



ISO/IEC JTC1/SC 27 **N7014**

ISO/IEC JTC1/SC 27/WG 3 **N954**

REPLACES: N

ISO/IEC JTC 1/SC 27

Information technology - Security techniques

Secretariat: DIN, Germany

**DOC. TYPE:** national body / expert contribution

**TITLE:** **Comment of the German National Body to additional documents according to resolution 5 of the 36th SC 27/WG 3 meeting held in Kyoto (April 2008)**

**SOURCE:** DIN, National Body of Germany  
**SOURCE of Att.1:** WG 3 Experts (M. Serowy, M. Bañón, B. Krüger)  
(the document is also endorsed by the German NB)

**DATE:** 2008-09-03

**PROJECT:** 15408

**STATUS:** This document is being circulated for consideration at the 37<sup>th</sup> SC 27/WG 3 meeting in Limassol (Cyprus) 6<sup>th</sup> – 10<sup>th</sup> October 2008.

**DISTRIBUTION:** P-, O- and L-Members  
W. Fumy, SC 27 Chairman  
M. De Soete, SC 27 Vice-chair  
E. Humphreys, K. Naemura, M. Ohlin, M.-C. Kang, K. Rannenber, WG-Conveners

**ACTION ID:** ACT

**DUE DATE:**

**MEDIUM:** Livelink-server

**NO. OF PAGES:** 1 + 1 - 12

## ISO/IEC JTC 1/SC 27 N7014

Text of the German National Body comment: Resolution 5 (see SC 27 N6640) of the 36th SC 27/WG 3 meeting held in Kyoto (April 2008) required three documents for the future development of 15408 to be delivered till 2nd July 2008:

1. A document "List of questions and answers concerning privacy components", which was delivered and distributed as SC 27 N6855
2. A document on "multiple assurance level per evaluation", which was delivered and distributed as SC 27 N6848
3. A document on "Architectural Security Requirements", which was not yet delivered.

Since the topic of the third document is regarded as a very important one for the future development of the Common Criteria, the German national body has asked some of the experts involved (see above) to deliver a version of this document.

This version was prepared by the experts and is attached.

The German national body has the following general comment to all three documents:

The German national body proposes to WG 3 to accept all three documents listed above as agreed results of the respective topics.

Moreover it proposes to deliver these documents to the CDDB with the request to implement these concepts into the next version of the Common Criteria.

**The concept of  
Architectural Security Requirements  
as recommendation for the further development  
of the CC (ISO/IEC 15408) towards version 4.0**

Version 1.2, August 2008

Miriam Serowy, Bundesamt für Sicherheit in der Informationstechnik (BSI), Germany  
Bertolt Krueger, Security Research and Consulting GmbH (SRC), Germany  
Miguel Bagnon, Epoche & Esprit, Spain

## Table of contents

1.	Introduction	5
2.	Recommendation to include Architectural Requirements into CC, Part 2	6
2.1	Justification of a modification of the CC	6
2.2	Introduction of the approach	6
2.3	Possible CC implementation of the approach	7
3.	Annex 12	
3.1	Note on the limits of classical functional testing	12
3.2	Implications for EAL considerations	13
4.	Bibliography	14

## Introduction

This document presents recommendations for the development of the Common Criteria towards version 4.0.

It is meant as input for the development process of the CC in DIN and ISO as well as the CC Community (CCMB/CCDB).

## **Recommendation to include Architectural Requirements into CC, Part 2**

This chapter describes and justifies the recommendation. This document does not contain exact text proposals for the new CC version but describes the necessary modifications on a conceptual level.

### ***Justification of a modification of the CC***

On the occasion of a discussion about the possible inclusion of additional SFRs into the CC a more general discussion was started regarding requirements for testability of SFRs. In particular some people argued that today's CC test methodology is mainly oriented towards "functional requirements" in the narrow sense, in other words requirements, which can be tested by calls to external interfaces and the observation of their results. The consequence is that requirements on the overall architecture of the TOE, which cannot (or can only hardly) be mapped to specific external interfaces of the TOE, cannot be modelled by SFRs.

This would hold for example for so called "negative requirements" which state that certain actions or events must not be possible. A similar problem holds for functional requirements, which can be mapped to a specific part of the TOE design, but are not directly visible at external interfaces.

Some participants in the discussion believe that already the present CC allows to use the kind of requirements discussed here, because it already allows a broader understanding of "testability" by other means.

However, the fact alone, that such different views are possible and that proposed SFRs can be rejected partly with the argument, that they were not testable, raises the necessity to improve the CC. Depending on the point of view this improvement clarifies the already existing possibilities of the CC or extends the possibilities of the CC.

In any case the goal is to explicitly allow the specification of architectural requirements (sometimes also called "properties") for TOEs.

### ***Introduction of the approach***

The approach requires that the definition of SFRs in part 2 of the CC is clarified or broadened in the direction, that besides functional requirements in the narrow sense (requirements testable at external interfaces) also other types of requirements are covered. In the following three types of requirements are discussed:

1. "Positive" functional requirements that are directly visible and testable at the external interfaces.
2. Functional requirements related to the internals of the TSF where the effect is not directly visible at external interfaces.
3. Functional requirements expressed by stating a property, which the whole TOE should enforce. This is often done by expressing what should not happen, therefore defining "negative" functional requirements.

Examples for the first type of requirements are the definition of rules for an access control policy or the specification of the externally visible behaviour of an authentication mechanism. Examples of the second type of requirements are the specification of internal functions and properties like the protection of data when transferred or stored internal within the TSF. Examples for the third type of requirements are the definition of information flow, or fault tolerance where the requirements are expressed using properties the whole TOE has to enforce.

The refinement of these requirements to the design will then be split between FSP (for the functional requirements in the narrow sense), ADV\_ARC (for the SFRs which cover architectural aspects) and directly to ADV\_TDS (for SFRs specifying internal behaviour of the design) .

The approach extends the concept of security requirements in part 2 in order to allow the specification of architectural and internal requirements.

Note, that one SFR may combine more than one of the types listed above. As an example for this issue we propose to look at FDP\_RIP.1 "Subset residual Information Protection". Here one can choose between making former contents unavailable at "allocation" time on the one hand or on "de-allocation" time on the other hand. For the "allocation" case one can think of functional tests at defined external interfaces, which show, that newly allocated resources don't contain former information. For example one can allocate storage, fill it with defined data, de-allocate it and then re-allocate it for a different process, checking the contents. However, to test in this way, that data were made unavailable at de-allocation time, is definitely not possible. This can only be seen by "looking inside the TOE" (e. g. by reading storage contents with special tools like debuggers, by validating the source code and so on).

In other words: The same SFR may cover security functions in the narrow sense as well as architectural requirements or internal requirements at the same time. It is even conceivable that the same requirement could be implemented by a security function in one TOE and by architectural properties in another TOE. At least this possibility shouldn't be excluded by construction of the CC.

### ***Possible CC implementation of the approach***

The "architectural parts" of a SFR are usually implemented in part by some supporting (testable) functionality within the TSF and in part (potentially) some design principles. Supporting functionality now can be traced down to the TOE design documentation and tested appropriately using the requirements of ATE\_DPT. The figure below will show the possible paths for tracing these aspects to design and testing.

Design principles may not be testable that easily. For example a design principle of defensive programming can be tested on the basis of a (usually quite small) set of samples, but the evaluation that this principle is generally applied needs to be analysed differently. A developer may use different methods to ensure that the principle has been implemented throughout the TSF. One method is to use a tool within the development process. For the example of defensive programming a developer could either use a code generation tool that automatically inserts the code to check the parameter values or he could use a tool that checks the code of individual developers for the inclusion of statements that perform the required checks. In those cases the evaluator can validate that the tool has been used consistently throughout the development (or for the development of those parts of the TSF where the design principle was required to be applied) and get the required evidence in this way.

The following figure outlines some aspects, which are important for realisation of the approach:

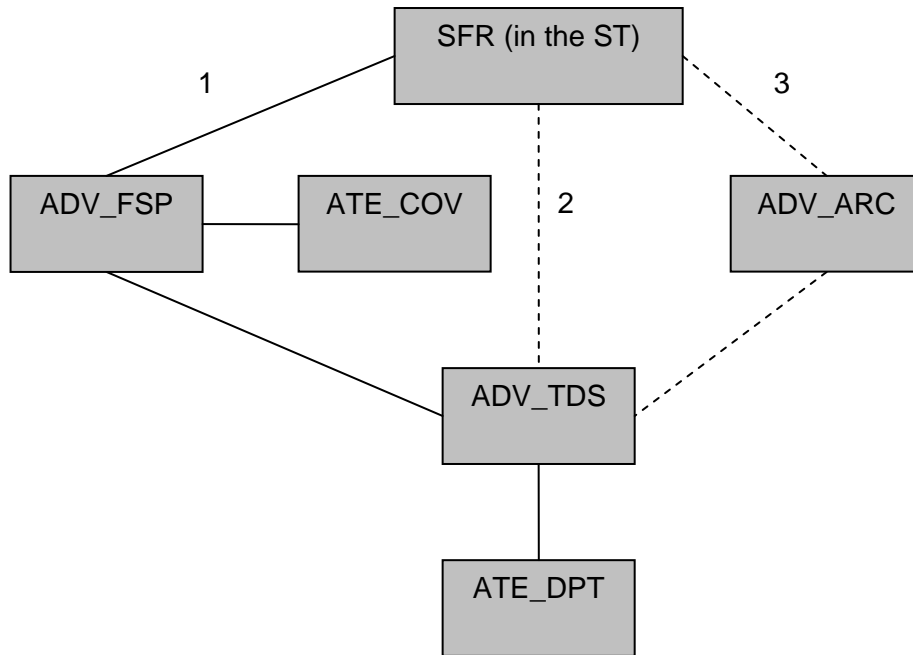


Figure 1: Refinement and Testing of different types of SFRs

### Explanation

- (i) The mappings from the level of SFRs in the Security Target to the level of FSP (labelled “1”), and from the FSP to TDS are identical to the refinement mappings, which are already constructed according to the CC today. New mappings (“2” and “3”) will be necessary from the ST to ADV\_TDS resp. ADV\_ARC. Here the developer will have to consider all SFRs, which can not (or can not completely) be mapped to external interfaces (and therefore to the FSP). Instead they can either be mapped directly to ADV\_TDS or the document Security Architecture Description (as required by ADV\_ARC) shall describe the realisation of these SFRs. In principle there are four possible cases for each SFR: It can be mapped completely to the FSP, completely to ADV\_TDS, completely to ARC or it can be realised in part by more than one of them.

As a summary the evaluator would need to map all SFRs to the functional specification, the TOE design and the security architecture according to the following rules:

1. All effects of an SFR visible at the external interfaces of the TSF need to be mapped to the TSFI.
2. All parts of an SFR that directly relate to testable functionality that is not directly visible at the external interfaces need to be mapped to the TOE design.
3. All parts of an SFR that relate to properties or "negative" requirements need to be mapped to the security architecture, which explains how the property or negative requirement is split up into testable functions and refined properties or negative requirements.

An additional mapping is necessary between ADV\_ARC and ADV\_TDS, because those aspects of properties (requirements of type 3), which are realised by mechanisms in the TOE design need to be traceable from ADV\_ARC to the TDS.

At first glance this construction seems to imply additional work for the developer, since ADV\_ARC needs to describe the realisation of architectural aspects of SFRs in addition

to the standard aspects of Start-up, Non-Bypassability, Domain-separation and Protection of the TSF. However, as far as today's SFRs already contain architectural aspects, this work was necessary anyway, for example by a specific argumentation in the FSP justifying that certain SFRs were not mapped to the TSFI but correctly realised otherwise. This modification should therefore only shift existing descriptions and even reduce the necessity for special case discussions.

- (ii) The connection between ATE\_COV and ATE\_DPT to FSP and TDS respectively, is the same as today. A new work item will consist of a section in ATE\_DPT, which justifies that not only the functions described traditionally in TDS but also the architectural properties described in ADV\_ARC are tested adequately.

Again this seems to be additional work, but this is put into perspective by two facts: Firstly, there is a connection between ATE\_DPT and ADV\_ARC already in CC 3.1, see ATE\_DPT.1-1, which talks about "explicitly cited mechanisms" in ADV\_ARC. Secondly chapter 14.2.2 ("Testing vs. alternate Approaches to verify the expected behaviour of functionality") of the CEM requires a justification, that functionality of the TOE, which is not externally visible, is tested adequately by alternative methods (e. g. module testing, source code review). All the cases, which are described as exceptions in today's CC, will become a normal case with Approach 2, which should mean no additional work for the developer. Once the procedure to cover ADV\_ARC in ATE\_DPT is routine, it should even be easier than having to discuss all architectural properties as exceptions from the "normal", externally visible case.

Remark: The preceding discussion shows again, that "internal requirements" and "architectural requirements" are already covered in CC 3.1, however, they are treated as exceptions rather than a normal phenomenon in various CC activities.

#### Which sections of the CC need modification

- Part 1 needs no relevant modifications (in particular the sections regarding STs and PPs stay unchanged), only the section "Terms and Definitions" may need small modifications, but we doubt even that, because all concept changes can be done inside Part 2.
- Part 2: The "paradigm" needs to be adapted, however even these changes are not as big as one might expect. It will be good to explicitly state that SFRs do not only cover security functionality regarding to TSFI but can also cover requirements for the architecture of the TOE. Moreover, it has to be clarified that the requirement that an SFR is testable doesn't imply functional tests in the narrow sense (triggering of external interfaces) but can also allow methods like simulation or source code review. So the requirement of testability means that there needs to be a possibility for the evaluator to determine, whether the SFR is implemented, but there is no predefined restriction on this possibility. It can be external testing, source code review, use of a debugger or a simulator etc.

In addition to this, guidance should be given to the developers and evaluators for each SFR defined in part 2 how to address a potential "architectural aspect" of the SFR. The existing appendices of part 2 could be used to provide this type of guidance. This would basically extend part 2 with additional guidance rather than restructuring the SFRs in part 2.

This would then allow to add new security functional requirements to part 2 that so far have been rejected since they address "negative" requirements or properties which were known to be hard to handle in the existing CC framework.

- Part 3 and CEM: Some amendments and modifications will be necessary for ADV\_FSP, ADV\_ARC and ATE\_DPT, as discussed before. Moreover it has to be checked, if cross references from other parts of the CC and CEM to these aspects need to be modified to reflect the more open approach.

Note: Since the developer vulnerability analysis was removed from AVA in CC 3.1, ADV\_ARC is now meant to cover the aspect of a "developer security analysis" of the

TOE's construction. We believe that a section, which clarifies that SFRs may not only require the implementation of specific functions at specific interfaces, but also the absence of undesired behaviour and the adherence to global security properties, will also help developers to understand the importance and the meaning of this "developer security analysis" in ADV\_ARC, because this analysis is an important part of the "testing" for the fulfilment of these aspects of SFRs.

The vulnerability analysis in AVA\_VAN might be slightly modified. The vulnerability analysis should analyse all parts of design and testing for violations of the Security Problem Definition. For example a design decision, which doesn't formally violate the SFRs might nonetheless violate the Security Problem Definition.

### Aspects of test methodology

The proposed modifications also give a good opportunity to describe the various test methods allowed by the CC in a more systematic way and to describe them as methods of equal validity.

The following test methods could therefore be described in more detail in the introduction for ATE (e. g. chapter 14.2.2 of the CEM):

- (i) Functional tests in the narrow sense (testing at external interfaces and the behaviour at these interfaces) are useful for those (parts of) SFRs, which can be mapped to TSFI and their behaviour.
- (ii) Tests of subsystems and modules using internal interfaces (for example using module-specific test software) are useful for those (parts of) SFRs, which are still mainly functional in the narrow sense, but not visible at external interfaces alone.
- (iii) Tests using tools like debuggers or simulators, which for example allow to inspect memory contents at run-time. In parts this is similar to the preceding category, but many tests, which are usually seen as penetration tests, can also be in this category (e. g. use of root-kits).
- (iv) Source code analysis: This is useful for verification of behaviour, which cannot be verified by one of the preceding methods. The source code analysis can exist in the variant of "reading and understanding by an expert", but it can also be tool-supported. For example there are various specific analysis tools to detect potential buffer overflow or stack overflow, tools, which allow to check, where in the memory specific data elements are handled, tools, which check, that specific exceptions can not occur, etc. See the document [Dewar], which was written for safety critical applications but is equally valid for security critical applications.
- (v) Penetration test: Tests, which simulate the behaviour of an attacker, who for example tries to bypass specified external interfaces or tries to provoke undesired behaviour at specified interfaces. Note, that such tests cannot only be used in the context of AVA but also in the context of ATE, because they can give assurance, that the TOE fulfils certain global properties.

These categories should be suitable to cover a state-of-the-art security level and to demonstrate, what it may mean, that an SFR is testable.

In addition, in CC Part 2 a reference to this section could be made as a hint for readers who want to understand, what the general requirement "an SFR has to be testable" means.

It is possible that additional text in ALC\_TAT might be useful in order to examine those tools mentioned in the preceding discussion, which are used to support source code inspection or defensive programming (e. g. tools which detect or prevent buffer overflow).

Results of applying such tools to a specific TOE would probably fit into ATE\_FUN. This broadens the scope of ATE\_FUN from interface testing to all kinds of applying test tools in a general sense to TOEs. Where applicable also theoretic analysis may replace tool based analysis as already allowed for ATE\_FUN today.

These Modifications in ALC\_TAT and ATE\_FUN address the following aspects:

1. Is the combination of tools and techniques used adequate to address the design principle?
2. Have the tools and techniques been applied for those parts of the TSF where they need to be applied (as defined in the security architecture documentation)?
3. Does the developer provide sufficient evidence that the results of the application of the tools within the development process demonstrate that the design principle is enforced?

## Annex

### ***Note on the limits of classical functional testing***

It is important to understand that even for classical functional requirements in the narrow sense, which are completely determined by their behaviour at the external interfaces, the purely functional testing at the external interfaces might not be sufficient.

The following thought experiment strengthens this statement:

A function of the TOE may have some parameter, which is longer than 16 Byte (e. g. an Input string). The programmer of this function has implemented a trojan horse, which works as follows: For exactly one value of the input parameter, which looks like a random value for outsiders, the function does something undesired (e. g. outputs secret internal data). For all other values of the parameter the function behaves absolutely correct.

The probability to find this undesired behaviour by functional testing is obviously not better than that for finding a secret key of 16 byte length by guessing or by brute force.

In other words: This behaviour is very unlikely to be detected by functional testing, although it is a purely functional behaviour.

Only a source code analysis would have a chance to identify this behaviour. (This could also be done tool based by using a tool, which highlights all sections of the code, where sensitive data are accessed).

The general rule behind this example is as follows: Functional testing (in the narrow sense) can only detect unintended, random errors. All types of errors, which are either intentional or not intentional, but based on a systematic problem (like an error on design or specification level) have a good chance to go undetected by functional testing.

In the interest of a good quality of the development process, it should not be left only to the evaluator's vulnerability analysis to detect such errors. Therefore specific test methods besides functional testing should be made part of development and therefore (in the CC language) of ATE, not only of AVA. The detection of security flaws only after development should be the exception.

These considerations are an important reason to make clear, that ATE should not handle functional testing in the narrow sense as the "rule" while other methods are only an exception.

In the literature (for example [Wilander et al]) the problem described above is often shown by the following picture:

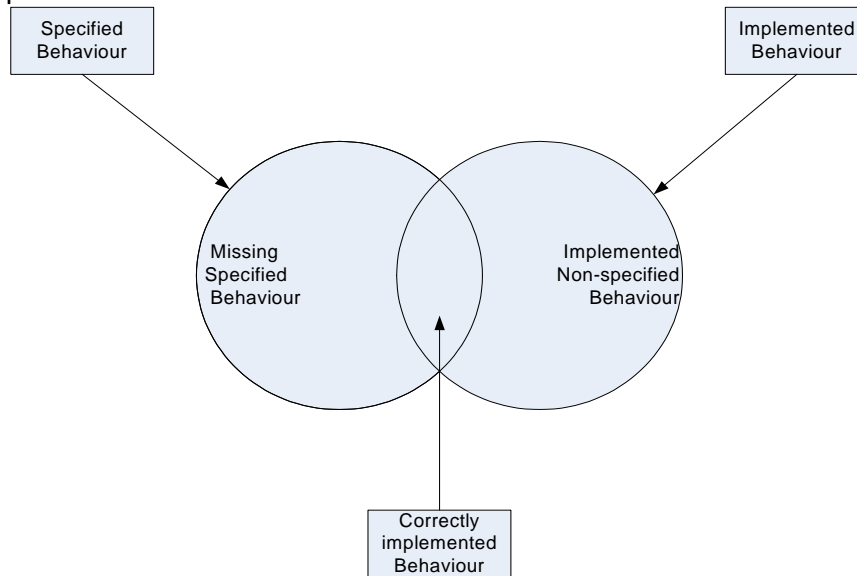


Figure 2: Relation between Specification and Implementation

The circle to the left represents the specified behaviour of a product, the circle to the right represents the implemented behaviour. The intersection of both circles then describes the correctly implemented behaviour. The area to the left describes the specified behaviour, which was not implemented, while the area to the right defines functionality, which was implemented but not specified. Both of these represent errors, but classical functional testing can only identify problems in the left side, not those to the right. On the other hand hackers and evaluators usually look for behaviour on the right side, which may represent security flaws. This shows that already during development methods are needed to reduce the amount of errors of the second type.

Further literature: [Wilander et al], [Firesmith].

### ***Implications for EAL considerations***

The implications of this approach is that any security functional requirement that needs to be treated in the evaluation via ADV\_ARC will only get tested via ATE\_DPT. SFRs that are of type 2 and 3 would therefore needed to be restricted to evaluations that include at least ADV\_ARC.1 and ATE\_DPT.1. One has to keep in mind that so far ADV\_ARC.1 is only included in EAL2 and higher and ATE\_DPT.1 is only included in EAL3 and higher. Unless this is changed, any useful evaluation of requirements of category 2 and 3 can only be performed at assurance levels of EAL3 and higher – which due to the nature of those requirements may make sense to not allow to specify those for low assurance levels.

## Bibliography

- [Dewar] Safety-critical design for secure systems, Robert B. K. Dewar, 2006, <http://www.embedded.com/columns/technicalinsights/190400498>
- [Firesmith] Engineering security Requirements, Donald G. Firesmith, Journal of Object Technology, 2003, [http://www.jot.fm/issues/issue\\_2003\\_01/column6](http://www.jot.fm/issues/issue_2003_01/column6)
- [Wilander et al] Security Requirements – A Filed study of Current Practice, John Wilander and Jens Gustavsson, Linköpings universitet, 2005 [http://www.ida.liu.se/~johwi/research\\_publications/paper\\_sreis2005\\_wilander\\_gustavsson.pdf](http://www.ida.liu.se/~johwi/research_publications/paper_sreis2005_wilander_gustavsson.pdf)