Software Quality Objectives for Source Code

A. *Patrick BRIAND*[5], B. *Martin BROCHET*[4,] C. *Thierry CAMBOIS*[2], D. *Emmanuel COUTENCEAU*[5] E. *Olivier GUETTA*[3], F. *Daniel MAINBERTE*[2], G. *Frederic MONDOT*[3], H. *Patrick MUNIER*[4], I. *Loic NOURY*[4], J. *Philippe SPOZIO*[2], K. *Frederic RETAILLEAU*[1]

1. **Delphi Diesel System** France s.a.s, 9 bd de l'Industrie 41042 BLOIS France
2. **PSA Peugeot Citroën**, 75 avenue de la Grande-Armée, BP01, 75761 PARIS
3. **Renault s.a.s**, 13/15 Quai Alphonse Le Gallo, 92100 BOULOGNE-BILLANCOURT
4. **The MathWorks**, 2 rue de Paris 92196 MEUDON France
5. **Valeo**, 43 Rue Bayen, PARIS 75017

Abstract: The MathWorks - Renault SA - PSA Peugeot Citroën - Delphi Diesel System - Valeo group wrote together a code quality standard from scratch. This document describes how the code standard places the proof of absence of run-time errors at the centre of its software quality model. It details how the following elements of the quality model co-exist together with the supplier code life cycle: MISRA-C coding standard, the absence of run-time errors and some code complexity metrics. Additionally, this document describes how the Automotive manufacturers and the suppliers have to agree on and achieve different Software Quality Objectives according to the code life cycle stage and the safety aspects of the application.

Finally, the document illustrates that standard with the PolySpace product and details how the product can help both the automotive manufacturer and the supplier working with this standard.

Keywords: Quality, run-time error, absence, formal method, MISRA, formal proof, software

1. Introduction

This document has been produced by a working group which includes the automotive manufacturers Renault SA and PSA Peugeot Citroën, the automotive suppliers Delphi Diesel Systems and Valeo, and software supplier The MathWorks[1].
The document defines a general and standard approach to measure the software quality of a product using criteria linked to code quality and dynamic execution errors.

N.B: This current work does not cover functional test activities.

[1] With Patrick Artola participation (DevQuality).

2. Origin of Software Quality Objectives

After the PolySpace company acquisition by The MathWorks in 2007, a new dynamic occurred. The MathWorks team worked as facilitator to organize a meeting with PSA Peugeot Citroën and Renault SA. The objective of this meeting was to get their feedback about PolySpace product usage. During this discussion PSA Peugeot Citroën and Renault presented their 4 PolySpace usages:

- Risk prevention of non-quality embedded software
- Acceptance criteria for product delivery or process embedded software
- Audit/Assess the quality of embedded software
- Investigate the possible causes of malfunctions

And these uses had several objectives:

- A "deterministic" implementation to know exactly the software quality integrated in vehicles.
- Keep a good technical/economic performance by maximizing the automation (e.g. reduce the number of warnings need to be analyzed manually)
- Compliance strategy (to obtain results) rather than means (tools to use)

From their experiences automotive manufacturer gathered a set of common difficulties:

- The analysis reports were not homogeneous, they were homemade, did not always indicate whether the preconditions of the analysis were validated and did not guarantee their integrity or completeness. (e.g. MISRA compliance). Incremental verification between 2 software deliveries and comparative treatment of the associated report were difficult
- The reports documented numerous measures and/or warnings without providing the method of analysis involved (see Process Area MA CMMi) which increased the presence of inaccuracies and false alarms
- The method of verification report was not formalized and validation of justification was difficult, it was primarily for validation (expert opinion). The report study did not provide the possibility to easily conclude on the software quality and made judgments “OK or Not OK” on the product embedded software.

In regard of this feedback the working group decided to:

- ⇨ Gather criteria, (complexity metrics, measure of unreachable code, etc) to be able to objectively measure “Software Quality”
- ⇨ Formalize relationships and communication between automotive manufacturers and their suppliers through a list of requirements.
- ⇨ Ask software suppliers to provide guidelines on how to use their tools (ex: PolySpace) to produce information required.

After two and a half years the working group got agreement between automotive manufacturers and suppliers on a standard requirement list and described it in the Software Quality Objective document version 2.0. This requirement list consists of six Software Quality Objectives (SQO) which are associated to four quality levels, Quality Levels 1 (QL-1 lowest quality) to Quality Levels 4 (QL-4 highest quality).

The following document will present the content of the Software Quality Objective. If you need more information about the software supplier implementation, please contact them.

3. Software Quality Objectives (SQO) Overview

The first working step was to define generic acceptance criteria.

Generic acceptance criteria

The criteria selected to build Software Quality Objectives are focused on static and dynamic software quality and also takes into consideration the organisational information, which helps the car manufacturer to better understand who performs the work, and how and where the work is done.

Static and dynamic criteria are spread over six others themes:

- Detailed design description,
- Code metrics,
- Coding rules,
- Unreachable branches,
- Run-time errors
- Dataflow Analysis.

4. Definition of 6 incremental Quality Objectives

Differences between Software Quality Objectives 1 and Software Quality Objectives 6 are the number of requirements to fulfil. The following table presents this distribution over the 6 Software Quality Objectives.

For the SQO-1 (Software Quality Objectives) the supplier has to provide information regarding Quality Plan, Detailed design description, Code metrics and Coding rules.

For higher SQO, some additional criteria come in.

For the SQO-3 the supplier has to provide information to the car manufacturer about systematic Run Time Error, Non terminating information and unreachable branches in addition to information already present in SQO-1.

For the SQO-6, the highest SQO, the supplier has to provide information about a bigger set of coding rules, non terminating construction, Dataflow Analysis and an advance level of review coverage for potential run-time errors.

The incremental process for quality levels is mapped to vehicle development processes which guarantee integration of items of increasing maturity.

5. Software Quality Objective vs. Quality Level

Software Quality Objectives (SQO) are associated with four quality levels, Quality Level 1 (QL-1 lowest quality) to Quality Level (QL-4 highest quality). It is up to the supplier and car manufacturer to negotiate and map SQO to their deliveries and completely integrate it to their schedule.

The supplier shall provide a quality plan which shall consist of a table showing for each module:

- The corresponding Quality Level (QL-1 to QL-4)
- The quality objectives for that level
- The number of times the module will be delivered during the project.

N.B.: the automotive manufacturer shall validate the software quality plan and the decisions taken within it.

The following example table shows the possible progress for each quality level to achieve the final Software Quality Objectives. The number of quality levels is fixed. The SQO associated to the first, last and penultimate deliveries are also fixed. The number of deliveries (table lines) is project dependant.

Delivery	Quality Level			
	QL-1	QL-2	QL-3	QL-4
First	SQO-1	SQO-2	SQO-3	SQO-4
X Intermediates	...	...	...	...
X Intermediates	SQO-2	SQO-3	SQO-4	SQO-5
X Intermediates	...	...	...	...
Penultimate	SQO-3	SQO-4	SQO-5	SQO-6
Last	SQO-3	SQO-4	SQO-5	SQO-6

N.B.: The penultimate and the last delivery have the same SQO. This is to increase the probability to obtain at the last delivery, the SQO decided at the beginning and associated to a quality level.

6. How to associate each level to your application?

As expressed before, at the beginning of the project the supplier and the car manufacturer have to associate a quality level to each software module. The document does not provide recommendation or criteria to do it.
To accomplish this work several criteria should be used and combined. The final objective is to achieve an agreement between both sides.

The SQO document does not exclude any kind of source code. All software modules have to be associated with Quality Level, whatever the type of code it may be: automatically generated, hand-written, legacy, or COTS.

Potential criteria:

- Re-use work already done with the ISO-26262 standard to define criticality (Level A to Level D)
- The module maturity (Proven in use, new module…)
- The Code origin (legacy code, COTS, hand written code, generated code, …)
- Module Quality Insurance
- Module functionality (User interface, Application layer, Platform layer, Operating system, Drivers…)

7. Practical example

Supplier is delivering one application with several modules to an automotive manufacturer. What should be done?

- Define the number of deliveries for application
- For each module define its Quality Level
- Associate the Software Quality Objective to intermediate deliveries for each module

After that both sides should respect these commitments.

8. Software Quality developed in criteria

Each Software Quality Objective consists of a set of 12 criteria listed in the table below

Criteria	Objectives					
	SQO-1	SQO-2	SQO-3	SQO-4	SQO-5	SQO-6
Quality Plan	X	X	X	X	X	X
Detailed design description	X	X	X	X	X	X
Code metrics	X	X	X	X	X	X
First MISRA-C:2004 rules subset	X	X	X	X	X	X
Second MISRA-C:2004 rules subset					X	X
Systematic run-time errors		X	X	X	X	X
Non terminating constructs		X	X	X	X	X
Unreachable branches			X	X	X	X
First subset of potential run-time errors				X	X	X
Second subset of potential run-time errors					X	X
Third subset of potential run-time errors						X
Dataflow Analysis						X

Criteria distributed over Software Quality Objectives

8.1. Quality Plan

This criterion describes the general information which shall be provided by a supplier. It covers information about the methods, tools and teams involved in the Software Quality Requirement fulfilment, as well as information about the project itself. This information shall help to better understand who performs the work, and how and where the work is done.

8.2. Detailed design description

The information provided in this criterion will help evaluate the architecture of the application and its maturity. This will form the basis for following criteria of the document. Three levels of information have to be provided:

- Application level
- Module level
- File level

8.3. Code metrics

This criterion shall help the automotive manufacturers evaluate the module characteristics and better understand the methods and tools used to demonstrate the application quality regarding the absence of runtime errors.

Some of the recommended metrics are:

- Comment Density; Cyclomatic complexity “v(G)”;
- Number of Calling Functions per Function;
- Number of Function Parameters;
- Number of call Levels;
- Number of return points within a function;
- …

N.B.: The set of recommended metrics was selected to contribute to respect most of **HIS** metrics initiative.

8.4. Two MISRA-C:2004 rules subsets

These two criteria shall help the automotive manufacturer to evaluate the code quality and maintainability.

The criterion shall help the automotive supplier to be more efficient regarding the reduction of the number of Run Time Errors.

Two MISRA-C:2004 rules subsets were defined; the first one covers 20 rules and the second one 29 rules.

The objective is to correct or justify all violations, i.e. zero remaining violations found by the tool or remaining violations unjustified.

8.5. Systematic run-time errors

This criterion shall prove the absence of systematic error in the supplier software. The supplier shall demonstrate that for all files within a module a review of systematic runtime errors has been performed and that errors which have not been corrected are justified, for the following categories:

- Out-of-bounds array access
- Division by zero
- Read access of non-initialized data
- Function returning non initialized value
- Integer overflow/underflow
- Float overflow
- De-referencing through null or out-of-bounds pointer
- Usage (read or dereference) of a non-initialized pointer
- Shift amount is in 0..7, 15, 31 or 63 and left operand of left shift is negative
- Wrong type for argument passed to a function pointer
- Wrong number of arguments passed to a function pointer
- Wrong return type of a function or a function pointer
- Wrong return type of an arithmetic function
- Non null this-pointer (for C++)
- Positive array size (for C++)
- Incorrect typeid argument (for C++)
- Incorrect dynamic_cast on pointer (for C++)
- Incorrect dynamic_cast on reference (for C++)
- Invalid pointer to member (for C++)
- Call of pure virtual function (for C++)
- Incorrect type for this-pointer (for C++)

8.6. Non terminating function calls and loops

This criterion shall help suppliers to present the amount of verified non terminating function calls and loops in their software.

If the code intentionally contains:

- Non terminating loops like 'while(1)' or 'for(;;)'
- Non terminating calls like 'exit', 'stop', 'My_Non_Returning_Function'

these should be justified.

8.7. Unreachable branches

The supplier shall demonstrate that files do not contain any unjustified dead code branches, as well as all defensive code and dead code intentionally contained in the application shall be justified.

8.8. Three subsets of potential run-time errors

This criterion shall help suppliers to present the amount of verified operation presents in their software.

The supplier shall demonstrate that for all files within a module, a review of potential runtime errors with review coverage level 1 (lowest), 2 or 3 has been performed and that potential errors which have not been corrected are justified

The 3 subset lists of run time error are the same as presented in the 8.5 Systematic run-time errors paragraph. Hereunder there is an example of percentage defined for review coverage level 1:

- Out-of-bounds array access: 80%
- Division by zero: 80%
- Read access to local non-initialized data: 80%
- overflow/underflow: 60%
- ...

N.B.: For runtime errors the review coverage is defined by a percentage, indicated after the runtime error category (example: "Division by zero: 80 %") which represents the number of operations concluded as proven safe or justified.

These conclusions could be drawn:

- Automatically (with a tool);
- Partially automatically and completed manually;
- Totally manually.

Example: let's take an application containing 60 divisions. Let's assume that the review coverage objective is "Division by zero: 80%". Then the 80% review coverage can be reached by proving that at least 80% of the divisions are "safe operations" or "potential runtime errors" that can be justified.

Let's consider that a tool is used, which proves automatically that 45 divisions out of the 60 are "safe operations". The review objective can be reached by demonstrating that at least 3 "potential errors" can be justified, because (45 + 3) / 60 = 80%.

8.9. Dataflow Analysis

This criterion shall help automotive manufacturer to see the dependencies in the supplier application files architecture (global variables read/write location) and show if all shared variables are protected.

The supplier shall provide for each module, the data flow analysis results.

This shall contain at least:

- Application call tree
- Dictionary containing read/write accesses to global variables
- List of shared variables and their associated concurrent access protection (if any)

9. Impact on PolySpace developments

The Document has already impacted the PolySpace development. For example release 2010a supports the following capabilities.

- Incremental review.
- Result justifications capabilities (annotations in source code)
- Formalisation of review through standardized acronyms
- Improved report generator
- Methodological assistant oriented towards quality objectives (on top of existing bug finding methodology)
- Provide facilities to model application environment (support function Stub in Data Range Specification functionality)
- Enlarged detection of unreachable code.

10. Conclusion

Currently “Software Quality Objectives” version 2.0 is available.

PSA Peugeot Citroën and Renault SA integrated Version 1.0 in their Quality Requirements in 2009 and already got some feedback from suppliers in October 2009.

Renault integrated it as rank 2 in their requirement but they are negotiating with Nissan to integrate it in their rank 1 requirements.

Valeo & Delphi Diesel System promoted this document internally and they are currently working on a solution to implement this standard in their process.

The MathWorks presented it to several companies from the automotive industry, Railway industry, defence industry, in France and abroad (Germany, Japan and USA)

Presentations have been given to several other software companies, including Coverity, KlocWork and Programming Research (QAC author).

Work is ongoing to evaluate the interaction between the SQO standard and ISO-26262 and establish how the SQO complements the ISO-26262 requirement and how it covers it.

The SQO standard version 2.0 is available; please contact the work group to get it and provide feedback

11. Glossary

COTS: **Commercial, off-the-shelf** is a term for software or hardware, generally technology or computer products, that are ready-made and available for sale, lease, or license to the general public. They are often used as alternatives to in-house developments or one-off government-funded developments. The use of COTS is being mandated across many government and business programs, as they may offer significant savings in procurement and maintenance. However, since COTS software specifications are written by external sources, government agencies are sometimes wary of these products because they fear that future changes to the product will not be under their control.

Automotive manufacturer: is a company that uses a component made by a second company in its own product, or sells the product of the second company under its own brand. It constitutes a federally-licensed entity required to warrant and/or guarantee their products, unlike "aftermarket" which is not legally bound to a government-dictated level of liability.

Supplier: automotive components manufacturer.

SQO: Software Quality Objectives

QL: Quality Level

MISRA: The Motor Industry Software Reliability Association (http://www.misra.org.uk/)

ISO 26262: Is an emerging ISO standard for safety systems in road vehicles engine (http://www.iso.org/)

HIS: Hersteller Inititiative Software. Initiative from German automotive manufacturers (Audi, BMW Group, DaimlerChrysler, Porsche and Volkswagen) whose goal is the production of agreed standards within the area of standard software modules for networks, development of process maturity, software test, software tools and programming of ECU's. HIS specifies a fundamental set of Software Metrics to be used in the evaluation of software.

See http://portal.automotive-his.de/images/pdf/SoftwareTest/his-sc-metriken.1.3.1_e.pdf