Joint use of static and dynamic software verification techniques: a cross-domain view in safety critical system industries

Emmanuel Ledinot[(1)], Jean-Paul Blanquart[(2)], Jean-Marc Astruc[(3)], Philippe Baufreton[(4)], Jean-Louis Boulanger[(5)], Cyrille Comar[(6)], Hervé Delseny[(7)], Jean Gassino[(8)], Michel Leeman[(9)],Philippe Quéré[(10)], Bertarnd Ricque[(4)]

(1) Contact author, Dassault Aviation, emmanuel.ledinot@dassault-aviation.com ;

(2) Astrium Satellites (3): Continental; (4): Sagem Défense Sécurité; (5): CERTIFER; (6) AdaCore; (7): Airbus; (8): Institut de Radioprotection et de Sûreté Nucléaire; (9): Valeo; (10): Renault.

Abstract:

How different are the approaches to combining formal methods (FM) and testing in the safety standards of the automotive, aeronautic, nuclear, process, railway and space industries? This is the question addressed in this paper by a cross-domain group of experts involved in the revision committees of ISO 26262, DO-178C, IEC 60880, IEC 61508, EN 50128 and ECSS-Q-ST-8OC.

First we review some commonalities and differences regarding application of formal methods in the aforementioned standards. Are they mandatory or recommended only? What kind of properties are they advised to be applied to? What is specified in the different standards regarding coverage (both functional and structural) if testing and formal methods are used jointly?

We also account for the return on experience of the group members in the six industrial domains regarding state of the art practice of joint use of formal methods and testing. Where did formal methods actually prove to outperform testing?

Then we discuss verification coverage, and more specifically the role of structural coverage. Does structural coverage play the same role in all the standards? Is it specific to testing and irrelevant for formal methods? What verification termination criteria is applicable in case FM-test mix?

We conclude on some prospective views on how software safety standards may evolve to maximize the benefits of joint use of dynamic (testing) and static (FM) verification methods.

Keywords: Safety standards, cross-domain comparison, software verification, formal methods, tests, coverage.

1. Introduction

While testing has for long been one of the major software verification means, the so-called static analysis methods have been gaining momentum since the early 2000s. They differ from dynamic analysis such as testing in the sense that they do not require actual execution of the code.

Static verification techniques include model checking, abstract interpretation and theorem proving. They perform symbolic computation on source code taking into account all the execution cases, but possibly even more.1. On this issue of exhaustiveness w.r.t. execution cases, dynamic and static software verification techniques are commonly and duly opposed to one another since the former samples software correctness, whereas the latter guarantees conformance to the formalized specification for the analyzed properties.

But from usage perspective, there is no point opposing static and dynamic techniques. As experienced in various safety related industrial domains and as even reflected in some of their software safety standards, joint use of testing and formal methods is beneficial and gaining adoption by software practitioners.

Joint use of testing and static analysis in six software safety standards (ISO 26262, DO-178C, IEC 60880, IEC 61508, EN 50128 and ECSS-Q-ST-8OC) is the issue addressed by the group who co-authored this paper. This group, formerly named CG2E group [1] is now affiliated to SYNTEC Informatique as part of Embedded France. After reviewing the main commonalities and differences between the six verification policies, we give an account on standard per standard basis. Finally we explore the role of structural coverage analysis, as a key verification activity to address when using FM and testing jointly.

2. Variability of approaches to verification

Before focusing on verification mix and the associated coverage issues, we first review high level differences between the standards that are noticeable at process level.

The six industrial domains may be split into two groups, depending on whether their respective software dependability standard is mainly a safety standard or a development assurance standard.

[1] Some additional impossible execution cases because of outer-approximation in set-based analysis.

The point at stake is the existence, or not, of a specific software level risk analysis (product and process). ISO26262, IEC 61508, EN 50128 require a software level hazard analysis to be performed. They also isolate the safety-related software requirements from the other requirements and ask for higher rigor in the way they are addressed.

Domain	Standard	Rationale
Aeronautic	DO-178	Development Assurance
Automotive	ISO 26262	Safety Standard
Nuclear	IEC 60880	Development Assurance
Process	IEC 61508	Safety Standard
Railway	EN 50128	Safety Standard
Space	ECSS-QST-80C	Development Assurance

Table 1: Overall orientation of the standards

To the opposite, aeronautic, space and nuclear consider safety exclusively at system level. The system level safety functions are allocated to the software units, and they have then to be implemented in a correct way. Conformance with these allocated system requirements i.e. ensuring accuracy and completeness of the software specification on the one hand, and correctness of the implementation against the specification on the other hand, are the main issues.

In such standards there is no point performing software risk analysis, nor distinguishing the requirements related to functional safety. The point is appropriate formalization, refinement, design and implementation of the system requirements into software, whatever they are.

The distinction between safety-oriented or assurance-oriented software standards correlates to some extent with the means-oriented vs. objective-oriented characterization, and with product-oriented (safety-case) vs. process-oriented argumentation. Safety-oriented standards are more inclined to prescription of means, and means prescriptive standards explicitly state where formal methods (FMs) have to be applied while process oriented standards leave more options open as to the acceptable means.

In the sequel, the discussion is limited to what concerns highest criticality software (DAL A, ASIL D, SIL 4, Class 1). In table 2, the specification (resp. implementation) refinement and correctness columns encompass development and verification activities. Correctness at specification level means verifying specification properties, for instance consistency, uniqueness (non ambiguity), completeness or conformance w.r.t. formalized system requirements. Correctness at implementation level is meant as conformance of software behaviour against the specification.

Table 2 shows the positioning of the standards w.r.t. formal methods: may or should they be used, and where?

Domain	Specifcation Refinement & Correctness	Implementation Refinement & Correctness
Aeronautic	Applicable	Partially Applicable
Automotive	Applicable	Applicable
Nuclear	Encouraged	Applicable
Process	Recommended	Recommended
Railway	Recommended	Recommended
Space	Applicable	Applicable

Table 2: Applicability of Formal Methods

The railway standard is the more inclined to formal methods as they are highly recommended for SIL3, in other words they are mandatory at specification, implementation and verification stages. French railway industry even developed a correct-by-construction technology where these steps are intertwined in a formally proven refinement process.

Automotive and process industry also promote formal methods, but for cost-effectiveness reasons they keep from making them mandatory.

Aeronautic and space do not prescribe any mean, even those that showed evidence of benefit for safety critical software. The new DO-333 provides guidance for using formal methods (cf. section 3), but it does not mean that they are recommended. It is now easier to have them accepted as means of compliance by airworthiness authorities.

In space like in aeronautics, testing is required for some verification activities (e.g., hardware/software integration testing in DO 178C).

3. Formal methods and testing: a domain wise survey

When formal methods are recommended or acknowledged as acceptable means of compliance, joint use of static analysis and testing is not specified in detail. It is implicitly assumed that a given type of properties is addressed either by testing of by formal methods but not both.

3.1. Aeronautics

3.1.1. Highlights on the formal methods technical supplement (ED-216/DO-333)

This supplement of DO-178C enables the use of formal methods in place of conventional methods by providing guidance on how to use them, by modifying existing objectives, by defining new objectives, and by describing the needed activities to meet the objectives. It also describes what evidence must be provided by the activities, gives information on the fundamentals of Formal Methods, and deals with characteristics proper to Formal Methods.

This supplement is applicable only when credit is sought from formal method to reach verification objectives. Formal methods are defined as descriptive notations and analytical methods used to construct, develop and reason about mathematical models of system or software.

This supplement is based on some definitions:

- A formal method is a formal analysis carried out on a formal model,
- A model is an abstract representation of a given set of aspects of a system or software that is used for analysis, simulation, code generation, or any combination thereof,
- A formal model is based on a formal notation,
- A formal notation has a precise, unambiguous, mathematically defined syntax and semantics.

In the scope of this supplement, formal analysis is the use of mathematical reasoning to guarantee that properties are always satisfied by a formal model..

After defining what could be considered as formal, this supplement describes where formal analysis could be used. Formal Analysis might replace:

- Review and analysis objectives
- Conformance tests versus HLR & LLR
- Robustness tests

Formal Analysis might help for verification of compatibility with the hardware. But formal analysis cannot replace HW/SW integration tests. Therefore HW/SW integration testing is still needed.

Where formal verification is used to replace testing activities, the former structural coverage objectives are superseded by new objectives stated by the standard, that aim to demonstrate that:

- Each requirement is completely covered
- The set of requirements is complete with respect to the intended functions
- There is no non expected dependencies between output and input data
- There is no dead code.

3.1.2. Field experience and feedback

At Airbus since 2001, a group has been transferring formal verification technology – tools and associated methods – from research projects to operational teams who develop avionics software.

The technology for verifying non-functional properties (such as stack analysis, worst case execution time assessment, absence of run-time errors, floating point accuracy) is not seen as an alternative to testing and will not be discussed here.

We focus instead on the technique of unit proof that we developed for verifying functional properties. This has replaced some of the testing activities for parts of critical embedded software on the A400M military aircraft and the A380 and A350 commercial aircraft.

Within the classical V-cycle development process of most safety-critical avionics programs, we use the technique of unit proof for achieving DO-178 objectives related to verifying that the executable code meets the functional LLRs. The term “unit proof” echoes the name of the classical technique it replaces: unit testing. Since 2002, this approach has been used industrially. It departed from the DO-178B standard (more accurately, it was treated as an alternative method of compliance), and therefore we worked with the certification authorities to address and authorize this alternative. The new DO-178C standard, together with the formal methods supplement DO-333, fully support the use of unit proof.

Unit proof is basically a three-step process.

1. LLRs are expressed formally as dataflow constraints between the inputs and outputs of a computation, and as contracts (pre- and post-conditions) in first order logic, during the detailed design activity of the development process.
2. A module is written to implement the desired functionality (this is the classical coding activity). The C language is used for this purpose.
3. The formal requirements of the C module and the module itself are given to a proof tool. This activity is performed for each C function of each C module.

As usual when performing a verification activity at source level instead of binary level, we perform an analysis of the object code generated by the compiler, including the effects of the compiler options on the object code, to substantiate that the compiler preserves in the object code the property proved on the source code. Note that within this development cycle the HLRs are expressed informally, so the verification of integration is done by testing. Overall, the technique of unit proof reduces the overall effort compared to unit testing, in particular because it facilitates maintenance.

This approach satisfies the four alternative objectives to structural coverage analysis stated in DO-333:

- COVER - Each requirement is expressed as a property, each property is formally proved, and each assumption made for formal verification is verified.
- COMPLETE - Completeness of the set of requirements is verified through the following elements:
  - a The verification of dataflow gives evidence that the data used by the source code is conformant with decisions made during design.
  - b Based on this guarantee, the theorem proving tool is used to verify that the formal contract defined in the design phase specifies a behavior for all possible inputs.
  - c We verify by manual review of the formal contracts that they are accurate and specify the value of each output for each execution condition.
- DATAFLOW - The dataflow verification gives evidence that the operands used by the source code are those defined at design level.
- EXTRANEOUS - Except for unreachable code (which cannot be executed), all the executable code is formally verified against the LLRs. Thus, the completeness of the properties and the exhaustiveness of formal proof guarantee that any code section that can be executed will have no other impact on the results of the function than the one specified in the LLRs. Identification of unreachable code, including dead code, is achieved by an independent focused manual review of the source code.

At Dassault Aviation formal methods have been used on production integrated software since 2000, in the military domain as in the civil domain, but without certification credit.

Contrary to Airbus, FMs are not used to replace testing activities, but to perform new verifications or to support manual analyses. Model-checking on finite state modules, and abstract interpretation intertwined with theorem proving for RTE detection and robustness analysis are the main use cases [4].

Testing is used jointly with abstract interpretation and theorem proving to locally speed-up the verification process.

3.2. Automotive

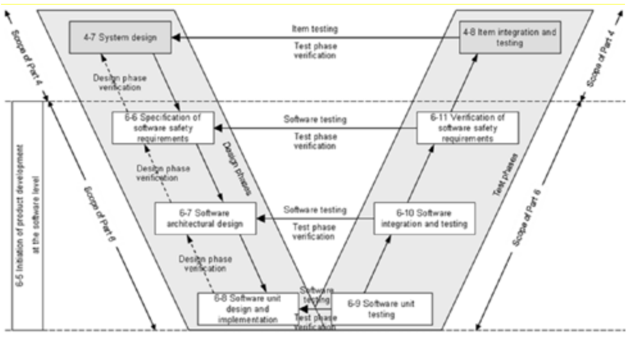
3.2.1. Requirements and recommendations of the standard

In the ISO 26262 the formal verification is defined as a method used to prove the correctness against the specification in formal notation. Two other formalism levels are defined: informal and semi-formal (semantics definition can be incomplete, but syntax is completely defined).

Formal verification is cited twice for software as a method for:

- verification of the software architectural design,
- verification of the software unit design and implementation

These activities are in the left part of the software V-Cycle represented in Figure 1.



4-7 System design
Item testing
Test phase verification
4-8 Item integration and testing
Scope of Part 4
Scope of Part 4
Design phase verification
6-6 Specification of software safety requirements
Software testing
Test phase verification
6-11 Verification of software safety requirements
6-5 Initiation of product development at the software level
Design phases
Test phases
Scope of Part 6
Scope of Part 6
Design phase verification
6-7 Software architectural design
Software testing
Test phase verification
6-10 Software integration and testing
Design phase verification
6-8 Software unit design and implementation
Software testing
Test phase verification
6-9 Software unit testing

Figure 1: reference phase model for the software development defined in the ISO 26262

In both cases, formal verification is considered as an alternative or a complement to other static verification methods such as control and data flow analysis or design reviews.

There is no recommendation of FM for low ASIL and when they are recommended, for high ASIL, they are always presented as alternative or complementary to other methods that are highly recommended. This means that either you don't use formal methods or you use them as a complement or you have to demonstrate that they can replace other static verification methods. In other words, the standard does not promote the use of formal methods.

In the same table detailing the methods for verification of the software unit and implementation we can also consider as a “formal verification method” the “semantic code analysis” which is described in a note as performing mathematical analysis of source code by use of an abstract representation of possible values for the variables.

To complete the picture about static verification at software unit design and implementation level, the static code analysis is highly recommended from ASIL C (verification of rules like MISRA C).

Concerning the testing activities, the standard classically requires unit testing, integration testing, and testing of the integrated software on the target hardware as represented in the right part of the V-Cycle in Figure 1.

Formal methods are not considered as alternative or complementary to test activities meaning that they are only considered in the left part of the software V-cycle.

3.3. Field experience and feedback

In the automotive field, the usage of formal languages is not a common practice and in consequence formal verification cannot today be considered as state of the art in this industry.

Verification of coding rules supported by tools is now a common practise. The most popular set of rules is the MISRA-C coding rules (since 1998).

Static verification using abstract interpretation for verification of the absence of RTE is also a widely applied practise. In some cases, a high number of potential errors for which the tool could not determine if it was a real error or just a warning, has led to some reluctance for the usage in the context of the industrial resource constrained projects. More precisely, the inaccuracy of the tool results obliged to perform manual verification. When the amount of verification to be performed manually increases, the confidence in finding the issues decreases. Progress made on the process and the tools (and its configuration) tend to overcome this issue.

Beyond the detection of RTE, the usage of formal verification and furthermore its possible combination with dynamic verification methods, are still mainly confined to the advance engineering teams.

3.4. Nuclear

3.4.1. Guidelines of the standards

In the nuclear filed, formal methods are not yet mandatory, except in very specific cases such as Static Timing Analysis for programmable electronics. Regarding pure software, the standards require the verification plan to include activities based on source code analysis and test.

Regarding design, the standards consider specifically the case of application-oriented languages associated with “code generators”, and provide specific guidance for this case. In practice, this leads to a significant reduction in unit verification effort, provided the language and code generators comply with the requirements of the standard.

Regarding verification and validation, the standards provide extensive guidance about testing methods for historical reasons, but mainly under the form of recommendations (“should”) rather than requirements (“shall”); thus other means such as formal methods are acceptable too, provided their suitability is demonstrated. Finally, the integration and mostly validation phases must include extensive testing, but other verification activities may be based on testing and/or formal proofs.

The standards do not promote a particular method or tool, but require the verification plan to demonstrate the adequacy of the overall verification strategy, and this demonstration is considered as highly important by the regulators. Thus the domain is ready to take benefit from the progress made in formal methods, for example tools for static detection of Run Time Errors (RTE) such as “divide by zero” are now used in practice, provided they are sound and complete (i.e. all possible RTE will be detected); another example is the use of static stack analyzers based on abstract interpretation to prove the correctness of stack sizing.

3.4.2. Field experience and feedback

In the nuclear field, application-oriented languages and associated code generators have been used since the end of the 1980s, especially with tools based on the formal language Lustre. The feedback of experience is actually very positive both in terms of development effort and confidence in the resulting object. An important success factor is certainly the graphical tools which encapsulate the formal language, preserving its formal properties while providing the user with familiar constructs and elementary objects. This feature helps the project participants (such as process engineers, system designers, software specialists) to understand each other.

Regarding verification activities, the experience of formal tools is not yet so wide, but this approach has already demonstrated at least results as good as testing in cases such as RTE detection and stack analysis. Active applied research is in progress to extend their application to structural properties (e.g. proving that the behaviour of the system software cannot be influenced by the application software) and functional properties, with very encouraging results.

Tool qualification may be reached by application to its development of the requirements applicable to the target software. Otherwise, it is based on the way the tool is used (can it introduce faults, fail to reveal faults, etc.), on its development process, and on the feedback of experience.

3.5. Process and Manufacturing

3.5.1. *Formal methods and tests in the standards* IEC 61508, IEC 61511 and IEC 62061:

A distinction has to done between the base standard (IEC 61508) and the application sectors standards (IEC 61511 and IEC 62061). The sector standards focus on the realisation of safety critical functions based on the use of IEC 61508 compliant devices.

This means that they focus on the coding of application specific programs and not on lower levels such as electronics, operating software and language issues. Formal methods and tests are not mandated nor recommended in the application sector standards. It is however expected to see an emergence of guidance focused on Model Based Design with an opening towards formal proof in the next release of IEC 61511. IEC 61508 recommends and highly recommends, i.e. mandates, formal methods and tests for SIL 3 and SIL 4 criticality level requirements for some activities of the safety life cycle.

3.5.2. State of the art in these industries:

The status of the use of tests and formal methods in the process industries and manufacturing industries varies much according to the actors. The variability is based on the academic background of the different stakeholders. The end-users don't have the teams to adress formal methods. This is not directly required by the regulatory bodies that inspect the machines or the plants. So they don't see the benefit and the need, except in some very large companies such as Oil and Gas industries having a centralised R&D team. In this case, formal methods are used at R&D level to anlyse particular architectures and to standardise them. But the formal method culture does not percolate down to the plants. The system integrators usually don't have access to this technology as well. It is not a clear direct requirement of their customers. The manufacturers of safety products such as safety Programmable Logic Controllers, or Smart Instruments, are supposed to have the suitable engineers within their team. But there is little visibility through literature, communications, as well as through third party tests, on regular activity concerning formal methods within the major players.

3.6. Railway

3.6.1. Guidelines of the standards

Since the first version of the standard CENELEC EN 50128 in 2001, formal methods have been introduced and highly recommended. In the new version of the standard (2011), the V-cycle is recommended and testing is the basic verification technique. But from specification to design it is possible to use formal methods.

The standard introduced the possibility to use the formal techniques (proof, model-checking, etc.) in place of tests. For verification, one can choose a combination of techniques. The standard proposes some best choices and if one decides to introduce a new combination one must explain why it is a good choice and why the proposed set of techniques has the same efficiency. Each time formal methods and/or formal techniques are used one must explain why the efficiency (for some error classes) is similar.

In railway, formal methods are used more and more and at different levels: specification, architecture, design, in replacement of unit tests and of software/software integration test, in data preparation, etc.

3.6.2. Field experience and feedback

Actually, in all railway projects model-based development and verification is used at different levels (system, software, complete software or for some specific functions), supported by SCADE, CONTROL-BUILD, or B-Method. For some projects, proof of properties is applied with some reduction of testing activities.

The most frequent and extensive used of proof in replacement of testing activities is related to data validation. Railway software includes a lot of data and configurations of data. Proof techniques turned out to be most efficient to data verification.

3.7. Space

3.7.1. Guidelines of the standards

In European space the applicable standards are the series of standards from ECSS, European Cooperation for Space Standardization and in particular for software, the software engineering standard ECSS-E-ST-40C and the software product assurance standard ECSS-Q-ST-80C. As a summary one can say that formal methods are recognized as possible means to support software development and validation and their justification. In particular for the highest two criticality categories (so-called “safety-critical software”) it is required to develop “a logical model” of the software, including a behavioral view using automata, and “an analyzable computational model’, and their verification. It is also indicated that for the highest three categories (so-called “critical software”), measures can include “formal design language for formal proof”. However in more general terms it is indicated that verification is based on a combination of testing and analysis (including reviews and inspections) and that the strategy for verification (including the intended combination and its justification) must be documented and justified. In usual practice, this generally means that the verification baseline is primarily through testing, possibly complemented by “analysis” (not necessarily and not usually formal proofs) when testing is not possible for some reason to justify. In particular one can note that testing is mandatory, up to structural coverage criteria varying according to the criticality category, not mentioning the ability to use formal methods as alternate means.

3.7.2. Field experience and feedback

In European space projects, applicable ECSS standards are always detailed, tailored and complemented under the form of specific requirements applicable to the project, also taking into account industry specific processes and practice, and research and development studies.

In this respect one should note an increasing importance of formal methods. However the major focus is more on model-based software engineering (and associated tool support in particular integrated development environment) rather than specifically on formal proofs even though in practice this is also covered through the progressive (formal) verification of modelling steps and formal model transformation techniques down to automatic code generation.

Acquired experience in advanced studies, is progressively introduced in operational practice even though this one remains nowadays somehow limited apart noticeable usage of static analysis for run-time errors, generation of code e.g., from mathematical models of control laws, or formal analysis of some specific critical behaviors (e.g., an initialization or deployment sequence for a complex spacecraft).

4. A cross-domain analysis of verification coverage

Little is said in the reviewed standards about joint use of testing and formal verification. In the recent DO-333, the approach is a split and cumulative policy: applicability of the core document on the testing part, applicability of DO-333 on the FM part. In case of FM-testing mix, both sets of objectives are to be met.

In this section, of more prospective nature, a cross-domain rationale of verification coverage is explored, keeping in mind three kinds of FM-Testing mix:

- *Partitioning*: testing and FM are used on different verification activities, they never overlap,
- *Substitution*: some tests are no longer performed, they are replaced by formal verification, on the whole activity,
- *Weaving*: FM and test are used on the same integrated verification activity. FM is the baseline, testing is used here and there to trade coverage against cost-effectiveness on time consuming proofs [4].

All standards resort to structural coverage analysis as a DAL-dependent testing completion criterion. The higher the criticality, the greater the structural coverage. It plays the role of rigor metrics, but little information is given on the rationale linking “structural coverage” to “verification coverage”.

What is verification coverage? How does structural coverage contribute to verification coverage? Is its contribution proper to testing or relevant for formal methods as well?

DO-178 is the only one of the six standards to emphasize three added values of structural coverage analysis, *as achieved by specification-based*[2] *testing* (Figure 2):

1. detection of some missing test cases to verify a requirement,
2. detection of some missing requirement,
3. detection of the code deactivated or potentially non reachable (so-called "dead code"). “Dead code” has to be chased as potential source of unintended functions.

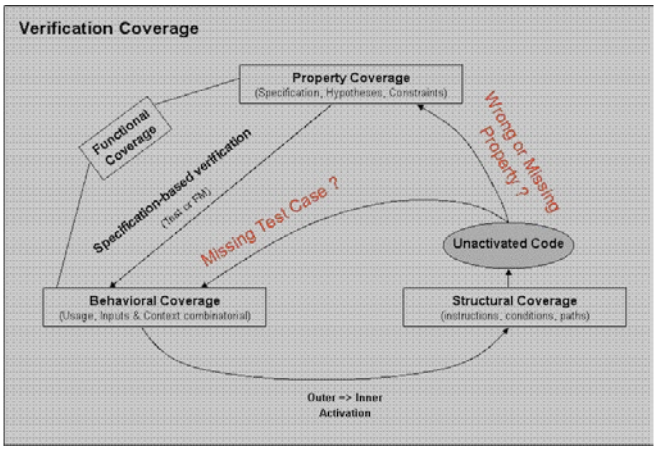

Verification Coverage
Property Coverage
(Specification, Hypotheses, Constraints)
Functional Coverage
Specification-based verification
(Test or FM)
Wrong or Missing Property ?
Missing Test Case ?
Unactivated Code
Behavioral Coverage
(Usage, Inputs & Context combinatorial)
Structural Coverage
(instructions, conditions, paths)
Outer => Inner Activation

Figure 2: Cross-domain rationale of verification coverage derived from DO-178

This detection is not claimed to be complete. It is regarded as a valuable enabler worth the high cost of high coverage ratios (e.g. MC/DC 100%).

In this rationale that stemmed from testing, what would be superseded by Formal Methods?

FMs solve issue 1. It may be argued that structural coverage analysis is not proper to testing because it is an enabler of specification completeness (issue 2 and 3) and specification completeness is key in specification-based verification as required by DO-178, ECSS-Q-ST-8OC, IEC 60880, EN 50128 and by ISO 26262 to some extent.

Figure 3 attempts to illustrate, in the behavioural space, the issue of unintended function detection. It applies to the integrated software and its actual execution environment (hardware, system, and the system's physical and human environment). The behavioural space is the infinite set of MIMO I/O (Multiple Inputs Multiple Outputs) traces that are observed at development time, and may be observed at operation time. Each point of the 2D

[2] Named requirement-based testing in DO-178

figure represents a black box observation at the software boundaries, for a domain-dependent duration (hours for transportation, days, months or years for energy, process and space).

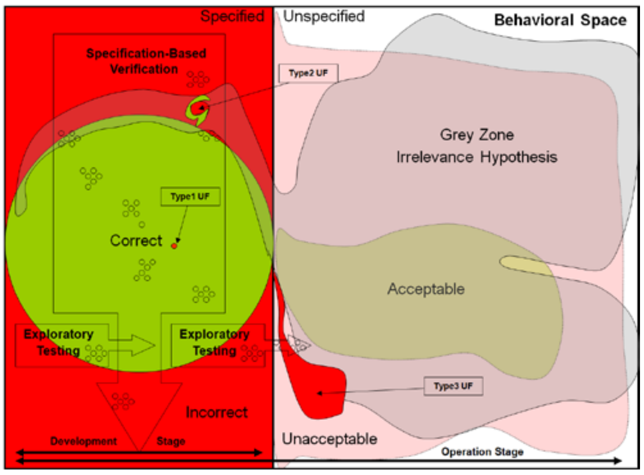


Figure 3: conceptual representation of three kinds of unintended functions

Economic constraints tend to:

- minimize the development-time (a few months or years) and the number of system verifiers (a few tens to hundreds),
- maximize the operation-time (decades) and the number of system users (thousands, millions or more)

The behavioural part of verification coverage could be intuitively defined as the proportion of operation-time behavioural space that was explored by the verification activities at development time.

The economic min-max duality, common to all domains but with significant and regulation dependent variability, is source of unbalance between the respective sizes of these two behavioural spaces. Likelihood of occurrence of unintended functions at operation time is proportionate to this unbalance.

Property space is the second component of verification coverage. Property coverage could be defined (loosely again, and independently of any tractability consideration) as the ratio of user-expected properties actually formalized at development time into test oracles or FM artefacts.

Correctness, used mainly in FM, is behavioural conformance with the explicit properties. The properties that are needed but remain implicit are the second source of unintended functions. FM and Testing are equally powerless to verify these properties, as both need computable predicates over concrete or abstract values.

In figure 3, Acceptable vs. Unacceptable is used in the implicit part of the property space to distinguish from Correct vs. Incorrect in the explicit part (green / red and transparent / dense).

As mentioned previously, most of the standards advocate specification-based verification and use structural coverage as a termination criterion. Fewer recommend non activated code analysis. DO-178 and ECSS-Q80 are singular in requiring dead code removal.

The implicit vs. explicit partitioning of the property space is not represented in figure 4. It should lead to a two-layered validity mapping on the whole unfolded behavioural space, i.e. a 3D figure 3. Each point has a dense and a transparent colour (green if all the properties of the specification are satisfied, red if any is not satisfied).

A hopefully insightful though possibly misleading partitioned fusion of the two status layers was introduced in Figure 3. The sharp rectangular frontier between the statuses of implicit and explicit properties attempts to figure out the situation at the end of development time:

I. All the explicitly specified properties have a status on the explored subset of the behavioural space. These statuses aggregate in a conformance status represented by each point's dense colour.

II. The status of all the needed but still implicit properties (the forgotten additional specification) on that behavioural subset is accessible through "God's eye view[3]" only. This second map lies underneath the dense one.

III. So it is for the status of the explicit properties on the executions cases that lie outside of that subspace, and that may occur for the first time in operation. FMs help minimizing this additional behavioural space not covered by verification.

This non covered behavioural space, that may exist even when using exhaustive verification means like FMs, is represented in a way commensurate to the min-max unbalance. Why?

Neighbourhood in Figure 3's 2D symbolic representation of the high dimensional trace space is trace neighbourhood, i.e. behavioural proximity. Two close traces may occur at very different dates of development or operation time. Conversely two distant traces may occur in a row, the very same day.

So, why suggesting that there is a link between behavioural space, usage intensity during operation,

[3] As used e.g., in estimation theory for absolute or asymptotic knowledge.

and verification coverage? Because of the role that the assumptions play in verification, by testing and even more so by formal methods. Behavioural properties (implicit and explicit) are of the form:

```
    condition(trace)=> constraint(trace)
```

where the condition is a predicate mostly on inputs, and constraint a functional predicate constraining the output trace w.r.t. the input trace. When the condition part is false the property's status is true because the constraint part of the property is not applicable in such an input context. The associated trace point is marked green on the map of explicit properties.

Assumption tuning depends on tuning verification cost. Assumption coverage ratio[4] depends on usage intensity at operation time. As the embedded system and its environment evolve over large periods of time and under stress of even larger user populations, the likelihood of still meeting all the assumptions made at development time is likely to drop.

The role of assumptions is symbolized by the light grey zone in figure 3, suggesting some sort of shadow that may doom the specification validity map. DO-333 (§4.3, 6.2.1, 6.7.1) assigns dedicated objectives to ensure controlled use of assumptions.

In spite of its acknowledged possibly misleading features, figure 3 attempts to depict three kinds of unintended functions (UFs):

1. Specified property violation by a non-tested execution case. This first kind is addressed by formal methods' exhaustiveness,
2. Specified property violation on parts of the operation-time behavioural space not included in the development-time behavioural space. Includes assumption coverage failure,
3. Property emergence, in two cases:
   - o in the behavioural space covered by verification, an operation-time execution case “pops-up” a property left in the implicit part of the property space, that should have been formalized and verified,
   - o same as before, but the triggering operational behaviour was not covered by verification.

DO-178 and some other standards more or less aligned on the same verification rationale (specification-based testing or analysis prior to structural coverage analysis), advocate a contribution to unintended function detection. On which kind of unintended functions?

Structural Coverage Analysis is an enabler for kind 1: the non activated parts of the structure may, once activated, violate some explicit properties. It may be an enabler for kind 2 if the non activated parts of the structure suggest valid weaker conditions on inputs that would activate them. Kind 3 may be regarded as the case of specification completeness explicitly at stake in section 6.7 of DO-333.

So Structural Coverage Analysis is neither direct nor complete for UF detection. It provides so to speak another viewpoint on the behavioural space covered by testing, an inner view on it. Admittedly, confronting two different views, the outer functional and the inner structural, on the same complex object may help to grasp it. In addition, it may also have positive side-effects on the position of the implicit/explicit frontier in the property space. Structural Coverage Analysis contributes to the two components of verification coverage.

5. Structural coverage in joint use of FM and testing

Should then Structural Coverage Analysis be extended to FM verification because a significant part of its value is not specific to testing?

Structural Coverage Analysis is costly and indirect in its effects. Moreover, introducing structural coverage analysis in static analysis tools would face some serious difficulties:

- Set-based accessibility analysis in code or model structures would have to be exact, which is either undecidable, or NP-hard, i.e. intractable.
- Cumulating the coverage metrics over different static analysis tools and testing tools would be exceedingly difficult,
- Qualifying the correctness of such hybrid structural coverage analysis tools would be at best very expensive, and at worst not convincing.

Since detection of type-2 (assumption violation) and type-3 (property emergence) unintended functions is the benefit of Structural Coverage Analysis to be preserved whatever verification method is used, why not looking for means directly aimed at these detections? FM and testing can both help detecting these types of unintended functions, i.e. check the assumptions on enlarged behavioural spaces, and foster property emergence.

FMs offer means to formally verify properties on the explicit (and formalized) part of specifications. As formalization foster both ambiguity removal and completeness, proving properties on the specification would enhance this property emergence effect. Double independent formalization of the specification, with or without proof of equivalence of the two different formulations, would be another property emergence enabler. Definitely more costly, it would probably also be more efficient than proving properties of properties.

[4] Probability of assumption violation in operation

The previous two methods foster the detection of unintended functions in the *property space* component of verification coverage, as FM can compute in this space. The third and last one is exploratory testing. Exploratory testing is akin to validation testing: it is oracle-free testing, a more "free-style" and user centric exploration of software behaviour, looking for behavioural oddities or worse. Since there is no or fewer plans, no or fewer oracles, there is less opportunity for mind biased by assumptions and specified properties. Exploratory testing operates on the *behavioural space.* It was represented on figure 3 as two arrows orthogonal to that of specification-based verification. Specification-less is orthogonal to specification-based.

Already extensively practiced in the six industrial domains at system validation stage, could a cross-domain rationale trading less structural coverage against more exploratory tests be elaborated? Like stress testing of physical equipments that for instance manage to exercise a 10-year lifespan in a 1-month stress-test campaign, might the same approach be attempted to explore a larger part of the behavioural space?

New means may be considered in this respect, though linear CPU time increase faces exponentials:

1. Serious gaming environments. They would make the software accessible through affordable on-line high accuracy operational environments, to a larger group of beta testers,

2. Usage-driven (machine learning on previous developments) stochastic testing on HPC facilities.

6. Conclusion and perspectives

There is a wide range of guidelines in the reviewed standards regarding testing, formal methods and verification coverage. Practice known to the authors varies a lot as well, but consistently with the standards, except in process industry where practice lags behind regulatory guidelines.

Railway, more recently Aeronautics, and to some extent Nuclear, are the three industrial domains where using formal methods, alone or jointly with testing, is effective in production software development. In case of joint use, three modes of combination may be considered, depending on whether one partitions, substitutes or intertwines the two verification means.

The paper reviewed some high level principles of the standards in their way to address safety, and then proceeded to greater detail on the six industrial domains separately. The last part, more conceptual, is a tentative framework to analyse verification coverage in a cross-domain way, applicable to FM and testing, to software and system. It figured out why structural coverage analysis may be regarded as not proper to testing but not worth the pain of being extended to FM.

Alternative and more direct means to address detection of unintended functions have been proposed FM-verification of the specification, double independent specification, and enhanced exploratory testing. They help improve, hopefully as effectively as structural coverage, property coverage and assumption coverage, two components of verification coverage that condition existence of unintended functions.

7. References

[1] 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, Toulouse, 19-21 May 2010, Toulouse, France.

[2] E. Ledinot, J. Gassino, JP. Blanquart, JL. Boulanger, P. Quéré, B. Ricque “A cross-domain comparison of software development assurance”, ERTS-2012, Toulouse, 1-3 February 2012, Toulouse, France.

[3] Y. Moy, E. Ledinot, H. Delseny, V. Wiels, B. Monate, "Testing or Formal Verification: DO-178C Alternatives". IEEE Software Engineering n° xx, May 2013.

[4] 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 Editor, Wiley 2012.

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

[ED12C/DO178C] “Software considerations in airborne systems and equipment certification”, EUROCAE ED-12 and RTCA DO-178, issue C, 01/05/2012.

[ED216/DO333] EUROCAE/RTCA. “Formal Methods Supplement to DO-178C/ED-12C and DO-278A/ED-109A”, ED-216/DO-333, (2011)

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

[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.

[ISO 26262] “Road vehicles – Functional safety” ISO 26262 Parts 1-9, first edition, 2011-11-15 ISO 26262 Part 10, first edition, 2012-08-01