Elektrobit

EB tresos Bootloader Generic Quality Level documentation

v2.0.0

Elektrobit Automotive GmbH
Am Wolfsmantel 46
91058 Erlangen, Germany
Phone: +49 9131 7701 0
Fax: +49 9131 7701 6333
Email: info.automotive@elektrobit.com

Technical support

https://www.elektrobit.com/support

Legal disclaimer

Table of Contents

1. Modification history

Version	Date	Description
1.0.10	2011-05-20	Initial release.
2.0.0	2015-04-21	Updates for OsekCore releases

Table 1.1. Modification history

2. Overview

The purpose of this document is to explain in detail the terms, definitions and concepts used in the evaluation of releases of the product EB Bootloader Quality Level and related projects.

This document is relevant for the development and verification processes of software units produced in the project EB Bootloader Quality Level and related projects.

During software development, a number of software quality metrics are collected. These metrics are then used to assign a quality level to the software product.

The software metrics that are collected during software development are described further in Chapter 3, “Measurement“. The quality levels and the measurement criteria required to achieve these levels are described in Chapter 4, “Quality levels“

3. Measurement

3.1. What is measured

The choice of software quality metrics collected is derived from standards that are applicable in the domain of automotive software. See for example [his_metrics], [his_subset], or [sqo].

The metrics used during ongoing product development are defined in Section 3.1.1, “Development checks”. The metrics that are used prior to each product release are described in Section 3.1.2, “Release checks”.

3.1.1. Development checks

During product development, the following software quality metrics are collected:

- cyclomatic complexity
- Number (#) of compiler warnings per 100 lines of code
- Percentage (%) of implemented tests that are executed successfully
- decision coverage

These metrics are used in the definition of the product quality level and have specific targets that must be achieved. For each metric and for each quality level, a quality criterion is defined. In addition, further metrics are collected for each quality level that do not have such quality criteria defined. The quality criteria and the additional metrics collected are described for each quality level in Chapter 4, “Quality levels“.

The software metrics are collected at different stages of the product development:

- During software unit testing on the lead platform for a software unit release.
- When the EB Bootloader Quality Level is ported to other target hardware architectures. Only those metrics that depend on the execution of test cases on the target hardware are collected, e.g. the number of compiler warnings.

Software metrics are not collected for software modules that are supplied by other vendors and which are only integrated, e.g. MicroController Abstraction Layer (MCAL) modules.

3.1.2. Release checks

For each release which is independent of its associated quality level, the following items are reviewed and accepted by a project-independent third party:

- The compiler chosen, the compiler version and the compiler options used for the verification of the release.
- The compiler warnings produced during a release test build.
- The [MISRAC2] compliance is reviewed and it is ensured that there are no undocumented and unjustified [MISRAC2] violations.
- The values of software quality metrics.
- The decision coverage for each software module.
- Further internal release checklists.

4. Quality levels

4.1. Use of quality levels

Each release has an associated quality level that is the basis for determining the *recommended use* of that specific release. The quality level for a particular release is defined in the document quality statement, which is contained in each delivery.

The quality statement aggregates all quality data obtained for a release into a single statement that maps quality data to the *recommended use* of the release.

There are two types of quality statement available:

1. The *generic* type quality statement: this provides the qualification for a Windows platform. The Windows version and toolchain (compiler/linker and settings) used are documented.
2. The QP1 type quality statement: this provides the qualification for a specific hardware platform (uC) and toolchain (compiler/linker). This type of quality statement is always provided in addition to a generic quality statement. The generic type quality statement must exist in advance for the modules that are to be qualified for QP1.

The metrics which are used are different and are dependent on the type of quality statement. For the generic type, the qualification is dependent on the full set of metrics as defined in tables in Section 4.2, “Quality levels and criteria”. For the QP1 type, the qualification is dependent on a reduced set of metrics. In the tables in Section 4.2, “Quality levels and criteria”, a note indicates when a metric is used for QP1 qualification.

NOTE

Safe and correct use of the release
If you intend to use the release for purposes other than the *recommended use*, ensure that additional verification measures are conducted. These measures shall be sufficient to ensure the safe and correct use of the release for the purpose that is intended.

4.2. Quality levels and criteria

There are four possible quality levels defined:

- Development drop, defined in Section 4.2.1, “Development drop”
- Ready for prototyping (RFP), defined in Section 4.2.2, “Ready for prototyping (RFP)”
- Ready for development (RFD), defined in Section 4.2.3, “Ready for development (RFD)”
- Ready for manufacturing (RFM), defined in Section 4.2.4, “Ready for manufacturing (RFM)”

4.2.1. Development drop

The quality level *Development drop* is assigned to a release that does not fulfill any defined set of quality criteria. A release with the quality level *Development drop* is neither ready for development nor fully tested.

Note: In Quality Statements, which have been created before 04-2015, this quality level is referenced as *Integration Snapshot*.

4.2.2. Ready for prototyping (RFP)

The quality level *ready for prototyping* is assigned to a release that is neither ready for development nor fully tested.

Note: In Quality Statements, which have been created before 04-2015, this quality level is referenced as *Prototype level*.

4.2.2.1. Ready for prototyping level metrics

The set of software quality metrics and associated criteria checked for a *generic* quality statement are as follows: the tables show the criteria defined for *ready for prototyping level*.

software quality metrics	**quality criterion**
called functions	<= 20
goto statements	== 0
call levels	<= 16
function parameters	<= 8
static path count	<= 4000
return points	<= 1
function statements	<= 100
cyclomatic complexity	<= 30
language scope	<= 20
recursions	== 0
direct recursions	== 0
number of compiler warnings [a]	<= 10
number of undocumented MISRA violations	<= 10

Table 4.1. Ready for prototyping level: code metrics with quality criterion

software quality metrics	**quality criterion**
decision coverage	>= 0.8
percentage of successful tests [a]	>= 0.7

[a]This criterion is also measured for QP1 quality statement.

Table 4.2. Ready for prototyping level: test metrics with quality criterion

software quality metrics	**quality criterion**
number of test cases which are not required	== 0
number of tracing fulfillments which are not required	== 0
number of implemented tests without test specification	== 0

Table 4.3. Ready for prototyping level: process metrics with quality criterion

4.2.3. Ready for development (RFD)

The quality level *ready for development* is assigned to a release that is ready for development, but not fully tested.

Note: In Quality Statements, which have been created before 04-2015, this quality level is referenced as *Development level*.

4.2.3.1. Ready for development level metrics

The set of software quality metrics and associated criteria checked for a *generic* quality statement are as follows: the tables show the criteria defined for *ready for development level*.

software quality metrics	**quality criterion**
called functions	<= 15
goto statements	== 0
call levels	<= 12
function parameters	<= 8
static path count	<= 2000
return points	<= 1
function statements	<= 75
cyclomatic complexity	<= 30
language scope	<= 12

software quality metrics	**quality criterion**
recursions	== 0
direct recursions	== 0
number of compiler warnings [a]	<= 1
number of undocumented MISRA violations	<= 1

Table 4.4. Ready for development level: code metrics with quality criterion

software quality metrics	**quality criterion**
decision coverage	>= 0.9
percentage of successful tests [a]	== 1.0

[a]This criterion is also measured for QP1 quality statement.

Table 4.5. Ready for development level: test metrics with quality criterion

software quality metrics	**quality criterion**
percentage of implemented requirements	>= 0.85
percentage of tested requirements	>= 0.75
number of test cases which are not required	== 0
number of tracing fulfillments which are not required	== 0
number of implemented tests without test specification	== 0

Table 4.6. Ready for development level: process metrics with quality criterion

4.2.4. Ready for manufacturing (RFM)

The quality level *ready for manufacturing* is assigned to a release that is ready for development and fully tested.

Note: In Quality Statements, which have been created before 04-2015, this quality level is referenced as *Production level*.

4.2.4.1. Ready for manufacturing level metrics

The set of software quality metrics and associated criteria checked for a *generic* quality statement are as follows: the tables show the criteria defined for *ready for manufacturing level*.

software quality metrics	**quality criterion**
called functions	<= 15
goto statements	== 0

software quality metrics	**quality criterion**
call levels	<= 12
function parameters	<= 8
static path count	<= 2000
return points	<= 1
function statements	<= 75
cyclomatic complexity	<= 30
language scope	<= 12
recursions	== 0
direct recursions	== 0
number of compiler warnings [a]	== 0
number of undocumented MISRA violations	== 0

Table 4.7. Ready for manufacturing level: code metrics with quality criterion

software quality metrics	**quality criterion**
decision coverage	== 1.0
percentage of successful tests [a]	== 1.0

[a]This criterion is also measured for QP1 quality statement.

Table 4.8. Ready for manufacturing level: test metrics with quality criterion

software quality metrics	**quality criterion**
percentage of implemented requirements	== 1.0
percentage of tested requirements	== 1.0
number of not implemented requirements	== 0
number of not tested requirements	== 0
number of test cases which are not required	== 0
number of tracing fulfillments which are not required	== 0
number of implemented tests without test specification	== 0
number of test cases which are not implemented	== 0
number of requirement tracing rules violations	== 0

Table 4.9. Ready for manufacturing level: process metrics with quality criterion

For a release with *ready for manufacturing level* all structural code branches that are not covered in module tests are reviewed and accepted by an independent third party. Such branches are usually the result of defensive programming.

Bibliography

[automotive_spice] *Automotive SPICE*, Process Assessment Model, automotiveSIG_PAM_v25.pdf , Version 2.5, 2010-05-10

[his_metrics] *HIS Source Code Metriken*, his-sc-metriken.1.3.1.pdf , Version 1.3.1, HIS AK Softwaretest, 2008-04-01

[his_subset] *Gemeinsames Subset der MISRA C Guidelines*, his_subset_misra_c_2.0.pdf , Version 2.0, HIS AK Softwaretest, 2006-02-14

[iso_c] *ISO/IEC 9899:1990*, Programming languages - C, International Organization for Standardization , 1990, Copyright © 1990 ISO,

[MISRAC2] *MISRA-C:2004 Guidelines for the use of the C language in critical systems*, http://www.misra.org.-uk/ , 2004-10, Copyright © 2004 MIRA Limited,

[MISRAC2TC1] *MISRA-C:2004 Technical Corrigendum 1*, Technical clarification of MISRA-C:2004, 2007-07-17, Copyright © 2007 MIRA Limited,

[misra_deviation] *MISRA deviation documentation*, EB tresos AutoCore Generic, v1.0.4, 2011-03-22, Copyright © 2011 Elektrobit Automotive GmbH,

[sqo] *Software Quality Objectives for Source Code*, 72337_Software_Quality_Objectives_V3.0.pdf , Version 3.0, 2012-05-02

[autosar_solutions] *EB Solutions*, PD_EB_Solutions_Overview.pdf , Version 1.0.1, 2014-11-19

Glossary

calling functions	Number of distinct callers of a function. Taken from [his_metrics].
called functions	Number of distinct functions called by a function. Taken from [his_metrics].
call levels	Depth of function nesting. Maximum depth of control structures within a function body. The value of 1 means either no control structure exists within a function body or all existing control structures are not nested within another control structure. Taken from [his_metrics].
comment density	Relationship of the number of comments (outside and within functions) to the number of statements. Taken from [his_metrics].
cyclomatic complexity	The *cyclomatic complexity* of a function is the count of the number of linearly independent paths through the source code. It is a measure about the structural complexity of a function. See Cyclomatic_complexity for further information. Taken from [his_metrics].
decision coverage	The *decision coverage* is a measure for structural code coverage. Based on the control flow graph of a program, it measures if each edge has been executed by at least one test case. See Code_coverage for further information.
defensive programming	See Defensive_programming for further information.
direct recursions	Number of direct recursions. Taken from [his_metrics].
function parameters	Number of parameters per function. A measure of the complexity of the function interface. Taken from [his_metrics].
function statements	Number of statements of a function, which is a measure of function complexity.

	Taken from [his_metrics].
goto statements	Number of `goto` statements per function. Taken from [his_metrics].
language scope	The language scope is an indicator of the cost of maintaining or changing functions. language scope := `(N1+N2) / (n1+n2)` where: ▶ `n1` = number of different operators ▶ `N1` = sum of all operators ▶ `n2` = number of different operands ▶ `N2` = sum of all operands Taken from [his_metrics].
lead platform	The *lead platform* is a target hardware for which the EB Bootloader Quality Level is actively developed. The software product is then only ported to other target hardware architectures.
lines of code	The lines of code is a software metric used to measure the size of a program. See Source_lines_of_code for further information.
lines of comment	The number of lines containing comments.
MicroController Abstraction Layer (MCAL)	The *MCAL* contains basic software modules that abstract the microcontroller specifics (*drivers*).
number of source files	Number of source files per software unit.
number of header file inclusions	Number of directly and indirectly included header files per software unit.
number of statement lines	The *number of statement lines* is the number of lines of code that include an executable statement.
number of requirements	The number of requirements per software unit.
number of requirements that need implementation	The number of requirements that need implementation per software unit.
number of requirements that are implemented	The number of requirements that are implemented per software unit.

number of requirements that need testing	The number of requirements that need testing per software unit.
number of requirements that are tested	The number of requirements that are tested per software unit.
number of not implemented requirements	The number of requirements that need implementation, but have not been implemented or are only partially implemented.
number of not tested requirements	The number of requirements that need testing, but have not been tested or are only partially tested.
number of test cases which are not required	The number of test cases that are specified and do not trace correctly to requirements.
number of test cases which are not implemented	The number of test cases that are specified and have not been implemented.
number of tracing fulfillments which are not required	The number of coverages in requirement tracings that are not required, e.g. there exists a test specification, but the requirement does not need a test case.
number of implemented tests without test specification	The number of implemented test cases that are not derived from a test specification.
number of requirement tracing rules violations	The number of requirement tracing rules that are violated.
number of compiler warnings	The number of compiler warnings per 100 lines of code.
number of undocumented MISRA violations	The number of unjustified violations against [MISRAC2] per 100 lines of code. This takes [MISRAC2TC1] into account.
percentage of implemented requirements	The percentage of requirements that need implementation and have been implemented.
percentage of tested requirements	The percentage of requirements that need testing and have been tested.
percentage of successful tests	Percentage of implemented tests that have been executed with a successful result.
QP1	module qualification package For a detailed description see [autosar_solutions].
quality criteria	See quality criterion.

quality criterion	A *quality criterion* is a definition of the lowest acceptable value for software quality metrics. This value is dependent on the quality level.
quality level	A *quality level* defines a set of quality criteria that shall be satisfied by a module. The module is then said to have this quality level.
quality statement	The *quality statement* is a document created by quality assurance that testifies that a release satisfies a quality level.
recursions	Call graph recursions. Number of call cycles over one or more functions. If one function is at the same time directly recursive (it calls itself) and indirectly recursive, the call cycle is counted only once. See also direct recursions. Taken from [his_metrics].
return points	Number of `return` points of a function. Taken from [his_metrics].
software quality metrics	A *software quality metric* defines a method to determine a value for a particular attribute of the software.
static path count	Estimated static path count of a function. Taken from [his_metrics].