming, dual monitored channel architecture running on two distinct computer types within primary flight control computer, and different hardware and software between the two flight control computer types).

In the second approach, the flight control system architecture provided robustness features to mitigate systematic hardware and software failure (two distinct software implementations running on distinct computer types within overall system architecture,). Software systematic failures induced by compilation layer are mitigated in the primary flight control computer. Both approaches are different and the risk mitigation means do not prevent the same kind of development errors. Nevertheless, in both cases, diversity is proposed in addition
to development assurance, and mitigation techniques are used at both the computer level and at system architecture level.

Therefore, the mitigation techniques used by both applicants do not rely on development assurance alone.

4.2 Air Data System

The air data system is composed of four smart probes with embedded software for data acquisition and processing. To cope with a potential systematic failure of the software running in these probes, the applicant provides robustness with the smart probes by using two dissimilar software applications running on two dissimilar hardware boards.

4.3 Door Controllers

The aircraft door controller system is considered critical due to the potential of the door opening or slide activating in-flight. So, all the critical functionality implemented in level A software provides commands that are consolidated within a simple analog voter by a programmable component, whose development assurance level is also level A.

4.4 Summary of Examples

The growing complexity of systems emphasizes the need to mitigate the risk against systematic failures and common points of failure. Some mitigation techniques have been used in already certified systems implementing full-time critical and complex functions. In any case, development assurance alone was generally deemed insufficient to provide an acceptable level of safety.

For future systems, integrated modular avionics (IMA) systems are being proposed that will dramatically increase functional integration and complexity. This introduces a high potential for failure propagation due to the extensive use of generic (and common) modules. This could potentially increase the common failure modes within the IMA system. Therefore, there is a risk of reduction of the current level of safety provided in federated system architectures, as well as possible reduction of robustness against “common” development errors, if appropriate mitigation techniques are not implemented within these types of systems.

The assessment of risk mitigation could become a key issue for the future certification programs, and it is important to understand the significance of the topic for its contribution to maintaining an acceptable level of aircraft safety.

5.0 Guidelines for Assessing Mitigation Techniques

Aviation regulations and policy require the use of appropriate safety techniques to mitigate the risk of failures and unacceptable failure conditions for critical

NOTE: This position paper has been coordinated among the software specialists of certification authorities from North America, South America, and Europe. However, it does not constitute official policy or guidance from any of the authorities. This document is provided for educational and informational purposes only and should be discussed with the appropriate certification authority when considering for actual projects.
functions. Fail-safe design techniques are required. The safety techniques are applicable to the design and do not exclude software from the scope. Software can cause common mode errors and the following approach may help to assess the proposed fail-safe design techniques and architectures used to mitigate these risks. Development assurance is generally not sufficient by itself. Some fault-tolerance techniques should be applied (such as, diverse and redundant implementations or simple design) to mitigate single failures, combination failures and common points of failure in the system.

Risk should be addressed by the applicant in their aircraft systems’ design and architecture, and certification authorities should assess the risk mitigation scheme for acceptability, using engineering judgment. The subsections propose objectives and guidelines for certification authorities to consider during the assessment task.

5.1 Assessment objectives

1. Ensure that an analysis of common cause of failure is performed on the design (including software and complex electronic hardware).

2. For complex critical systems (associated with catastrophic and hazardous failure conditions), ensure that evidence is provided to show that an acceptable level of fault tolerance and fault detection techniques is achieved.

5.2 Assessment guidelines

1. Certification authorities should assess mitigation means for common causes of failure in the design (including software and complex electronic hardware). The systematic errors avoided or mitigated should be documented.

2. Certification authorities should assess all design techniques used and check application of fault detection and fault tolerance for complex and critical systems.

3. Certification authorities should realize that:
   - Use of architectural means are acceptable to achieve fault detection and fault tolerance.
   - If architectural means are not proposed, alternative means of compliance should be assessed for acceptability, and the findings documented. Alternative means of compliance should be reviewed against the failure modes, to confirm whether they are sufficient to mitigate the risks both at the system level and at the aircraft level.
4. The certification authorities should review and approve the system safety assessment (SSA). The risk mitigation level should be documented and justified in the SSA.

6.0 Conclusion

Development assurance alone is not necessarily sufficient to establish an acceptable level of safety for complex and full-time critical functions implemented in software or complex hardware. Some best practices currently provide protection against development errors, and also protect against common failure modes. In-service experience shows that traditional safety assessment techniques identify only a fraction of the failure modes that can occur in the actual operational environment. Therefore, mitigation techniques, in addition to development assurance, are typically required for system implementation of complex and full-time critical functions.

7.0 References


NOTE: This position paper has been coordinated among the software specialists of certification authorities from North America, South America, and Europe. However, it does not constitute official policy or guidance from any of the authorities. This document is provided for educational and informational purposes only and should be discussed with the appropriate certification authority when considering for actual projects.
Appendix A The Diversity Concept

This appendix on diversity highlights this concept as one possible, but not the only, approach to provide mitigation against the potential effects of system development errors.

Design diversity is a traditional defense against development faults in safety critical functions. Where such functions are implemented by two functional failure paths, it is a design concept that: (1) enhances the level of independence of the functional failure path (FFP), and (2) reduces the risk that the failure of one FFP stemming from development error causes the failure of the other FFP. A design implementation of two FFPs is diverse, when a failure needs both FFPs to behave erroneously to cause a catastrophic event. Diversity can include functional segregation, system redundancy, system back up and many others techniques.

Diversity is one means to help ensure that a system satisfies the objectives of fault tolerance and fault detection. Diversity can provide evidence that “common” software and hardware errors of system architectural components have been addressed by the developers and that the system will provide the necessary safety robustness and safety properties (fail-safe properties). This approach is compliant with the regulatory requirements that recommend a fail-safe design, and encourages the use of safety techniques like fault tolerance (fault accommodation), fault avoidance, fault detection, and fault removal. The extent of the usage of safety techniques should be based on "best practices" and the state-of-the-art known by industry and certification authorities.

4.1 Why use diversity?

The following extracts from scientific studies support the diversity concept.

[2] states: Diversity between redundant subsystem is, in various forms, a common design approach for improving system dependability. Diversity is a common design approach for protection against common-mode failures in redundant systems, mostly used in critical applications. It is hoped that if redundant channels are implemented in different ways (diverse “versions”), the risk of common design flaws causing common failures will be reduced. The growing adoption of software based systems, and the attendant doubts about the risk caused by development faults in the software, justify increased interest in diversity. Well-known examples of diversity in software are in the aerospace and railways industries, but some form of diversity is present in many software systems.

Although diversity improves reliability, the knowledge that diversity is present brings no quantifiable advantage during assessment.
[3] states: The known experiment with software diversity confirm indeed that fault-tolerant software employing diversity is “on average” more reliable that a single software version. Utilising design diversity when high reliability is required is, nevertheless problematic (“complexity”).

[4] states: Whilst there is clear evidence that these approaches can bring benefits when compared with unitary systems, these benefit can be difficult to quantify. At the very simplest level, where components can be replicated and their failures in operation can be assumed to be statically independent, we know that we can build arbitrarily reliable systems with arbitrarily unreliable components.

As can be concluded from the studies, diversity is generally recognized as a means to improve system integrity and system robustness, and is a mitigation technique used to reduce the common-mode failures. The diversity concept reduces the risk that a failure of a single item causes a catastrophic event.

Diversity is used to improve fault tolerance and fault detection; it also improves the system reliability, and thus, contributes to enhance the overall system safety. The diversity benefit is qualitative and remains difficult to quantify; nevertheless, the system robustness gain can be assessed concretely and system fault tolerance is enhanced.

4.2 What are the benefits of diversity?

[2] states: Diversity obviously improves dependability. Functional diversity is an effective way to pursue high reliability. What is not possible, however, is to claim that functional diversity is sufficient in itself to justify an assumption of independence in the version failures. It leaves the system assessor with the task of evaluating precisely how dependent the versions are before he/she can evaluate the reliability of the system. This is not easy, as we have seen in other contexts.

The benefit of diversity usage is generally the improvement of the system integrity and reliability. However, this can sometimes decrease the system availability. Thus, when using diversity as a mitigation technique, the applicant must demonstrate that the diverse system architecture still complies with availability specification requirements. Diversity can be considered as a safety enhancement, when used in addition to traditional quality process and when shown to add qualitative value.

4.3 Diversity disadvantage and alternative methods

There are some disadvantages associated with diversity usage, and they should not be ignored. This sub-section expands the discussion to address the
disadvantages of diversity and to highlight that diversity should not be considered as the only acceptable mitigation means.

The complexity added by implementing diversity increases the technical effort required during initial development and during in-service modifications. Therefore, before using diversity as a mitigation technique, simple design approaches should be considered.

Diversity is difficult to establish, and the error detection thresholds are often complex to determine. Therefore, applicants must address these difficulties in order to demonstrate added value from diversity usage. Applicants should show how diversity will be maintained during in-service modification without introducing errors (which could occur because of the complexity of a diverse design).

Diversity enhances the error detection capabilities (predictable failure modes and voting); however, when a system detects errors, the detection is performed with dedicated thresholds. If the thresholds are not correctly set and do not comply with the system availability requirements, diversity could introduce an unacceptable reduced availability.

The additional effort for initial development and in-service modification means that diversity is potentially more costly. Therefore, diversity usage should be carefully assessed for its added value in terms of safety.

4.4 Summary of diversity

Diversity should be understood in the broad meaning. It may include, but is not limited to, diverse programming (which may have minimal benefit alone and is not considered an acceptable approach on its own). Diverse programming is a way to increase the level of independence of redundant items and contributes to the final safety argument. When a system provides a level of diversity with dual monitored channels (control and monitor), with diverse programming it supports the regulatory intent recommending that a single failure should not affect all the redundant channels. Diversity is a mitigation technique that provides a level of independence between the redundant channels.

The diversity contribution to system integrity improvement is generally tangible, but is never quantifiable. The qualitative benefit assessment is mainly based on engineering judgment and experience.

Diversity is never complete, as there is always a remaining part of similarity in any diverse design. The diversity concept has limitations and should not be considered as the only acceptable mitigation technique.

NOTE: This position paper has been coordinated among the software specialists of certification authorities from North America, South America, and Europe. However, it does not constitute official policy or guidance from any of the authorities. This document is provided for educational and informational purposes only and should be discussed with the appropriate certification authority when considering for actual projects.
Diversity is more expensive, complex, and difficult to achieve. Therefore, added development cost and burden must be balanced with the aircraft loss potentially avoided, with life saved during operation, and with the final safety level reached.

When a system is complex (complex functions), the combination of possible remaining errors in the design is so important that development assurance is not sufficient alone to mitigate the risk. Diversity is one of the acceptable means used to decrease the risk to an acceptable level.