**EUROPEAN SPACE AGENCY**
OPS
GI


# SOFTWARE QUALITY AND CODING RULES

Reference: EGOS-QA-XX-TN-9007
Version: 1.3 Draft A
Date: 2005-04-25

| | | | |
|---|---|---|---|
| **Document Title:** | Software Quality and Coding Rules | | |
| **Document Reference:** | EGOS-QA-XX-TN-9007 | | |
| **Document Version:** | 1.0 Draft B | **Date:** | 2004-12-01 |
| **Abstract** | | | |
| | | | |

**Approval Table:**

| Action | Name | Function | Signature | Date |
|---|---|---|---|---|
| Prepared by: | Eduardo Gomez | | | 2004-10-29 |
| Verified by: | | | | YYYY-MM-DD |
| Approved by: | Please update EgosAuthorisationList property | | | YYYY-MM-DD |

**Authors and Contributors:**

| Name | Contact | Description | Date |
|---|---|---|---|
| Eduardo Gomez | Eduardo.Gomez@esa.int | Author | 2004-10-29 |
| Nuno Sebastião | Nuno.sebastiao@esa.int | Contributor | 2004-12-01 |

**Distribution List:**



**Document Change Log**

| Issue | Date | Description |
|---|---|---|
| | YYYY-MM-DD | |

**Document Change Record**

| | | | |
|---|---|---|---|
| **DCR No:** | 00 | **Originator:** | |
| **Date:** | | **Approved by:** | |
| **Document Title:** | Software Quality and Coding Rules | | |
| **Document Reference:** | EGOS-QA-XX-TN-9007 | | |
| **Page** | **Paragraph** | **Reason for Change** | |
| | | | |

# TABLE OF CONTENTS



# TABLE OF TABLES



 

# TABLE OF FIGURES

**Error! No table of figures entries found.**

 

# 1. Introduction

## *1.1 Purpose*

The objective of this document is to provide the definition of a software quality and static model.

The quality model is expressed either by a set of values that can be measured on the source code and has to fall within a given range (e.g. comment frequency) or as a set of specific conditions that need to be verified at one or more points in the lifecycle (e.g. “the architectural documentation has to be based on the object-oriented approach” or “all the requirements have to be covered by a validation mechanism”).

The static model defined in this document presents a classification of the rules defined in [] and presents methods for checking the compliance of software code:
In general, measurements on the source code are referred to as “metrics”, while the verification of requirements that need to be met are referred to as “checks”.

## *1.2 Scope*

This quality model is applicable to all development projects related to infrastructure software to be accepted by OPS-GI. The quality requirements implicitly or explicitly contained in this document apply

- To the work performed by the contractor personnel
- To the acceptance process performed by OPS-GI personnel

## *1.3 Glossary*

### 1.3.1 Quality model (software)

Set of characteristics and the relationships between them which provide the basis for specifying quality requirements and evaluating quality [ISO/IEC 9126--1:2001] (from [RD-2]).

### 1.3.2 Static model (software)

Set of coding standards that, in accordance to the quality model, define the rules to which the software product source code shall comply with [ISO/IEC 9126--1:2001] (from [RD-2]).

### 1.3.3 Metric

Defined measurement method and the measurement scale
NOTE 1 Metrics can be internal or external, and direct or indirect.
NOTE 2 Metrics include methods for categorising qualitative data.
[ISO/IEC 9126--1:2001] (from [RD-2])

### 1.3.4 Reusability

Degree to which a software module or other work product can be used in more than one computer program or software system [IEEE 610.12:1990] (from [RD-2]).

### 1.3.5 Portability (a quality characteristic)

capability of software to be transferred from one environment to another (from [RD-2]).

### 1.3.6 Check

Verification of a quality requirement. Note: this definition is specific to this document

### 1.3.7 Acronyms

| Acronyms | Description |
|---|---|
| | |

 

### 1.3.8 Definition of Terms

| Terms | Description |
|---|---|
| | |

## *1.4 References*

### 1.4.1 Applicable documents

| Ref. | Document Title | Issue and Revision, Date |
|---|---|---|
| [AD-1] | | |

### 1.4.2 Reference documents

| Ref. | Document Title | Issue and Revision, Date |
|---|---|---|
| [RD-1] | Space Engineering: software – Part 1: Principles and requirements | ECSS-E-40 part 1B, 28.11.04 |
| [RD-2] | Space product assurance: software product assurance | ECSS-Q-80B, 10-10-03 |
| [RD-3] | SPEC TN3, Space Domain Specific Software Product Quality Models, Requirements and Related Evaluation Methods | TBS, 20-02-2002 |
| [RD-4] | Software engineering—Product quality—Parts 1, 2, 3 and 4 | ISO/IEC 9126, 15-06-01 |
| [RD-5] | Software Development Practices | 1, 2004-12-01 |
| [RD-6] | C++ Naming and Coding Conventions | S2K-MCS-TN-9003-TOS-GIC, issue 1.1, 04-11-2003 |
| [RD-7] | Generic Coding Conventions | S2K-MCS-TN-9004-TOS-GIC, issue 1.1, 22-10-2003 |
| [RD-8] | Rulechecker C++ Reference Manual | Logiscope 6.1, May 2004 |
| [RD-9] | C and C++ Coding Standards | BSSC (2000)1, issue 1, 30, 03,2000 |

## *1.5 Document Overview*

Section 2 of the document defines the simplified quality model and how to apply it. More specifically:

- Paragraph 2.1 provides an introduction and some background details;
- Paragraph **Error! Reference source not found.** explains how to apply the model from a practical point of view;
- Paragraph 2.3.2 explains how the metrics and checks defined in this document map into quality properties as defined in [RD-3](SPEC model)
- Paragraph 2.5 defines the specific checks that need to be applied

Detail traceability information between the SPEC model and the metrics and checks defined here can be found in 0, while Annex A contains a table that summarizes the SPEC model.

 

# 2. Definition of the quality model

## 2.1 Introduction

A quality model is composed of several main properties (sometimes also called factors) that characterised the software quality. These properties are further characterised by sub-properties (sometimes known as criteria), which, in turn can be measured using a specific methodology, known as a metric. The quality model is completed by assigning a value (or range) to each metric. The quality model presented in this document is based on [RD-3] and [RD-4].

There are two kinds of metrics. The first group consists of measurements that can be obtained by inspecting the code (e.g. number of lines per method). Such measurements will provide a numerical result that has to fall within the specified range. These metrics are suitable for automation using a tool. The name "**metric**" is used in this document to refer to them.

The selection of the values for the metrics is mainly based on the values provided in [RD-3]. If the metric in question is not present in this document, then a suitable value has been obtained based on common practice. All the values provided for the metrics will need to be tuned, as experience is gained.

The metrics are based on C++, but the same values have been applied for JAVA. This may need to be reviewed in the future based on the experience gained. Some of the metrics are not relevant for Java. The list of specific metrics to be applied for C++ and for Java is provided in **Error! Reference source not found.**

The second kind of metrics consists of **checklists**. The output of the metric is discrete, in the sense that it can be 1 if the checklist is fulfilled or 0 if not. Not all the items in the checklists maybe always applicable, therefore some items in the checklist may simply be disregarded if proper justification is provided. This set of checklists is strictly based on [RD-3] and has been adapted to the type of developments typically carried out under OPS-GIC control. Safety related items, in particular, have been disregarded in most cases, since infrastructure software developed under OPS-GIC contracts has not yet been validated to be used on environments with safety implications.
For clarity reasons **Metrics defined in** [RD-3] (SPEC) **are referred to as SPEC metrics and the model defined in the same document as the SPEC quality model**. The complete quality model tree as defined in [RD-3], with properties, sub-properties and SPEC metrics is presented in Annex A.

## 2.2 The metrics

This document attempts to simplify the application of the SPEC quality model by defining 10 checklists (instead of the 85 metrics defined in [RD-3] which are listed in Table 1.

| Documentation Checks |
|---|
| Effectiveness checks |
| Operability checks |
| PA checks |
| Process checks |
| Reliability checks |
| Reusability checks |
| Safety checks |
| Static analysis |
| Traceability checks |

Table 1 - List of checks to be applied

In addition to these checklists, 56 metrics and their values have been defined. The collection of these metrics can be easily automated. A configuration file for the tool Logiscope (from Telelogic) can be provided. The configuration file has adopted as metric value the one in [RD-3] if the equivalent SPEC metric exists and the default Logicope value if an equivalent SPEC metric does not exist or if the default

 

Logiscope value is more restrictive (or reasonable). The full list of metrics is provided in Table 2. The last column in the table specified whether the metric is also applicable to Java code.

| name | Logiscope aggregated metric | max value | min value | applicability | Applicable to Java |
|---|---|---|---|---|---|
| Attribute hiding factor (MOOD) | ap_ahf | 1 | 0.7 | application | No |
| Attribute inheritance factor (MOOD) | ap_aif | 0.6 | 0.3 | application | No |
| Coupling between objects | ap_cbo | | | application | No |
| Number of Levels in the Call Graph | ap_cg_levl | 9 | 2 | application | No |
| Number of application classes | ap_clas | | | application | No |
| Coupling factor (MOOD) | ap_cof | 0.18 | 0.03 | application | No |
| Hierarchical Complexity of the Inheritance Graph | ap_inhg_cpx | 2 | 1 | application | Yes |
| Number of Levels in the Inheritance Graph | ap_inhg_levl | 4 | 1 | application | Yes |
| Method hiding factor (MOOD) | ap_mhf | 0.4 | 0.1 | application | No |
| Method inheritance factor (MOOD) | ap_mif | 0.8 | 0.6 | application | No |
| Polymorphism factor (MOOD) | ap_pof | 1 | 0.3 | application | No |
| Rate of class autonomy | AUTONOM | 100 | 30 | class | No |
| Average coupling between objects | AVG_CBO | 10 | 0 | application | No |
| Average of the VG of the application's functions | AVG_VG | 5 | 0 | application | Yes |
| Average size of statements | AVGS | 7 | 1 | funcion | No |
| Relative call graph Hierarchical complexity | cg_hiercpx | 5 | 1 | funcion | No |
| Number of relative call graph levels | cg_levels | 12 | 1 | funcion | No |
| Relative call graph Structural complexity | cg_strucpx | 3 | 0 | funcion | No |
| Relative call graph System testability | cg_testab | 1 | 0 | funcion | No |
| Coupling between classes | cl_cobc | 12 | 0 | class | No |
| Number of dependent methods | cl_dep_meth | 6 | 0 | class | No |
| Lack of cohesion of methods | cl_locm | | | class | No |
| Weighted Methods per Class | cl_wmc | 25 | 0 | class | Yes |
| Comments frequency | COMF | 0.99 | 0.3 | funcion | Yes |
| Class Comments Frequency | COMFclass | 0.99 | 0.3 | class | Yes |
| Number of destructuring statements | ct_bran | 0 | 0 | funcion | Yes |
| Number of out statements | ct_exit | 1 | 0 | funcion | Yes |
| Number of nestings | ct_nest | | | funcion | Yes |
| Number of paths | ct_path | 60 | 1 | funcion | Yes |
| Cyclomatic number (VG) | ct_vg | 12 | 1 | funcion | Yes |
| Number of direct used classes | cu_cdused | 7 | 0 | class | Yes |

 

| name | Logiscope aggregated metric | max value | min value | applicability | Applicable to Java |
|---|---|---|---|---|---|
| Number of direct users classes | cu_cdusers | 4 | 0 | class | Yes |
| Number of callers | dc_calling | 7 | 0 | funcion | No |
| Number of direct calls | dc_calls | 5 | 0 | funcion | No |
| Number of local variables | dc_lvars | 5 | 0 | funcion | No |
| Encapsulation rules | ENCAP | 5 | 0 | class | No |
| Fan In | FAN_IN | 4 | 0 | funcion | No |
| Fan in of a class | FAN_Inclass | 15 | 0 | class | No |
| Fan Out | FAN_OUT | 4 | 0 | funcion | No |
| Fan out value of a class | FAN_OUTclass | 20 | 0 | class | No |
| Number of parameters passed by reference | ic_paradd | 2 | 0 | funcion | No |
| Number of function parameters | ic_param | 5 | 0 | funcion | Yes |
| Number of parameters passed by value | ic_parval | 2 | 0 | funcion | No |
| Number of distinct uses of external attributes | ic_varpe | 2 | 0 | funcion | No |
| Number of base classes | in_bases | 3 | 0 | class | Yes |
| Number of children | in_noc | 2 | 0 | class | Yes |
| Number of relative call graph call-paths | IND_CALLS | 30 | 1 | funcion | No |
| Number of statements | lc_stat | 20 | 1 | funcion | Yes |
| Number of levels | LEVL | 4 | 1 | funcion | Yes |
| Percentage of non-member functions | NMM_Ratio | 10 | 0 | application | No |
| Ratio of recursive edges on the call graph | RECU_Ratio | 5 | 0 | application | No |
| Specializability | SPECIAL | 25 | 0 | class | Yes |
| Testability | TESTAB | 100 | 0 | class | Yes |
| Ratio of repeated inheritances in the application | URI_Ratio | 10 | 0 | application | Yes |
| Usability | USABLE | 10 | 0 | class | Yes |
| Vocabulary frequency | VOCF | 4 | 1 | funcion | No |

Table 2 - List of metrics (based on C++)

In summary, applying this quality model consist of:

- Measure the value of the metrics in Table 2, combine them using the provided models and identify these areas that should be improved. It should be noted the failure of one single metric does not provide enough information to evaluate the code. Metrics have to be combined using the models. The measurement can be fully automated using Logiscope and batch processing (see [RD-5]).
- Go through the checklists at the beginning and end of the lifecycle phases of the project ensuring that the relevant requirements are taken into account and verified

## *2.3 The quality models*

The experience in using the model has shown that the maintainability metrics cannot be used without further processing. One single metric does not provide reliable information about a piece of code. A

 

combination of them has to be used instead. Unfortunately [RD-3] does not define any criteria to classify the combination of the values of the metrics. Therefore two approaches are proposed. The first approach is based in using the model proposed by Logiscope (with some minor corrections taken from SPEC). The second approach is based on mapping SPEC metrics to Logiscope metrics and defining a "SPEC" quality model based on them so that the code can be automatically reviewed.

### 2.3.1 The "logiscope" quality model for maintainability

The so-called maintainability model is based on factors (goal properties in [RD-3] terminology) and each factor is based on a number of criteria (properties in [RD-3] terminology). Each criterion corresponds to a weighted combination of metrics. The next sections will show the models used at the levels of application, class and methods.

For each factor and criterium, Logiscope provides a scale of results (poor, fair, good and excellent). Results labelled poor (be it a function, class or application level) should be investigated and corrected or justified.

#### *2.3.1.1 Logiscope model for applications*

| Factor | Criterion |
|---|---|
| application_MAINTAINABILITY | application_TESTABILITY |
| | application_STABILITY |
| | application_CHANGEABILITY |
| | application_ANALYZABILITY |

Table 3 - Factors at aplication level in the Logiscope model

| Criterion | Metric | Weight |
|---|---|---|
| application_ANALYZABILITY | RECU_Ratio | 1.0 |
| | ap_cof | 1.0 |
| | ap_mif | 1.0 |
| | ap_aif | 1.0 |
| | AVG_CBO | 1.0 |
| | ap_inhg_levl | 1.0 |
| application_CHANGEABILITY | ap_mif | 1.0 |
| | ap_pof | 1.0 |
| | NMM_Ratio | 1.0 |
| | URI_Ratio | 1.0 |
| | ap_inhg_levl | 1.0 |
| application_STABILITY | ap_cof | 1.0 |
| | ap_ahf | 1.0 |
| | ap_mhf | 1.0 |
| | ap_inhg_cpx | 1.0 |
| | AVG_CBO | 1.0 |
| application_TESTABILITY | ap_cg_levl | 1.0 |
| | ap_ahf | 1.0 |
| | ap_mhf | 1.0 |
| | NMM_Ratio | 1.0 |
| | AVG_VG | 1.0 |

Table 4 - Criteria at application level in the Logiscope model

#### *2.3.1.2 Logiscope for classes*

| Factor | Criterion |
|---|---|
| class_MAINTAINABILITY | class_TESTABILITY |

 

| | |
|---|---|
| | class_STABILITY |
| | class_CHANGEABILITY |
| | class_ANALYZABILITY |
| class_REUSABILITY | class_ANALYZABILITY |
| | class_SPECIALIZABILITY |
| | class_USABILITY |

Table 5 - Factors at class level in the Logiscope model

| Criterion | Metric | Weight |
|---|---|---|
| class_ANALYZABILITY | COMFclass | 1.0 |
| | FAN_OUTclass | 1.0 |
| | FAN_INclass | 1.0 |
| | cl_dep_meth | 1.0 |
| | in_bases | 1.0 |
| | cl_wmc | 1.0 |
| class_CHANGEABILITY | SPECIAL | 1.0 |
| | USABLE | 1.0 |
| | ENCAP | 1.0 |
| class_STABILITY | cu_cdusers | 1.0 |
| | cl_cobc | 1.0 |
| | In_noc | 1.0 |
| | AUTONOM | 1.0 |
| class_TESTABILITY | cu_cdused | 1.0 |
| | TESTAB | 1.0 |
| | in_bases | 1.0 |
| class_USABILITY | AUTONOM | 1.0 |
| | ENCAP | 1.0 |
| | USABLE | 1.0 |
| class_SPECIALIZABILITY | AUTONOM | 1.0 |
| | ENCAP | 1.0 |
| | SPECIAL | 1.0 |

Table 6 - Criteria at class level in the Logiscope model

#### *2.3.1.3 Logiscope foir methods and functions*

| Factor | Criterion |
|---|---|
| function_MAINTAINABILITY | function_TESTABILITY |
| | function_STABILITY |
| | function_CHANGEABILITY |
| | function_ANALYZABILITY |
| relativeCall_MAINTAINABILITY | relativeCall_TESTABILITY |
| | relativeCall_STABILITY |
| | relativeCall_ANALYZABILITY |

Table 7 - Factors at method level in the Logiscope model

| Criterion | Metric | Weight |
|---|---|---|

 

| | | |
|---|---|---|
| Function_TESTABILITY | ic_param | 1.0 |
| | ct_path | 1.0 |
| | LEVL | 1.0 |
| | dc_calls | 1.0 |
| function_STABILITY | ic_param | 1.0 |
| | dc_calls | 1.0 |
| | ct_exit | 1.0 |
| | ic_varpe | 1.0 |
| | dc_calling | 1.0 |
| Function_CHANGEABILITY | ct_bran | 1.0 |
| | VOCF | 1.0 |
| | dc_lvars | 1.0 |
| | ic_param | 1.0 |
| function_ANALYZABILITY | COMF | 1.0 |
| | AVGS | 1.0 |
| | lc_stat | 1.0 |
| | ct_vg | 1.0 |
| relativeCall_ANALYZABILITY | cg_levels | 1.0 |
| | cg_strucpx | 1.0 |
| RelativeCall_STABILITY | cg_hiercpx | 1.0 |
| | IND_CALLS | 1.0 |
| relativeCall_TESTABILITY | IND_CALLS | 1.0 |
| | cg_testab | 1.0 |

Table 8 - Criteria at method level in the Logiscope model

### 2.3.2 The "SPEC" model for maintainability

Using the mapping defined in A.2, it is possible to create a second model that will report any non-compliance with a SPEC maintainability model. The evaluation of the results of this model is based on a discrete failed/passed output for each factor and criterium which can be traced to a specific metric. Non-compliances with SPEC shall be investigated and corrected or justified. A known problem of SPEC is the use of switch constructs as they can easily break the maximum cyclomatic number. If such an error is reported it can be ignored.

#### *2.3.2.1 Logiscope metrics used in SPEC*

The Logiscope metrics used to build the SPEC model are reported in A.2. To simplify the reading of the tables, the equivalent SPEC metric has been provided in an extra column.

#### *2.3.2.2 SPEC for applications*

| Factor | Criterion |
|---|---|
| application_SPEC | application_SPEC_modularity |

 

Table 9 - Factors at application level in the SPEC model

| Criterion | Metric (logiscope) | Metric (SPEC) | Min | Max | Weight |
|---|---|---|---|---|---|
| application_SPEC_modularity | ap_cbo | MAIN.MO.M3 | - | - | 1.0 |

Table 10 - Criteria at application level in the SPEC model

#### *2.3.2.3 SPEC for classes*

| Factor | Criterion |
|---|---|
| class_SPEC | class_SPEC_Modularity |
| | class_SPEC_Documentation |

Table 11 - Factors at class level in the SPEC model

| Criterion | Metric (Logiscope) | Metric (SPEC) | Min | Max | Weight |
|---|---|---|---|---|---|
| class_SPEC_Documentation | COMFclass | DOQ.AD.M1 | .3 | 1 | 1.0 |
| class_SPEC_Modularity | cu_cdused | MAIN.MO.M1 | 0 | 7 | 1.0 |

Table 12 Criteria at class level in the SPEC model

#### *2.3.2.4 SPEC for methods and functions*

| Factor | Criterion |
|---|---|
| SPEC | function_SPEC_verifiability |
| | function_SPEC_modularity |
| | function_SPEC_Documentation |
| | function_SPEC_Analysablity |

Table 13 - Factors at method level in the SPEC model

| Criterion | Metric (logiscope) | Metric (SPEC) | Min | Max | Weight |
|---|---|---|---|---|---|
| function_SPEC_Analysablity | LEVL | MAIN.AN.M4 | 1 | 4 | 1.0 |
| | lc_stat | MAIN.AN.M7 | 1 | 20 | 1.0 |
| | AVGS | MAIN.AN.M6 | 1 | 7 | 1.0 |
| | ct_vg | MAIN.AN.M2 | 1 | 12 | 1.0 |
| function_SPEC_Documentation | COMF | DOQ.AD.M1 | .3[1] | 1 | 1.0 |
| function_SPEC_modularity | lc_stat | MAIN.AN.M7 | 1 | 20 | 1.0 |
| function_SPEC_verifiability | ct_vg | MAIN.AN.M2 | 1 | 12 | 1.0 |

Table 14 - Criteria at method level in the SPEC model

## *2.4 Mapping to SPEC*

This section presents a simplified mapping to SPEC where only goal properties are taken into account. A detailed mapping (based on metrics) can be found in section 3. Metrics are only relevant for the property maintainability[2]. The mapping between checks and goal properties is provided in Table 15. For C++ code, the metric URI_Ratio (Ratio of repeated inheritances in the application) can be mapped to the existing coding standard rule PC.26.C++.D, which completely forbids diamond shaped inheritances (the value of this metric should therefore been set to zero for C++).

| Goal property as defined in [RD-3] | Checklist used |
|---|---|

[1] Value changed with regard to the original SPEC proposal

[2] The metric number of application classes (ap_class) is also mapped to re-usability, however, there are no values imposed for this metric and therefore it is not considered.

 

| Documentation Quality | Documentation Checks |
|---|---|
| Functionality | Traceability checks<br>PA checks<br>Documentation checks<br>Static analysis |
| Maintainability | Static analysis<br>Documentation checks<br>PA checks |
| Operability | Documentation checks<br>Operability checks |
| Reliability | Reliability checks |
| Re-usability | Reusability checks |
| Suitability for safety | Safety checks |
| Software Development Effectiveness | Process checks |
| System Engineering Effectiveness | Effectiveness checks |

Table 15 - Association of checks and goal properties

## *2.5 Checklists*

### 2.5.1 Documentation Checks

The check procedure shall ensure that:

ARCHITECTURAL DESIGN DOCUMENTATION
- There are no ambiguities
- The object oriented approach is applied consistently.
- A top-down approach I used to decompose the software into components
- Design trade-offs are properly documented.
- The architectural design contains a `physical model', which describesthe design of the software using implementation terminology.
- For each component the following information is detailed:data input;functions to be performed;data output.
- Re-used components are clearly identified.
- COTS are clearly identified.
- Data structures that interface components are defined.
- Data structure definitions include the:
* Description of each element (e.g. name, type, dimension);
* relationships between the elements (i.e. the structure);
* range of possible values of each element;
* initial values of each element.
- The control flow between the components is defined
- The architectural design defines the major components of the software and the interfaces between them.
- The architectural design defines or references all external interfaces
- The architectural design documentation covers all the software requirements
- CPU and memory shall be confirmed in accordance with the margins set by the system requirements.
- A table cross-referencing software requirements to parts of the architectural design is provided
- The architectural design is sufficiently detailed to allow the project leader to draw up a detailed implementation plan and to control the overall project during the remaining development phases
- The detailed design defines all components

REQUIREMENTS
- There are no conflicting requirements
- There are no ambiguous requirements
- The set of requirements defines the system completely
- There are no replicated erquirements

 

USER DOCUMENTATION
- The User manual contains an overview of the system
- Each operation is described in the User manual
- Cautions and warnings are described in the User manual
- For each operation the User manual provides:
* set-up and initialisation
* input operations
* What results to expect
- The User manual provides probable errors and possible causes
- The User manual reports error messages and recovery procedure
- The user manual covers all software requirements
- The complete set of documentation is available on-line and can be easily accessed by typing keywords or using a context-based help environment

OTHER DOCUMENTATION
- Traceability matrixes ensure that each input to a phase is traceable to an output of that phase
- Traceability matrixes ensure that each output to a phase is traceable to an input of that phase
- The validation tests are properly documented reporting test design, cases, procedures and reports
- The test documentation include traceability (validation tests shall be traceable to the user requirement or upper level technical specification)

### 2.5.2 effectiveness checks

The check procedure shall ensure that:
- The system components interfaces have been analyzed
- All Interface control Documents are available
- The correct implementation of the interfaces of the software component
with other system components has been verified

### 2.5.3 Operability checks

The check procedure targets the installation and checkout phases and shall ensure that

- A reliable estimate of how long the installation/upgrade should take is described in the installation kit
- The installation documentation includes an accurate specification of all of:
* required platform
* Operating system
* Other SW required
* other prerequisites
- The software automatically checks that all required installation files etc. are present on the distribution medium and advices the operator if files are missing
- There a set of step-by-step instruction in a manual computer-based installation guide or easily accessed file in the distribution medium that efficiently and unambiguously guides the installation
- If options relating to the customisation of the software are offered to the operator, default options are proposed and explained in understandable terms
- Help is provided to operators on the choice of non-default options
- Installation is allowed in operator-specified directory or disk
- The operator can roll-back to a previous step in the procedure without prejudice (i.e. it is not necessary to re-start whole procedure if , for example, the operator changes his mind about a choice of option)
- If the installation/upgrade process automatically modifies existing files, such as those associated with the environment (e.g. PATH), the operator is informed
- The installation procedure be cancelled at any time
- If new directories are automatically created, the operator is informed
- If new files are automatically added to existing directories the operator is informed
- Product-parts are suitably identified, inventoried and referenced without confusion in installation/upgrade instructions
- The distribution medium is protected so that operators cannot inadvertently overwrite it
- The execution of the installation procedure is problem free
- Names of application-specific menus, functions, commands, keywords etc. are self-explanatory.

 

- Name of Application-specific menus, functions, commands, keywords are easy to remember
- Application-specific Icons and other displayed objects have an obvious meaning
- Actions on application-specific objects are generic
- At least 90% of each of the following conform to a common layout:
* Data entry panels and forms
* Data display panelsFields
* Menus
* Command-lines
* Prompts
* Error messages
* Help information
- There are consistent conventions for:
* Operation of selection mechanisms
* Actions of non-application-dependent function keys, buttons
* Operations on windows
- Display screens and panels are well laid-out and uncluttered
- Character-sets, icons, graphics, etc. have good legibility
- Window manipulation functions include a:
* Close function
* Moving function
* Restore function
* Resize function
* Minimise function
* Maximise function
- Allows operator to save files in operator-specified directory/filename
- Prompts operator to save work whenever closing or quitting (i.e. does not just exits)
- Open files with the proper extension
- Files export to other applications
- Print to networked printer
- On completion of a function, the operator is told (or can easily find out) which functions he/she can invoke next (e.g. by means of a function menu-bar)
- On completion of a function, the operator can directly move the program into a familiar, 'home' state (e.g. into a root-menu)
- The design of menus, commands, selection mechanisms, etc. lets the operator move quickly to where he/she wants to go:
* minimum number of elementary operator-actions required to move between the two most 'distant' functions
* number of elementary operator-actions required to select the 'deepest' function from the opening program-state
- Input data-fields are validated to the maximum possible extent given the context
- Elementary input-device mis-operations are trapped and signaled appropriately
- Interface recover gracefully form anticipated operator errors (e.g. (invalid inputs)
- Operator error-messages show the location and type of errors, and explain how to correct the error (this information can be either on-line or located in the operator documentation)
- The system automatically proposes corrections to clearly correctable operator-errors
- Operator error-correction and roll-back functions are adequate (e.g. during an update, original data should be recallable)
- The system warns the operator before implementing operator-actions that could have serious consequences
- Where important, the system forces the operator to take data 'backups' to regularly change passwords and to take other essential security measures.
- Instructions and functions are provided to restore the application after all but the most unlikely types of system and program 'crashes'.
- There is immediate feedback to show that the operator's action is understood by the system
- The system keeps the operator informed on the progress of operations that require more than 5 consecutive seconds of internal processing (i.e. the operator is not left wondering if the system has 'died' during lengthy 'silent' functions)
- Advanced functions are 'hidden' or packaged so that beginners are not overwhelmed
- Operators are offered suitable default values and/or assistance in setting parameter values

 

- 'Fast-Track' is available to experienced operators:
* command-language
* operator-definable macros
* operator-definable function-keys
- WYSIWYG by output editing/formatting functions is used
- Elementary operator-actions are processed quickly.
- The operator can suspend a current function, invoke other functions, and then resume the suspended function
- The operator can choose his/her own names for persistent information-sets to which he/she makes direct reference (data-files, directories, tables, etc.)
- The operator can change at least 90% of the following aspects of the operator-interface :
* The colors of displayed elements and objects
* The position of displayed elements and objects on the screen
* The size of displayed elements and objects
* The font size of alphanumeric characters
* The mapping of functions to key, buttons etc
* The name of operator-functions
* The arrangement of data in display-panels
* The arrangement of data in printed/plotted output
* The rate at which information is displayed
* The selection of I/O devices
- Technical support information is identical to that stated in documentation
- Detailed Help information is available
- Help system is on line and easy to use
- Hypertext links jump to proper subject
- Glossary and search capabilities work correctly
- No specific Hw set-up procedure shall be needed
- There shall be no need to re-boot the hardware after an error
- It shall be possible to execute all functions without any Hw constraint
Function execution shall not depend on the Hw
- Software outputs shall not be related to Hw configuration
- The software shall not require Hw related parameters
- Software shall not stop stop because of lack of resources (e.g.: memory, disk space)
- The software shall be easy to configure
- The man/machine interface software shall be in line with standards
- the usage of software shall be described in a user manual
- Error messages sent by the software shall be documented in a user manual
- Error messages shall not be related to Hw
- It shall be possible to identify easily which version of software is running
The user manual shall be « self sufficient »

### 2.5.4 PA checks

The check precedure shall ensure that
- All verification activities planned in the PA plan have been executed and documented in the progress reports
- Verification activities take into account the criticality of the software
- Verification activities ensure that the product meets the quality, reliability, maintainability and safety requirements (stated in the requirements)
- Walkthrough inspection are executed.
- Internal audits are executed before the release of the software
- Progress reports include statistics on:
* time spent on SPR corrections. An estimate of the number of hours spent for every 1000 lines of code shall be provided.
* mean time to diagnose (MTTD) the cause of the failure in hours
* Number of problems introduced as a consequence of SPR correction

- The report warns if any of the following values have been exceeded:

 

* 15 hours per 1000 lines of code
* 4 hours for MTTD
* more than 1 new problem for each solved SPR

### 2.5.5 Process checks

The check procedure shall ensure that:
- All the parties involved are ISO 9001 certified or present a S4S profile compliant with the following values for the related processes (if the processes are relevant to their activities)
CUS.1.1-Acquisition preparation 3
CUS.1.2-Supplier selection 3
CUS.1.3-Supplier monitoring 3
CUS.1.4-Customer acceptance 4

- Coding standards are enforced and verified
- Reliability, maintainability and safety targets are defined (e.g. in requirements and checked)
- There are clear guidelines for configuration and change management and they are respected
- There is a software quality assurance plan that clearly establishes measurable quality objectives and the methods to measure them
- There is a Software Development Plan that defines the objectives, standards and software life-cycle(s) to be used in the software development process
- There is a Software Configuration Management Plan that establishes the methods to be used to achieve the objectives of the software configuration management process throughout the software life cycle.
- There is a Software Verification Plan specifying the verification procedures to satisfy the software verification objectives (note: this may be presented as part of the software quality assurance plan).

- The Outputs of Software Requirements and architecture engineering Process are verified regarding:
* compliance of software requirements with system requirements
* accuracy and consistency of software requirements
* compatibility of software requirements with target computer
* verifiability of software requirements
* conformity of software requirements to standards
* traceability of software requirements to system requirements
* accuracy of algorithms
* software architecture compatibility with requirements
* software architecture compatibility with target computer
* software architecture verifiability
* software architecture conformity to standards
* confirmation of software integrity

- The Outputs of Software design and implementation engineering Processes are verified regarding
* source code compliance with requirements
* source code compliance with software architecture
* source code verifiability
* source code conformity to standards
* source code traceability to requirements
* source code accuracy and consistency
* The outputs of the software integration completeness and correctness

- The inputs of the software validation process are verified regarding
* Completeness of the testing specification
* Completeness of the validation activities (coverage of requirements)

- The outputs of the software validation process are verified regarding
* Completeness in the execution of the testing activities
* Completeness in the execution of the validation activities

 

### 2.5.6 Reliability checks

The check procedure targets the operation of the system and shall ensure that:
- Illegal operations are always trapped and handled by the software
- Data corruption never occurs
- Mean Time To Failure (MTTF) is greater than 1000 hours
- No critical failure occurs
- Mean Time To Restart is defined in the requirements and tested successfully
- Mean Time To Recover is defined in the requirements and never exceeded

Design components and software modules are associated to specific reliability values and tested successfully (MTTR and MTTF)
- A dependability analysis has been executed during software specification and the software requirements have been enhanced in line with this analysis (the analysis should provide at least the list of critical functionality)
- Test execution is run in accordance to the module criticality classes
- Possible fault patterns have been identified (during the dependability analysis) and failure avoidance mechanisms have been implemented and successfully tested

### 2.5.7 Reusability checks

The check procedure targets the documentation and shall ensure that:
A) In case of software design for reuse
- There are requirements targeting the future reuse
- The organization and content of the user manuals is suitable for reuse of the software (in-line with requirements)
- The definition of reusability limitation and criticality class reusability is stated
- Software units intended to maintain its correct behaviour in another environment are tested for that

B) In case of software reused from another mission/source
- Justification of re-use with respect to requirements baseline is provided

### 2.5.8 Safety checks

The check procedure shall ensure that:
- A safety analysis has been executed
- Criticality Classes are identified (often by using traces from critical requirements to architecture). The architectural design documentation shall clarify at least which components are mission critical
- A Software Safety Plan is defined (if applicable), specifying activities to be carried out, the implementation schedule and the resulting products.
- Hardware/software interactions for safety are identified and evaluated (if applicable)
- ISVV activities plan are provided (if applicable), to identify, taking into account the outcomes of the software safety analysis, the ISVV activities to be performed. The ISVV activities include reviews, inspections, testing and audits
- Software safety requirements are defined (involving analysis of system safety requirements, hardware, software and user interfaces, and areas of system performance)

### 2.5.9 Static analysis

The analysis shall ensure that:
- There are no SPRs open in the system
- The ration SPR/lines of code is less than 0.001 (I.e. no more than 1 SPR every 1000 lines of code)
- The code complies with mandatory coding standards

This analysis can be supported by tools and SPR statistics. SPR refers to SPRs detected after delivery

 

### 2.5.10 Traceability checks

| |
|---|
| The check precedure shall ensure that<br>- All software requirements are traced to system requirements<br>- All software requirements have been successfully tested<br>- All requirements are implemented and tested (this includes requirements on performance, resources, etc) |

 

# 3. Definition of the static model

## *3.1 Introduction*

In order to perform the verification of the software coding rules a mechanism was depicted to classify and verify the compliance of delivered source code against the established coding standards, [RD-6] and [RD-9].

In addition to this classification and in order to ease the conformance verification process the coding rules text was analysed and two classes of verification were identified: automated verification and manual verification. This is detailed in section 3.3.

## *3.2 Rule classification*

An analysis of the rules defined in [RD-6] was performed and these were aggregated in three different categories:

| Category ID | Description |
|---|---|
| Core rules | These rules are considered mandatory and shall be followed in the software development projects. |
| Recommended Rules | These rules should be followed in the software development projects. |
| De-scoped rules | These rules are de-scoped due to their non-compliance with the existing code base and should not be followed in software development cycles. |

Table 16 - Static Rules Classification

The prioritisation of the rules is done in Table 17 - Core Static Rules, Table 18 - Recommended Static Rules and Table 19 - De-Scoped Static Rules. For each rule a short description is also provided but the user is deferred to [RD-6] for the rule complete text and examples.

### 3.2.1 Core Rules

| Rule ID | Rule title |
|---|---|
| Appendix 1 | Header and implementation file outlines |
| GC.1.C++ | Never break a rule without documenting it. |
| GC.10.C++ | Provide meaningful, saying comments in the source code and make sure that they are kept up to date. |
| GC.12.C++ | Use appropriate tools to ensure that the code conforms to the rules and to catch potential problems as early as possible. |
| GC.13.C++ | Write the comments in the common language of the project. |
| GC.14.C++ | avoid "fancy-layout" comments because they require time and effort to maintain. |
| GC.14.C++.A | Comments must add to the code, not detract from it. |
| GC.15.C++ | Comments should never be used for "commenting out" code |
| GC.2.C++ | Maintain the source code and associated files under configuration control system and document cm information in the files. |
| GC.3.C++ | Use a "makefile" or its equivalent for building the application |
| GC.4.C++ | Write the software to conform to the coding language international standards |
| GC.5.C++ | Do not rely on compiler specific features. |
| GC.7.C++ | Use independent tools to provide additional warnings and information about the code. |
| GC.8.C++ | Always identify the source of warnings and correct the code to remove them. |
| GC.9.C++ | Do not attempt to optimize the code until it is proved to be necessary. |
| GL.11.C++ | Each file shall contain a standard comment header block. |
| GL.12.C++ | Each file header block shall contain information on configuration management, tracking of changes, spr numbers, name of author etc. |

 

| | |
|---|---|
| GL.15.C++ | The interface file shall contain declarations only. |
| GL.2.C++ | Each nested block of code, including one-line blocks, shall be identifiable through the code layout. |
| GL.2.C++.A | All aspects of indentation and formatting shall be consistent within a project. |
| GL.4.C++ | Each project shall define its own indentation rules (e.g. the contents of each nested block shall be indented by 3 spaces compared to the token which delimit the block). |
| GL.5.C++ | Tabs are not allowed. |
| GL.6.C++ | Use separate folders for each of the subsystems and libraries defined during the design phase. |
| GL.7.C++.A | Never use goto, longjmp(), setjmp(), malloc(), free(), realloc() |
| GL.9.C++ | Source code shall be separated into an interface and an implementation file |
| NC.5.C++ | Names for system global entities shall contain a prefix, which denotes which subsystem or library contains the definition of that entity. |
| NC.6.C++.B | Variables with large scopes are not allowed to have generic names. |
| NC.7.C++ | Each project shall define its own specific rules for naming conventions. |
| NC.7.C++.A | Any operation that matches one of the descriptions below should use the corresponding term as the first part of its name. |
| NC.7.C++.B | Accessor functions for an attribute shall always be based on the attribute name without the m_ prefix . |
| NC.8.C++ | User defined type and class names consist of one or more words where each word is capitalised plus an appropriate prefix or suffix. |
| PC.10.C++ | All local variables shall be initialised in their declaration. |
| PC.10.C++.A | Avoid global data if at all possible. static class data provides a much better alternative. |
| PC.10.C++.B | No direct access to neither public nor private class variables are allowed. |
| PC.13.C++ | Conditional expressions must always compare against an explicit value with boolean expressions as an exception. |
| PC.13.C++.A | The programmer must not override the comma, &&, \|\| and ?: operators. |
| PC.14.C++ | The programmer shall make sure that the order of evaluation of the expression is defined by typing in the appropriate syntax, by using parenthesis. |
| PC.15.C++ | The programmer must use parentheses to make intentions clear, when it is needed for improving readability |
| PC.18.C++.A | Allocation using shall use new/delete. |
| PC.18.C++.B | If the call to new uses [] then the corresponding call to delete must also use []. |
| PC.18.C++.C | After calling delete set the pointer to null (or 0) |
| PC.20.C++ | The return values of functions should be checked for errors. |
| PC.22.C++.A | When fallible functions fail, they shall indicate that they have failed by returning an error code or an out-of-bound value |
| PC.22.C++.B | Fallible functions should never return a reference when fallible functions fail, they shall indicate that they have failed by returning an error code or an out-of-bound value |
| PC.22.C++.C | When fallible functions fail in ways that are fully described by their return value, they should not raise an msg error or warning. |
| PC.22.C++.D | When infallible functions fail, they should raise a fatal error. |
| PC.22.C++.E | Whenever a fallible function is called, the return code shall always be checked. |
| PC.22.C++.G | C++ exceptions shall never be used without an appropriate design concept |
| PC.23.C++ | the programmer should use "problem domain" types rather than implementation types. |
| PC.24.C++ | Use non-portable code shall be minimized. |
| PC.24.C++.A | The programmer may only assume range(char) < range(short) <= range(int) <= range(long). |
| PC.24.C++.B | The programmer may only assume that range(float) <= range(double) <= range(long double) |
| PC.24.C++.C | The programmer may not assume knowledge of the representation of data types in memory, which implies that the use of memory dumps are forbidden. |
| PC.24.C++.D | The programmer may not assume that different data types have equivalent representations |

 

| | |
|---|---|
| | in memory. |
| PC.24.C++.E | The programmer may not assume knowledge of how different data types are aligned in memory. |
| PC.24.C++.F | The programmer may not assume that pointers to different data types are equivalent. |
| PC.24.C++.G | The programmer may not mix pointer and integer arithmetic. |
| PC.24.C++.H | It is not allow to use void pointers. |
| PC.24.C++.I | The programmer must use a wider type or unsigned values when testing for underflow or overflow. |
| PC.24.C++.J | the programmer must be careful when assigning "long" data values to "short" ones. |
| PC.25.C++.A | each case within a switch statement must contain a break statement or a "fall-through" comment. |
| PC.25.C++.B | All switch statements shall have a default clause. |
| PC.26.C++ | In class declarations shall the declaration of public, protected and private data and functions be clearly separated and only one section for each type. |
| PC.26.C++.B | Classes for which it is not intended to instantiate any objects should be abstract - i.e. they should contain at least one pure virtual function. |
| PC.26.C++.D | Diamond-shaped inheritance hierarchies are not allowed. |
| PC.27.C++ | class member variables must not be declared "public". |
| PC.27.C++.E | Inline functions should be defined within the class definition. |
| PC.28.C++ | It shall be ensured that an object of a class is created in a controlled manner |
| PC.29.C++ | It shall be ensured that an object of a class is deleted in a controlled manner. |
| PC.3.C++ | Global entities must be declared in the interface file for the module. |
| PC.30.C++.A | Base classes (capable of being derived) should have virtual destructors. but there is no need to have virtual destructor when the class may not be derived from |
| PC.31.C++ | It shall be ensured that an object of a class is copied in a controlled manner. |
| PC.31.C++.B | Any non-member, non-global function shall be explicitly declared static |
| PC.31.C++.C | In a function declaration, the names of formal arguments shall be specified and should be meaningful. if the function definition uses the parameters, the names should match the declaration. |
| PC.32.C++ | Member functions shall have a standard layout. |
| PC.33.C++ | Constructor functions which explicitly initialize any base class or member variable should not rely on a particular order of evaluation. |
| PC.34.C++ | Objects should be constructed and initialized immediately if possible rather than be assigned after construction. |
| PC.35.C++ | The class should always declare an assignment operator |
| PC.36.C++ | The assignment operator(s) must check for assigning an object to itself. |
| PC.37.C++.A | When possible, always use initialisation and initialisation list instead of assignment. this means the copy constructor is called, rather than the default constructor followed by an assignment operator. |
| PC.4.C++ | Declarations of "extern" variables and functions may only appear in interface files. |
| PC.47.C++ | A full function prototype shall be declared in the interface file(s) for each globally available function. |
| PC.48.C++ | Each function shall have an explanatory header comment |
| PC.48.C++.A | Each function shall have an explicit return type. a function, which returns no value, shall be declared as returning "void". |
| PC.5.C++.A | Declarations of static or variables and functions may only appear in classes. |
| PC.5.C++.B | All non-global variables and constants shall be explicitly declared static. |
| PC.51.C++ | A function may not return a reference or pointer to one of its own local automatic variables. |
| PC.7.C++ | Symbolic constants shall be used in the code. "magic" numbers and strings are expressly forbidden. |
| PC.7.C++.A | Never use numbers in code, nor any 'hardcoded' string, except when the use of these numbers is obvious - for instance in mathematical expressions, loop counter initialisation and limit checking. |

 

| | |
|---|---|
| PC.8.C++ | All symbolic constants shall be declared using an enumeration technique or the const keyword if the actual language provides such a facility. |
| PC.9.C++ | Each variable shall have its own declaration, on its own line. |
| PM.1.C++ | Each filename shall provide indication of modular relationship as well as functionality implemented. |
| PM.1.C++.A | For each class the implementation shall be distributed in two files, one for the interface specification and one for the actual implementation respectively with the .h and .c extensions. |
| PM.1.C++.B | The first 3-5 letters of the filename shall indicate to which module the code belongs. |
| PM.2.C++.A | The programmer shall consciously use the namespace facilities to organize his name scopes. |

Table 17 - Core Static Rules

### 3.2.2 Recommended Rules

| Rule ID | Rule title |
|---|---|
| GC.11.C++ | Minimize any debugging code. |
| GL.1.C++ | Each project shall define the maximum length in characters of the source code lines (e.g. each line of source code shall be no more than 80 characters in length). |
| GL.10.C++ | The interface file must be the first included in its own corresponding implementation file. |
| GL.13.C++ | The public interface file shall be self-contained and self-consistent. |
| GL.8.C++ | Use a utility or proforma to provide the starting point for all files within each particular subsystem, library or module. |
| NC.1.C++ | Names shall in general not start with an underscore character (_). |
| NC.11.C++ | Each enumeration within an enumerated type shall have a consistent prefix. |
| NC.2.C++ | Names shall be meaningful and consistent. |
| NC.3.C++ | Names containing abbreviations should be considered carefully to avoid ambiguity. |
| NC.4.C++ | Avoid using similar names, which may be easily confused or mistyped. |
| NC.6.C++.A | Names, which have wide scope and long lifetimes, should be longer than names with narrow scope and short lifetimes. |
| PC.11.C++.B | Every c++ program using exceptions must use the function set_unexpected() to specify which user defined function must be called in case a function throws an exception not listed in its exception specification. |
| PC.16.C++ | The programmer must always use parentheses around bitwise operators. |
| PC.17.C++.B | Other operators should be surrounded by white space. |
| PC.19.C++ | The programmer should validate function parameters where possible. |
| PC.2.C++ | Entities should be declared to have the shortest lifetime or most limited scope that is reasonable. |
| PC.21.C++ | Diagnostic code should be added to all areas of code, which "should never be executed". |
| PC.22.C++ | Error messages are not allowed to be hard coded, but shall be handled through some sort of central error message definition. |
| PC.26.C++.A | Any collection of data that does not warrant the work of writing a full class (e.g. defining accessor functions) should be defined as a struct. |
| PC.26.C++.C | Use of inheritance from non-abstract classes shall be minimised. |
| PC.28.C++.A | Avoid the use of global objects with constructors. the order in which global objects are initialised is not defined and can lead to 'chicken and egg' problems. |
| PC.31.C++.A | Stick to established conventions for overloaded operators. |
| PC.37.C++ | The assignment operator(s) must also assign base class member data. |
| PC.38.C++ | The assignment operator(s) should return a reference to the object. |
| PC.39.C++ | Symmetric operators, with the exception of assignment operator, should be defined as friend functions. All asymmetric operators (i.e. (), [], unary * and unary ->) must be defined as member functions. |
| PC.40.C++ | Member functions, which do not alter the state of an object, shall be declared ""const"." |

 

| | |
|---|---|
| PC.41.C++ | Public member functions must not return non-const references or pointers to member variables of an object. |
| PC.43.C++ | Member functions shall only be declared as ""inline"" if the need for optimization has been identified” |
| PC.44.C++ | A function shall not be declared as ""inline"" within the class definition itself." |
| PC.45.C++ | Generic units should be encouraged as a convenient way of reusing code. |
| PC.45.C++.A | Templates should only be used if all instantiations of the template will use the same algorithms. |
| PC.45.C++.B | There should be no functions in a template that do not depend on the type the template is instantiated for. |
| PC.45.C++.E | If templates are used then auto_ptr pointers should be preferred to normal pointers |
| PC.45.C++.F | All generic code shall work without modification when passed a valid subclass of a class it is expecting as an argument. |
| PC.45.C++.G | All functions should use references or pointers to base classes wherever possible. |
| PC.46.C++ | The use of type conversion shall as widely as possible be avoided in order to maintain compiler specific type checking. |
| PC.46.C++.A | Do not write code that force people to use explicit casts. |
| PC.49.C++ | A parameter, which is not changed by the function, should be declared ""const""." |
| PC.50.C++ | The layout of a function shall be well defined and used throughout the project. |
| PC.51.C++.A | In order to pass an object of type t as a function argument, use type: |
| PC.52.C++ | Records/structs should be converted to explicit tagged types/classes where possible. |
| PC.53.C++ | The use of design patterns shall be done as much as possible |
| PC.6.C++ | Declarations should appear in predefined order (e.g. constants and macros; types, structs and classes; variables; and functions). |

Table 18 - Recommended Static Rules

### 3.2.3 De-scoped Rules

| Rule ID | Rule title |
|---|---|
| NC.10.C++ | Class names shall begin with "c" or end with "_c" or "_class"." |
| NC.9.C++ | User defined type names shall begin with "t" or end with "_t" or "_type"." |
| PC.22.C++.F | Calls to new should be assumed to succeed. |

Table 19 - De-Scoped Static Rules

## *3.3 Compliance methodology*

To verify the standards compliance each rule was analyzed to identify a method for checking compliance, being two major categories depicted:

- Automated check:
  This will make use of automated testing tools, in our case the Logiscope suite from Tau Telelogic.
- Manual inspections:
  This comprises the manual inspection of build outputs, log files, documents and source code.

### 3.3.1 Automated checks

The table below presents the mapping between the rules from [RD-6] to the Tau Telelogic tool Logiscope set of checks. As can be observed, there are several cases where one single Logiscope check, e.g. ansi, verifies more than one coding convention rule and where one rule is verified in more than one logiscope check. The reader is deferred to [RD-8] for the detailed description of the Logiscope checks.

In the cases where the mapping to the C++ Naming and Coding Conventions [RD-6] is not possible it is attempted to map against the BSSC C and C++ Coding Standards [RD-9] rules. This mapping is presented in the comment column.

 

| Logiscope check | Rule Ids | Comment |
|---|---|---|
| ansi | PC.33.C++, PC.32.C++ | |
| ansi | PC.31.C++.C | |
| asscon | PC.13.C++ | |
| boolean | PC.13.C++ | |
| cast | EMPTY | Not in C++ coding conventions but Rule 91 of the BSSC |
| cmclass | PC.26.C++ | |
| cmdef | GL.9.C++, GL.15.C++ | |
| const | PC.7.C++.A, PC.7.C++, PC.8.C++ | |
| constrcpy | PC.31.C++ | |
| constrdef | PC.28.C++ | |
| constrinit | PC.34.C++, PC.37.C++.A, PC.28.C++ | |
| ctrlblock | Gl.3.C++ | |
| delarray | PC.18.C++.B | |
| destr | PC.29.C++ | |
| dmaccess | PC.10.C++, PC.27.C++, PC.10.C++.B | |
| exprparenth | PC.14.C++, PC.15.C++ | |
| fntype | EMPTY | Not in C++ coding conventions but Rule 61 of the BSSC |
| funcres | GL.7.C++.A | |
| headercom | GL.11.C++ | |
| hmclass | GL.9.C++, PC.6.C++ | |
| hmdef | PM.1.C++.A, PC.6.C++, GL.9.C++ | |
| hmstruct | GL.11.C++ | |
| identfmt | NC.1.C++ | |
| imptype | PC.32.C++, PC.33.C++ | |
| inldef | PC.27.C++.E, GL.15.C++ | |
| mname | PM.1.C++ | |
| multiass | EMPTY | |
| multinher | PC.26.C++ | |
| nonleafabs | PC.26.C++, PC.26.C++.B | |
| nostruct | PC.52.C++ | |
| operass | PC.35.C++ | |
| overload | PC.13.C++.A | |
| ptrinit | PC.34.C++ | |
| refclass | PC.38.C++ | |
| returnthis | PC.36.C++ | |
| sectord | Appendix 1 | |
| sgdecl | PC.10.C++, PC.9.C++ | |
| slcom | EMPTY | Not in C++ coding conventions but Rule 14 of the BSSC |
| swdef | PC.25.C++.B, PC.25.C++.A | |
| swend | PC.25.C++.A | |
| varinit | PC.37.C++, PC.10.C++ | |
| virtdestr | PC.30.C++.A | |
| voidptr | PC.24.C++.H | |

 

Table 20 - Automated Static Checks

### 3.3.2 Manual Inspections

There are rules that, due to their nature, cannot be verified by means of automated tools such as Tau Telelogic's Logiscope Rulechecker [RD-8] and where the manual intervention is required. These rules, as defined in [RD-6], are presented in Table 21 - Manual Static Checks.

| Rule ID | Inspection methodology |
|---|---|
| GC.1.C++ | Manual Inspection (Visual Observation) |
| GC.10.C++ | Logiscope Auditing + Manual Inspection (Visual Observation) |
| GC.13.C++ | Manual Inspection (Visual Observation) |
| GC.14.C++ | Manual Inspection (Visual Observation) |
| GC.14.C++.A | Manual Inspection (Visual Observation) |
| GC.15.C++ | Manual Inspection (Visual Observation) |
| GL.11.C++ | Logiscope Rulechecker + Manual Inspection (Visual Observation) |
| GL.12.C++ | Manual Inspection (Visual Observation) |
| GL.6.C++ | Documentation + Manual Inspection (Visual Observation) |
| NC.5.C++ | Manual Inspection (Visual Observation) |
| NC.6.C++.B | Manual Inspection (Visual Observation) |
| NC.7.C++.A | Manual Inspection (Visual Observation) |
| NC.7.C++.B | Manual Inspection (Visual Observation) |
| NC.8.C++ | Manual Inspection (Visual Observation) |
| PC.18.C++.C | Manual Inspection (Visual Observation) |
| PC.20.C++ | Manual Inspection (Visual Observation) |
| PC.22.C++.A | Manual Inspection (Visual Observation) |
| PC.22.C++.B | Manual Inspection (Visual Observation) |
| PC.22.C++.C | Manual Inspection (Visual Observation) |
| PC.22.C++.D | Manual Inspection (Visual Observation) |
| PC.22.C++.E | Manual Inspection (Visual Observation) |
| PC.23.C++ | Manual Inspection (Visual Observation) |
| PC.24.C++ | Manual Inspection (Visual Observation) |
| PC.24.C++.C | Manual Inspection (Visual Observation) |
| PC.24.C++.D | Manual Inspection (Visual Observation) |
| PC.24.C++.E | Manual Inspection (Visual Observation) |
| PC.24.C++.F | Manual Inspection (Visual Observation) |
| PC.24.C++.G | Manual Inspection (Visual Observation) |
| PC.24.C++.I | Manual Inspection (Visual Observation) |
| PC.24.C++.J | Manual Inspection (Visual Observation) |
| PC.3.C++ | Manual Inspection (Visual Observation) + Logiscope Audit |
| PC.4.C++ | Manual Inspection (Visual Observation) (grep tool) |
| PC.5.C++.A | Manual Inspection (Visual Observation) (grep static) |
| PC.5.C++.B | Manual Inspection (Visual Observation) + Logiscope Audit |
| PC.51.C++ | Manual Inspection (Visual Observation) |
| PM.1.C++ | Manual Inspection (Visual Observation) |
| PM.1.C++.B | Manual Inspection (Visual Observation) |

Table 21 - Manual Static Checks

 

# DETAILED MAPPING TO SPEC

## A.1 *SPEC metrics mapped to checks*

| Goal property | Property | SPEC metric | Check |
|---|---|---|---|
| Documentation Quality | Development & maintenance documentation quality | DOQ.AD.M2 | Documentation Checks |
| Documentation Quality | Development & maintenance documentation quality | DOQ.AD.M3 | Documentation Checks |
| Documentation Quality | Operation- related documentation quality | DOQ.OD.M1 | Documentation Checks |
| Documentation Quality | Operation- related documentation quality | DOQ.OD.M2 | Documentation Checks |
| Documentation Quality | Requirements quality | DOQ.RQ.M1 | Documentation Checks |
| Documentation Quality | Requirements quality | DOQ.RQ.M2 | Documentation Checks |
| Documentation Quality | Requirements quality | DOQ.RQ.M3 | Documentation Checks |
| Documentation Quality | Requirements quality | DOQ.RQ.M4 | N/A |
| Documentation Quality | Requirements quality | DOQ.RQ.M5 | Documentation Checks |
| Functionality | Completeness | FUN.CM.M1 | Traceability checks |
| Functionality | Completeness | FUN.CM.M2 | Traceability checks |
| Functionality | Completeness | FUN.CM.M3 | Traceability checks |
| Functionality | Completeness | FUN.CM.M4 | PA checks |
| Functionality | Completeness | FUN.CM.M5 | Documentation checks |
| Functionality | Correctness | FUN.CR.M1 | EMPTY |
| Functionality | Correctness | FUN.CR.M10 | Traceability checks |
| Functionality | Correctness | FUN.CR.M11 | EMPTY |
| Functionality | Correctness | FUN.CR.M2 | EMPTY |
| Functionality | Correctness | FUN.CR.M3 | EMPTY |
| Functionality | Correctness | FUN.CR.M4 | EMPTY |
| Functionality | Correctness | FUN.CR.M5 | EMPTY |
| Functionality | Correctness | FUN.CR.M6 | |
| Functionality | Correctness | FUN.CR.M7 | Static analysis |
| Functionality | Correctness | FUN.CR.M8 | N/A |
| Functionality | Correctness | FUN.CR.M9 | Traceability checks |
| Functionality | Efficiency | FUN.EF.M1 | Traceability checks |
| Functionality | Efficiency | FUN.EF.M2 | Traceability checks |
| Functionality | Efficiency | FUN.EF.M3 | Traceability checks |
| Functionality | Efficiency | FUN.EF.M4 | Traceability checks |
| Maintainability | Analysability | MAIN.AN.M1 | Static analysis |
| Maintainability | Analysability | MAIN.AN.M3 | Documentation checks |
| Maintainability | Analysability | MAIN.AN.M5 | Static analysis |
| Maintainability | Analysability | MAIN.AN.M8 | Documentation |

 

| Goal property | Property | SPEC metric | Check |
|---|---|---|---|
| | | | checks |
| Maintainability | Changeability | MAIN.CH.M1 | PA checks |
| Maintainability | Changeability | MAIN.CH.M2 | PA checks |
| Maintainability | Changeability | MAIN.CH.M3 | PA checks |
| Maintainability | Changeability | MAIN.CH.M4 | PA checks |
| Maintainability | Portability | MAIN.PO.M3 | N/A |
| Maintainability | Verifiability | MAIN.VE.M1 | N/A |
| Maintainability | Verifiability | MAIN.VE.M3 | N/A |
| Operability | Usability | OPE.US.M1 | Documentation checks |
| Operability | Usability | OPE.US.M2 | Documentation checks |
| Operability | Usability | OPE.US.M3 | N/A |
| Operability | Usability | OPE.US.M4 | Operability checks |
| Operability | Virtuality | OPE.VI.M1 | Operability checks |
| Reliability | Integrity | REL.IN.M1 | reliability checks |
| Reliability | Integrity | REL.IN.M2 | reliability checks |
| Reliability | Maturity | REL.MA.M1 | reliability checks |
| Reliability | Maturity | REL.MA.M2 | reliability checks |
| Reliability | Maturity | REL.MA.M3 | N/A |
| Reliability | Recoverability | REL.RC.M1 | reliability checks |
| Reliability | Recoverability | REL.RC.M2 | Reliability checks |
| Reliability | Reliability Evidence | REL.RE.M1 | Reliability checks |
| Reliability | Robustness | REL.RO.M1 | Reliability checks |
| Reliability | Robustness | REL.RO.M2 | Reliability checks |
| Re-usability | Portability | REU.PO.M3 | N/A |
| Re-usability | Re-usability Documentation | REU.RD.M1 | Reusability checks |
| Re-usability | Self-contained functionality | REU.SF.M1 | Reusability checks |
| Suitability for safety | Safety Evidence | SAF.SE.M1 | Safety checks |
| Suitability for safety | Safety Evidence | SAF.SE.M2 | N/A |
| Suitability for safety | Safety Evidence | SAF.SE.M3 | N/A |
| Suitability for safety | Safety Evidence | SAF.SE.M4 | N/A |
| Software Development Effectiveness | Software development process level | SDE.DL.M1 | Process checks |
| Software Development Effectiveness | Software development process evidence | SDE.PE.M1 | Process checks |
| Software Development Effectiveness | Software development process evidence | SDE.PE.M2 | Process checks |
| System Engineering Effectiveness | Interfaces management | SEE.IN.M1 | effectiveness checks |
| System Engineering Effectiveness | Requirements propagation | SEE.RQ.M1 | N/A |
| System Engineering Effectiveness | System engineering process evidence | SEE.SE.M1 | N/A |
| System Engineering Effectiveness | System engineering process evidence | SEE.SE.M2 | N/A |
| System Engineering Effectiveness | System engineering process evidence | SEE.SE.M3 | N/A |
| System Engineering | System engineering process evidence | SEE.SE.M4 | N/A |

 

| Goal property | Property | SPEC metric | Check |
|---|---|---|---|
| Effectiveness | | | |

Table 22 - Mapping between SPEC metrics and checks in this model

## A.2 *SPEC metrics mapped to metrics*

| Goal property | Property | SPEC metric | Metric name | Logiscope reference |
|---|---|---|---|---|
| Documentation Quality | Development & maintenance documentation quality | DOQ.AD.M1 | Comments frequency | COMF |
| Maintainability | Analysability | MAIN.AN.M2 | Cyclomatic number (VG) | ct_vg |
| Maintainability | Analysability | MAIN.AN.M4 | Number of nestings | LEVL |
| Maintainability | Analysability | MAIN.AN.M6 | Average size of statements | AVGS |
| Maintainability | Analysability | MAIN.AN.M7 | Number of statements | lc_stat |
| Maintainability | Modularity | MAIN.MO.M1 | Number of direct used classes | cu_cdused |
| Maintainability | Modularity | MAIN.MO.M2 | Number of statements | lc_stat |
| Maintainability | Modularity | MAIN.MO.M3 | Coupling between objects | ap_cbo |
| Maintainability | Modularity | MAIN.MO.M4 | Lack of cohesion of methods | cl_locm |
| Maintainability | Portability | MAIN.PO.M1 | Number of application classes | ap_clas |
| Maintainability | Portability | MAIN.PO.M2 | Number of application classes | ap_clas |
| Maintainability | Verifiability | MAIN.VE.M2 | Cyclomatic number (VG) | ct_vg |
| Re-usability | Portability | REU.PO.M1 | Number of application classes | ap_clas |
| Re-usability | Portability | REU.PO.M2 | Number of application classes | ap_clas |

Table 23 - Mapping between SPEC metrics and metrics measured in this model

## A.3 *Additional metrics (not in SPEC)*

| Metric name | Logiscope reference |
|---|---|
| Ratio of repeated inheritances in the application | URI_Ratio |
| Percentage of non-member functions | NMM_Ratio |
| Average coupling between objects | AVG_CBO |
| Average of the VG of the application's functions | AVG_VG |
| Ratio of recursive edges on the call graph | RECU_Ratio |
| Method hiding factor (MOOD) | ap_mhf |
| Attribute hiding factor (MOOD) | ap_ahf |
| Method inheritance factor (MOOD) | ap_mif |
| Attribute inheritance factor (MOOD) | ap_aif |

 

| Metric name | Logiscope reference |
|---|---|
| Polymorphism factor (MOOD) | ap_pof |
| Coupling factor (MOOD) | ap_cof |
| Number of Levels in the Inheritance Graph | ap_inhg_levl |
| Hierarchical Complexity of the Inheritance Graph | ap_inhg_cpx |
| Number of Levels in the Call Graph | ap_cg_levl |
| Fan in of a class | FAN_Inclass |
| Fan out value of a class | FAN_OUTclass |
| Class Comments Frequency | COMFclass |
| Encapsulation rules | ENCAP |
| Usability | USABLE |
| Specializability | SPECIAL |
| Rate of class autonomy | AUTONOM |
| Testability | TESTAB |
| Weighted Methods per Class | cl_wmc |
| Number of base classes | in_bases |
| Number of dependent methods | cl_dep_meth |
| Number of children | in_noc |
| Coupling between classes | cl_cobc |
| Number of direct users classes | cu_cdusers |
| Vocabulary frequency | VOCF |
| Number of levels | LEVL |
| Fan In | FAN_IN |
| Fan Out | FAN_OUT |
| Number of distinct uses of external attributes | ic_varpe |
| Number of destructuring statements | ct_bran |
| Number of parameters passed by value | ic_parval |
| Number of parameters passed by reference | ic_paradd |
| Number of out statements | ct_exit |
| Number of paths | ct_path |
| Number of direct calls | dc_calls |
| Number of callers | dc_calling |
| Number of local variables | dc_lvars |
| Number of function parameters | ic_param |
| Number of relative call graph levels | cg_levels |
| Relative call graph Hierarchical complexity | cg_hiercpx |
| Relative call graph Structural complexity | cg_strucpx |
| Number of relative call graph call-paths | IND_CALLS |
| Relative call graph System testability | cg_testab |

Table 24 - List of metrics that cannot be found in SPEC

# Annex A QUALITY MODEL TREE

The table below provides the complete quality model as defined in [RD-3].

| Goal property | Property | Metric | Code |
|---|---|---|---|
| Documentation Quality | Development & maintenance | Code Comment frequency. | DOQ.AD.M1 |
| | | Documentation clarity | DOQ.AD.M2 |

 

| Goal property | Property | Metric | Code |
|---|---|---|---|
| | documentation quality | Documentation suitability | DOQ.AD.M3 |
| | Operation- related documentation quality | Documentation clarity | DOQ.OD.M1 |
| | | Documentation suitability | DOQ.OD.M2 |
| | Requirements quality | Number of conflicting requirements | DOQ.RQ.M1 |
| | | Requirements clarity | DOQ.RQ.M2 |
| | | Requirements completeness | DOQ.RQ.M3 |
| | | Requirements volatility rate | DOQ.RQ.M4 |
| | | Requirements duplication | DOQ.RQ.M5 |
| Functionality | Completeness | Technical specification/ software requirements mapping rate | FUN.CM.M1 |
| | | Functional implementation coverage | FUN.CM.M2 |
| | | Requirements/program units mapping rate | FUN.CM.M3 |
| | | Verification activities mapping rate | FUN.CM.M4 |
| | | Functional requirements/ user manual items mapping rate | FUN.CM.M5 |
| | Correctness | Statement coverage | FUN.CR.M1 |
| | | Interface testing completeness | FUN.CR.M10 |
| | | Run-time error verification | FUN.CR.M11 |
| | | Module branch coverage | FUN.CR.M2 |
| | | Condition coverage | FUN.CR.M3 |
| | | Test completeness | FUN.CR.M4 |
| | | Verification coverage | FUN.CR.M5 |
| | | Faults removed | FUN.CR.M6 |
| | | Fault density | FUN.CR.M7 |
| | | Computational Accuracy (dynamic) | FUN.CR.M8 |
| | | Successful interface testing rate | FUN.CR.M9 |
| | Efficiency | Timing margin | FUN.EF.M1 |
| | | Memory margin | FUN.EF.M2 |
| | | Throughput | FUN.EF.M3 |
| | | Ressources utilisation | FUN.EF.M4 |
| Maintainability | Analysability | Problem cause understandability | MAIN.AN.M1 |
| | | Cyclomatic complexity or Module complexity | MAIN.AN.M2 |
| | | Staff hours to inspect the code | MAIN.AN.M3 |
| | | Nesting level | MAIN.AN.M4 |
| | | Code understandability | MAIN.AN.M5 |
| | | Average Size of statements. N/lc_stat | MAIN.AN.M6 |
| | | LOC (lines of Code) | MAIN.AN.M7 |
| | | MTTD (Mean Time To Diagnose) | MAIN.AN.M8 |
| | Changeability | Regression rate | MAIN.CH.M1 |
| | | MTTR (mean time to repair) | MAIN.CH.M2 |
| | | MTTC (Mean Time to Change) | MAIN.CH.M3 |
| | | Complexity of changes | MAIN.CH.M4 |
| | Modularity | Modular span of control. Average of cu_cdused | MAIN.MO.M1 |
| | | Modularity size profile. Average of lc_stat | MAIN.MO.M2 |
| | | Modular coupling. | MAIN.MO.M3 |

 

| Goal property | Property | Metric | Code |
|---|---|---|---|
| | | Modular cohesion | MAIN.MO.M4 |
| | Portability | Environmental software independence. Identify units with conditional compilation clauses that depend on the operating system (#ifdef SOLARIS) and divide by total number of units (total number of units provided by ap_clas) | MAIN.PO.M1 |
| | | System hardware independence. Identify units with conditional compilation clauses that depend on the hardware (#ifdef SOLARIS) and divide by total number of units (total number of units provided by ap_clas) | MAIN.PO.M2 |
| | | Installability | MAIN.PO.M3 |
| | Verifiability | Complexity of changes | MAIN.VE.M1 |
| | | Cyclomatic complexity | MAIN.VE.M2 |
| | | Verification Facilities | MAIN.VE.M3 |
| Operability | Usability | Off line Tutorial Readiness | OPE.US.M1 |
| | | On line Tutorial Readiness | OPE.US.M2 |
| | | Operator's Error Frequency | OPE.US.M3 |
| | | Operator's Judgement | OPE.US.M4 |
| | Virtuality | User operation virtuality | OPE.VI.M1 |
| Reliability | Integrity | Access controllabil ity; | REL.IN.M1 |
| | | Data corruption indicator | REL.IN.M2 |
| | Maturity | MTTF | REL.MA.M1 |
| | | Cumulative critical failure profile | REL.MA.M2 |
| | | Functional implementation stability | REL.MA.M3 |
| | Recoverability | MTTRestart (Mean Time To Restart) | REL.RC.M1 |
| | | MTTRecover (Mean Time To Recover) | REL.RC.M2 |
| | Reliability Evidence | Process Reliability activities adequacy | REL.RE.M1 |
| | Robustness | Failure avoidance | REL.RO.M1 |
| | | Failures tolerance | REL.RO.M2 |
| Re-usability | Portability | Environmental software independence. Identify units with conditional compilation clauses that depend on the operating system (#ifdef SOLARIS) and divide by total number of units (total number of units provided by ap_clas) | REU.PO.M1 |
| | | System hardware independence. Identify units with conditional compilation clauses that depend on the hardware (#ifdef SOLARIS) and divide by total number of units (total number of units provided by ap_clas) | REU.PO.M2 |
| | | Installability | REU.PO.M3 |
| | Re-usability Documentation | Reuse Documentation | REU.RD.M1 |
| | Self-contained functionality | Functional independence | REU.SF.M1 |
| Suitability for safety | Safety Evidence | Safety Planning adequacy | SAF.SE.M1 |
| | | Safety Analysis adequacy | SAF.SE.M2 |
| | | Safety Tecnique adequacy | SAF.SE.M3 |
| | | ISVV activities adequacy | SAF.SE.M4 |

 

| Goal property | Property | Metric | Code |
|---|---|---|---|
| Software Development Effectiveness | Software development process level | Process Maturity | SDE.DL.M1 |
| | | Software development activities adequacy | SDE.PE.M1 |
| | | Software verification activities adequacy | SDE.PE.M2 |
| System Engineering Effectiveness | Interfaces management | Interface management | SEE.IN.M1 |
| | Requirements propagation | Derivation of software product requirements from system requirements | SEE.RQ.M1 |
| | System engineering process evidence | Analysis and Planning activities quality | SEE.SE.M1 |
| | | System engineering activi ties quality | SEE.SE.M2 |
| | | Organisation and Management activities quality | SEE.SE.M3 |
| | | System-software requirements traceability rate | SEE.SE.M4 |

Table 25 - The full SPEC model (goal properties, properties and metrics)

 