Lessons Learned on Quality (of) Standards

Rainer Gerlich, Ralf Gerlich

BSSE System and Software Engineering, Auf dem Ruhbuehl 181,
88090 Immenstaad, Germany, Phone +49/7545/91.12.58, Mobile +49/171/80.20.659, +49/178/76.06.129
Fax +49/7545/91.12.40, e-mail:Rainer.Gerlich@bsse.biz, Ralf.Gerlich@bsse.biz URL: http://www.bsse.biz

ABSTRACT:

Standards are used to describe and ensure the quality of products, services and processes throughout almost all branches of industry, including the field of software engineering. Contractors and suppliers are obligated by their customers and certification authorities to follow a certain set of standards during development. For example, a customer can easier actively participate in and control the contractor's process when enforcing a standard process..

However, as with any requirement, a standard may also impede the contractor or supplier in assuring actual quality of the product in the sense of fitness for the purpose intended by the customer.

This is the case when a standard defines specific quality assurance activities requiring a considerable amount of effort while other more efficient but equivalent or even superior approaches are blocked. Then improvement of the ratio between cost and quality exceeding miniscule advances is heavily impeded.

While in some parts being too specific in defining the mechanisms of the enforced *process*, standards are sometimes too weak in defining the principles or goals on control of *product* quality.

Therefore this paper addresses the following issues: (1) Which conclusions can be drawn on the quality and efficiency of a standard? (2) If and how is it possible to improve or evolve a standard? (3) How well does a standard guide a user towards high quality of the end product?

One conclusion is that the analyzed standards do interfere with technological innovation, though the standards leave a lot of freedom for concretization and are understood as technology-independent.

Another conclusion is that standards are not only a matter of quality but also a matter of competitiveness of the industry depending on resulting costs and time-to-market. When the costs induced by a standard are not adequate to the achievable quality, industry encounters a significant disadvantage.

1 INTRODUCTION

1.1 Scope

Today, product assurance on software focuses on the quality of the process rather than on the quality of the end product. Related software standards such as DO-178B or ECSS suggest "best practices" but while giving much freedom for specialisation in a project, they may be too restrictive regarding technological evolution. While such freedom may be well appreciated from the perspective of a project, the question may be raised on how well the quality of the end product is driven by such standards and use of new technology is supported or even encouraged. Further, as standards adherence imposes costs, another question may be raised on the efficiency of the process.

The ultimate goal is to get an end product of high quality at low costs. In the following sections this potential conflict between costs and quality is discussed in the context of experience gained with current standards. The standards we have analyzed are: DO-178B [1], ECSS E-40 [2] and Q-80 [3], EN9115 [4] and the ESA ISVV Guide [5] and further ECSS standard documents on project management.

DO-178B is a standard on "Software Considerations in Airborne Systems and Equipment Certification", E-40 on "Space Software Engineering", Q-80 on "Software product assurance", and EN9115 on "Deliverable Software" as supplement to EN 9100 [6].

We did not make a full analysis of these standards but limited the scope to requirements related to verification and validation and project management and other matters identified as relevant for our activities.

1.2 Assessment Issues

In order to minimize the impact on industry, standards intend to define a minimum of activities required to achieve a certain level of quality. It is evident, that the higher the quality level is, the more activities have to be performed. Such activities may be of purely manual nature or automated based on tools. The share between manual and automated activities impacts significantly the costs, meaning the higher the degree of automation

is, the lower are the costs. Therefore it is extremely important to which degree standards support reduction of costs while demanding a high level of quality.

Currently, every domain where safety and dependability is an issue has its own standards. As long as all companies apply the same standards, they all produce under the same conditions. Then the resulting price only impacts the customers whether they are able or willing to accept the price level imposed by the standards for the desired level of quality.

However, if more than one set of standards is applied in one domain the issue of competition comes in. When another set of standards supports production at lower costs and higher quality, such companies will have a competitive advantage. Therefore, standards should be considered as a matter of competitiveness in a global market, and it should be an important issue to know how efficient standards are in terms of imposed costs for a given level of quality.

Consequently, standards should be evaluated on how efficiently they impact quality and costs of the end product (benchmarking of standards). In this context we will consider how deterministically a certain standard will ensure improved quality. E.g. we will analyze a procedure aiming to increase quality by use of independent tools (cf. 2.3.1.3).

The evaluation result is that it cannot be proven that the use of two tools which fulfill the given definition of "independency" will really lead to higher quality at the end. There might be a chance but without a precise definition of the term "independence" in the context of product quality, an efficient use of tools is impossible. In the definition of the analyzed standards "independence" is defined in the context of tool development but not in the context of quality criteria of the end product. In this example, standards are based on conclusions made some decades ago, but they do not reflect recent findings which allow a clearer picture and a more precise assessment on tool capabilities.

2 LESSONS LEARNED ON STANDARDS

2.1 Quality and Efficiency Issues

Only very little information is published about the impact of standards on quality of the end product and the efficiency of the production process. R. Feldt et al [7] stated that according to their investigations about 17% of costs were spent on efforts to adhere to the ECSS standard [8] which did not add any value to the end product, neither by quality nor by increasing confidence in the quality. In case of the two highest cost contributors engineers concluded that about 50% is adherence cost.

Refering to software engineering terminology "activities not adding a value to the quality of the end product" can be seen as similar to "dead code".This is an issue tackled by the standards in order to avoid it, with one of the reasons being maintenance overhead. This raises the question whether similar requirements of eliminating "dead activities" should be applied to the standards as the standards apply to the software development process in context of a quality management system. If such requirements are already addressed in the standardization process, the results obtained by Feldt et al indicate that a re-evaluation of the success of such purging would be in order.

When standards impose activities on projects which do not add a value, this implies wasting of costs and time. Of course, when a customer makes standards applicable and does accept to pay for it, the contractor should not have a problem, especially if he is being paid in man-hours spent to the project.

However, in a competitive market higher costs and extended time-to-market may imply loss of contracts. Therefore it should be worthwhile to think about the efficiency of a quality management system, where in our understanding the term "efficiency" means "quality of the end product" in relation to "costs and time required to achieve it" (cf. 3.75 in ECSS P-001 [9]).

2.1.1 Definition of Terms

Neither in DO-178B nor in ECSS E-40 and Q-80, ESA ISVV Guide and EN9115 an explicit definition of the term "quality" can be found.

In the basic document ISO 9000 [10] and also in 3.160 of P-001 the following definition is given for quality:

> "*degree to which a set of inherent characteristics fulfills requirements*",

where a requirement can be considered as a need or an expectation.

Consequently, evaluation of quality requires an identification of a set of relevant characteristics and a metrics from which the "*degree*" can be evaluated. This should be reflected in the standards, either inherently or by imposing corresponding measures on the derived assurance activities.

In P-001 the term "efficiency" is defined as

> "*relationship between the result achieved and the resources used*"

which is consistent with our definition given above.

Further analysis will show what are the measures the standards suggest for quality and efficiency.

2.1.2 Complexity and Understandability

An issue on readability is raised due to making extensive references to other standards acting as a supplement. In addition, such links are not visible in the overall architecture of the respective documents.

Part 2 of the SPICe standard [11] is made applicable in Q-80 Clause 5.7.2.2. As the whole document is referenced it is to be assumed that the whole standard is applicable. It is difficult to check whether requirements of both documents are compliant and do not overlap. However, we suppose that such a check has been performed when the SPICe standard was introduced.

The same applies to Clause 5.2.6.1 on conformances where ECSS Q-10-09 [12] is made applicable, a document of roughly 10 pages regarding applicable requirements.

In Q-80 Clause 5.4.1.1 on supplier selection, reference is made to ECSS Q-20 [13]

Relationship to both documents is not explicitly shown in the overall view on "Structure of this Standard". This makes it difficult to get an idea on what is really applicable and where all the information comes from when trying to understand a standard.

In EN 9115, mainly Sect. 5, many references to EN9100 can be found. This is a consequence of being a supplement document specializing EN9100 towards software. In other cases e.g. Sect. 7.2 clarifications for software are added, but the basic contents is not visible which is required to fully understand the requirement.

Every user of EN9115 needs to merge both documents to get a full view on what is made applicable.

It is acceptable to keep visibility on differences between EN9115 and EN9100 by only placing full text in EN9115 which differs from EN9100. However, in context of word processing systems it should not be a big issue to derive a synthesis and to provide it as a fully filled-in standard together with the basic document. Manual merging on the basis of a PDF-file means duplicating of effort by imposing the synthesis on every user.

In contrast, DO-178B is self-consistent and does not make applicable other standards by reference.

In case of source code or software documentation such references hiding the full context would be classified as poorly readable and understandable according to the quality requirements. Moreover, insufficiently documented links are considered as making maintenance of the standards difficult.

2.1.3 Control of Product Quality and Efficiency

In this section we analyze the requirements of the standards regarding product quality and efficiency, especially how strongly these goals are directly demanded in the standards.

In the introduction of ECSS P-00 the goal of standards is defined as:

> "*The goal of the ECSS Standardization System is to minimize life cycle cost, while continually improving the quality, functional integrity and compatibility of all elements of a project, by applying common standards for hardware, software, information and activities in projects.*"

DO-178B states its purpose in Sect. 1.1:

> "*The purpose of this document is to provide guidelines for the production of software for airborne systems and equipment that performs its intended function with a level of confidence in safety that complies with airworthiness requirements. These guidelines are in the form of:*
>
> - *Objectives for software life cycle processes.*
> - *Descriptions of activities and design considerations for achieving those objectives.*
> - *Descriptions of the evidence that indicate that the objectives have been satisfied."*

2.1.3.1 Product Quality

Clause 3.163 of P-001 defines "quality control" as

> "*part of quality management focused on fulfilling quality requirements*"

In this context it remains unclear whether control is applied to product quality or process quality.

DO-178B states in Ch. 8 on the "*Quality Assurance Process*":

> "*The SQA process assesses the software life cycle processes and their outputs to obtain assurance that the objectives are satisfied, that deficiencies are detected, evaluated, tracked and resolved, [...]*"

and further:

> "*The objectives of the SQA process are to obtain assurance that:*
>
> *a. Software development processes and integral processes comply with approved software plans and standards."*

Hence, the quality objectives of the first paragraph, which are still focusing on the end product in terms of

"*deficiencies*", are redirected to process quality by the following paragraph on the objectives of the SQA process

The following paragraph states as a further objective of the SQA process, that

> *"The transition criteria for the software life cycle processes are satisfied."*

This may look like a hook for actual product quality requirements driving the process, as transition criteria can be defined in such terms. However, transition criteria in DO-178B are criteria guarding the entry into parts of the software life cycle process. The examples given in Sect. 3.3 for such criteria are purely process-centric:

> *"[...] that the software verification process reviews have been performed; the input is an identified configuration item; and a traceability analysis has been completed for the input."*

This casts doubts on the conjecture that the objectives of the DO-178B SQA implicitly focus on or include the quality of the product instead of the quality of the process.

But no evidence is given why from "process quality" directly the "quality of the end product" follows. Also, no metrics are directly identified by which such inheritance of quality properties shall be measured and controlled. This control may happen e.g. on the level of quality assurance planning but no explicit requirement was found demanding such measurements on lower levels regarding product characteristics and requirements.

The only activities in the processes supporting measurement of quality of the end product are the verification and validation processes. However, the metrics, which can be found in the standards and in practice, are often related to conformance to standards rather than to quality of the end product.

However, the ESA ISVV Guide directly focuses on this goal by aiming to find faults in the end product thereby complementing ECSS.

2.1.3.2 Efficiency

Only a few requirements to measure the efficiency of standards can be found. For DO-178B no requirements were found supporting measurement of efficiency, at all. DO-178B, according to its own purpose definition (Sect. 1.1) primarily focuses on airworthiness requirements. It should be noted, however, that higher efficiency in the software development lifecycle may allow introduction of more demanding safety goals without additional cost, thus enhancing the airworthiness of systems.

In Sect. 6.2.5 of ECSS Q-80 we found process metrics in terms of duration and costs.

However, apart from [7] we did not find more published information on efficiency of standards in terms of concrete figures. Even if more publications do exist, the lack of found references may indicate that not many may exist.

2.1.4 Summary

The standards focus on process quality while it is still open whether conformance with the process will inherently ensure similar quality of the end product.

The efficiency of the process in terms of effort and duration vs. achieved quality of the end product is insufficiently addressed in all standards analyzed. Nearly no efficiency figures we did find, so that conclusions on the degree of efficiency cannot be drawn, at all, e.g. on how efficiency depends on size and complexity of a product and how costs will evolve in future, facing growing size and complexity of software products.

2.2 Evolution vs. Restrictions

In this section we consider several examples where improvement of standards may be desirable and discuss if and how this can be established.

In the course of evolution two cases may arise regarding evolution of standards from a principal point of view:

- new methods are in conflict with current standards, or
- they may be considered as an extension / enhancement of current standards.

There is a third case between both: new development results may not be in conflict with the general standards and be considered as an enhancement or a variation, but they may be in conflict with customized standards on lower level which have become de-facto standards.

In any of the cases above a software supplier will have a problem with liability or contractual constraints. When the production process deviates from the standard it is up to the supplier to demonstrate that the modified process is superior and will not become a source of quality degradation. Obviously, this is an obstacle preventing suppliers from modifications, even if quality is enhanced by the improvement. This is especially an issue when process requirements are driven by certification authorities – as is the case in DO-178B. Here, non-compliance – even without any negative impact on the actual airworthiness – may lead to a risk just by itself, namely the risk of failure of certification and thus barred market entry for the product.

If a supplier is part of a supplier hierarchy, confirmation is required on conformance with contractually imposed standards already when submitting a proposal, otherwise the supplier may not enter the evaluation phase at all.

Therefore the question is how the conflict between the goals of conformance and evolution can be solved, especially, how the maturity of the proposed evolution can be demonstrated representatively without applying it in an actual project driven by conformance requirements on the current process.

By two examples we will show what are the obstacles in detail and how the related issues may be solved.

According to ECSS Q-80 process evolution is part of the general process. Clause 5.7.3.1.a states:

> *"The results of the assessment shall be used as feedback to improve as necessary the performed processes, to recommend changes in the direction of the project, and to determine technology advancement needs."*

Clause 5.7.3.2.a states that the process improvement shall be conducted according to a documented process. In practice, this implies that the improvement should be exercised in a separate activity and not in the course of a project due to the constraining project schedule.

Such separate activities often take the form of a study of reduced scope. The consequently reduced representativity may lead (1) to acceptance of modifications which have shown improvements in the study but are not mature enough in practice, or (2) the wrongful rejection of actual improvements due to non-representative conditions leading to doubts in the results of the study, even if those are positive.

DO-178B – like ECSS Q-80 – considers process improvements as part of the Software Quality Assurance Plan but defines only a form for documentation of suggested – project specific or general – improvements. In its wording DO178B takes "should" instead of "shall" to indicate openness for improvements. But in practice it may as difficult as in case of Q-80 to apply improvements in a project, because a documented process is required as well.

In the following we consider two suggested improvements of the test process.

2.2.1 Modified Test Process: Compliant or Not?

This example refers to the test process as defined in DO-178B and ECSS E-40.

2.2.1.1 Test Process of DO-178B

DO-178B states in Sect. 6.3.6.b about "*Reviews and Analyses of the Test Cases, Procedures and Results*":

> *"The objective is to verify that the test cases were accurately developed into test procedures and expected results."*

DO-178B suggests in Sect. 6.4.2

> *"Requirements-based testing is emphasized because this strategy has been found to be the most effective at revealing errors."*

Thus, according to DO-178B derivation of test-cases from the specification is preferred over other methods. However, no reference to data supporting the claim of superior fault-detection effectiveness of specification-based testing is provided, so that this assertion and its reasons cannot be verified.

However, DO-178B also remarks in Sect. 6.4.4.2 on "Structural Coverage Analysis":

> *"The requirements-based test cases may not have completely exercised the code structure, so structural coverage analysis is performed and additional verification produced to provide structural coverage."*

This is refined in Sect. 6.4.4.3 on "*Structural Coverage Analysis Resolution*":

> *"Structural coverage analysis may reveal code structure that was not exercised during testing. Resolution would require additional software verification process activity."*

The root of the problem here is that the specification by definition applies a higher level of abstraction than the final implementation, and therefore the specification may lack distinctions which need to be applied in the code. For this reason the test cases derived from the specification may not be sufficient to provide the required coverage on code level and additional test cases need to be defined.

Similarly, the code may be faulty in the form that it does not cover all the cases defined in the specification. Such deficiencies could not be revealed by testing to code coverage only without checking coverage of the specification.

So DO-178B and other standards requiring this two-fold approach acknowledging that test cases need to be designed to cover both the specification and the code.

The question remains why the emphasis is laid on deriving test cases from the specification. Contrary to what is said in DO-178B, our experience from representative experiments indicates that a code-based test approach can result in both higher fault-detection

effectiveness and reduced effort than in specification-based testing (see also the discussion in Sect. 2.3.1).

We refer to our fully automated test cycle as described in [13] where the test cases are derived from automatically generated test stimuli based on code coverage criteria. What has to be done manually in this case is the consolidation of observed and expected results as defined by the specification.

Assuming that coverage of the specification is analyzed during confirmation of test-cases and counter-measures such as correction of insufficient distinction in the code or additional test-cases are taken afterwards, the procedure provides both code- and specification-coverage. It is therefore functionally equivalent to the procedure suggested in DO-178B.

However, the alternative procedure emphasizes code-based selection of test-stimuli and -cases, which is formally non-compliant with DO-178B or at least does counter the suggestion by the standard.

2.2.1.2 ECSS Test Process

Similarly to DO-178B, ECSS E-40 requests provision of test cases before the test campaign is started in Sect. 4.2.6, para 5 on the "*Software Validation Process*":

> *This process can include a test readiness review (TRR) to verify that all test facilities and test cases and procedures are available before each significant test campaign, and under configuration control.*

Although application of the code-based test-procedure effectively leads to the test-cases being available after the actual test run – by comparing the observed to expected results instead of preparing the expected results first – the test-run could also be re-declared as a test-generation process, which together with the consolidation with the specification leads to the test cases.

Thus before TRR, test cases could be derived from the results of the automatically executed tests and their approval in context of the specification, which then could be presented on the TRR together with the automatically generated test environment. This is a procedure usually executed on a host environment.

After TRR the suggested test cases and the automatically generated test drivers would be automatically re-executed after TRR on the target environment, declared as the actual test. Further, as the results of the tests are already known before TRR, the outcome of the tests is predictable and analysis can be completely based on the material prepared during the test-generation phase prior to TRR.

From this point of view the improved process would be fully compliant with E-40, except that tests are already executed before TRR. When interpreting the "test campaign" as "the test campaign on target" the new process would be fully compliant.

The E-40 approach is further detailed in the following clauses:

Clause 5.5.2.9 on "*Definition and documentation of the software unit test requirements and plan*" states:

> *The supplier shall define and document ..., test design and test case specification for testing software units.*

This may be easily fulfilled with the automated approach, just making a reference to the test tool and the automatically generated documentation.

Clause 5.5.3.1 on "*Development and documentation of the software units*" states:

> *The supplier shall develop and document the following:*
>
> *...; the build procedures to compile and link software units;*

So both these tasks are executed automatically and may be compliant when the tool is accepted as producer of the test specification and the test environment.

Clause 5.5.3.2 on "*Software unit testing*" states:

> *a. The supplier shall develop and document the test procedures and data for testing each software unit.*
>
> *b. The supplier shall test each software unit ensuring that it satisfies its requirements and document the test results.*

The same conclusions can be drawn as for 5.5.2.9 and 5.5.3.1.

Clause 5.6.3.1 on "*Development and documentation of a software validation specification with respect to the technical specification*" states:

> *The supplier shall develop and document, for each requirement of the software item in TS (including ICD), a set of tests, test cases (inputs, outputs, test criteria) and test procedures including: [...]*

The test procedures are those applied by the testing tool itself. Therefore these requirements can be fulfilled if the documentation of the tool and further documentation of the adaptations are accepted as test procedure documentation, and the obligation of the supplier to provide developed test cases and documentation is not interpreted as a manual task to be performed by engineers.

2.2.1.3 1Finding a Non-anticipated Fault by an Automaton

Fig. 2-1 lists a function which includes a non-anticipated fault found by the automaton of the fully automated test cycle mentioned above. We assume further that the function corresponds to the following requirement:

> *The algorithm tbd shall be executed in a loop starting from a lower value until the higher value is reached. The number of loop cycles shall not exceed 100 cycles.*

Before we explain, where the fault is, we encourage the reader to answer the following question:

Assuming that a loop cycles takes on 1 ms, what is the upper bound for the execution time? Obviuously, it is 100 ms because the 'if' before the loop ensures that not more than 100 cycles can be executed. However, this is wrong. The Worst Case Executiuon Time (WCET) is about 1.3 years or 4294967295 ms. Why?

```
void myFunc(int il, int iu)
{
   int ii;
   if ((iu-il)>100)
   return;
   for (ii=il;ii<iu;ii++)
       ; // tbd algorithm
   return;
}
```

Fig. 2-1: A Function Including a Non-anticipated Fault

To understand what may happen consider the following inputs shown in Fig. 2-2:

```
il              =-2147483648
iu              = 2147483647

iu-il           =-1

iu-il>100=false
```

Fig. 2-2: Inputs Activating the Non-Anticipated Fault

As soon as the difference between iu and il exceeds the maximum positive value possible for a signed integer, it is interpreted as a negative number by the processor. Therefore the check fails and more than 100 cycles will be executed up to 4294967295.

This fault is hard to detect or can even not be detected when applying all of the usual test requirements:

1. To achieve simple statement coverage and MC/DC (Modified Conditional Decision Coverage) two test cases are sufficient, e.g. iu=50 and il=0 and iu=200 and il=0. To exercise one loop cycle only as further singular case, iu=1 and il=0 could be added. There is no need to think about above fault activation.
2. Derivation of test cases from the specification also would not activate the fault, It is likely that the same testcases as in (1) would be selected.
3. Also, it is likely that it will not be detected during a review because a reviewer will apply the mathematics from a logical point of view and does not recognise the limited representation capabilities of the computer.

Actually, the fault was detected by a different fault identification method, which is simple but helps a lot: the fault was flagged by a timeout set as upper limit for test execution, in order to prevent a single test to block thousands of following tests over night.

The automaton which built the stimulation environment is advised to apply the full range of the input domain including the minimum and maximum value. Thererfore it was very easy to activate the fault condition.

A critical reader may argue that such exceptional conditions may not really occur under nominal operational conditions. That may be reasonable. However, it is essential to know that a condition may occur which violates the requirement on the upper bound of the execution time.

Moreover, this fault instructs us to be more caeful with types, because the results may not be as could be expected from the point of view of mathematics. It is just the difference between theory and practice, what we have learned about.

2.2.1.4 Summary

When applying an extensive interpretation of the requirements, the fully automated test approach of [13] may be considered as compliant. In case of DO-178B reference may be made to alternative process steps inherently proposed in the standard (Sect. 1.4 and Sect. 3 of Annex A), but leaving it open how such alternatives may be approved.

For ECSS, firstly, it is a matter of interpretation on how the supplier shall provide the required material on suggested modifications and maturity. Secondly, in the considered case there is an obvious conflict with Sect. 4.2.6, which, is more a matter of strict adherence to the standard, without having any impact on the quality of the end product.

However, in any case the standardisation bodies and the customer need to be contacted and to agree in advance.

2.2.2 Extended Test Process

This example discusses an extension of the test process aiming to efficiently detect faults which require high

effort for detection or may remain hidden in the traditional test process. The discussion focuses on the interpretation of current standards whether they have to be considered exhaustively and thereby exclude extensions, or non-exhaustively in which case an extension would be part of customization.

The point of discussion is related to the test environment. So far testing is requested on the target platform or a representative platform. As a matter of fact, budget limits currently constrain testing on other platforms, as this – at first glance – would only introduce additional cost. However, in the context of an alternative test process costs may be even saved when adding tests on a non-representative platform, in contrast to the current understanding.

Costs may be saved firstly due to limitation of the number of manual activities when taking an automated approach including platform porting, and secondly by reduction of false alarms and the related overhead due to a sophisticated strategy. Thirdly, use of non-representative platforms may be possible earlier in the development cycle – before the complete target environment is available – and allow detection of faults which are difficult to find on the target.

Finding bugs easier leads to reduced cost for detection, and finding bugs earlier leads to reduced cost for fixing them. Enhanced visibility of dormant faults may also help to reduce maintenance risks and costs and thereby improve maintainability.

If organized properly, e.g. by auto-porting, the benefits of non-target testing may be achieved with only low additional cost for this additional test step but yield an overall reduction of effort in the whole development cycle. This is an experience made in the past.

The keyword for the point of discussion is “platform diversification” as described in [13] and [14]. This experience has demonstrated that a non-representative environment, i.e. an independent one regarding platform characteristics, can have higher efficiency in finding some of the faults than a representative one. This is especially true for dormant faults.

Variation of the following basic platform characteristics was found helpful: operating system, processor architecture, compiler. They are extended by specific characteristic of the test environment, e.g. in adding more checking capabilities than available on the target platform.

The reason for the added value of platform diversification is that testing can only prove the presence but not the absence of bugs, which results in a major difference between acceptance testing and testing for defect detection. While acceptance testing relies strongly on a representative environment to be convincing – after all, detecting no defects does not mean that no defects are present. Therefore any defect found and confirmed by any method is convincing by itself.

Therefore all methods which help to find defects are allowed, even if they do not rely on a representative environment. Of course, false positives may be introduced by the non-representativity of the platform, but at the same time, additional true positives may be made visible which would stay invisible in traditional testing.

As long as the effort for filtering the false positives is lower than the effort for finding the additional true positives in traditional testing plus the costs saved due to the reasons laid out above, platform diversification is beneficial.

Further, a higher fault-detection probability of any given method also increases the confidence in the results of a passed acceptance test.

In the following we discuss possible conflicts of such an approach with DO-178B and ECSS standards.

2.2.2.1 DO-178B

DO-178B states in Sect. 6.3.1.c “*Compatibility with the target computer*”:

> *The objective is to ensure that no conflicts exist between the high-level requirements and the hardware/software features of the target computer, especially, system response times and input/output hardware.*

and in Sect. 6.4 on “*Hardware/software integration testing*”:

> *To verify correct operation of the software in the target computer environment.*

And further as an refinement thereof in Sect. 6.4.1 “*Test Environment*”:

> *More than one test environment may be needed to satisfy the objectives for software testing. An excellent test environment includes the software loaded into the target computer and tested in a high fidelity simulation of the target computer environment.*
>
> *Selected tests should be performed in the integrated target computer environment, since some errors are only detected in this environment.*

Obviously, 6.4.1 only requires the test on the target system with a representative (here: “high fidelity”) environmental simulation. The potential of testing on other platforms regarding improved fault identification capabilities is not acknowledged and its application is not requested. Further, the declaration of the target with environment simulation as an “excellent test

environment" may be read to imply that other environments are inferior to this configuration – contrary to what the experience laid out in Sect. 2.2.2 indicates.

As a matter of fact, the quality assurance requirements of DO-178B are already perceived as very costly and thus any activity not imposed by the standard and thus solely left to the discretion of the supplier is likely to be excluded for cost reasons.

2.2.2.2 ECSS

ECSS Q-80 states in Clause 7.3.6 on "*Testing on different platforms*":

> *a. Where the components developed for reuse are developed to be reusable on different platforms, the testing of the software shall be performed on all those platforms.*

This could leave the door open for testing on different platforms for other cases than reusable software.

Similarly, ECSS-E-ST-10-02 [15] states in sect. 5.2.2.1 on "*Verification methods*":

> *c. Verification of software shall include testing in the target hardware environment.*

The wording "*shall include*" may not exclude testing on another environment, so an extension may be possible in the context of customization. However, again the use of other platforms for testing is left to the discretion of the supplier.

2.2.2.3 Summary

The background of the requirements referenced above is obviously the acceptance test aiming to demonstrate compliance of the end product to the specification. As testing in principle can only reveal the presence of faults, the value of acceptance testing relies on achievement of sufficiently small doubt in its results. One of the required preconditions for this is the use of a representative test environment – typically the actual target computer – for acceptance testing.

However, the text of the standards may be interpreted such that the best selection for testing in general is the target environment. In consequence, testing on any other environment might be considered inferior from a "best practices" point of view.

Due to the high efforts of testing and the costly requirement to test on the target computer, in practice only the target computer or a representative emulation thereof is considered as relevant test environment.

Still, the optimization of the test process through platform diversification may be an achievable goal for such standards.

2.2.3 Risk Reduction in Project Management

This example deals with already rather detailed standards on project management issues in ECSS, which, however, block an overall optimization due to separation of parts of project management into different documents without defining an interface between them which would allow to get an integrated and harmonized view on all matters of project management.

ECSS standards on project management define requirements on planning (document ECSS M-10 [16]), cost and schedule (document ECSS M-60 [17]). Document ECSS M-80 [18] defines the process for risk management.

Unfortunately M-10 and M-60 introduce data structures which are not compliant to each other and prevent tracing of dependencies between planning, costs and schedule thereby inherently increasing the risks which shall be tackled by the process defined in M-80.

Hence, the conclusion is: a more systematic structure of data could remove risks related to project management.

This optimization is blocked for the following reasons:

- The project management issues have been broken down into two parts, which deemed to be independent according to traditional processes and organizational structures.
- The M-10, M-60 and M-80 documents are maintained by different and independent teams.
- Dependencies are dealt with informally and manually.

Due to separation of concerns – which is reasonable from an architectural point of view – an integrated handling of all management aspects is not supported. Such an approach would have several advantages, such as:

- Planning and cost figures could be collected bottom-up from the location where they are originating to the upper level.
- Metrics could be applied to check consistency of schedule and resource utilization.
- In case of changes their impact and potential inconsistencies can be identified immediately on all issues of project management.
- In summary, this will lead to a significant reduction of project management risks.

This optimization potential was recognized when BSSE defined a tool [SPM] to cover all the management issues in an integrated manner, based on ESA's principal approach.

This is the current situation:

The Cost Breakdown Structure (CBS) as defined in App. A2 of M-60 is incompatible with the Work Breakdown Structure (WBS) and the Work Package Description (WPD) as defined in App. C.2 and D.2 of M-10. Mainly, this is a matter of incompleteness of the WPD template. These are some examples:

- A definition of the required and provided dates is missing for the inputs and outputs. This may lead to inconsistencies between (man-power) resources, scheduling of the work package itself and the availability of deliverables. Also, the costs of a delivery cannot be associated with a delivery itself at its origin.
- A consideration of meetings and travels as well of related costs is missing to be covered in the work package.
- A consideration of usage of facilities is missing, the related availability and the number of instances.

In consequence, there is no link possible in such a way, that the costs can be collected from their source in the WPDs and immediately be transferred into the CBS.

However, the required extensions would not be in conflict with current standards.

2.3 Guidance towards Quality

This section puts the focus on the guidance of a developer towards high quality when applying a given standard. Such guidance requires precise metrics providng a feedback on the actual quality. We discuss such guidance in the context of requirements on the test processes.

Here the relevant point of discussion is how well the standards guide a developer to be sufficiently efficient in finding faults.

We will explore the standards regarding their contribution to fault identification. This issue is twofold: firstly, the question is how well the standards enforce identification of faults, secondly, what they demand to achieve sufficient confidence in the verification results.

The understanding of the different mechanisms contributing to fault identification is fundamental to find a high number of faults. This should be reflected in the standards or supplements thereof. We will discuss how current standards do support or interfere with the exploitation of such mechanisms.

As was stated in Sect. 2.2.1 the test approach as currently defined in the standards is specification-based. For more critical software the verification objective is to find faults as stated in the ESA ISVV Guide Sect. 2.1 on "*Objectives of ISVV*":

> *"As with any verification and validation activity, the objective of ISVV is to find faults and to raise confidence in the software subject to the ISVV process."*

which also draws attention to fault identification as an objective of the standard test process.

The following section 2.3.1 introduces in the issues discussed in sections 2.3.3 - 2.3.5 later.

2.3.1 Issues of Fault Identification

Fault identification is a matter of verification and validation. Verification is subdivided in the context of code verification into static and dynamic analysis, the latter of which is mainly implemented by testing. For both areas methods and tools exist aiming to detect faults in the end product, where a fault is considered as a non-compliance between expected and observed characteristics of the end product.

In this context we consider code as the end product of the software development cycle (apart from other results such as Operations Manual etc.).

2.3.1.1 Verification through Analysis and Test

Static analysis includes methods which do not require execution of the code. Instead the code is inspected or executed symbolically and the results are checked for compliance with a set of given rules addressing desired or undesired characteristics of the end product. Any non-compliance is considered as a fault.

Dynamic analysis is based on execution of the code on a given platform. Therefore the code is exposed to additional, different conditions compared to static analysis. The results of testing are the outputs of the test subject and additional – undesired – detected anomalies. Outputs are checked against the specification. Anomalies flag unexpected events through e.g. exceptions or error messages.

All rule- and specification-based checks focus on anticipated faults, because introduction of a rule requires knowledge about a fault or fault type. Rule-based checks can often identify the location of a fault directly. Anomalies and generic output-based checks indicate existence of a fault without necessarily pointing to its origin in the code or identifying its type. Anomalies may flag a non-anticipated fault.

Static analysis methods and tools – at first sight – seem to be superior to testing because in theory they can assess the fault-potential in the whole set of possible states of the code, while testing has to focus on samples from this set. However, theory and practice for static analysis may differ considerably because tools may be faulty or resource limitation may prevent actual exhaustive assessment of the state set for a fault-type

supposed to be supported. Abstractions are safe approximations which may enhance performance or even turn an undecidable problem into a decidable one, but they may also lead to false positives.

Moreover, static analysis does not support detection of non-anticipated faults and platform aspects not modelled in the underlying theory. Specifically, the latter are a very difficult problem, for example timing, behaviour of hardware drivers or of operating system primitives. In consequence, static analysis and test complement each other. As was shown in [13] this is also true for different analysis methods and tools and test stimulation and evaluation methods.

2.3.1.2 Coverage of Fault Types

For testing we classify faults into two major non-overlapping categories: product-dependent (or application-dependent) and product-independent faults.

Product-dependent faults are a consequence of discrepancies between expected and observed charcteristics of a product in the functional or non-functional domain. Product-independent faults are caused by violations of generic quality requirements, possibly also providing narrow indication for the presence of product-dependent faults.

Product-independent faults can be classified into a known set of fault types, where the real, full (super-)set of fault types is not necessarily known. However, the set of known types can be extended over time based on findings during testing thereby approaching incrementally the superset. However, systematic detection of faults based on testing for a subset of known fault types relevant for a product is a valuable goal compared to not knowing which fault types can be found at all in the current testing environment.

Due to the finite number of (known) fault types an assessment of methods and tools is possible regarding their sensitivity to identify certain fault types (cf. [13]]). Although not all fault types may be known, it is of extreme importance to know which spectrum of fault types is actually supported. If the evaluation yields incomplete support of (known) fault types, this helps to improve the set of verification methods and tools towards full coverage.

Classification of methods and tools according to the supported spectrum of faults is a quality criterion complementing the criterion on the quality of a test set as used in context of mutation testing.

2.3.1.3 Independent, Equivalent and Complementary Methods and Tools

The goal of using independent tools is to increase confidence in the verification process based on the assumption that independent tools supporting the same verification method will not fail all together to identify the same fault type.

The goal of using complementary methods and tools is to achieve full coverage of known fault types. However, if the coverage of fault types is not known for a tool, nothing can be said about the increase in confidence when adding another tool to the test environment.

When two methods support identification of the same set of fault types, they are equivalent regarding the fault types. Similarly, two tools are equivalent if they are supporting identification of the same set of fault types. Independence in case of equivalence can be used to increase the chance that at least one of both tools will detect a fault out of the same set of fault types ought to be supported.

When two methods are complementary, they are supporting identification of different fault types. Therefore they are independent per se. Independence in case of complementarity means that both tools extend the set of supported fault types and thereby increase the confidence in product quality once the product passes all the analysis without any relevant faults being detected.

Of course, mixed forms of equivalent and complementary methods and tools may exist in practice. In any case, it is extremely important to know about their sensitivity on fault types. This is similar to how a hardware engineer needs to know about the capabilities and the accuracy of the measuring equipment applied.

Hence, the usefulness of being “independent” can only be assessed in context of knowing details about equivalence and complementarity, but not on its own.

2.3.1.4 Metrics

Metrics represent the focal point to assess and to improve quality. The classification of fault types and the assessment of methods and tools regarding their sensitivity is one example to measure to which degree quality in terms of a high fault removal rate can be achieved.

Metrics may be applied to a process or to the end product. When applied to the process, the dependency between process quality and quality of the end product needs to be proven in the sense that high quality of the process – according to measured process properties – implies high quality of the end product – according to measured product properties. Until this proof is delivered the added value of a standard based on process quality remains rather doubtful.

In current practice, most metrics focus on the quality of the source code in terms of readability and understandability assuming that an improvement in these areas will increase the fault detection rate during reviews and manual code inspections.

2.3.1.5 Summary

The understanding of the different mechanisms contributing to fault identification as described above is fundamental to find a high number of faults. This should be reflected in the standards or supplements thereof. In the following section we will discuss how current standards do support or interfere such mechanisms.

2.3.2 DO-178B

Regarding independence DO-178B states in Sect. 12.3.3.4 on dissimilar software tools and their qualification that

- *"each tool is to be obtained from a different developer,*
- *tool designs have to be dissimilar".*

Regarding fault identification Sect. 2.3.3 on "*Safety Monitoring*" states on "*System Fault Coverage*":

> *Assessment of the system fault coverage of a monitor ensures that the monitor's design and implementation are such that the faults which it is intended to detect will be detected under all necessary conditions.*

This addresses operational capabilities of a product and is not directly applicable to fault identification, but expresses the same idea. Therefore an extension requiring similar analysis for the tools applied as expressed in Sect. 2.3.1.2 should not raise a principal conflict.

Sect. 4.4.1 on the "*Software Development Environment*" demands:

> *b. The use of qualified tools or combinations of tools and parts of the software development environment should be chosen to achieve the necessary level of confidence that an error introduced by one part would be detected by another. An acceptable environment is produced when both parts are consistently used together.*

Here the basic idea of fault detection capability analysis is expressed in a general manner. Such a requirement should be flanked by common activities establishing the information required for this assessment.

DO-178B requests for verification tools in Sect. 12.2:

> *Tools that cannot introduce errors, but may fail to detect them. For example, a static analyzer, that automates a software verification process activity, should be qualified if the function that it performs is not verified by another activity. Type checkers, analysis tools and test tools are other examples*

This requirement acknowledges that tools themselves may be faulty in that their function may in principle be able to detect a fault, but they may not be properly implemented.

Similarly, Sect. 12.2.2 on "*Qualification Criteria for Software Verification Tools*" requires that tools are verified according to their own specification:

> *The qualification criteria for software verification tools should be achieved by demonstration that the tool complies with its Tool Operational Requirements under normal operational conditions.*

whereas Sect. 12.2.3.2 defines "*Tool Operational Requirements*":

> *Tool Operational Requirements describe the tool's operational functionality. This data should include:*
>
> *a. A description of the tool's functions and technical features.*

Demonstration of compliance with requirements, however, is subject to the same doubts as any other verification activity and thus is not sufficient in its own right, but rather must be complemented with safety provisions as required in Sect. 12.2.And in Sect. 12.2.4 on "*Tool Qualification Agreement*":

> *The certification authority gives its agreement to the use of a tool in two steps:*
>
> *• […] For software verification tools, agreement with the Plan for Software Aspects of Certification of the airborne software.*
>
> *• […] For software verification tools, agreement with the Software Accomplishment Summary of the airborne software.*

This requires certification of the tool according to general rules, but does not request demonstration of fault identification capabilities. Also, DO-178B opens the door through Sect. 12.3 on "Alternative Methods". Though a final set of such methods is discussed as known at the time the standard was established, it seems that other alternative methods are not excluded from a principal point of view.

The only metric regarding quality of the end product we found in DO-178B is the metric on coverage as described in Sect. 6.4.4. Coverage is not a direct measure for this quality, but of course finding a defect by testing necessarily requires that the erroneous code is actually executed. In addition, the defect must be activated by the conditions under which the code is executed, the fault must become visible and must be noticed.

In this context it is worth noting that while full path coverage of the code is impossible in principle and thus the required levels of test coverage are always an issue of balance between effort and value, not even the coverage criteria required for the highest criticality levels in these standards are sufficient to capture the basic complexity of the code. None of the criteria can distinguish a loop from a branching statement: Both can be covered using the same test cases.

More metrics may be defined in the Software Quality Assurance Plan which is made applicable through Sect. 11.5.

Regarding quality of the process we found traceability matrix in Sect. 5.5 and 6.2, and report tracking in Sect. 7.2.3.

Examples for known fault types can be found in Sect. 6.4.3.

2.3.3 ECSS and ESA ISVV Guide

Regarding independence ECSS Q-80 states in Clause 5.6.1.1a on "*Methods and Tools*":

> *Methods and tools to be used […](including […] validation, testing, […]) shall be identified by the supplier and agreed by the customer.*

The ESA ISVV Guide refers to IEEE Standard 1012 [19] which defines the "*technical independence for software tools*" as

> *"For software tools, technical independence means that the IV&V effort uses or develops its own set of test and analysis tools separate from the developer's tools".*

In consequence, this gives full degree of freedom to the top-level customer, at least. The good point is that customized, more precise criteria can be added without being in conflict with the standard. The weak point is that a more qualified approach is not enforced.

Clause 5.6.1.2.a of Q-80 states:

> *The choice of development methods and tools shall be justified by demonstrating through testing or documented assessment that:[…]*
>
> *2. the tools and methods are appropriate for the functional and operational characteristics of the product,*

Fault identification capabilities are not explicitly listed.

However, Clause 5.6.1.3.a states:

> *The correct use of methods and tools shall be verified and reported.*

From this perspective the support of the complete set of fault types should follow, in principle. However, as fault types are not introduced in the document, this aspect is out of scope in this context.

Again, the good point is that extension towards guidance is not a conflict, at all, but the document lacks guidance at this point.

We found product metrics in Sect. 6.2.5 and 7.1.5 of Q-80. In E-40 code quality metrics are requested for the Software Reuse File in Annex N.2.

Metrics on the overall product quality are defined in Clause 6.2.5.4.a of Q-80 (number of faults detected) and in 6.2.7.4.a.12 (requesting a figure on code quality), but not stating what this figure actually shall be.

Note that the number of faults detected is in principle neither a valid metric for the quality of the product nor for the quality of the process. A low number of faults detected may lead to the assumption that the product is of high quality, while another reason may be a low fault sensitivity of the process applied to find faults.

Vice versa, a high number of faults detected may inspire both confidence into the fault sensitivity of the process and into the assumption that most of the faults should have been identified already. However, a high number of faults detected could just as well be an indication of a bad codebase, which could by itself imply a high number of faults. The assumption, that the high number of faults detected implies a low number of faults remaining may be fallacious. In contrast, a bad codebase could as well imply introduction of new faults by fixing detected faults.

A more plausible, although still not completely valid metric would be the development of the number of faults detected over time. A high fault detection rate at the beginning of V&V activities declining over the time could be seen as indication that a high number of faults has been detected and the number of remaining faults is small. However, this does not exclude the possibility of the remaining faults simply being difficult to find with the method applied, without any reassurance of the remaining faults being irrelevant or even non-critical.

Clause 7.1.5.a of Q-80 suggests as elements of basic metrics: size and complexity of code, the number of faults detected and fault density, code coverage. Note that specifically fault density may be affected by fault detection effort varying over the set of modules and by numeric errors introduced by different module sizes. Similar to measurements in nature sciences, these sources of measurement errors have to be taken into consideration when interpreting derived data.

The ESA ISVV Guide lists metrics explicitly in Sub Task MAN.VV.T4.S4: the cyclomatic complexity and the number of references to a unit. Definition of further metrics is requested in the ISVV Plan, sect. 6.1.5 and traceability in Annex F.11.

2.3.4 EN9115

Regarding verification tools EN9115 states in Sect. 6.3 on "*Infrastructure*":

> *The organization shall determine, provide, and maintain an infrastructure, as appropriate, to support the software life cycle.*
>
> *Organization infrastructure includes, as applicable: […]*
> *b) software verification tools and utilities, including test equipment and test software;*

There are no specific quality requirements imposed on verification tools other that such tools shall be applied. Sect. 7.1 on "*Product Realization*" reads:

> *Software planning shall address software related activities from project planning through product delivery and maintenance, including the following, as appropriate:*
> *a) quality objectives and requirements expressed in measurable terms, including critical items and key characteristics;*
> *b) the software life cycle;*
> *d) evaluation, qualification, verification, and approval of non-developmental and support software;*
> *f) monitoring, evaluation, and audit of software and related activities;*
> *g) the level of criticality for software, as based upon the contribution of software to potential failure conditions;*
> *h) safety and security requirements for the product and data;*
> *i) standards (e.g., design and coding standards), rules, practices, conventions, techniques, and methodologies for development and test;*

which just defines a mandatory corridor within which the projects can define what they find appropriate for the purpose.

In Sect. 7.3.6.1 on "*Design and development verification and validation testing*" the required process steps are identified while the contents have to be filled in by the projects:

> *The test environment shall be documented and controlled to ensure repeatability.*
>
> *NOTE 1 Verification and validation testing should be appropriate to the size, criticality, and scope of the product.*
>
> *NOTE 2 An approach for regression testing should be documented for retesting software aggregates that have been changed. Regression testing should be appropriate to the size, criticality, and scope of the change.*

Here the general requirement is quite unspecific. The expression of further specializations as notes instead of a proper requirement makes the latter stand out as non-normative. Still, even these specializations do not specify further how the appropriateness of the desired activities should be assessed in detail.

In Sect. 7.6 on "*Control of monitoring and measuring equipment":*

> *The organization shall determine and document how test equipment used for validation, verification, or acceptance of deliverable software product is developed, maintained, and controlled.*

Again, only general requirements on documentation are introduced, but no rigorous methodology is defined by which the adequacy of the measures is to be assessed. The appropriate determination and discussion – exceeding the requirement of documentation – of, e.g., fault detection capabilities is not required.

2.3.5 Summary

In general, DO-178B and ECSS lack guidance towards optimization of identification of fault types regarding the product-independent faults and provision of relevant metrics. The optimization is left to an engineer or a project, a fact which is dissatisfying because a systematic approach will significantly increase the probability of fault identification. Especially, the knowledge on incomplete coverage of such fault types by a toolset will be extremely useful, because then it becomes obvious that a number of faults cannot be detected at all. In consequence, the project gets a chance to improve this uncomfortable situation.

DO-178B, ECSS and ESA ISVV Guide all rely on independency of tools while the term "independence" is not precisely defined. This is in part a matter of IEEE 1012, which is used as a reference. It remains unknown whether "independent" refers to the same set of fault types or to complementary fault types. The current standards just suggest to take another, similar tool without applying any metrics to measure whether such a combination can increase the confidence.

Further, as fault types are not considered in these standards it cannot be decided whether tools are really independent, because the only criteria on independence used are "tool supplier" and "design" in general. It seems that the expectation is to meet both, independence regarding the same subset of fault types and complementarity regarding support of the full set.

A few metrics are defined in the standards, which are not sufficient to control the quality of the product. Primarily, definition of metrics is considered as a matter of the projects. From this it follows directly that these

standards do not have the capability to ensure a certain level of quality.

The systematics of fault identification described in sect. 0 can be considered as an extension of current standards without an obvious conflict. It may even be possible to put such an extension in a separate document to which can be referred to explicitly.

Regarding the definitions on “dissimilar software tools” and “independence” in the standards an update is required, because in the current shape the added value to the verification process cannot be measured.

3 CONCLUSIONS

Above considerations show that current standards

- bear some weaknesses in supporting an efficient lifecycle, and do not sufficiently encourage measurement of process efficiency and identification of the need and possibility for potential process improvements,
- lack completeness and precision regarding metrics for measurement of product quality,
- may block innovative and more efficient approaches, even if compliance can be achieved by “creative” interpretation of the standards,
- lack support and requirements of systematic fault identification.

It seems that more benchmarking on the effects of standards and the achieved efficiency is required to identify more reasonable potentials for improvement of standards. The traditional, manual development process implies a lot of human communication and intervention which is reflected in the standards and which thereby preserves a process structure based on outdated and partly inefficient technology, although the standards intend to be technology-independent.

The future issue should be to benchmark the standards as much as possible, to publish results and to open the door for more efficient technologies based on precisely defined and mature modification procedures enabling improvements in a short- and mid-term perspective.

More specific conclusions are sub-divided and given in the following sections.

3.1 Quality focus of standards

The analyzed standards focus on quality of the process and leave it up to the projects how they want to control the quality of the product. If at all, they prescribe that quality of the product is to be verified and controlled, but without providing specific guidance and standardization in this regard.

3.2 Evolution of Standards

Part of the processes is the adaptation of standards according to project needs. Specifically, ECSS gives detailed support on this subject. Extensions aiming to optimize efficiency of the process and/or guidance towards better product quality are possible in some cases but then require “creative” interpretation of the standards to be not in a conflict. In other cases a conflict cannot be avoided and modification or permission for deviation has to be asked for formally.

3.3 Non-conformance to Standards

A specific problem in improving standards is their embedding in a top-down supplier chain. As standards are (usually) made applicable already in tender conditions, it is practically impossible to suggest non-conforming improvements in the context of a competitive bidder process

The supplier-chain is not the right place for bottom-up propagation of feedback and improvements. By passing standards top-down within a legal context it is unlikely that improvements of standards are introduced. Only, when negotiations on contract conditions are possible or significant parts of standards can be established by the contractor, local evolution is possible or even may reach bottom-up the level where changes can be put in effect.

Another opportunity to introduce improvements – more reasonable – is through activities in parallel to projects, e.g. in evaluation studies, and active participation to the standardization process. This implies consideration of innovation in a mid- to long-term perspective, in addition to the time to be considered for solution of the primary technical problems.

3.4 Complexity

In the ECSS documents references to other standards (e.g. from Q-80 to SPICe) were found through which a large number of additional standards are made applicable. Referencing is an efficient approach to avoid redundancy and to ease maintenance, however the understandability of the reader suffers and it is difficult to get a full view on what is applicable.

Similarly, excessive referencing was found in EN9115.

DO-178B seems to be more concise in this respect. No such references were found.

Such weakness normally is marked as poor by the standards when being applied in the context of quality measurement. This raises the question why the standards themselves should not be subject of the same quality control issues as they impose on.

3.5 Efficiency Issues

The current standards focus on a manually-driven process for historical reasons. Consideration of efficiency issues of a process we found only addressed in ECSS Q-80 with metrics related to duration and effort. However, here the intention of metrics collection is only the comparison with the planning. Comparison between different development and quality models and investigation whether effort found in later phases is induced by previous phases, also as a matter of insufficient guidance by the standards, e.g. due to late fault identification, is not addressed at all.

According to Feldt et al [7] there is a non-neglible portion of standards currently applicable which according to the opinion of interviewed engineers do not contribute to the quality of the product. Such non-contributing activities are similar to "dead code" in software products and should be found and removed.

3.6 Standards and Competitiveness

If standards binding a whole industry – such as the European Space Industry – inhibit by their nature the enhancement of efficiency, this also has a negative impact on the competitiveness of the branch.

Today, developers of new technologies not only have the – reasonable – burden of showing the applicability, practicability and soundness of their new methods as a principal matter, but in presence of technology-dependent standards they also have to either prove adherence to these standards or they have to initiate and advocate the changing of these standards.

In addition to the cost of development and the cost of proof of fitness, any development now includes the cost and effort of standardization. In tightly standardized industry branches like the European space industry or the aviation industry, these are additional costs and thus risks of investment before any financial gain can be achieved by selling or applying the technologies in projects.

This clearly raises the bar of entry for new technologies for mainly formal reasons and thus may deter introduction or even development of such new technologies in the first place.

In other branches, the standardization process has also become a strategic tool for companies to control the market, as can be seen, e.g., in the conflict about XML-based standards for office applications.

As a consequence companies which can afford the cost and effort of participation in the standardization process may gain a competitive advantage just be working towards having their principal practices fortified in the standards, independent of whether they participate in the development of new technologies in the area addressed by the standards they contribute to. Additionally, a customer may also bear the disadvantage of loss of competitiveness imposed by the reliance on standards with such effects. It should therefore be in the best interest of a customer to benchmark the standards and to avoid such standards with such implied negative effects.

A similar situation exists regarding competition between industry working on the same subject, but applying different standards. Then one branch may have competitive advantages or disadvantages due to imposed standards. Therefore standards are not only a matter of quality but of competition, too, which is a matter of efficiency impacting costs, flexibility and time-to-market.

3.7 Technology Independence

The standards considered do not precisely define the development and quality assessment activities. The intent is to leave a sufficient degree of freedom up to the projects to identify the optimum approach for the actual case. This may lead to divergence of lower-level de-facto standards which, however, is contrary to the intent of standardization.

In their present shape standards are only organizing definition of further standards in the form of assurance plans which may heavily vary depending on the domain and the project team.

ECSS supports the concept of tailoring, meaning that sub-sets are derived from a super-set as part of a standard, thereby avoiding divergence at lower levels. However, there is still a lot of freedom to define own standards as far as these are compliant with the overall process corridor.

Though being unspecific to the extent possible, aiming to cover a broad area of methods and to be open for technical evolution, full technological independence was not found. The essential point is that the overall process corridor is based on manual execution of tasks, which implies a certain structure not compatible with newer technologies like automation of the lifecycle, for which a different structure is required to be fully efficient.

Examples are the separation of tools into two classes of development and verification tools and a traditional understanding of fault identification mechanisms as discussed in the following section.

3.8 Tooling

In the considered standards two types of tools are considered: development and verification tools. While for development tools the same standards apply as for the software they are supporting, verification tools are subject of tool qualification and certification, which

may be as costly as the development of application software.

The insufficient understanding of (in DO-178B terminology) “dissimilar software” leads to inefficient use of tools regarding fault identification and high costs for certification and qualification. Once the mechanism is understood the use of equivalent and complementary tools could really increase the confidence at lower costs because additional manual checks complementing the tool results are no longer needed.

Also, the separation into two different classes of tools, development and verification, may be misunderstood and may exclude performing development and verification activities in the same tool, which does increase quality and efficiency rather than compromising quality of the end product. This improvement is currently discussed in context of Model-Driven Development.

In contrast, the traditional approach is based on primarily manual activities, possibly to be performed by two independent teams, which implies the separation. However, today more powerful tools can be more efficient in terms of cost and time as well as of fault identification.

These are examples where restructuring – or reconsideration at least – of the imposed process is required.

3.9 Guidance towards Product Quality

As the main focus is set on process quality and the assessment of product quality, e.g. through metrics, is left to projects, only a small part of the requirements in the standards considered deal with product quality. Product quality is a matter of comparison of observed results against the values derived from the specification, code analysis and code coverage.

A systematic approach to classification of fault types and related sophisticated detection methods in combination of methods increasing the fault identification rate and related metrics is missing in all standards.

3.10 Organization of Standards

As discussed for the ECSS standards on project management matters in Sect. 2.2.3 above, partitioning of standards into a number of documents may have an impact on harmonization of interfaces between different subsets of standards, especially if the documents are maintained by different teams. In the referenced case an integrated view is not possible because different parts were isolated from each other, though they are related to each other. This makes it impossible to identify inconsistencies in a formal manner.

3.11 Metrics

Definition of metrics is not a central topic in either of the standards. This may imply that metrics diverge and evaluation of results does not drive improvements.

3.12 Quality of Standards

The analyzed standards mainly focus on the quality of the process aiming to ensure that further refinements based on these standards achieve a sufficient quality regarding the process implemented by the projects’ responsible. This is completely in-line with the ideas of ISO9000/9001 of which EN9115 is a further refinement towards “deliverable software”, but still remains on the level of a corridor. DO-178B and ECSS go a step further and refine the principal process deeper but without reaching the level required for accurate control of quality of a product by metrics at reasonable budgets.

All these processes are fully in-line with ISO 9001 regarding the control of the production process in the sense that they can identify nonconformances regarding the process allowing an organization to improve the degree of conformance. However, DO-178B and ECSS lack advice regarding process improvement itself. The requirements on identification of process efficiency are scarce and without obligation. It even remains undefined what efficiency means. What can be found in ECSS Q-80 is a definition of efficiency by compliance of planned and achieved duration and effort. But efficiency in the sense of achieved quality of the end product vs. consumed effort is not addressed.

This leads to frozen processes where the aim is to increase quality by spending more effort on verification and validation of the code rather than to improve quality during generation of the code and to reduce the manual effort required to achieve the desired quality level efficiently (cf. the discussion in Sect. 3.8 above).

The current process as supported in the standards still represents the traditional structure induced by manual development. ECSS E-40 considers auto-coding but lacks an integrated approach to the overall process at a higher level of automation. For example, reviews of the models are still required (Clause 5.3.2.4.a), which induce manual effort and increase the verification costs, while integrated automatic code-generation and testing would allow a development cycle with low turn-around time and provide a way of validating the results of what is defined in the model in a concrete manner.

This is to a certain degree a matter of doubts in the automation tools due to insufficient capabilities to control their quality, i.e. the quality of the product (in this case the tool itself), which in turn is a matter of the current processes mainly focusing on the quality of a process.

From an overall perspective the current standards do not define or enforce a process by which the efficiency can be measured and quality of the end product and efficiency of the process subsequently can be improved. They admit insufficient efficiency by compromising quality goals to keep the effort and budgets within acceptable limits.

The analysis indicates that standards add only poorly to checking the actual quality of the end product, not only in the sense of "fitness for the purpose", but also regarding the compliance of specification and product. If the focus of our analysis is put on pure process quality, then the quality is reasonable apart from weaknesses as discussed above.

EN9115 has the highest abstraction level of all four standard documents analyzed. It appears more as a check list and a guide to establish concrete standards. Its basic intention is to provide a harmonized baseline for standardization of processes in the area of aerospace. It describes a mature, abstract process for software products, but nothing more.

Above considerations already address weaknesses of quality as observed in the standards. An important aspect for understanding the scope of standards is the intention to leave as much as possible to projects and to introduce a basic corridor which ensures a long-term stability.

However, there are valid aspects of specific topics which could be included while not violating this goal. Such a topic is the extension of fault identification from purely specification-based testing towards a systematic approach as explained in Sect. 2.3.1.2 above. Guidance w.r.t. fault identification based on classification of fault types and metrics should be considered as an important matter of a standard because this addresses directly the product quality.

Moving the focus from the quality of the process to the quality of the product could also lend more freedom to suppliers in applying new and possibly more efficient technology, without compromising in terms of quality. Use of metrics on product quality would make it easier to decide whether an extension is acceptable or not without enforcing much bureaucracy.

Standards could define systematic cornerstones to ensure that conclusions are backed by proper observations – such as proof or hypothesis and contradiction by experiment as well as the assessment of measurements in terms of conclusiveness and accuracy – and define concrete benchmarks to be achieved, without actually defining the concrete process by which these results should be achieved. Concretizations for specific application domains would be necessary in any case, just like required safe load factors for fixed-wing aeroplanes.

4 PANEL DISCUSSION

Parts of this paper addressing evolution of standards were presented during the panel session on "Quality of Standards". The example of sect. 2.2.1.3 was used to explain why the current identification methods as suggested by the standards ECSS and DO178B are not sufficient. Then three cases were discussed regarding compliance with standards and evolution:

1. The modified process of the fully automated test cycle [13] which does not derive test cases from a specification but from the code (sect. 2.2.1.
2. The potential of platform diversification (sect. 2.2.2) based on automated porting.
3. The potential of a database for known fault types and metrics on the capabilities of tools regarding fault identification (sect. 2.3.1).

In a summary the following obstacles regarding evolution were mentioned:

a. The process of getting an agreement from standardization bodies is an open issue. The discussion was limited to ECSS.

b. Compliance with the standards can be considered as a "safe harbor" for a contractor, as compliance is sufficient to get rid of any liability and legal aspects. This leads to missing motivation to spent effort to achieve higher product quality by evolving standards.

c. There is lack of guidance towards higher efficiency in terms of more quality per €.

d. Standards should be subject of benchmarking.

The ESA position can be summarized as:

I. The modified test process (fully automated test cycle) was declared as compliant with ECSS E-40 and Q-80.

II. The potential of platform diversification was doubted. Software suppliers expressed their concern that the effort for porting of code and the number of false alarms due to a non-representative platform would be expected as too high. Therefore ESA stated not to discuss this approach.

III. The use of knowledge about known fault types for assessment of fault identification capabilities of tools to forecast which of the fault types can be identified at all is considered as unfeasible.

First concern is that tool vendors will not support this issue.

Second concern is that the exchange of information between software supplier and ISVV contractor is not

allowed. This would imply that none of both can receive information on the capabilities of the respective tool of the other one. Therefore no conclusion on equivalent or complementary capabilities can be made.

The position of ESA on the issue to know whether the verification and test toolset completely covers the known fault types is left open.

IV. The process of evolution of standards was not explained because (1) is considered as compliant and (2) and (3) are considered as out of scope – so no need for such a process to be explained.

V. To give more guidance the handbook for E-40 (which is in preparation) was considered in context of the fully automated test cycle.

VI. The issue on benchmarking of standards was not discussed.

VII. ESA declared that concerns of software suppliers have to be respected which consequently limits the evolution of standards (see also II and IIIabove).

5 REFERENCES

[1] Software Considerations in Airborne Systems and Equipment Specification, DO-178B

[2] Space Engineering, Software, ECSS-E-ST-40

[3] Space Product Assurance, Software Product Assurance, ECSS-Q-ST-80

[4] Quality Management Systems, Requirements for Aviation, Space and Defense Organisations, Deliverable Software, EN9115 (Supplement to EN9100)

[5] ESA Guide for Independent Software verification and Validation, ESA ISVV Guide

[6] Quality Management Systems, Requirements for Aviation, Space and Defense Organisations, Construction, Development, Production, Assembly and Maintenance, EN9100

[7] R.Feldt, N.Torstensson, E.Hult, L. Green: “Analyzing the Cost of Complying to the ECSS Standards for Software Development”, presented on DASIA’10, Data Systems in Aerospace, managed by Eurospace, June 2010, Budapest, Hungary, but not included in the proceedings

[8] ECSS Standardization Policy, ECSS-P-00

[9] ECSS Glossary of Terms, ECSS-P-001

[10] ISO 9000:2005, “Quality management Systems - Fundamentals and Vocabulary”

[11] SPICe, “Software Process Improvement and Capability Determination” ISO 15504

[12] Space Product Assurance – Nonconformance Control System, ECSS-Q-ST-10-09

[13] R.Gerlich, R.Gerlich: “Fault Identification Strategies”, presented on DASIA’09, Data Systems in Aerospace, managed by Eurospace, June 2009, Istanbul, Turkey

[14] R.Gerlich, R.Gerlich, Th.Boll, K.Ludwig, Ph.Chevalley, N.Langmead: "Software Diversity by Automation", DASIA'05 "Data Systems in Aerospace", 30 May – 2 June, 2005, Edinburgh, Scotland

[15] Space Engineering, Verification, ECSS-E-ST-10-02

[16] Space Project Management, Project Planning and Implementation, ECSS-M-ST-10

[17] Space Project Management, Cost and Schedule Management, ECSS-M-ST-60

[18] Space Project Management, Risk Management, ECSS-M-ST-80

[19] IEEE 1012:1998: Software Verification and Validation