Work Instruction

MISRA and Code Metrics Product Development PES

Author	Hartmut Hörner, Andreas Raisch		
Version	2.1 of 2013-12-17	**Number of pages:**	35
Status	released	(in process / inspected / released)	
Release	Hartmut Hörner, ProcessOwner	2013-04-22	Signed: Hartmut Hörner
Published by	© 2013 Vector Informatik GmbH Ingersheimer Str. 24 70499 Stuttgart		

Document Management

Topic	Content
Purpose (of document)	Defines MISRA and HIS metric compliance procedure
Scope (of document)	VI-PES
Document Management for this document	SVN (https://vistrpessvn3.vi.vector.int/svn/Org/Prc/70_Topic_MISRA/base/01_wi/trunk)
Quality requirements on this document	Review and release before publishing on the Intranet
Open issues	• none

Review	
Name, position	**Date**
Hartmut Hörner; Senior Manager	2013-12-17
Markus Schwarz; Product Architect	2013-12-17
Danilo Assmann; Quality Assurance	2013-12-17

Revision list				
Version	**Editor**	**Date**	**Section**	**Changes, comments**
0.1	visHH	2007-05-03	all	Initial version
0.2	visHH	2007-10-10	all	Review comments by Kip incorporated
0.3	visHH	2007-10-23	all	Review comments by Smu, Gp, Ra and Dgm incorporated
0.4	visHH	2009-10-26	1.3	References updated
			2.4	Clarify handling of code annotations. They can remain in delivered code. Examples added.
0.5	visHH	2010-01-22	2.1 2.3 2.4	Added test report generated by CDK Describe workflow variant with CDK Additional code annotation at end of file specified
0.9	visRa	2011-08-03	all	No change in general usage strategy compared to version 0.5 but clarifications added and focus set on “justification in code”. Tasks and roles added. Background information added, tool usage added, code annotation described in more detail. References changed to new locations on server.
1.0	visRa	2011-08-04	all	Review remarks added (visEt, visMas, visDan)
1.1	visHh	2011-11-16	6.6	ISO26262 chapter reworked.
1.2	visRa	2012-05-11	6.7	Hints for GOTO justification added
2.0	visRa	2013-04-22	al l	Links adopted, chapter “Code Metrics” completely reworked, chapter “Help on MISRA Justification Activity“ added, usage of “ “ between comments as mandatory added, recommendation to name MISRA rule in justification text added. Link on MICROSAR DeviationDocument replaced by link to ComplianceDocument.

Revision list				
Version	**Editor**	**Date**	**Section**	**Changes, comments**
2.1	visRa	2013-12-17	3	Keyword #ifdef `VECTOR_CHECK_MISRA` for #pragma usage added.

Contents

Figures

Tables

1 Introduction

The Motor Industry Software Reliability Association (MISRA) has published coding rules for embedded software development C projects [MISRA-C]. These rules shall help to produce more robust and reliable C code for Electronic Control Units (ECUs) and to improve reusability and portability of single software components or whole designs.

The purpose of this work instruction is to define the process how the "MISRA-C:2004 guidelines for the use of the C language in critical systems" are applied in component, product, and program development projects (further referred as project within this document).

It is in general the goal for all embedded software within PES to be MISRA compliant. But there are always constraints in projects that prevent to fulfill all MISRA rules at a 100%. Thus this document will also give instructions how MISRA deviations have to be handled. Furthermore this document describes the necessary steps to get a MISRA test report using different tool chains to enable us to claim MISRA compliance for the product.

Because [MISRA-C] strongly recommends the use of code metrics and the used tool provides code metrics calculation, this document also describes the measured of code metrics (*5 Code Metrics*).

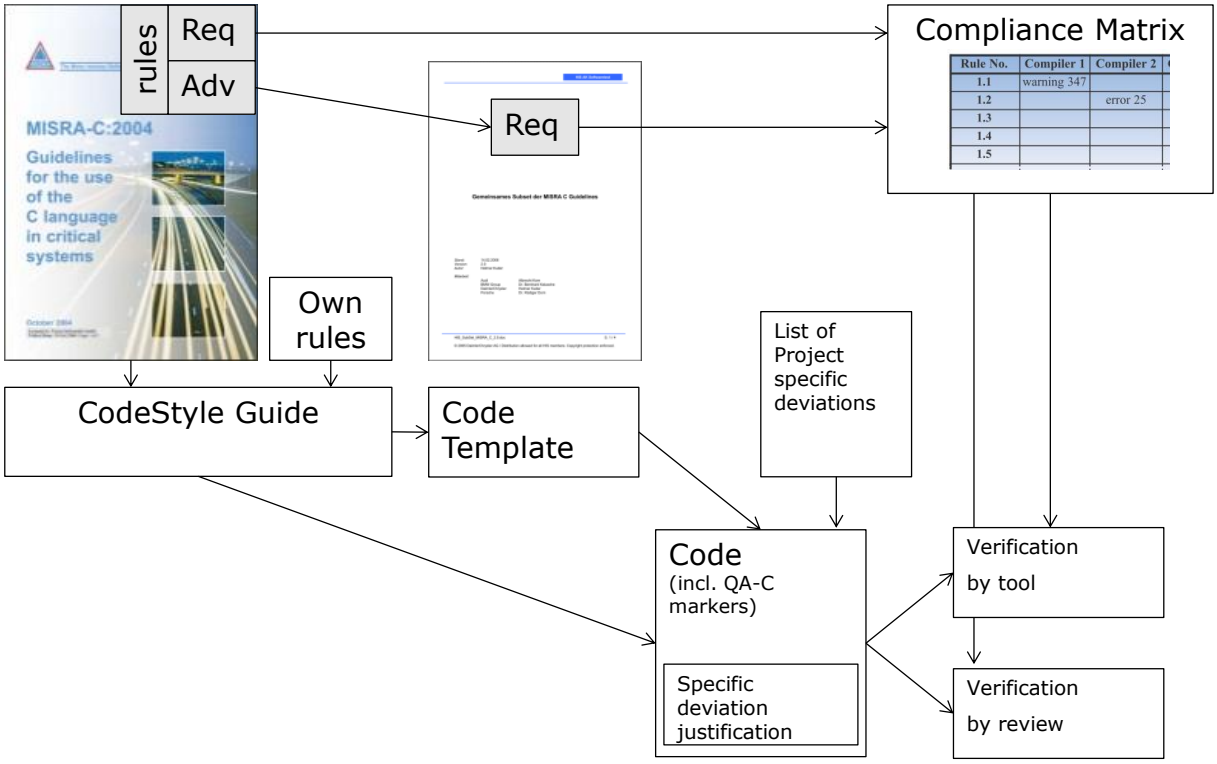


Figure 1-1 Overall MISRA process: MISRA standard - code style guides - code - compliance matrix - ...

TIER1s are forced to report the MISRA status of their ECU project to the OEM. PES embedded code is part of that ECU project and is thus analyzed, too. The approach for MISRA analysis and justification in the code described here addresses that need.

MISRA analysis and justification activities are located in Q IV of the product-line process model. The expected outcomes of that activity are:

- Code with as less as possible MISRA deviations
- Code that contains justifications for all remaining MISRA deviations
- A local MISRA analysis report as evidence data that the analysis has been executed

Based on the data produced in QIV, PES is able to claim MISRA compliance for the delivered products (SIPs). Basically, we claim the compliance and refer to our development process. If customers need more and detailed evidence, the process is described in *6.7 How to handle customer requests on MISRA Analysis for Product Deliveries*.

1.1 Scope

This document is directed to all employees of the Vector Group Product Line PES (Embedded Software) involved in implementation activities within component, product and program development.

See *6.2 Application Areas* for more information on the scope.

1.2 References

No.	Document
[MISRA-C]	https://vglobpessvn1.vg.vector.int/svn/Specifications/Standards/MISRA/MISRA-C_2004/trunk/misra_C_2004.pdf
[HIS-WEB]	http://www.automotive-his.de
[HIS-MISRA]	https://vglobpessvn1.vg.vector.int/svn/Specifications/Standards/HIS/HIS_SubsetMISRA_C/trunk/HIS_SubSet_MISRA_C_2.0.pdf
[HIS-METRIC]	https://vglobpessvn1.vg.vector.int/svn/Specifications/Standards/HIS/HIS_SourceCodeMetrics/tags/1.03.01/his-sc-metriken.1.3.1.pdf
[ComplianceCBD]	http://www-serv.vi.vector.int/portal/medien/process_management/process_documents/pes/Development/Standard_Component_Development_1/TP_Dev_ProjectDocumentation/doc/30_ImplementationPhase/Implementation/WI_MISRADeviationsCmpDevCBD.pdf
[ComplianceMSR]	http://www-serv.vi.vector.int/portal/medien/process_management/process_documents/pes/Development/Standard_Component_Development/TP_Dev_ProjectDocumentation/doc/30_ImplementationPhase/Implementation/WI_MISRA_Compliance_MSR.pdf
[AppNote]	https://vglobpessvn1.vg.vector.int/svn/zCommon/ApplicationNotes/MICROSAR/AN-ISC-8-1135_StaticCodeAnalysisMISRACheck/base/Doc_ApplicationNote/trunk/AN-ISC-8-1135_StaticCodeAnalysisMISRACheck.pdf
[WIKI_MISRA]	http://wiki.vi.vector.int/wiki-pes/CDK/MISRA
[WIKI-QAC]	http://wiki.vi.vector.int/wiki-pes/Tools/QAC
[WIKI-DOC]	http://wiki.vi.vector.int/wiki-pes/QAC_MISRA_Analysis
[WIKI_ProcessManagement]	http://wiki.vi.vector.int/wiki-pes/ProcessManagementInformation
[WIKI-GenTool]	http://wiki.vi.vector.int/wiki-pes/GenTool%20GENy/MISRAComments
[PES-Intranet]	http://www-serv.vi.vector.int/vi_pm_development_pes_en,,186.html
[CDK-MsgRef]	\\vistrpesfs1\Project2\CAN_Base\CANtate\zCDK\TestReport_References\Code_Qac_MsgRef.html
[CDK-Justifications]	\\vistrpesfs1\Project2\CAN_Base\CANtate\zCDK\Data\MisraJustification.html
[CDK-UsageLog]	\\vistrpesfs1\Project2\CAN_Base\CANtate\zCDK\Data\CDK_DashBoard.html
[QAC-UsageLog]	\\vi.vector.int/backup/PES/DevelopmentTools/GraphitePesLicensesView/index.html
[QAC-HIS-Guide]	https://vglobpessvn1.vg.vector.int/svn/Specifications/Standards/HIS/HIS_SubsetMISRA_C/trunk/QAC_HIS_Quick_Guide.pdf

No.	Document
[QAC_26262]	http://www.programmingresearch.com/news/iso26262_nov.html
[AUTOSAR]	https://vglobpessvn1.vg.vector.int/svn/Products/MICROSAR/AUTOSAR_Specification/tags/3.02.01/02_Auxiliary/AUTOSAR_SWS_C_ImplementationRules.pdf
[QAC-Guide]	\\vi.vector.int\backup\PES\DevelopmentTools\others\PRQA\qac7.0\doc\pdf\QAC-7.0-Win-UsersGuide.pdf
[QAC-MsgSummary]	\\vi.vector.int\backup\PES\DevelopmentTools\others\PRQA\qac7.0\doc\pdf\MCM-4.1-QAC-7.0-QAC-Message_Usage_Summary.pdf
[QAC-RuleEnforcement]	\\vi.vector.int\backup\PES\DevelopmentTools\others\PRQA\qac7.0\doc\pdf\MCM-4.1-QAC-7.0-Rule_Enforcement.pdf
[QAC-RuleRef]	\\vi.vector.int\backup\PES\DevelopmentTools\others\PRQA\qac7.0\doc\pdf\MCM-4.1-QAC-7.0-Rule_Xreference.pdf
[C99]	https://vglobpessvn1.vg.vector.int/svn/Specifications/Standards/ISO/ISO9899_ProgrammingLanguages_C99/trunk/INCITS%2bISO%2bIEC%2b9899-1999.pdf

2 Enforcing MISRA-C Compliance

2.1 Overview & Mapping

Because most readers of this WI are interested in "what" instead "why", the "why" information is located in chapter *6.1 Process Mapping "Using MISRA-C"*.

2.2 Deviation Procedure

The MISRA standard already states that a strict adherence to all rules in an embedded code is unlikely and specifies how to handle deviations (see chapter 4.3.2 Deviation procedure in [MISRA-C]).

2.2.1 Project Deviation

Project specific deviations are explicitly mentioned in [MISRA-C] as a method to permit relaxation of the rule set for a specific scope. PES has set up product specific compliance documentation containing the product specific Project Deviation list, e.g. [ComplianceCBD], [ComplianceMSR].

They are focused on specific products but can be referenced by other projects, too.

2.2.2 Specific Deviation

In addition to the commonly defined project deviations, specific deviations may occur on component level.

The MISRA standard specifies how specific deviations have to be justified. This includes

- Number of the rule which is violated
- Reason why a deviation occurs and is not avoidable
- Potential risks which may arise from the deviation
- Prevention of risks where applicable

2.3 HIS Subset

The MISRA-C:2004 "subset" defined by HIS [HIS-MISRA] requires compliance with all rules defined in [MISRA-C]. Thus all rules have the status "required" and not "advisory" independent of the assignment in [MISRA-C].

2.4 PES MISRA Compliance Rules

The standard PES MISRA rule subset consists of all MISRA rules.

Deactivation of rules in the QA-C message personality file is forbidden.

Table 2-1 describes the priority of the rules to be applied when analyzing MISRA deviations.

Priority	Rule	Rationale
1	Embedded software developed by PES shall be fully MISRA compliant. Thus deviation from MISRA rules shall be prevented wherever possible.	A MISRA deviation has to be checked each time an analysis is executed and thus may create additional effort. In addition, customers may run an own analysis and request to justify each deviation that also will result in additional effort.
2	If a MISRA deviation occurs, try to change/extend the code to become MISRA compliant.	If code is MISRA compliant, there is no additional effort for justification, review of justification, discussion with customers …
3	If fixing the deviation is not a solution, mark the deviation as known and add a justification into the code.	There might be more important requirements than "MISRA compliance" like e.g. stick to standards (ISO, AUTOSAR, OEM specification, …), preventing inefficiency (RAM/ROM footprint, runtime, …), PES single-source maintainability (our big #ifdef system, …) that prevents us from fixing the deviation – so we have to spend time in analysis, justification, review of justification …

Priority	Rule	Rationale
4	If the occurrence of a deviation depends on specific configurations, add the justification **without additional #ifdef or other checks (e.g. by CodeGenerator)** into the code.	Keep the code as simple and readable as possible! Do not add #if statements to prevent or enable a QA-C marker or justification in the code for a specific configuration! If a justification is in the code that is sometimes not true for a given configuration, describe this (briefly) in the justification text. Do not spend effort to remove the justification depending on the configuration!

Table 2-1 Priority of PES MISRA compliance rules

3 MISRA Code Annotation and Justification Handling

If the code contains either a *Project Deviation* or a *Specific Deviation* that cannot be avoided, the deviation in the code has to be marked as "known" and a justification has to be provided.

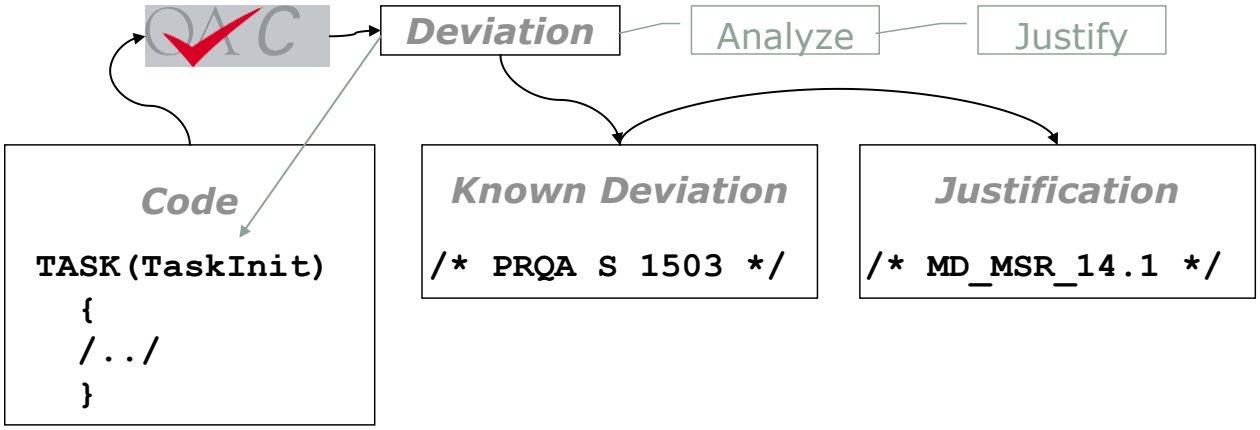


Figure 3-1 Wording for the steps and actions of a MISRA analysis

QA-C provides different commands to mark a deviation as *known* in the report:

- Single line suppression: `/* PRQA S <rule> */`
- Range suppression using a tag: `/* PRQA S <rule> <tag> */`
  …
  `/* PRQA L:<tag> */`
- Range suppression using line counting: `/* PRQA S <rule> <lines> */`

The use of single line suppressions is the most recommended style.

Range suppressions, especially "Range suppression using line counting" shall be used carefully and only for short areas (less than 60 lines) due to possible side-effects when code is e.g. reworked for maintenance.

QA-C commands are added to the code by default without pre-processor encapsulation, i.e. the QA-C command is always "active".

It is not recommended to use `#pragma PRQA_MESSAGES_OFF` and `#pragma PRQA_MESSAGES_ON`, but if they are used, a justification comment is necessary and the #pragma block has to be encapsulated with a specific #ifdef block. I.e. the block shall look like:

```
#ifdef VECTOR_CHECK_MISRA
   #pragma PRQA_MESSAGES_OFF abcd
#endif
```

Each QA-C command to mark a deviation as known shall be commented as described in *2.2.2 Specific Deviation*.

The necessary amount of justification text will break the readability of the code. Thus PES has defined a rule how to add all this information into the code.

The basic idea is that the justification is as short as possible and the additional justification text is located in a standard deviation document or is in the code but not necessarily at the location of the deviation. Figure 3-2 explains the idea and dependencies.

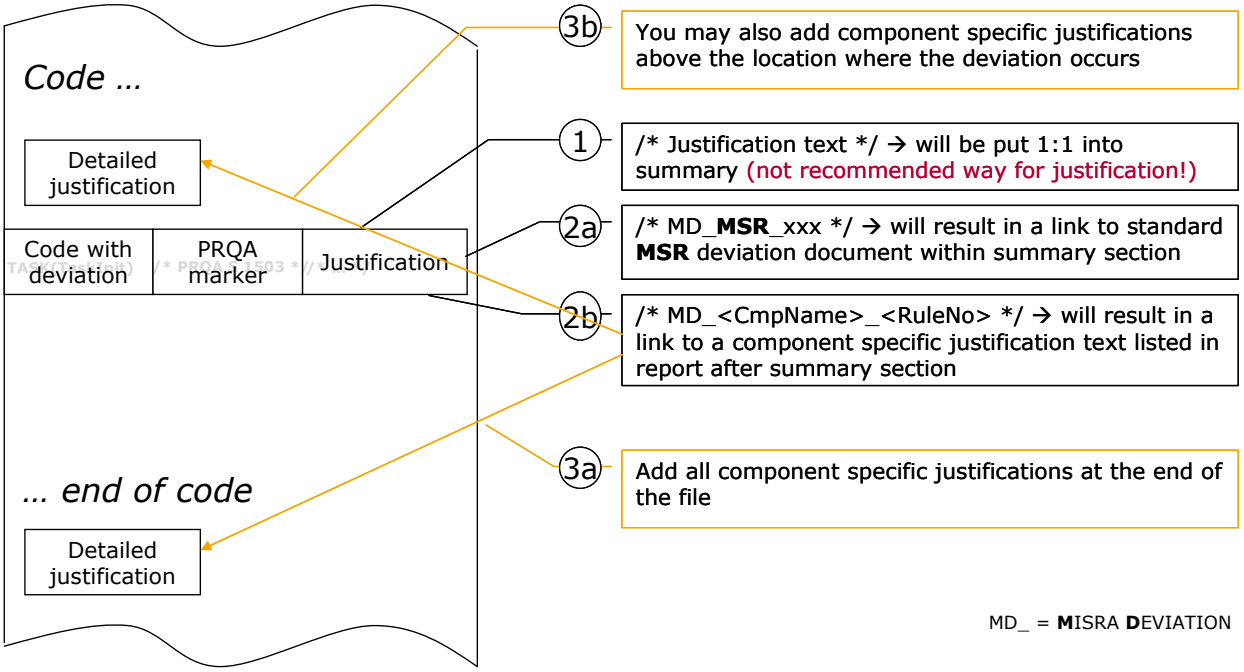


Figure 3-2 Location of justification comments in code

#	Justification style	Usage recommended?
1	/* Justification text */ will be used 1:1 into MISRA report (Add at least one blank between the PRQA marker comment and this comment!)	No – old style, mentioned just for compatibility to existing code. Do not use for new components or new justifications in existing code.
2a	/* MD_**MSR**_xxx */ or /* MD_**CBD**_xxx */ will result in a link to assigned compliance document containing the *Project Deviation* list. No further justification text necessary within the code. See *3.1 Deviation is listed in <Project Deviation* for more details.	Yes - for <Project Deviation>
2b	/* MD_<CmpName>_<RuleNo> */ is interpreted as a link to a component specific justification text within the code. For this type of justification style, the justification text location can be chosen according Table 3-2. See *3.2 Deviation is a <Specific Deviation>* for more details.	Yes - for <Specific Deviation>

Table 3-1 Justification styles in code (according Figure 3-2)

The " " (blank) between the two comments is mandatory due to some compilers have issues with two consecutive comments without at least one white-space character in-between!

`I.e. use this style: /* PRQA S 5087 */ 1_BLANK_HERE /* MD_MSR_19.1 */`

#	Justification location	Usage recommended?
3a	All component local justifications are located at the end of the (static) .c file.	Yes – recommended location
3b	Each justification block is close to (above) the code line where the deviation occurs. Due to this will reduce the readability of the code, it is not recommended for new components. In addition, if a justification matches for more than one deviation it has to be repeated many times in the code.	No –not recommended due to it will break the readability of the code

Table 3-2 Justification location in code (according Figure 3-2)

Note: [AUTOSAR] describes in [PROG_086] a commenting style similar to our "old style – not recommended" one. Nevertheless, our "new" style with the link fits into the requirement, too, and has the advantage that our code remains maintainable (due to better readability) and we can add more and better information into our justifications.

3.1 Deviation is listed in <Project Deviation>

If a reported deviation is on the *Project Deviation* list of the product, check if the explanation of the "Reason" is also sufficient for the locally found deviation.

If deviation is not on the project deviation list or explanation does not fit, see *3.2 Deviation is a <Specific Deviation>* how to handle the deviation.

If deviation is on the list and the explanation is correct, too, use this way to add a justification into the code:

```
CodeCodeCodeCode  /* PRQA X <RuleNo> */ /* MD_<MSR|CBD>_<MISRA_rule> */
```

i.e. it may look like this in real code:

```
#include "MemMap.h"  /* PRQA S 5087 */ /* MD_MSR_19.1[1] */
```

The (middle-fix part of the) name of the justification link is used to select the specific project deviation list:

- /* MD_**CBD**_xxx */ references into the CANbedded project deviation list in [ComplianceCBD].
- /* MD_**MSR**_xxx */ references into the MICROSAR project deviation list in [ComplianceMSR].

3.2 Deviation is a <Specific Deviation>

If a reported deviation is **not** on the *Project Deviation* list or the standard explanation does not fit, use this way to add a justification into the code:

```
CodeCodeCodeCode  /* PRQA X <RuleNo> */ /* MD_<CmpName[2]>_<RuleNo[3]> */
```

i.e. it may look like this in real code:

```
"case NM_STATE_READY_SLEEP:  /* PRQA S 2003 */ /* MD_CanNm_2003 */"
```

If more than one MISRA deviation occurs in one line of code, this usage is supported, too:

```
CodeCodeCodeCode  /* PRQA X <RuleA>,<RuleB> */ /* MD_<CmpName>_<RuleA>,
MD_<CmpName>_<RuleB> */
```

The justification comment MD_<CmpName>_<RuleNo> is used as a link to the justification text within the code.

The name of the justification comment shall be a descriptive name according the rules listed here:

- The name of the justification link shall always **start** with MD_ (MD = MISRA Deviation).
- It is strongly recommended to use the component name <CmpName> as the links` **middle-fix** (e.g. CanNm_). At least the chosen name has to be unique also in an integrated project consisting out of many different components (aka SLP).
- It is strongly recommended to use the PRQA rule number <RuleNo> as the links` **postfix** (e.g. 2003) to have a common understanding between different implementation engineers and also the review engineers.

The very same justification can be used for more than one deviation, i.e. if the very same deviation occurs on more than one line in the code (including matching reason, risk and prevention) only one justification text is necessary.

On the other hand, if a specific deviation occurs more than once in the code but different reasons/risks/preventions are necessary, use a unique justification and justification text for each deviation (e.g. MD_CanNm_2003_DeviationType1, MD_CanNm_2003_DeviationType2)

---

[1] Example uses the name of a standard project deviation – use always the name defined in the assigned compliance document.

[2] CamelCase style for <CompName> is recommended; if <ModuleShortName> is defined, use it!

[3] For own, specific deviations, RuleNo shall be the number of the QA-C rule to be able to distinguish between different deviation types in the same line and in code. (See example with QA-C rule 2003 and MD_CanNm_2003)

3.3 Justification Handling for <Specific Deviation>

The justification text for a *Specific Deviation* in the code has to match a given format so that our reporting tools can easily access the specific justifications.

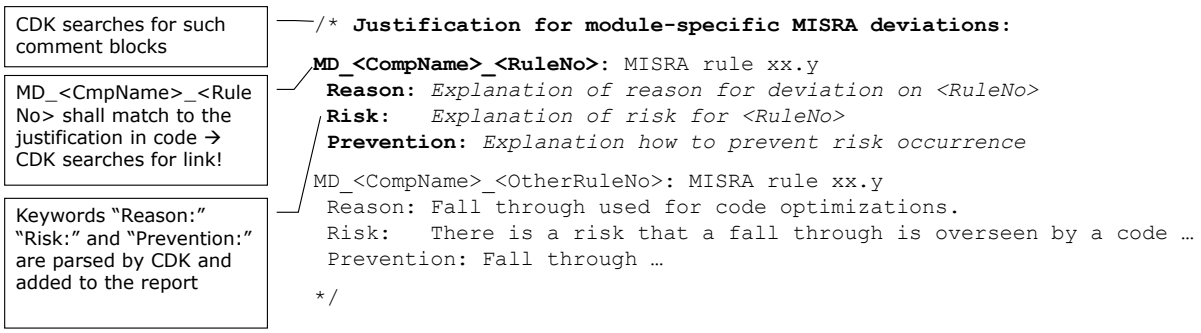


Figure 3-3 Keywords for justification text of <*Specific Deviation*>

Justification comment block:

- Each justification comment block shall start with "Justification for module-specific MISRA deviations:".
- More than one justification text is allowed within one justification comment block.
- If there is only one justification comment block at the end of the file, one such marker is sufficient. If more than one justification comment block is used, each block shall start with such a comment.
- One justification comment block per component is sufficient. I.e. if MISRA deviations occur in the static code and in the generated code, justifications for all deviations can be located in the static code (.c file) or each code can have its own justification comment block (i.e. in each .c file).

Justification link:

- The justification link used within the code has to be system-wide uniquely (and matching the used name at the deviation!).
- See *3.2 Deviation is a* <*Specific Deviation*> for naming rules (e.g. MD_<CmpName>_<RuleNo>)
- Add also deviated MISRA rule in the same line. E.g. MISRA rule 19.13

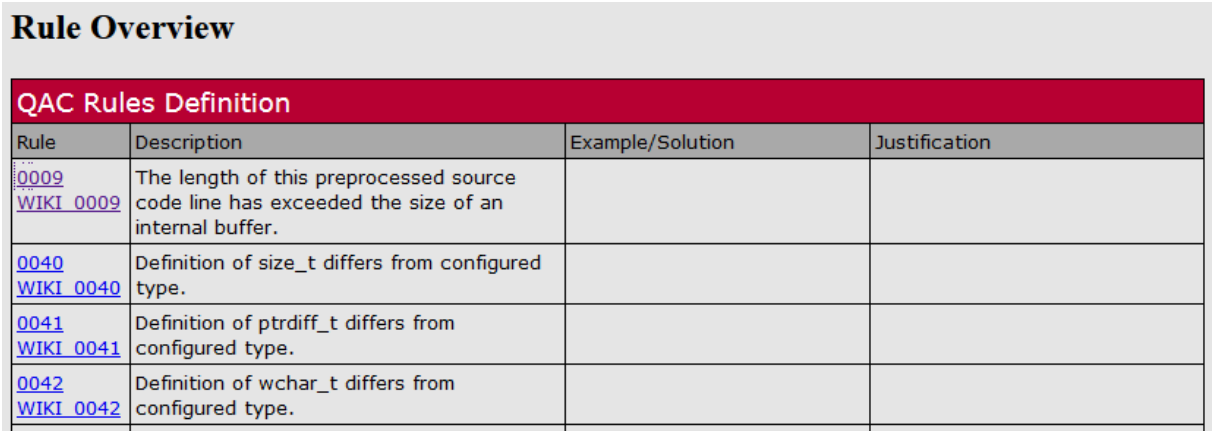
Keywords and justification text content:

- The keywords "Reason:", "Risk:" and "Prevention:" trigger the separation of the different topics for the report generation.
- Do not use the keywords twice in the justification text!
- Keep the justification text short but precise.
- Help on creating justification text can be found in [CDK-MsgRef].

The PES MISRA reporting tools parse all .c files for justification comment blocks, then for the justification text links and then for the "Reason:", "Risk:" and "Prevention:" keywords. Then they summarize the findings and create a HTML file with all information linked.

3.4 Help on MISRA Justification Activities

Code_Qac_MsgRef.html [CDK-MsgRef] provides a QA-C rule overview including a link on the original/complete text for each rule into MISRA standard [MISRA-C] and another link on a rule-specific page in the PES WIKI.

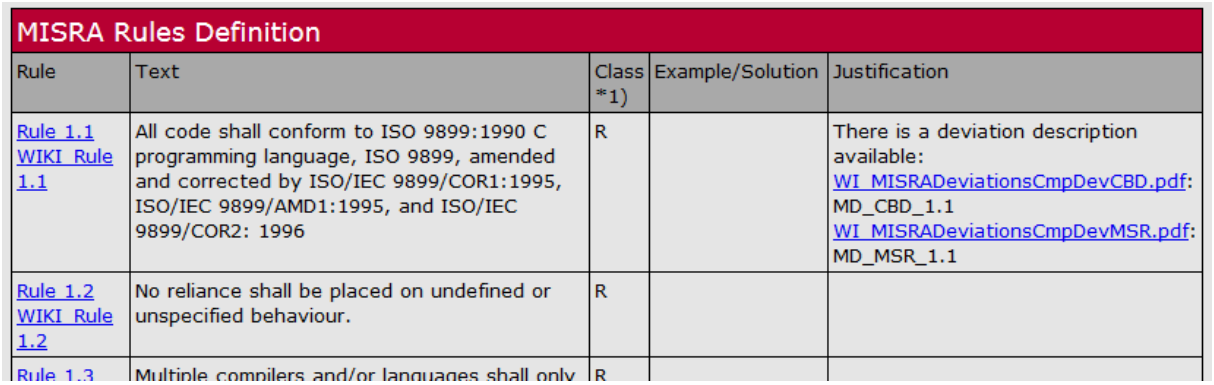


Rule Overview

QAC Rules Definition

Rule	Description	Example/Solution	Justification
0009 WIKI_0009	The length of this preprocessed source code line has exceeded the size of an internal buffer.		
0040 WIKI_0040	Definition of size_t differs from configured type.		
0041 WIKI_0041	Definition of ptrdiff_t differs from configured type.		
0042 WIKI_0042	Definition of wchar_t differs from configured type.		

Figure 3-4 ScreenShot of PES-internal QA-C message help page



MISRA Rules Definition

Rule	Text	Class *1)	Example/Solution	Justification
Rule 1.1 WIKI_Rule 1.1	All code shall conform to ISO 9899:1990 C programming language, ISO 9899, amended and corrected by ISO/IEC 9899/COR1:1995, ISO/IEC 9899/AMD1:1995, and ISO/IEC 9899/COR2: 1996	R		There is a deviation description available: WI_MISRADeviationsCmpDevCBD.pdf: MD_CBD_1.1 WI_MISRADeviationsCmpDevMSR.pdf: MD_MSR_1.1
Rule 1.2 WIKI_Rule 1.2	No reliance shall be placed on undefined or unspecified behaviour.	R		
Rule 1.3	Multiple compilers and/or languages shall only	R		

Figure 3-5 ScreenShot of PES-internal MISRA rule help page

The link onto the rules shall help to quickly access the concrete description of the QA-C and MISRA rules. The table shows also if a project specific justification is available.

The link into the WIKI enables everybody to support and annotate issues and information concerning the used tooling, specific rule usage in our software and more[4].

An overview on ALL used MISRA justifications in ALL products and ALL components is available in MisraJustification.html [CDK-Justifications]. The report is created on request by the product architect.

[4] The content of the WIKI is not reviewed and released by QA. The user is responsible to validate the information before using it. In case of unclear information, contact the product architect for clarification.

4 MISRA Analysis Process

4.1 Roles and Tasks

This chapter describes the tasks and roles for MISRA analysis on component level. If MISRA analysis shall be executed on other development levels, it shall be used in a similar way.

Component Manager

- is responsible to define for which work products of the component (i.e. sub-packages Implementation, GenTool_Xxx, TsiXxx, TscXxx ...) a MISRA analysis is necessary (see *6.2 Application Areas*).
- is responsible to define the configuration and use-cases for which a MISRA analysis is to be executed (see *6.3 Configuration and UseCases*).
- is responsible to define if for the current development task a MISRA analysis is to be executed or if the last MISRA report shall still be valid (see *6.4 Frequency of MISRA Analysis*).

Implementation Engineer

- is responsible to execute the MISRA analysis on all defined work products and with all defined configurations and use-cases
- is in charge to analyze reported deviations, fix or justify them
- creates the MISRA test report summary (see *6.5 MISRA Analysis Report*)

Review Engineer

- is responsible to review the MISRA test report summary as part of the code inspection concerning
  - "Is the reported deviation really not avoidable or can it be fixed?"
  - "Is the assignment as Project Deviation or Specific Deviation correct?"
  - "Is the justification for a Specific Deviation understandable and correct?"
- logs review findings in ReviewLogSheet_Code.xls (or similar document where code inspection results are logged)

Quality Engineer

- is responsible to check in quality gate Implementation, if a current MISRA test report summary is available or a justification for reuse of an older MISRA test report summary is part of the component release documentation
- logs quality gate findings in appropriate QG log

Architect

- is responsible to define if a Project Deviation can be applied on a project (i.e. in the context of this WI this is a new product like MSR, CBD, ...) or a new project deviation document is to be created (see *6.8 Hints for Project Specific Deviation Documents*)

4.2 Workflow, Inputs and Outputs

#	Step	Input	Output
1	Run QA-C to execute MISRA analysis	Source code QA-C configuration (generated by BRS or CDK)	MISRA analysis report (see *6.5 MISRA Analysis Report*)
2	Fix deviations	Source code MISRA analysis report	Modified source code

#	Step	Input	Output
3	Justify deviations (by Project Deviation or Specific Deviation). The QA-C message reference [CDK-MsgRef] and the PES WIKI [WIKI_MISRA] provides specific help for this task	Source code MISRA analysis report	Annotated source code with justifications: PRQA marker to mark deviation as known Justification link in line where deviation occurs Justification text to explain reason, risk and prevention as described in Table 3-1 and Table 3-2.
4	Repeat step 1 to 3 until all deviations are fixed or justified		
5	Execute code inspection (see *4.1 Roles*)	Annotated source code with justifications MISRA analysis report incl. justification Code inspection checklist	Code inspection report
6	Execute quality gate Implementation (see *4.1 Roles*)	MISRA analysis report incl. justification Code inspection report	QG report

Table 4-1 Input and output documents for MISRA analysis

5 Code Metrics

5.1 CodeMetrics 04/2013 and later

See presentation CodeMetrics_at_PES for description of the metric measurement and justification process. This WI will updated in a later release!

5.2 CodeMetrics until 04/2013

Code metrics are "expected to be necessary for good software quality". HIS [HIS-WEB] has set up a document to describe requirements on code metrics [HIS-METRIC] with a first draft in 2005 and the latest release in 2008.

Software of TIER1 for HIS members is expected to comply with HIS metrics. Thus, the TIER1 has to report and justify deviations on function and file level. PES software is known to often exceed the HIS metric thresholds.

PRQA QA-C is explicitly mentioned in [HIS-METRIC] and supports most of the required code metrics. Each time a MISRA analysis is executed, QA-C calculates code metrics, too.

On violation of a metric, QA-C generates the message 4700 "Metric value out of range".

Priority	Rule	Rationale
1	Measure the metric	It is good to known what happens in my code …
2	Analyze the results for code metrics that exceed the specific HIS threshold	There might be more important requirements than "keeping HIS threshold" like e.g. implementing standards (ISO, AUTOSAR, OEM specification, preventing inefficiency …).
3	If no more important reason prevents a change – fix it	If a code metric is in its "green" are no special action is necessary and also no customer justification request came up …
4	There is no need to justify code metrics that exceed a specific HIS threshold.	

Table 5-1 Priority of PES HIS metric compliance rules

Metric deviations can be used in the component development process to set focus on specific areas of the code, e.g.

- check if the design/coding is sufficient or not
- what & where are the complex areas of your component
- define code inspection areas based on code areas highlighted here
- assign additional test cases on code areas highlighted here…

See *6.10 HIS Metric Compliance Matrix* how a HIS metric report created by QA-C and visualized by CDK may look.

6 Appendix

6.1 Process Mapping "Using MISRA-C"

[MISRA-C], chapter 4, describes the necessary definitions, activities and evidence artifacts to claim compliance to MISRA-C for a project[5] (product). This chapter maps those requirements onto the solution used by PES.

- Requirements described in chapter "4.1 The software engineering context" are covered by the AUTOMOTIVE SPICE Level 3 development process used at PES for product development.
- Requirements described in chapter "4.2.1 Training"
  - Training on the use of C for embedded applications → [WIKI_ProcessManagement]
  - Training on the MISRA analysis tool usage and how to reach compliance → [WIKI_ProcessManagement]
- Requirements described in chapter "4.2.2 Style guide"
  - Use of code-style guide → [PES-Intranet]
- Requirements described in chapter "4.2.3 Tool selection and validation"
  - Complier selection → see *6.9 MISRA Compliance Matrix*, justification for rule 1.4
  - Code checker tool →*6.11 Tool PRQA QA-C*
  - Metrics → *5 Code Metrics*
- Requirements described in chapter "4.2.4 Source complexity metrics"
  - Metrics → *5 Code Metrics*
- Requirements described in chapter "4.2.5 Test coverage"
  - Concept described in WhitePaper_Test [PES-Intranet]
- Requirements described in Chapter "4.3.1 Compliance matrix"
  - Compliance matrix → *6.9 MISRA Compliance Matrix*
- Requirements described in Chapter "4.3.2 Deviation procedure"
  - Devine a deviation procedure →*2.2 Deviation Procedure*
- Requirements described in Chapter "4.3.3 Formalization with quality system"
  - Formalize working practice → this document
- Requirements described in Chapter "4.3.4 Introducing the subset"
  - Code-style guide → [PES-Intranet]

6.2 Application Areas

6.2.1 "Static" and "Generated" Embedded C Code

All embedded C code (.c and .h) that is delivered to customers and is intended to be used in serial-production shall be analyzed for MISRA deviations.

This definition typically includes the "static" code and the "generated" code of a component.

The handling for "static" code parts of a component is straight-forward as described in this document.

The handling for "generated" parts of a component in the tools is described in [WIKI-GenTool]. As already mentioned, the justification text can be located within the static code parts which are easier to change and update. Due to MISRA deviations (and especially the reason/risk for a deviation) are not so volatile between different releases

[5] MISRA itself focuses on ECU projects and uses thus the word "project". PES develops products – in this context we use "product". In the context of this WI both words address the embedded software developed by PES.

of a component; this additional dependency (i.e. between packages Implementation and GenTool_Xx) does not break the package independency concept.

6.2.2 Embedded C Code in Test Environment and Demo

Test environment and demo code is usually not intended to be used in serial production. To reduce the amount of MISRA deviations reported in a project it is recommended (in the meaning of optional) to be MISRA compliant, too.

It is thus not required for test environment and demo code to:

- analyze reported MISRA deviations
- mark deviations as known
- create justifications

6.2.3 Files not originated in the Component Context

Header files of other software components or of compiler libraries which do not belong to the component under analysis shall not be analyzed by excluding them from the QA-C configuration.

6.3 Configuration and UseCases

The MISRA analysis shall be applied on at least one (maximum feature) configuration of a project.

If a project has many different features that cannot be analyzed at once it is strongly recommended to apply the analysis on a basic set of configurations to get at least a line coverage of 100% enabled code lines (merged over all configurations of the project).

Additional rule for component development scope:

If a deviation is found during an analysis, check the code (and the code generator) for occurrence of similar deviations and try to handle all locations at once – not only the code lines currently reported. This approach allows us to minimize the number of to be analyzed configurations per project and reduces the risk of late change requests for fixing other, similar deviations in the code.

6.4 Frequency of MISRA Analysis

As explained in the *Introduction* the goal for PES is to have 100% MISRA compliant code, either by deviation-free code or by code + justification of deviations.

MISRA analysis and creation of a report is mandatory for:

- release 1.00.00 of a component
- each main version release (aa++.00.00)
- sub-version releases (xx.bb++.00) voted to have MISRA relevant changes

It is recommended to execute the MISRA analysis for each release of a component to verify that no new deviation has occurred and to have an up-to-date report of the MISRA status available.

Hint: to reduce risk of larger changes to gain MISRA deviation free code, it is strongly recommended to execute the MISRA analysis in parallel to the development activities and not only as a release action.

6.5 MISRA Analysis Report

Due to PES is applying the MISRA analysis for many years, different methods to execute and document MISRA analysis has been introduced and are still in use. This WI describes only two methods but also older types of MISRA reporting are allowed to be used.

The HTML based, generated MISRA report including all justifications is the latest and thus recommended way to document the analysis results.

6.5.1 HTML based MISRA Analysis Report

If a component applies the code annotation as described in *3 MISRA Code Annotation and Justification Handling*, the most efficient way to create a MISRA analysis is to apply the HTML based MISRA report created via CDK (or a similar tool).

If code annotations are not possible, appropriate justifications shall be added in the general test report of the component.

The name of the HTML based MISRA report is TestReport_MISRA2004.html.

The file shall be located in the 30_ImplementationDocu/TestReports folder of the component.

Note: if applying the HTML based MISRA report make sure to remove older files (e.g. TestReportMISRA2004.doc) in the configuration management.

6.5.2 MS-WORD based MISRA Analysis Report

The MS-Word based MISRA analysis report is based on a template (see PES process pages in intranet). The template is filled out manually.

The name of the MS-word based MISRA report is TestReportMISRA2004.doc.

The file shall be located in the 30_ImplementationDocu folder of the component.

Note: if applying the MS-Word based MISRA report do not store other MISRA report results on the same level. Use a sub-directory called TestReports or WorkingProducts for other information.

A batch file is available that ease up the creation of the MS-Word based MISRA report. See [WIKI-DOC] for more information.

6.6 Mapping on ISO 26262

Some requirements from ISO 26262-6 can be covered by MISRA rules. The explanations in the following sub-sections give guidance which MISRA rules are important in safety related projects. It has to be checked in the project specific context if the following mapping is sufficient or if additional measures are required. The following references refer to the FDIS version of ISO26262.

6.6.1 Requirement 5.4.7 – Table 1

ISO 26262, Part 6: Product development: software level describes in “Table 1- Topics to be covered by modeling and coding guidelines”:

Technique/Measure	ASIL A	ASIL B	ASIL C	ASIL D
1a Enforcement of low complexity	++	++	++	++
1b Use of languages subsets	++	++	++	++
1c Enforcement of strong typing	++	++	++	++
1d Use of defensive implementation techniques	o	+	++	++

Table 6-1 ISO 26262-6, “Table 1- Topics to be covered by modeling and coding guidelines”

The objectives of method “Use of languages subsets” are

- Exclusion of ambiguously defined language constructs which might be interpreted differently by different modelers,
- programmers, code generators or compilers.
- Exclusion of language constructs which from experience easily lead to mistakes, for example assignments in conditions
- or identical naming of local and global variables.
- Exclusion of language constructs which might result in unhandled run-time errors.
- As a result of this definition, MISRA is the (for us) only available concept to cover such requirements.

In the following table a mapping of MISA rules to measures required by ISO26262 is shown. All MISRA rules are covered by the tool QA-C unless otherwise stated.

Technique/Measure	MISRA rules	comment
1a Enforcement of low complexity	5.3 9.2 9.3 12.4 12.5 12.10 13.1 14.2 19.5 20.1 20.2 20.4	Additionally the source code metrics described in chapter *5 Code Metrics* shall be applied.
1b Use of languages subsets	all	s. explanations above
1c Enforcement of strong typing	6.1 6.3 11.1 18.1	
1d Use of defensive implementation techniques	14.10 15.3 16.10 20.3 21.1	Note, that rule 20.3 is not checked automatically by QA-C, for details refer to chapter *6.11.3 Rules checked by QA-C*.

Table 6-2 ISO 26262-6, Table 1- mapping to MISRA rules

6.6.2 Requirement 8.4.4 – Table 8

In "Table 8- Design principles for software unit design and implementation" the following measures are listed:

Technique/Measure	ASIL A	ASIL B	ASIL C	ASIL D
1a One entry and one exit point in subprograms and functions	++	++	++	++
1b No dynamic objects or variables, or else online test during their creation	+	++	++	++
1c Initialization of variables	++	++	++	++
1d No multiple use of variable names	+	++	++	++
1e Avoid global variables or else justify their usage	+	+	++	++
1f Limited use of pointers	o	+	+	++
1g No implicit type conversions	+	++	++	++
1h No hidden data flow or control flow	+	++	++	++
1i No unconditional jumps	++	++	++	++
1j No recursions	+	+	++	++

Table 6-3 ISO 26262-6, "Table 8- Design principles for software unit design and implementation"

As already mentioned in the footnote of table 8 those measures can completely or partially be achieved by MISRA rules. Note, that table 8 refers to both design and implementation whereas MISRA only covers the implementation.

MISRA rules are covered by the tool QA-C unless otherwise stated.

Technique/Measure	MISRA rules	comment
1a One entry and one exit point in subprograms and functions	14.7 20.7	
1b No dynamic objects or variables, or else online test during their creation	20.4	
1c Initialization of variables	9.1	
1d No multiple use of variable names	5.2	
1e Avoid global variables or else justify their usage	8.7 8.10	These rules cover only partial aspects; main verification is usually by code inspection.
1f Limited use of pointers	17.3 17.6	Could be extended to 17.x Note that rule 17.3 is not covered by the tool QA-C, for details refer to chapter *6.11.3 Rules checked by QA-C*.
1g No implicit type conversions	10.1	
1h No hidden data flow or control flow	12.2 12.4 14.2	These rules cover only partial aspects; main verification is usually by code inspection.
1i No unconditional jumps	14.4 14.5 20.7	
1j No recursions	16.2	

Table 6-4 ISO 26262-6, Table 8- mapping to MISRA rules

6.6.3 Requirement 8.4.5 – Table 9

In “Table 9 - Methods for the verification of software unit design and implementation” the following relevant measures are listed:

Technique/Measure	ASIL A	ASIL B	ASIL C	ASIL D
1g Static code analysis	+	++	++	++

Table 6-5 ISO 26262-6, “Table 9- Methods for the verification of software unit design and implementation”

QA-C is a static code analysis tool which supports measure 1g.

Technique/Measure	MISRA rules	comment
1g Static code analysis	all	This method is covered by application of the tool QA-C and the related MISRA analysis process.

Table 6-6 ISO 26262-6, Table 9 - mapping to MISRA rules

6.6.4 Usage of QA-C and MISRA Results for Argument related to Safety Requirements

In addition to the before mentioned elements of ISO26262-6 analysis results may also be used as an argument that a specific safety requirement is fulfilled.

Example: During safety analysis it was found that freedom of interference is threatened by pointer arithmetic. Therefore it was decided in the safety concept hat pointer arithmetic shall not be used. This can be shown by a MISRA report which contains no violation of rules 17.x.

Note, that QA-C is not a proving tool which guarantees that code where no warning is detected is fully correct.

6.6.5 Tool Qualification Aspects

For all tools used in the development process the tool qualification according to ISO26262-8 clause 11 has to be analyzed. The following rating can typically be assumed for QA-C since it has a central role in source code checking:

TI = TI2

Justification for TI2: If the tool does not work correctly warnings in the code may remain undetected

TD = TD2

Justification for TD2: The code is also analyzed by other tools (several compilers, other static analysis tools) and it is subject to code inspection and runtime tests. Therefore a medium probability exists that not detected warnings are found otherwise.

Note that this rating has to be confirmed project specific depending on the usage of QA-C.

The resulting TCL is TCL2 and "ISO26262-8; Table 5 — Qualification of software tools classified TCL2" has to be applied.

Technique/Measure	ASIL A	ASIL B	ASIL C	ASIL D
1a Increased confidence from use in accordance with 11.4.7	++	++	++	+
1b Evaluation of the tool development process in accordance with 11.4.8	++	++	++	+
1c Validation of the software tool in accordance with 11.4.9	+	+	+	++

Table 6-7 ISO 26262-8, "Table 5 - Methods for the verification of software unit design and implementation"

The following arguments can be used for the tool qualification:

Technique/Measure	comment
1a Increased confidence from use in accordance with 11.4.7	Tool has been used for MISRA analysis of > 100 software modules over the last 4 years. Related MISRA reports can be found in the related repositories. Tool issues are collected in a central list [WIKI-QAC]. Tool usage is reported and logged in [CDK-UsageLog] and [QAC-UsageLog].
1b Evaluation of the tool development process in accordance with 11.4.8	QA-C has been certified for ISO26262 compliance by TÜV SÜD, the evaluation included an analysis of the development process at PRQA and a test of the tool [QAC_26262].
1c Validation of the software tool in accordance with 11.4.9	QA-C has been certified for ISO26262 compliance by TÜV SÜD, the evaluation included an analysis of the development process at PRQA and a test of the tool [QAC_26262].

Table 6-8 ISO 26262-8, Table 5 - arguments

6.7 How to handle customer requests on MISRA Analysis for Product Deliveries

TIER1s are required to provide MISRA compliance reports for their ECUs to the OEMs. Because part of their software is delivered by Vector, they ask us for MISRA compliance reports for our part. Due to we develop the software in a product-line approach, we cannot simply produce SIP specific MISRA compliance evidence. The problem to handle such a request is described in [AppNote].

#	Step	Input	Output
1	Customer request MISRA compliance evidence for a specific delivery.		Email with [AppNote] to customer. Also the product related compliance statement (e.g. [ComplianceMSR], [ComplianceCBD], ...) shall be send.
2	If step 1 does not satisfy the customer, it has to be checked if it is possible to send summary MISRA analysis reports created during the component development phase.		(Depends on customer acceptance and "go" by Program Manager) ZIP file with component MISRA reports.
3	If step 2 is not possible, the customer shall receive a project quotation for a SIP specific MISRA compliance report.		Further handling see *6.7.1 Customer Order "SIP specific Compliance Report"* for details.

Table 6-9 Steps for customer request "Need MISRA compliance information"

6.7.1 Customer Order "SIP specific Compliance Report"

#	Step	Input	Output
1	Customer request SIP specific MISRA compliance evidence.		Quotation to customer for a SIP specific MISRA analysis with tool QA-C.
2	Define configuration setup and confirm with customer. The setup contains: • Components • MISRA rule subset • Configuration data (Vector delivery test setup or customer generated data)	Order	Definition for to be analyzed components and rules.
3	Prepare analysis setup	Agreed configuration setup	
3a	Create folder - Use short names and paths, e.g. C:\CBDyyxxxxx\		
3b	Copy all delivered component .c and .h files into the very same folder.		
3c	Add configuration files (.c, .h) into the same folder.		
3d	(Especially if customer config is used) Check configuration data and remove all not used .c files[6]. Do not remove unused .h files.		Folder containing all "to be analyzed" files.
3e	Configure QA-C (recommended: via CDK Folder-Support)		QA-C project
3f	Run QA-C to detect missing header files[7].		List of missing header files

[6] SIP may contain more components than used by the customer in the requested project (e.g. SIP has CAN/LIN/FR) but customer uses just FR. This will reduce the amount of to be analyzed c. files.

#	Step	Input	Output
3g	Create missing headers as empty files with required names to satisfy QA-C preprocessor.		
4	Run QA-C to detect unjustified MISRA deviations		
4a	Vector code: analyze deviation and justify or create issue ticket. Start with deviations caused by Vector header files to get strong reduction quickly.		
4b	Non-Vector code: disable checks for the related files or add justification that this is no Vector code		
5	If all deviations are analyzed, create justification report.		MISRA compliance report
6	Deliver results to customer	• Folder with all .c/.h files (Files including the added justifications, but otherwise unchanged) • MISRA compliance report	Email with data as ZIP file incl. short notes onto the gained project result.
7	Inform PES sales for "project done" and triggering the invoice process.		

Table 6-10 Steps for customer order "SIP specific MISRA Compliance Report"

[7] Especially customer projects may reference header files which are not available and are also not helpful regarding MISRA compliance analysis results.

6.8 Hints for Project Specific Deviation Documents

Any deviations have to be avoided where possible; however specific circumstances may lead to the violation of some rules. This is outlined in the following sections. Note that this is thought as a hint for setting up the chapter *Project Deviation* in the product specific compliance documents, it does not replace those.

6.8.1 Usage of Macros

Since efficiency is a primary implementation target it is necessary to use macros. This leads typically to violations of the rules 19.7 (function should be used instead of macro) and 19.4 (reserved C keywords or braces / parentheses in macro). If macros are used in different code variants it may happen that after resolving the macro only a null statement remains. This happens typically if assertions are implemented by macros and leads to the violation of the rule 14.3 (empty statement must appear on a line on its own).

6.8.2 Direct Hardware Interface

To access directly to hardware which is required to develop operating systems, drivers and flash boot loaders the usage of special compiler features is usually required. This may lead to a violation of rule 1.1 (ISO C standard compliance).

6.8.3 Software Component vs. System

Since the actual usage of the software component by the customer is not known some of the provided interfaces may not be used. Also during MISRA analysis there may be some provided APIs which are not in use. This may lead to a violation of the rules 8.10 (global symbol only referenced in one translation unit) and 14.1 (unreachable code - API function not used in project).

6.8.4 Complex Systems

Some MISRA-C:2004 rules are outdated compared to the complexity of nowadays embedded C-projects and automotive standards like AUTOSAR. In addition, the code provided by PES is just a part of such a system (e.g. BasicSoftware, BSW) so the final complexity in the customers project is again higher than we can measure for our software part.

Conclusion is, that compilers with limitations as addressed by those rules are not usable in such projects. This affects partly rule 1.1 (ISO C standard compliance).

By large projects typically affected QA-C rules are:

- 0810: “#include causes nesting to exceed 8 levels - program is non-conforming”
- 0828: “Maximum '#if...' nesting exceeds 8 levels - program is non-conforming”
- 0857: “Number of macro definitions exceeds 1024 - program is non-conforming”

6.8.5 Usage of Unions

For an efficient implementation of protocols the usage of unions is required (violation of rule 18.4). This is covered by the two acceptable deviations described in [MISRA-C], rule 18.4 (packing and unpacking of data, variant records).

6.8.6 Control Flow

Rule 14.7 (single point of exit at end of function) may lead to an inefficient code because the structure of the function can become more complex. Therefore it may be necessary to violate this rule for efficiency reasons.

6.8.7 Existing Code from old Projects

In old projects the MISRA rules were not taken into account. This can lead to large number of different violations in such a source code. These codes have been verified, some are in production since many years. Here MISRA improvements should be used with care since changes in the code always go along with the risk to introduce issues. In specific it should be avoided to change the structure of the software. If a major rework of such a component is necessary for other reasons (e.g. to incorporate new features) it should also be considered to improve the MISRA compliance.

6.8.8 Use of GOTO

GOTO shall not be used due to this often lead to bad programming style and unreadable code. Nevertheless, GOTO might be necessary in rare cases to IMPROVE the readability, maintainability and/or runtime efficiency of the code. One such situation is for components developed according the PES HighLevel/LowLevel code concept where maintaining the different possible scenarios for the LowLevel code variants causes too much maintenance effort to write that with if-else or switch-case blocks due to the closing brackets will need the same, complex #ifdef structure again.

In addition, for very runtime critical code parts the use of GOTO might improve speed to prevent check of terminating conditions in multiple nested loop statements.

Main rule: Prevent use of GOTO wherever possible (i.e. try different solutions first)

Condition 1: Use GOTO as unconditional jump at the end of a code block to a later (terminating) block.

Condition 2: Apply meaningful justification for this MISRA deviation (due to GOTO implies "bad coding style" explain the usage reason with care, e.g. reason: "Unconditional jump to a terminating code used to prevent break of readability by too complex if-else and ifdef structures.", risk: "Customer may complain about use of GOTO.", prevention: "Use GOTO only in the described way to improve readability and maintainability.".

Limitation for GOTO usage: It is not allowed to jump into control structures where values of variables are then differently initialized depending on the control flow or to implement a "loop" (i.e. a jump back to the top of the function).

(See http://en.wikipedia.org/wiki/Goto for links and details).

6.9 MISRA Compliance Matrix

This chapter shall be part of each product's compliance documentation.

The compliance matrix for the product is:

- All rules not explicitly listed in *Table 6-12* are checked by the tool QA-C (125 rules)
- Rules listed in *Table 6-12* and marked "Code inspection" are manually checked (8 rules)
- Rules listed in *Table 6-12* and marked "Not applicable" are not checked (8 rules); the justification is added within the table.

Rule	Description	Enforcement strategy
1.3 (req)	Multiple compilers and/or languages shall only be used if there is a common defined interface standard for object code to which the languages/compilers/assemblers conform.	Not applicable Justification: Multiple compilers or languages are not used in embedded software components developed at PES. The only exception is the usage of assembler code. Here however the assembler which belongs to the compiler is used so there is no issue with the object code interface.
1.4 (req)	The compiler/linker shall be checked to ensure that 31 character significance and case sensitivity are supported for external identifiers.	Not applicable Justification: Since we use the compiler which is prescribed by our customer, we are not free in the compiler selection. There are still compilers which support only 6 characters, so the software has to support this. Since especially during implementation of higher layers the used compiler is not known it is not possible to check this in the development phase. This is verified during delivery test with the prescribed compiler.

Rule	Description	Enforcement strategy
1.5 (adv)	Floating-point implementations should comply with a defined floating-point standard.	Not applicable Justification: Floating point operations are not used in embedded software components developed at PES.
2.4 (adv)	Sections of code should not be 'commented out'.	Code inspection
3.2 (req)	The character set and the corresponding encoding shall be documented.	Not applicable Justification: Character strings are not used in embedded software components developed at PES.
3.3 (adv)	The implementation of integer division in the chosen compiler should be determined, documented and taken into account. Comment: There is a potential issue in case of the division of negative signed integers because the compiler behavior is not standardized.	Not applicable Justification: Division of negative signed integers is not used in embedded software components developed at PES.
3.5 (req)	If it is being relied upon, the implementation-defined behavior and packing of bit-fields shall be documented.	Not applicable Justification: In the embedded environment the handling of bit-fields is documented for each compiler and hardware in the compilers' user manual. Beyond this, tests minimize the risk indicated by this rule.
3.6 (req)	All libraries used in production code shall be written to comply with the provisions of this document, and shall have been subject to appropriate validation.	Not applicable Justification: The C standard library or other libraries are not used in embedded software components developed at PES.
5.7 (adv)	No identifier name should be reused. Comment: the term "system" in [MISRA-C] relates to the unit on which the MISRA analysis is applied, i.e. a software component.	Code inspection
10.5. (req)	If the bitwise operators ~ and << are applied to an operand of underlying type unsigned char or unsigned short, the result shall be immediately cast to the underlying type of the operand	Code inspection
17.1 (req)	Pointer arithmetic shall only be applied to pointers that address an array or array element.	Code inspection
17.2. (req)	Pointer subtraction shall only be applied to pointers that address elements of the same array	Code inspection
17.3 (req)	>, >=, <, <= shall not be applied to pointer types except where they point to the same array.	Code inspection
18.2 (req)	An object shall not be assigned to an overlapping object. Comment: This is related to use the same memory by different objects (variables) at the same time without using unions.	Code inspection

Rule	Description	Enforcement strategy
18.3 (req)	An area of memory shall not be reused for unrelated purposes. Comment: This is related to use the same memory for different purposes at different points of time (example: dynamic memory management) without using unions.	Code inspection
20.3 (req)	The validity of values passed to library functions shall be checked.	Not applicable Justification: The C standard library is not used in embedded software components developed at PES. Our own libraries provide sufficient internal checking in contrast to the C standard library.

Table 6-11 Rules not checked by MISRA analysis tool PRQA- QA-C

6.10 HIS Metric Compliance Matrix

This chapter shall be part of each product's compliance documentation

The code for the product shall comply with HIS source code metrics [HIS-METRIC].

Analysis of the HIS source code metrics in the product context has shown that only few of the metrics are helpful to measure and vote the products quality.

The product has a high configurability, so metric deviations may occur in all or in just a few, specific setups. The components of the product are thus analyzed for a set of defined use-cases and metric deviations are analyzed. If the analysis result is "accept metric deviation", a justification is added to the code.

HIS Metric (QA-C metric)	Rationale and Enforcement Strategy
Comment Density (COMF)	HIS: COMF > 0.2 This metric is measured but deviations are not justified. Justification: - Product consists out of static and generated code files, some very small, some large, some with just tables and defines, some with many complex code structures. Comments shall be applied where necessary to improve the readability, not to satisfy a metric. - Adding MISRA justifications comments in the code results (in relevant files) in more than 20% → metric target is typically fulfilled but not the intended improvement in maintainability. - Vector focuses on the outcome of code inspection to improve also maintainability concerning good and helpful comments.
Estimated static path count (STPTH)	HIS: 1..80 This metric is measured and justified per deviation or class of deviations.
Cyclomatic Complexity (STCYC)	HIS: 1..10 This metric is measured and justified per deviation or class of deviations.
Number of GOTOs (STGTO)	HIS: 0 This metric is measured but deviations are not justified. Justification: - Use of GOTO is allowed in Vector code for limited use-cases. - Use of GOTO has to be already justified as MISRA deviation per use. - Mandatory metric STBAK is applied to check for illegal use of GOTO

Number of functions calling this function (STM29)	HIS: 0..5 This metric is measured but deviations are not justified. Justification: - The metric is aimed to detect "up to now unknown" functions in a "black-box system" which were critical due to be called by many other functions and shall thus be under specific focus. - By contrast, the product is designed to provide functionality to an (ECU) project based on defined standards (ISO, AUTOSAR, ...). Thus the usage of APIs is known, predefined and not in our control. Justifying the usage will indicate only the usage in e.g. our test-projects but not that in the concrete final project.
Number of distinct function calls (STCAL)	HIS: 1..7 This metric is measured and justified per deviation or class of deviations.
Number of function parameters (STPAR)	HIS: 0..5 This metric is measured and justified per deviation or class of deviations.
Number of statements in function (STST3)	HIS: 1..50 This metric is measured but deviations are not justified. Justification: - Project analysis has shown that if STST3 is out of limit and the function is complex, also STCYC and STPATH are out of limits. So we focus on STCYC and STPATH for analysis.
Maximum nesting of control structures (STMIF)	HIS: 0..4 This metric is measured and justified per deviation or class of deviations.
Number of exit points (STM19)	HIS: 0..1 This metric is measured but deviations are not justified. Justification: - Safe programming requires runtime checks. Due to runtime efficiency, the checks have to be disabled for production code. Thus the DET checks are typically macros and are added to the start of a function. To keep code structure simple, additional returns are allowed here, too. - Use of return statement has to be already justified as MISRA deviation per use. - Vector focuses on the outcome of code inspection to check for correct usage of return statements.
Language set (VOCF)	HIS: 0..4 This metric is measured but deviations are not justified. (Metric calculation done according PRQA specification) Justification: - Value for very complex components has been found in HIS range (= ok), but nearly empty components have been reported as dramatically out of range (e.g. >2500). - Summary is, that analysis of metric values in the product context has not shown any significant hint on improving code maintainability or other benefits.
Number of recursions across project (STNRA)	HIS: 0 This metric is measured but deviations are not justified. Justification: - Vector coding guidelines forbid the use of recursions to implement functional behavior. - Vector develops standard software according e.g. AUTOSAR, so API usage definition is mainly out of our scope

	- Vector code is just a part of a "final" ECU code, so recursions caused by customer-project specific API or configuration data use can not be detected in our selected product test use-cases. Measurement is done to be aware of potential recursions. - (The tool QAC has no reliable capability to detect the absence of recursions due to insufficient pointer analysis)
Number of MISRA subset violations (NOMV)	HIS: 0 This metric is measured but deviations are not justified. Justification: - MISRA deviations are already justified per occurrence during the development process of each component. - The components are assembled to a large number of different projects with different configurations resulting in different number of MISRA deviations. Due to the product-line approach and the component-local handling, creating justification above the component level is not reasonable.
Number of MISRA subset violations per rule (NOMVRP)	HIS: 0 This metric is measured but deviations are not justified. Justification: - Same justification as for NOMV
Stability Index „S_i "	HIS: <=1 This metric is measured but deviations are not justified. Justification: - Basic idea of this metric is to show that the code has less changes from release to release. This is typically true for an concrete ECU project in its given timeline. The time-boxed product-line development has features of different products developed at the same time and each release is typically "production". In addition, the code consists on a static part and a generated part. The generated part depends on the used configuration database. The reference database is updated with each time-box, too. Due to those effects creating justifications for the product is not reasonable.

Table 6-12 HIS code metric compliance matrix

6.11 Tool PRQA QA-C

The tool PRQA QA-C has been selected to perform the MISRA compliance analysis. The tool is well-known and widely used in the automotive software development business.

6.11.1 Installation of PRQA QA-C

Step 1: Installation of QA-C Analyzer

See [WIKI-QAC] for used tool version and latest details.

Root folder for tool setup is \\vi.vector.int\backup\PES\DevelopmentTools\others\PRQA.

Passwords for tool installation are in readme.txt in parallel to the setup file.

Step 2: Installation of MCM (MISRA Compliance Module)

See [WIKI-QAC] for used tool version and latest details.

Root folder for tool setup is \\vi.vector.int\backup\PES\DevelopmentTools\others\PRQA.

Passwords for tool installation are in readme.txt in parallel to the setup file.

Step 3: Update qac.usr.m2cm[8]

Change qac.usr.m2cm in the QA-C tool folder .\m2cm\messages to the file stored here: https://vglobpessvn1.vg.vector.int/svn/zCommon/zCantate/zCantate_CDK/trunk/Tools/QAC/m2cm/messages/qac.usr.m2cm to apply PES compliant MISRA warning levels.

Step 4: PES is using a license server. On first start of QA-C, the license server information has to be entered.

See \\vi.vector.int\backup\PES\DevelopmentTools\others\PRQA\install\QAC\QAC 7.0\Licenseserver.txt for license server information.

6.11.2 Personality Files

QA-C uses different files to configure the analysis engine.

The latest version of released files can be found here https://vglobpessvn1.vg.vector.int/svn/zCommon/zCantate/zCantate_CDK/trunk/Scripts/Prj.

A default analysis shall apply

- https://vglobpessvn1.vg.vector.int/svn/zCommon/zCantate/zCantate_CDK/trunk/Scripts/Prj/CheckQAC_CANoeEmu.p_c
- https://vglobpessvn1.vg.vector.int/svn/zCommon/zCantate/zCantate_CDK/trunk/Scripts/Prj/CheckQAC_General.p_a
- https://vglobpessvn1.vg.vector.int/svn/zCommon/zCantate/zCantate_CDK/trunk/Scripts/Prj/CheckQAC_m2cm.p_s

and for the m2cm mapping https://vglobpessvn1.vg.vector.int/svn/zCommon/zCantate/zCantate_CDK/trunk/Tools/QAC/m2cm/messages/qac.usr.m2cm

6.11.3 Rules checked by QA-C

For an overview which MISRA rules are checked by QA-C and which internal QA-C rules are available refer to [QAC-MsgSummary], [QAC-RuleEnforcement] and [QAC-RuleRef].

[8] In QA-C MISRA warnings belong to message level 4 by default. The message browser GUI displays only the highest warning level of one and the same message. If a MISRA warning also belongs to a higher message level, the MISRA warning is shadowed by the higher level and may be invisible in the GUI. To solve this problem higher message levels can be lowered by adapting the configuration user message file. This change has been done for PES in the referenced m2cm file.

6.11.4 QA-C GUI Usage

The QA-C GUI shows the number of “active”, i.e. unknown (i.e. not marked by PRQA S ...) and “total” deviations to a given rule.

If all deviations are marked as known (PRQA S ...), the number of “active” elements is 0.

The CDK report contains all deviations and checks in addition for availability of a justification.

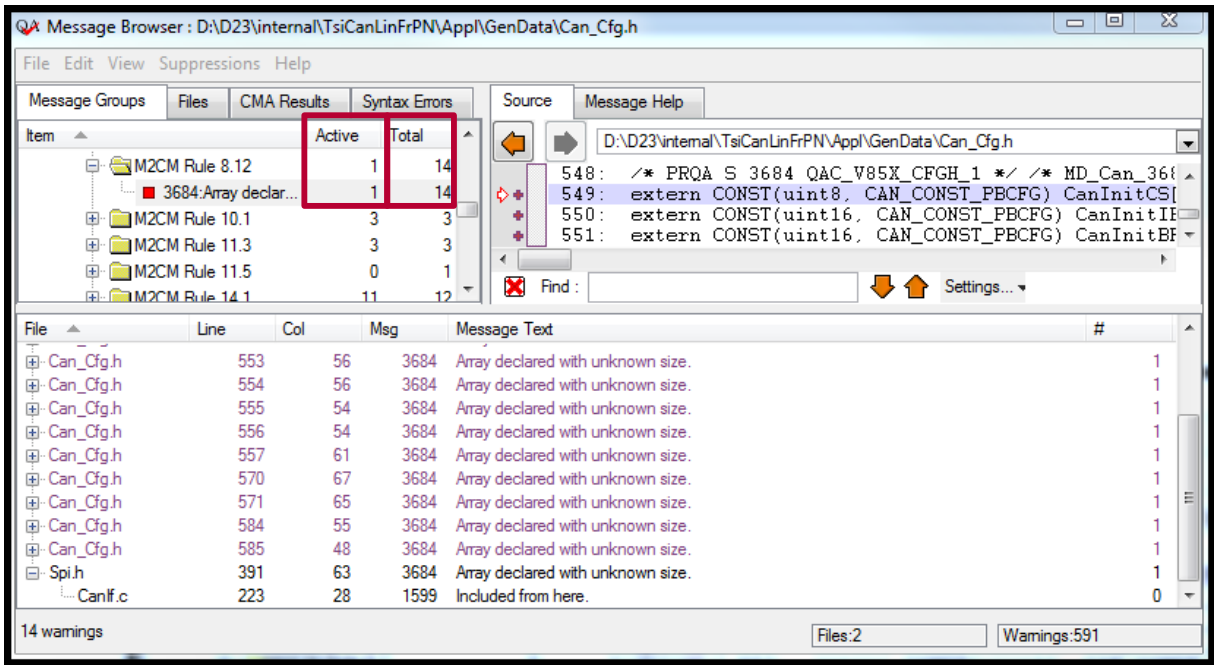


Figure 6-1 QA-C GUI displays "active" and "total" deviations

6.12 Frameworks supporting QA-C Usage

6.12.1 CDK (Component Development Kit)

Installation and usage of CDK is explained in https://vglobpessvn1.vg.vector.int/svn/Presentations/SoftwareDevelopment/InternalTraining_ComponentDevelopmentKit_CDK/trunk/zCDK_TrainingSlides.ppt and in http://wiki.vi.vector.int/wiki-pes/CDK.

6.12.2 MAKESUPPORT

Installation and usage of MAKESUPPORT is explained in https://vglobpessvn1.vg.vector.int/svn/zCommon/zBrs/zBrs_EmbeddedRunTimeSystem/base/Doc_UserMan/trunk/PresentationBRS.ppt and in http://wiki.vi.vector.int/wiki-pes/MakeSupport.

6.13 Example for Code Metric Report

File metrics

	info-only metrics		measured-only metrics													
file	CDN	*COMF	BME	DIF	ECT	FCO	FNC	M22	M28	M33	SCT	SHN	TLN	TPP	VAR	*VOCF
BswM.c	0.764	1.168	4.982	86.84	12	44	16	131	153	68	5	9530	305	1311	237	60.02
BswM_LCfg.c	0.145	0.301	4.329	143.62	16	38	31	306	92	44	69	13507	877	1166	189	55.38
Can.c	0.626	3.647	35.912	26.31	36	52	31	538	1962	1672	5	52688	1426	6799	825	16.23
Can_Irq.c	0.400	41	4.746	66.78	0	7	4	4	176	139	0	1458	16	1259	57	46.40

Figure 6-2 Example for File Metric Report generated by QA-C and visualized by CDK

Function metrics

	supervised metrics						info-only metrics				measured-only metrics									
function	PTH	CYC	CAL	PAR	MIF	BAK	GTO	M19	RET	ST3	AV1	KNT	LCT	LIN	M07	M29	SUB	UNR	UNV	XLN
ApplCanTimerLoop	4	4	2	1	3	0	0	1	1	11	9	0	1	86	1		2	0	0	12
ApplSetAcitveSchdTbl_SC_LIN1	1	1	1	1	0	0	0	0	1	2	9	0	0	2	1		1	0	0	2
ApplSetAcitveSchdTbl_SC_LIN2	1	1	1	1	0	0	0	0	1	2	9	0	0	2	1		1	0	0	2
ApplSetEiraEraIpduGroupMode	1	1	1	1	0	0	0	0	1	2	9	0	0	2	1		1	0	0	2
ApplXcpGetPointer	2	2	0	2	1	0	0	1	1	4	12	0	0	29	1		0	0	0	4
BswM_CanSM_CurrentState	16	5	3	2	1	0	0	3	4	14	4	3	1	26	1		5	0	0	8

Figure 6-3 Example for Function Metric Report generated by QA-C and visualized by CDK

The report contains file and function based code metrics. A code metric that exceed the specific HIS threshold is highlighted in the report.