A cross-domain comparison of software development assurance standards

Emmanuel Ledinot[(1)], Jean-Marc Astruc[(2)], Jean-Paul Blanquart[(3)], Philippe Baufreton[(4)], Jean-Louis Boulanger[(5)], Hervé Delseny[(6)], Jean Gassino[(7)], Gérard Ladier[(8)], Michel Leeman[(9)], Joseph Machrouh[(10)], Philippe Quéré[(11)], Bertrand Ricque[(4)]

(1) Contact author, Dassault Aviation, emmanuel.ledinot@dassault-aviation.com ;
(2): Continental; (3): Astrium Satellites; (4) Sagem Défense Sécurité; (5): CERTIFER; (6): Airbus;
(7): Institut de Radioprotection et de Sûreté Nucléaire; (8): Aerospace Valley; (9): Valeo; (10): Thales; (11): Renault

Abstract:

This paper compares the influence of Development Assurance Levels (DALs) on the prescribed objectives, activities, methods and tools of six different software development assurance standards, indeed that of civil aviation, automotive, space, process automation, nuclear and railway.

Through an inventory of their respective requirements, we attempt to compare the software safety levels ensured by each standard for its lowest and highest DALs.

We first explain the rationale of the comparison, i.e on what basis we compare the securing effects of the various process-based or product-based requirements issued by the six software development assurance standards. Then we review the DAL-dependent variability of each standard and finally outline some tentative cross-domain equivalence classes or ranking.

Keywords: safety, software, DAL, SIL, ASIL, SSIL, processes, standards, cross-domain comparison.

1. Introduction

We present an analysis of the following software safety standards: aeronautics' DO-178/ED-12, automation's IEC 61508, automotive's ISO 26262, nuclear's IEC 60880, railway's EN 50128 and space's ECSS-Q-ST-80C, paying particular attention to the way Development Assurance Levels drive the increase of prescribed objectives, activities, methods or safety mechanisms to be implemented.

We then attempt to answer the following questions: are these standards comparable with respect to the software safety levels they claim to ensure? Does the DAL-dependent progressive construction of software safety rely on the same principles in the various industrial omnibet domains?

We first explain the rationale of the comparison, then we give, domain after domain, an overview of the DAL-dependent gradual construction of development rigor. We highlight their main commonalities and differences on supporting processes (also called integral processes), development processes and verification processes.

Regarding the question of cross-domain comparison of software DALs, we restrict ourselves to the lowest (respectively E, SIL0, ASIL A, class 3, SSIL0, 4) and the highest levels (A, SIL4, ASIL D, class 1, SSIL4, 1).

A forthcoming more comprehensive version of this work will provide an all DAL inclusive comparison, and attempt to define some cross-domain equivalences.

2. Comparison rationale

We first compare the probabilistic system safety levels to which the highest software development assurance levels are supposed to be compatible with. As explained in [Baufreton et al, 2010], all the standards dismissed the notion of probabilistic software failure, as well as any probabilistic quantification of DAL-dependent likelihood of residual software fault. However, all the standards implicitly state that the various software development assurance levels are compatible, or consistent, with corresponding quantified system safety levels.

These system safety levels are highly sensitive to the physics, to the severity and exposure issues of the different industrial domains. Because of these great differences, (see the companion paper "*Criticality categories across safety standards in different domains"* [Blanquart et al., 2012]), it may be the case that the highest safety objectives of the different software development assurances are indeed different, whatever means they define to meet these objectives. So it is worth comparing their respective safety objectives before listing and categorising their numerous requirements.

For each domain we point out which safety assurance requirements are DAL dependent, and which are not, grouping the requirements into three categories: applicable to supporting processes, development processes or verification processes.

When discussing the possible cross-domain equivalences or ranking between the highest software development assurance levels we will put in two different categories the standards that require external conformance assessment from that which do not. External assessment does not necessarily assume assessment by a regulation authority. Assessment by peer review within a company is also considered as "inner external" assessment and valued, though to a lesser extent than review by authority officials.

3. Aeronautics

3.1. Software safety objectives

DO-178/ED-12 defines five software criticality levels, also called Development Assurance Levels, from E to A upward. Consistently with CS 25.1309 and ARP 4761, DAL A aims at developing airborne safety critical software so that one can assume that system level catastrophic failure conditions directly or indirectly caused by software malfunctioning may occur at most once per billion of flight hours.

3.2. Rationale of gradual software safety assurance

DO-178/ED-12 is a process-based software development assurance standard. Through a DAL-dependent set of activities, quality objectives and development work products, it strives to ensure that all the system requirements allocated to a given piece of software are implemented in the executable code loaded in a defined equipment, and nothing else (no dead code, no unintended function).

This is the reason why DO-178/ED-12, like IEC 60880 or EN 50128, ensures a *context-dependent* property of software. There is no notion of intrinsically qualified software, i.e. independently of an upstream system and a downstream hardware equipment.

However and surprisingly enough, the context-dependent system safety requirements allocated to software are not isolated and handled with greater care in DO-178/ED-12.

And since its foundational principle is to limit normative guidance to quality objectives and process activities, in other words to refrain from putting any constraint on development and verification means, its DALs have no influence on methods and tools contrary to IEC 61508, IEC 60880, ISO 26262 and EN 50128.

It is still the case with its new C version, in spite of the fact that three mean related technical supplements now complement the core document for Object Oriented Technologies (OOT), Model Based Development and Verification (MBDV), and Formal Methods (FM).

These techniques, especially MBDV and FM, are not regarded as of superior efficiency so that they should be applied to the development of high criticality software. They are regarded as new development or verification means that may contribute to meet the quality assurance objectives of the core document, that were kept unchanged from version B to C.

On one side potential benefit regarding software safety is credited by the standard to these techniques, but on the other side they are suspected of error-prone tooling sophistication and of inspiring overconfidence in their benefits, especially formal verification w.r.t. testing.

In the end, *none* of these techniques are recommended, even formal methods for DAL A, to the opposite of EN 50128 and IEC 60880.

Likewise, and contrary to most of the other standards, DO-178/ED-12 does not contain a DAL-dependent list of software threats and associated forbidden programming traits or mandatory analyses. Some potentially dangerous programming constructs are highlighted, like interrupts or dynamic memory allocation. The standard mandates demonstration of safe usage of these features whenever used. *Applicants* forbid them, not the standard, as a mean to cope with the otherwise insuperable safety demonstration objectives.

Finally, DALs condition independence requirements regarding verification activities w.r.t. development activities.

3.3. DAL-dependent requirements on supporting processes

DALs have a significant influence on supporting processes i.e. planning, documentation, configuration management, relation to Authority etc.

Level E software is not subject to any development constraint, so it will not be discussed any further.

As defined in section 4 and table A-1 of the core document [ED12B/DO178B][1], planning is required nearly to the same extent for levels D to A. The requirements are exactly the same from C to A. Level D is relieved of defining the software life-cycle environment, the development standards, and the coordination of plan revision.

The purpose of planning is to define the software production means that will ensure compliance with the system requirements at DAL-specified confidence level. In these plans particular attention has to be paid to the development and verification means of multiple-version dissimilar software, deactivated code, user-modifiable code, and parameter data items.

The configuration management process is DAL-*independent* but the software lifecycle work products this unique process applies to are DAL-dependent. Two categories of Change Control, CC1 and CC2, are defined for items whose configuration is managed. CC2 is a lighter version of CC1, i.e. the configuration management activities required by CC2 are a subset of that of CC1.

The higher the DAL, the more work products are subject to configuration management, and the more often CC1 is required instead of CC2.

Software quality assurance is handled evenly for the three higher DALs: the plans and standards have to be defined and applied to the processes, the activity records have to be generated, the transition criteria monitored, and a process conformity review conducted.

Regarding the certification liaison process, the requirements are the same for all DALs.

[1] By the time of publication of this paper ED-12C/DO-178C is accepted by RTCA and EUROCAE but not yet published.

3.4. DAL-dependent requirements on the development process

Originally, DO-178/ED-12 required only the High Level Requirements and the executable object code integrated on target computer to be developed. This is the reason why these objectives are uniformly required for all the levels.

Then software became larger and programming evolved towards general purpose languages so that intermediate specification refinement steps or development artefacts were introduced between the HLRs and the executable object code: Low Level Requirements, software architecture, source code. Producing these three intermediate work products became also required, but only for levels C to A.

So the DALs have a great influence on the development process: light-weight process for E & D on one side, heavier step-wise refinement process for C, B, A on the other side (table A-2).

Resorting to model-based development, object oriented technologies or formal methods, with or without automatic code generation, has no influence on this [E,D], [C,B,A] split of the development processes. These three technologies are regarded as new means of producing the HLR, LLR, architecture or source code artefacts. Very few development assurance objectives were added in the Technical Supplements (TS) of version C. MBDV TS introduced three new objectives (MB8-10 in table MB.C-2) that indeed boil down to a sole one: whatever model is used for HLR, LLR, or software architecture, the elements of the model that will not be implemented in the piece of software must be marked as such.

This new requirement applies to levels D to A for HLR models, and for architecture models[2] as well.

As expected, it applies only from C upward to the LLR models.

3.5. DAL-dependent requirements on the verification process

DALs' influence is even greater on the verification process as it modulates:

- the work products that are to be verified (tables A-3 to A-6)
- the activities that are to be verified (table A-7)
- the independence of verification teams w.r.t. development teams (tables A-3 to A-7)

Consistently with the [E,D] vs. [C,B,A] split at development level, the same split underpins the A-3 to A-6 verification tables: verification of the HLRs against the allocated system requirements and verification of the executable object code against the HLRs are mandatory at all levels but E, whereas verification of the intermediate work products (LLR, architecture, source code) and verification of the executable object code against the LLRs are mandated only from C to A.

Verification of verification activities is still called verification of testing[3] in the core document whereas it applies also to verification by simulation in the MBDV supplement and to verification by proof techniques in the FM supplement.

Verification of verification cases, procedures and results on one side, verification of functional and structural coverage obtained *after* requirement-based verification on the other side, are the main verification of verification activities. At D level, only verification of HLR functional coverage is required (A-7.3). Verification of LLR functional coverage is required from C upward. The required structural coverage (A-7.5 to A-7.9) gradually increases from C (statement) to A (MC/DC).

Consistently with the way new development technologies were considered on the development process, very few new quality assurance objectives were added on the verification process by the three technical supplements:

- MBDV TS added verification of simulation cases, procedures and results wherever model simulation is used (3x3 new objectives),
- OOT TS added verification of local type consistency and robustness of dynamic memory allocation (OO.9-10 in table OO.A-7), with independency for levels A and B.
- FM TS, like MBDV TS, added verification of proof cases, procedures and results wherever proof techniques are applied (FM.8-9 in FM.A-3 to A-6). More significant are the new formalization correctness (FM-1x) and method appropriateness (FM-1y) objectives in tables FM.A-3 to A-6, and even more so the new structure coverage and property preservation objectives of table FM.A-7 (FM.5-8 and FM.9) [LE 2011]. Property preservation and method appropriateness apply to all the DALs. All the other new verification objectives are required only from C upward.

Regarding the rationale of DAL-dependent modulation of verification independency we have the following: independence of verification of HLR consistency, accuracy and compliance with system requirements are mandated for A *and B*.

It is also the case for algorithms' accuracy and compliance at all the refinement steps, from the HLRs to the executable object code.

Independent verification of instruction and decision coverage is also required for A *and B*.

Because A and B are very close, decomposition of a DAL A system into two independent B-items (sub-

[2] Though software architecture is not required at level D. But if done, and done by means of a model, then extra work is required on this non mandated artefact.

[3] In DO-178/ED-12 testing mandates execution on the target airborne equipment. Execution of the binary code on virtualized target hardware accounts for simulation.

system or equipment) is authorized in ARP 4754 (see [Blanquart et al., 2012]).

Independent verification of MC/DC coverage, and removal of additional executable code non traceable to source code are required exclusively for DAL A.

There is no independency requirement from C downward.

4. Automotive

4.1. Software safety objectives

As explained in [Blanquart et al., 2012], a distinctive feature of ISO 26262 w.r.t. the five other standards is its criticality allocation granularity.

In automotive, criticality may be allocated to the safety requirements. This fine grained criticality allocation policy at system level entails possibility of DAL mix at software level, i.e. possibility of DALs defined at module, class, or procedure level.

However, some directed influence constraints between the components of the DAL-mix must be guaranteed and duly justified ([ISO 26262] part 9, clauses 5 and 6), and like in aerospace nuclear or railway, the default approach in automotive is to develop a piece of software at the DAL which fits the most critical of its allocated safety requirements. It is especially the case when the properly oriented dependence condition[4], required for a DAL-mix can't be demonstrated.

Automotive most critical embedded software, ASIL D software, has to be consistent with a system safety level quantified by a probability rate of 10^{-8} per driving hour.

4.2. Rationale of gradual software safety assurance

Contrary to DO-178/ED-12 where process objectives and activities depend on DAL, ISO 26262 part 6 does not modulate the development assurance requirements according to ASIL.

ASIL-dependent variability applies to the methods and tools that are accepted, recommended, or highly recommended to meet the various objectives of the unique ASIL-independent development and verification processes.

In particular, the work products to be supplied for confirmation measures[5] are the same for ASIL A and ASIL D software. The ASILs modulate the level of formality recommended for requirement capture and software verification, they also drive the applicable coding standards and testing coverage criteria.

Last but not least, ASIL D also enforces a few product-based design requirements such as implementation of error detection (part 6 table 4), error handling (part 6 table 5, and table C.1 for calibration data error detection), and partitioning mechanisms (part 6 clause 7.4.1 and table 5). Partitioning is especially important as a means of substantiating absence of interference between ASIL-heterogeneous software elements.

4.3. ASIL-dependent requirements on supporting processes

They are addressed by ISO 26262 part 8 (8-7, 8-8, 8-10).

Planning includes:

- documentation management plan
- configuration and change management plans
- software integration and verification plans,
- unit and integration testing plans,
- testing plan,
- safety requirement verification plan,
- tool usage plans,
- component qualification plans.

Regarding documentation, there is no ASIL-dependent list of documents or work products to be supplied.

Configuration management (part 8 chapter 7) can be applied according to ISO/TS 16949, ISO 10007 and ISO 12207. There are specific, but ASIL-independent, requirements for configuration and calibration data.

4.4. ASIL-dependent requirements on the development process

For safety requirements specification, an ASIL-dependent appropriate combination of natural language, semi-formal, or formal notation is required (part 6 table 2). For ASIL C and D, a semi-formal notation is highly recommended. Contrary to railway, formal specification is never highly recommended, even for ASIL D. For the lower ASILs, informal notation can be sufficient to comply with the standard.

In ISO 26262, the DO-178/ED-12 notion of derived requirement, i.e. a requirement traceable to a design choice but not traceable to a system requirement, does not exist.

Software architectural design is peculiar in ISO 26262 as the standard applies to the whole piece of software, contrary to all of its other chapters that only apply to its safety-related aspects (part 6 chapter 7).

The notation and the architectural principles regarding interface, module size, hierarchical decomposition, control and data coupling, interrupt handing, scheduling etc. are ASIL dependent (tables 2 and 3).

ASIL D software architectural requirements must be verifiable by simulation of its dynamic part (part 6 table 6). Formal notation is recommended, whereas *semi*-formal notation is highly recommended. Software modelling and coding guidelines are also ASIL-dependent (part 6 table 1).

4.5. ASIL-dependent requirements on the verification process

The requirement verification methods are ASIL-dependent (part 8, table 2). From ASIL C upward, semi-formal verification of the requirements is highly

[4] A DAL X item may depend from a DAL Y item only if $X \leq Y$.

[5] Functional safety related quality assurance measures (review, audit, assessment).

recommended. Walk-through is enough for ASIL A, but inspection is highly recommended from ASIL B to D. Formal verification of requirements is never highly recommended.

Traceability is explicitly required along the whole hierarchical safety requirement structure and to the specification of verification (part 8, figure 2).

Formality in source code verification methods grows with ASIL, but formal verification of source code is never highly recommended.

Like in DO-178/ED-12, requirement based testing is highly recommended for all the levels, and detecting the unintended functions is a verification objective.

For unit testing, structural coverage is required. Statement coverage for ASIL A and B, branch coverage for ASIL B to D. MC/DC is highly recommended only for ASIL D (part 6 table 12).

For integration testing, structural coverage is highly recommended only for ASIL C and D (function or call coverage, part 6 table 15).

Fault injection tests, resource usage tests, and model-code back-to-back tests (when applicable) are highly recommended for ASIL D.

5. Industry Automation

By automation we understand the continuous process industries such as nuclear facilities (beside energy production), non nuclear energy, metals, cement, oil and gas and chemicals, the manufacturing industries with the exception of automotive and the batch production industries such as pharmaceuticals and food and beverage. These industries are relevant of IEC61511 for the continuous and batch processes, and of IEC62061 for manufacturing industries. Both standards are derivates of IEC61508 and, as they are not self supporting, refer to IEC61508.

These three standards address only the electric, electronic, programmable electronic systems under the concept of functional safety, that is systems distinct from the controlled equipment (plants, machines, and processing lines) contributing to risk reduction.

The standards are performance oriented. This means that, as for the other industries, the functions contributing to risk reduction are classified in 4 levels (SIL) according to the impact of a failure on safety. The requirements are thus increasingly stringent with the SIL number.

The central concept of these standards is to achieve the targeted safety integrity and performance by putting requirements in different fields encompassing:

- project management,
- quality assurance,
- hardware design,
- architecture et software development.

The standards assume that there are two types of failures. The failures that are introduced before the commissioning of the systems that are only systematic failures, and the failures occurring after system commissioning and that can be either systematic or random. The standard thus addresses:

- incorrect specifications of the system, hardware or software;
- omissions in the safety requirements specification;
- random hardware failure mechanisms;
- systematic hardware failure mechanisms;
- software errors;
- common cause failures;
- human error;
- environmental influences;

Part 3 (IEC 61508-3) is dedicated to software.

5.1. Software safety objectives

When software is involved in a safety related E/E/PE system, the standards put requirements on its robustness and its integrity concerning systematic failures. The safety objectives of a given safety related function are defined during the safety requirements specification phase in terms of safety functional and integrity requirements at the level of the safety related function itself. They are then refined in hardware and in software safety requirements.

Appendix B of part 2 acknowledges that exhaustive detection of systematic software failures introduced during development, as well as quantification of the efficiency of the software fault avoidance policy are intractable. It states that performance is more achieved by means of development environment, techniques and methods, rather than by quantification attempts. Objectives and recommended means are thus set for properties such as:

- Completeness with respect to the safety needs;
- Correctness with respect to the safety needs;
- Freedom from intrinsic specification faults, including freedom from ambiguity;
- Understandability of safety requirements;
- Freedom from adverse interference of non-safety functions with the safety needs;
- Capability of providing a basis for verification and validation.

However, and consistently with the aforementioned intractability statement of *prior* process efficiency quantification, the informative appendix D of Part 7 presents a statistical method to estimate *posterior* safety integrity levels, especially when importing software COTS.

5.2. Rationale of gradual construction of software safety

According to the targeted SIL level, IEC 61508 requires for each lifecycle phase, the selection and application of methods and techniques aiming to provide a suitable context to achieve the expected performance level.

The link between the expected emerging software properties and the applied techniques and methods is (qualitatively) described in the informative part 7, appendix C.

5.3. SIL-dependent requirements on supporting, development and verification processes

Ten lifecycle phases or activities are covered by the standard:

- specification,
- architecture design,
- supporting tools,
- design,
- tests and software integration,
- software/hardware integration,
- modification
- verification, validation,
- safety evaluation.

The standard defines indirectly 48 possible emerging properties for the software. The crossing of the 10 lifecycle phases with the relevant expected properties per phase produces 73 requirements on emerging properties.

The link between the normative methods and techniques and the achievement of the expected properties is detailed in the informative appendix C of part 7. The standard recognises that the achievement of a property is obtained through a combination of techniques and methods and with their application with the proper level of rigour, indirectly driven by the targeted SIL number.

Roughly one hundred different detailed techniques and methods are required; ranging from formal proof to dynamic addressing avoidance. The techniques and methods occur in 184 distinct requirements, 87 of them being without any alternative (cf. the 18 tables of the normative appendices A and B of part 3).

Screening the Highly Recommended (HR) methods in these tables one notices that:

- most of the methods related to supporting process activities apply uniformly to the 4 SILs,
- most of the specification, programming, testing or tooling related means either apply to SIL1 and SIL2, or to SIL3 and SIL4. A few ones apply to SIL1, SIL2, SIL3. Static analysis is HR from SIL2 to SIL4.
- the very few HR techniques exclusively at SIL 4 are formal specification, formal proof, and probabilistic testing.

As for SIL-modulated verification independency, either at person, department, or institution level, it happens to be used for functional safety evaluation (part 1, § 8.2.12 to 14).

5.4. Conclusion

There is little difference between SIL 3 and SIL 4. There is a large difference between SIL 2 and SIL 3. These two facts are acknowledged by the railway industry. 96 % of SIL 4 prescriptions are necessary for lower SILs. From a project management point view, if one is targeting SIL 3, it is probably simpler and more effective from a cost point of view to realise the whole project at SIL3 level. This is acknowledged by IEC 61511 for continuous process industry, with the penalty of the cost of methods and techniques frequently oversized.

6. Nuclear

IEC 61226 defines how to allocate the severity categories A, B, C to the safety-related functions of a nuclear plant, and then IEC 61513 defines, depending on the system architecture, how to allocate its DAL, called class, to each item. There are three levels: class 1 to class 3, mapped to the A, B, C criticality levels.

The guidance for class 2 and class 3 software development is stated in IEC 62138, and that of class 1 in IEC 60880. Hence understanding the increment of rigor enforced for class 1 compared to class 2 needs to sort out which requirements are asserted in IEC 60880 while not in IEC 62138.

6.1. Software safety objectives

Like with any other software safety standard there is a postulated "compatibility" principle of classes with probabilistic analysis at system level. A class 1 software item, possibly flawed by some residual systematic failures, is assumed to be consistent with a 10^{-4} failure on demand reliability.

6.2. Rationale of gradual construction of software safety

The higher the class, the longer the list of issues to be addressed in the specification, the more safety mechanisms to be implemented, the greater formality in the development and the higher coverage of software verification.

IEC 60880 for class 1 and IEC 62138 for classes 2 and 3 enforce processes based on the requirements of the system standard IEC 61513. These processes are similar (based on the V-cycle) but not identical because additional activities and even teams are required for the highest classes. Class-dependent variability also affects the content of the specification, of the implementation, and of the development means as well as the extent of the documentation and verification.

6.3. Class-dependent requirements on supporting processes

There is no significant influence of classes on the principles of the supporting processes. Let us mention briefly configuration management and documentation.

Configuration management must be performed according to documented provisions. Configuration management must be applied in particular to the items related to the correctness of software, to the documents subject to verification, to the components needed to build the executable code and to the software tools.

Whatever the class, the Software Requirements Specification, the Design Specification, the Verification Plan, the Validation Plan, the results of the verification actions and all the items related to software correctness must be documented.

Anyway, when going from class 3 to class 1, much more aspects are required to be documented, justified, verified and placed under configuration management. Also, the need for an independent verification team in class 1 induces differences in communication and reporting means, and therefore in the supporting processes.

Thus, although similar in principle, supporting processes are different in practice across classes.

6.4. Class-dependent requirements on the development processes

Regarding software specification, class 3 requires it to conform to the allotted system requirements in a verifiable way, and to address a list of issues: the interfaces, the functions, the behavioural modes, including that in error or detected failure cases, etc. Class 2 adds low complexity constraints and quality objectives (clarity, precision etc.) to be enforced by authoring rules and standards. Class 1 still adds a complementary list of technical issues to be addressed by the specification like self-supervision, the plant's special operating conditions, hardware software integration constraints, etc.

As far as software architecture, design, coding and integration are concerned, class 3 mainly requires the overall organisation and behaviour to be documented, the safety related requirements to be met in all specified conditions, and to apply documented authoring rules. Class 2 adds a list of technical items to be documented, and verifiable rules to be applied, especially that aiming at early detection and containment of software errors. Class 1 adds supplementary and mandatory design rules, in particular regarding modularity and verifiable determinism. It also requires documenting all the dependencies between module verification and verification of the integrated software.

6.5. Class-dependent requirements on the verification processes

Class 3 and class 2 require that at least specification and design be verified by competent and independent persons (at activity level). Class 1 requires an independent verification team placed under independent management and verification of each development phase with respect to its inputs. Class 1 requires the techniques, tools and pass criteria to be documented. Adequacy of the architecture to the safety requirements must be substantiated.

IEC 62138 accepts class 3 and class 2 source code to be tested on the host hardware, or in a software engineering environment. Class 3 and class 2 mandate documented verification, but class 2 adds requirements on the verification of the applicable rules and standards, on the test sufficiency and on the justification of non conformances. Class 1 requires analysis in addition to testing, detailed documentation and justification of the test cases, procedures, results and coverage criteria. Module verification must show that each module performs its intended functions and does not perform unintended functions.

Finally, object code verification is performed during system integration and validation in the target Instrumentation & Control environment. Validation must demonstrate that the software complies with its functional and interface specification derived from the system needs. Some requirements are stated only at class 2 and class 1 levels. Class 2 requires demonstration that, in the target I&C system, the integrated software conforms to each functional, performance and interface statement of the software specification, and contributes as designed to the satisfaction of the system requirements. Class 1 adds management independency of the validation team, and comprehensive coverage of the signal ranges, of the ranges of calculated parameters, of the voting and logic combinations.

For classes 3 and 2, this validation testing (i.e. final testing of the integrated system) may be performed on a hardware platform identical to the one of the actual final system if adequate justification is provided. For class 1, validation testing must be performed on the actual final system.

7. Railway

Railway systems integrate more and more programmable numerical equipment including consequently software. Some of these systems are subjected to RAMS requirements (especially safety requirements). It is in particular the case of on-board control/command systems known as “safety critical” whose failures can cause serious damage to people or to goods, as well as systems with very high availability targets (telecommunications networks in particular).

The software integrated in such systems consequently also undergoes RAMS constraints. There are several techniques making it possible, on one hand, to avoid or eliminate the development faults and, and on the other hand to make the execution of the software applications safe in case of occurrence of physical or environmental faults. These techniques include in particular tests, simulation, proofs, and design of safe and reliable architectures including the RAMS analyses (Failure Modes Effects and Criticality Analysis, Software Error Effects Analyses, Fault trees…).

The standard for the railway domain is decomposed in three parts:

- CENELEC EN 50126 establishes a method for the specification and demonstration of reliability, availability, maintainability and safety (RAMS), for railway domain.
- CENELEC EN 50129 provides general guidance to demonstrate the safety of electronic systems and

to construct the safety case for signalling railway application.

- CENELEC EN 50128 provides requirements for the software used in signaling railway application.

7.1. Software safety objectives

The standard CENELEC EN 50128 is particularly dedicated to the software development for the railway field. Notice that a new version of this standard was published in October 2011. This version is stricter and introduces some enhancement in quality management, tool qualification and software maintenance management to address deployment.

The Safety Integrity Level (SIL) becomes SSIL (Software SIL) with levels from 0 (not critical) to 4 (critical), and for each SSIL, the specific development activities (including verification and validation) are prescribed.

For a component of a given SSIL, EN 50128 describes the processes, methods and tools to be implemented during the development. It is about an obligation of means, which is added to the obligations of quantitative and/or qualitative results.

Software certification demonstrates the reliability, or safety of software systems in such a way that an independent authority can check it with sufficient trust in the techniques and tools used in the certification process itself. It can be built on existing validation and verification techniques but introduces the notion of explicit software certificates, which contain all the information necessary for an independent assessment of the demonstrated properties. Software certificates support a product-oriented assurance approach, combining different techniques and forms of evidence (e.g., fault trees, safety cases, formal proofs,...) and linking them to the details of the underlying software.

7.2. Rationale of gradual construction of software safety

Within the framework of critical systems (SIL 3 and 4), the design principles to ensure safety generally go in opposition to system availability. This is the consequence of a “fail stop” design principle aiming to stop the system in case of failure and therefore ensuring a “fail safe” behaviour.

As example, in the railway field the plausible failures will generally have the effect of “stopping the train(s)” which has a strong impact on the system availability. This feature, characteristic of applications (like ground transportation and energy production) having a “rest state” identified as safe and reachable by (relatively) simple means and (relatively: 3km and 1mn30s to stop a high-speed train at 300km/h) fast, is not shared in other fields (like aeronautic) where some vital functions must remain available in all circumstances.

As to software, only subject to design faults because of it’s immaterial nature, preventing and eliminating these faults by the various prescribed methods for high SSIL levels (SSIL 3 or SSIL 4), also contributes to improve its reliability level by a better control of it’s complexity and quality.

For the non-critical (SSIL 0) and not much critical (SSIL 1 and SSIL 2) applications, the design process of software is on the other hand less constrained (as well for the programming language and tools as for Verification and Validation process) inducing a less quality of software, often causing unavailability scenarios. For such applications, the use of “Commercial Off The Shelf” (COTS) components is allowed and therefore frequent. The control of the quality of these COTS components, which has consequently a direct impact on system availability, remains consequently a crucial question, in a context of increasing search for profitability.

7.3. SSIL-dependent requirements on integral processes

The CENELEC EN 50128 requests some plans to manage software safety:

- Software assurance quality plan,
- Configuration and change management plan,
- Verification and validation plan,
- Unit testing plan,
- Integration testing plan,
- Overall testing plan,
- Tool qualification plans.

Configuration management can be applied according to ISO 9001:2008, ISO 9000-3.

7.4. SSIL-dependent requirements on the development process

Phases, inputs, outputs, transition criteria are defined by the standard.

A list of documents is defined as well as a list of milestones. Some definite analyses are mandated (e.g. Software Error Effect Analysis for SSIL3-4).

Regarding documentation, the SSIL have an impact on the list of documents (for example, for SSIL0 32 documents are required, for SSIL1 to SSIL4 46 documents are required).

The other work products to be supplied are also conditioned by the software SIL:

- for SSIL0, the ISO 9001:2008 defines the set of activities,
- for SSIL3-4, all the activities are HR, so that the development cost is roughly twice as much as for SSIL1-2.

7.5. SSIL-dependent requirements on the verification process

Verification includes software verification but does not include the verification of conformity to the standard.

The conformity to the standard is defined in a specific section of the standard and called “software assessment”. The software assessment is a specific activity realized by a person independent from the project that examines the process and all the parts of the product (source code, tests scenarios and results,

documentations, competencies, organization, etc.). The assessor can request some activities (specific tests, safety analysis, etc.).

Traceability is explicitly required from the specification to the code (vertical traceability) and for each phase from the development documentation to the test documents (horizontal traceability).

Formality in source code verification methods grows with SSIL and the formal verification of source code is highly recommended.

For unit testing, structural coverage is required. Statement coverage for SSIL 0, Branch coverage for SSIL 1-2, Path coverage for SSIL 3-4. In the new version of the standard, MCDC is highly recommended only for SSIL 3-4.

For integration testing, interface coverage is highly recommended for all SSILs.

Like DO-178/ED12 and ECSS-Q-ST-80C, EN 50128 requires detection and removal of unreachable code.

The standard recommends using some of the available quality models and associated metrics (e.g. size of code; complexity of design and code, test coverage, number of failures, etc.)

8. Space

Compared to the other standards ECSS-Q-ST-80C has two distinctive characteristics:

1. it pays particular attention to the customer-supplier relationship,
2. on many safety assurance issues it only requires them to be dealt with by the customer and the supplier, without any constraint on the way it is done.

To some extent ECSS-Q-ST-80C is some sort of "meta" software safety assurance standard (or framework), as it compels the customer and the supplier to define their own project specific software safety assurance programme. The standard requires only the list of topics to be addressed in their agreement.

This list is DAL-dependent (table D-2 appendix D), but weakly dependent.

The list of mandatory topics, called clauses in ECSS-Q-ST-80C terminology, is exactly the same for levels A and B, and nearly identical to that of C, except two clauses peculiar to safety critical software.

This weak dependency is due to the "meta-level" nature of ECSS-Q-ST-80C: it defines what is to be defined, not what will be done on the project. Whatever the criticality of the software to be developed, the development assurance issues to be discussed are the same, except for the very few ones that are meaningful only if the software is of high criticality.

This introduction to ECSS-Q-ST-80C is not completely fair. Some of its clauses are precise process-based or product-based development constraints so that the content of the customer-supplier agreement on these issues is much more constrained by the standard than those left open by its overall "meta-level" flavour.

8.1. Software safety objectives

Table 6-1 of ECSS-Q-ST-40 defines the mapping of the 4 severity levels (failures propagation, loss of mission, etc.) to the 4 DALs (A to D for software, ECSS-Q-ST-80C) downward. Some corresponding probability targets are explicitly mentioned in the standard, but their values are not defined by the standard. They are set on a project by project basis.

Hence in the space domain the postulated and tabulated "consistency principle" between software development assurance levels and probabilistic system safety levels applies differently from aeronautics or railway. DAL A is consistent with a variable probability level whereas DAL A or SSIL 4 is consistent with a constant one.

8.2. Rationale of gradual construction of software safety

As explained previously, the DALs of ECSS-Q-ST-80C have little explicit influence in the standard, but of course they are instrumental in what is agreed on all the clauses by the customer and the supplier.

ECSS-Q-ST-80C is both a process-based and product- based framework. It resorts to defining quality objectives, development activities, constraints on applicable methods, or mechanisms to be implemented in the software (e.g. non corruption check and check sum in 6.2.4.8 to 10).

Like DO-178/ED-12 it does not singularize the safety requirements among all the software requirements.

8.3. DAL-dependent requirements on supporting processes

Planning is extensive, like with all the other standards:

- product assurance plan,
- development and verification plan,
- configuration management plan,
- test plan and acceptance test plan

Change control, configuration management, documentation management are obviously addressed.

8.4. DAL-dependent requirements on the development process

Phases, inputs, outputs, transition criteria have to be defined by the customer and the supplier, they are not defined by the standard.

A list of documents is defined as well as a list of milestones. Some definite analyses are mandated (e.g., Hardware Software Integration Analysis).

Methods and tools to be used for all activities of the development life-cycle are to be identified by the supplier and agreed by the customer.

Quality requirements, including safety and dependability, must be defined in quantitative terms.

A distinction is made between manual and automatic coding. If code generators are used some dedicated technical issues are to be addressed (6.2.8: testing, tool qualification aspects, V&V documentation)

8.5. DAL-dependent requirements on the verification process

Verification includes software verification and verification of conformity of the process to the standard (6.2.6.7).

The few requirements that are only applicable to DAL 1 and 2 are verification requirements:

- Validation tests must be re-executed on non instrumented code (6.2.3.8),
- Independent software verification must be performed by a third party and be a combination of reviews, inspections, analyses, simulations, testing and auditing (6.2.6.13),
- Independent software validation must be performed by a third party (6.3.5.28).

But on a structural coverage of testing one reads "*based on the criticality of the software, test coverage goals for each testing level shall be agreed between the customer and the supplier and their achievement monitored by metrics*" (6.5.3.2).

Like DO-178/ED-12, ECSS-Q-ST-80C requires detection and removal of unreachable code. In addition it requires analysing the need of post removal re-verification and re-validation (6.2.3.6, 6.2.6.5).

The standard recommends using some of the available quality models and associated metrics (e.g. size of code; complexity of design and code; fault density and failure intensity, test coverage, number of failures, estimated and verified numerical accuracy, etc.).

9. Synthesis

When comparing the six standards, one notices that their respective DALs condition:

- the quality assurance objectives,
- the processes' activities,
- the processes' work products,
- the development means (methods, tools, rules, standards),
- the independence of some verification activities (software or process' conformity to the standard),
- and last but not least, the content of the software product itself (defensive programming, error detection mechanisms etc.).

Hence the DALs influence software safety construction on six different dimensions.

There are significant differences between the industrial domains, and in some cases even opposite choices: DO-178/ED-12 enforces a DAL-dependent process supported by DAL-independent means (since there is no mean recommendation), whereas IEC 61508, ISO 26262, CENELEC EN 50128 and IEC 62136 & 60880 enforce DAL-independent processes supported by DAL-dependent means.

Space is a bit particular as it is the only standard that allows client and supplier to agree on some process and means on a project by project basis.

Among the four mean-prescriptive standards, a tentative synthesis of the DAL-modulation of the development and verification means is the following:

	Auto.	Auta	Nucl	Rlw
Design & programming rules	X	X	X	X
Requirement and architecture notations	X	X	X	X
Methods of verification by analysis	X	X	X	X
Types and methods of testing	X	X	X	X
Testing environments	X	X	X	X

They all modulate the authoring standards, the design rules and the verification means.

They also all resort to verification independence, but on a varying perimeter: process conformity only, or process conformity and software verification activities.

10. Conclusion

Thus we observe many significant cross-domain differences in the manner of minimizing the risk of residual software development or verification errors.

The discrepancies between the six standards are not a matter of degree: more or less planning, more or less rules and standards, more or less structural coverage or verification independency etc.

Some major discrepancies are a matter of principles: definition of requirements vs. requirement of definitions, modulation of activities vs. modulation of means.

Some positioning of the standards with respect to the six dimensions is possible:

	DAL-dependency		
	None	**Medium**	**High**
Product/content	Aero	Automation Automotive, Space Railway Nuclear	
Process/ Quality objectives		Automation Automotive, Railway	Aero Nuclear Space*
Process/ activities	Automation Automotive, Railway	Nuclear	Aero Space*
Process/ means	Aero	Nuclear Space*	Automation Automotive, Railway
Process/ Independence/ V&V		Aero Automation Automotive Nuclear	Space Railway
Process/ Independence/ Conformity		Automation Automotive Nuclear	Aero Space Railway

The * means an indirect DAL-dependency, i.e. not defined by the standard but practiced within customer-supplier agreements.

Because of the these great foundational differences between the aeronautic standard, the space standard, the nuclear standards, and the three other ones considered as a reasonably homogeneous group, comparing the six highest DALs to one another seems hopeless. The question of cross-domain equivalences or ranking formulated in the introduction may be given an answer if one restricts its scope to the group of standards that derive from, or at least were influenced by, IEC 61508. Even so, it will remain difficult, but will be attempted in further work of our CG2E group.

Another approach, that would encompass the six domains, would be to rely on accident statistics. Since aeronautics, nuclear, railway and space have now been experiencing their standards over 20 to 30 years, some experimental confirmation, or invalidation, of their efficiency could be looked for in the respective safety agencies' fatal accidents statistics. Then some cross-domain comparison of these statistical data could be attempted. Thus return on experience, provided some sound statistical and probabilistic arguments are found, might show that the different rigor enforcement policies are indeed efficient, and that they would lead in some definite cases to admittedly comparable safety levels. Such an approach will also be attempted in future work.

11. Acknowledgement

The authors wish to thank Sébastien Bardoz (Dassault Systèmes), Jean-Louis Camus (Esterel Technologies) and Cyril Comar (AdaCore) for their valuable contributions to the working group.

12. References

[Baufreton et al., 2010] P. Baufreton, JP. Blanquart, JL. Boulanger, H. Delseny, JC. Derrien, J. Gassino, G. Ladier, E. Ledinot, M. Leeman, J. Machrouh, P. Quéré, B. Ricque, “Multi-domain comparison of safety standards”, ERTS-2010, 19-21 May 2010, Toulouse, France.

[Blanquart et al., 2012] JP. Blanquart, JM. Astruc, P. Baufreton, JL. Boulanger, H. Delseny, J. Gassino, G. Ladier, E. Ledinot, M. Leeman, J. Machrouh, P. Quéré, B. Ricque, “Criticality categories across safety standards in different domains”, ERTS-2012, 1-3 February 2012, Toulouse, France.

[Ledinot and Pariente 2011] E. Ledinot, D. Pariente "Formal methods and compliance to the DO-178C/ED-12C standard in aeronautics" in Static Analysis of Software, J.L Boulanger Ed. Wiley 2011.

[Machrouh et al., 2012] J. Machrouh, JP. Blanquart, P. Baufreton, JL. Boulanger, H. Delseny, J. Gassino, G. Ladier, E. Ledinot, M. Leeman, JM. Astruc, P. Quéré, B. Ricque, “Cross domain comparison of System Assurance”, ERTS-2012, 1-3 February 2012, Toulouse, France.

[ECSS-Q30] “Space product assurance – Dependability”, European Cooperation for Space Standardisation, ECSS-Q-ST-30C, 6/3/2009.

[ECSS-Q40] “Space product assurance – Safety”, European Cooperation for Space Standardisation, ECSS-Q-ST-40C, 6/3/2009.

[ECSS-Q80] “Space product assurance – Software product assurance”, European Cooperation for Space Standardisation, ECSS-Q-ST-80C, 6/32009.

[ED12B/DO178B] “Software considerations in airborne systems and equipment certification”, EUROCAE ED-12 and RTCA DO-178, issue B, 1/12/1992.

[ED79A/ARP4754A] “Guidelines for Development of Civil Aircraft and Systems”, EUROCAE ED-79A and SAE ARP 4754A, 21/12/2010.

[ED80/DO254] “Design Assurance Guidance for Airborne Electronic Hardware”, EUROCAE ED-80 and RTCA DO-254, 4/2000.

[ED135/ARP4761] “Guidelines and methods for conducting the safety assessment process on civil airborne systems and equipment”, EUROCAE ED135 and SAE ARP 4761, 12/1996.

[EN 50126] “Railway applications – The specification and demonstration of reliability, availability, maintainability and safety (RAMS)”, CENELEC, EN 50126, 1999 AMD 16956, 28/2/2007

[EN 50128] “Railway applications – Communications, signalling and processing systems – Software for railway control and protection systems”, CENELEC, EN 50128:2001, 15/5/2001

[EN 50129] “Railway applications – Communications, signalling and processing systems – Safety related electronic systems for signalling”, CENELEC, EN 50129:2003, 7/5/2003

[IEC 60880] “Nuclear power plants – Instrumentation and control systems important to safety – Software aspects for computer-based systems performing category A functions”, IEC 60880, edition 2.0, 2006-05.

[IEC 61226] “Nuclear power plants – Instrumentation and control important to safety – Classification of instrumentation and control functions”, edition 3.0, 2009-07.

[IEC 61508] “Functional safety of electrical/electronic/ programmable electronic safety-related systems IEC 61508 Parts 1-7, Edition 2.0, 4/2010.

[IEC 61511] “Functional safety – Safety instrumented systems for the process industry sector. IEC 61511 Parts 1-3, edition 1.0, 3/2003

[IEC 61513] “Nuclear power plants – Instrumentation and control for systems important to safety – General requirements for systems”, edition 1.0, 22/3/2001.

[ISO 26262 “Road vehicles – Functional safety” ISO 26262 Parts 1-9, first edition, 2011-11-15 ISO/FDIS 26262 Part 10, 2011-07-20

13. Glossary

ARP	Aerospace Recommended Practice
ASIL	Automotive Safety Integrity Level
CC	Change Control
CENELEC	European Committee for Electrotechnical Standardisation
CG2E	Club des Grandes Entreprises de l’Embarqué
COTS	Commercial Off-The-Shelf (component)
DAL	Development Assurance Level
E/E (/PE)	Electrical/Electronic (/Programmable Electronic)
ECSS	European Cooperation for Space Standardisation
EUROCAE	European Organisation for Civil Aviation Equipment
FM	Formal Methods
HLR	High Level Requirement
I&C	Instrumentation and Control
IEC	International Electrotechnical Commission
ISO	International Organisation for Standardisation
LLR	Low Level Requirement
MBDV	Model Based Development and Verification
MCDC	Multiple Condition Decision Coverage
OOT	Object Oriented Technologies
RTCA	Radio Technical Committee for Aeronautics
SAE	Society of Automotive Engineers
SIL	Safety Integrity Level
SSIL	Software Safety Integrity Level
TS	Technical Supplement