SW Construction

**Change History**

| **Version** | **Date** | **Change Description / Reason** | **Author** |
| --- | --- | --- | --- |
| 1.0 | 2017-12-08 | Initial version based on Hella Process (State Avenue)  Add sub-process description in chapter 3.3  Add overview process flow chart | Hae-Min, Woo |
|  |  |  |  |
|  |  |  |  |

**Table of Contents**

[1. Purpose Description 4](#_Toc492993979)

[1.1 Purpose 4](#_Toc492993980)

[1.2 Main description 4](#_Toc492993981)

[1.3 Result of Process 4](#_Toc492993982)

[2. Major Roles Acting in this Process 4](#_Toc492993983)

[3. Process 5](#_Toc492993984)

[3.1 Process Input – Output Definition 5](#_Toc492993985)

[3.2 Introducing Aspects 5](#_Toc492993986)

[3.3 Process Flow Chart 6](#_Toc492993987)

[3.3.1 Task 1: Initial SW Module Release 9](#_Toc492993988)

[3.3.2 Decision 1: Adapt Build Parameters 9](#_Toc492993989)

[3.3.3 Task 2: SW Module Integration (Build) 9](#_Toc492993990)

[3.3.4 Task 3: SW Module Configuration & Integration (QA-C) 10](#_Toc492993991)

[3.3.5 Task 4: Review Configuration & Integration 11](#_Toc492993992)

[3.3.6 Task 5: Check Tool Bugs 11](#_Toc492993993)

[3.3.7 Task 6: Create SW Units & Document Source Code 12](#_Toc492993994)

[3.3.8 Task 7: Build & Debug SW Module 13](#_Toc492993995)

[3.3.9 Task 8: Code Analysis (QA-C/CMA)/(MISRA/Style) 14](#_Toc492993996)

[3.3.10 Task 9: Configuration Update & SW Module Integration (static code Analysis) 15](#_Toc492993997)

[3.3.11 Decision 2: Preventable Findings 15](#_Toc492993998)

[3.3.12 Decision 3: SW Module Baseline 16](#_Toc492993999)

[3.3.13 Task 10: Generate Static Code Analysis Reports & Annotate Warning/Metrics Deviations 16](#_Toc492994000)

[3.3.14 Task 11: Code Review 17](#_Toc492994001)

[3.3.15 Task 12: Baseline SW Module 18](#_Toc492994002)

[4. Changes, References, Appendix, Terms 18](#_Toc492994003)

[4.1 References 18](#_Toc492994004)

[4.2 Template 18](#_Toc492994005)

[5. Supplement A: Assignment of the Failure Types 19](#_Toc492994006)

[5.1 Overview 19](#_Toc492994007)

[5.2 Correlation of static analysis and test methods 19](#_Toc492994008)

[5.3 Assignment of Metrics and Error Types to QA-C 19](#_Toc492994009)

[5.4 Assignment of Error Types to Tessy/CTE Tool 20](#_Toc492994010)

[5.5 Assignment of Use Cases and Error Types to Polyspace Tool 21](#_Toc492994011)

# Purpose Description

## Purpose

The purpose of this process step is to create binaries that realize the software module and unit design and that fulfill the specified quality.

## Main description

This MHE procedure describes the software construction process for the construction of executable software units that properly reflect the software design.

## Result of Process

As a result of the successful implementation of this process:

* Construction of the software modules
* Analyzing the software modules with the static code analysis tool, reviewing the results, eliminating the violations against QA-C messages and metrics, and commenting the remaining results
* Executing a code review for checking MISRA rules which can’t be checked automatically
* Executing a code review for checking the compliance of the source code with the C Style Guide which can be found in MHE-PE-23.

# Major Roles Acting in this Process

|  |  |
| --- | --- |
| **Role** | **Contribution and Responsibilities** |
| SW Project Manager | * Responsible for the execution of the activities of this process (software module configuration and integration, software construction, static code analysis and code reviews) * Monitors process execution * Creates and maintains software project schedule * Has to ensure that the remaining work results are kept consistent if changes are made to individual work results |
| SW Integrator | * Responsible for the static code analysis tool configuration. * Responsible for specification and implementation of the build environment * Responsible for the integration of the static code analysis tool into the build process * Execution of the static code analysis tool configuration |
| SW Developer | * Responsible for the execution of the following activities of this process: * Construction of the software modules * Execution of the static code analysis. * Review of static code analysis results, this includes documenting the remaining results * Review of correct code realization |

# Process

## Process Input – Output Definition

The following process input is required:

|  |  |
| --- | --- |
| **Process input** | **From supplier** |
| SW Module and Unit Design | SW Developer |
| SW Requirements Specification | SW Analyst |

|  |  |
| --- | --- |
| **Process output** | **To customer** |
| Executable software modules and documentation | SW Project Manager / SW Integrator |
| Report of the configuration and results of the static code analysis tool. | SW Quality Planner |

## Introducing Aspects

As defined in the process description, all work results are reviewed.

Bidirectional traceability is established between the detailed software design and the code, i.e. for each design element it must be clear in which part of the code it is realized and for each software module it must be clear on which design it is based.

The traceability between the software design and the code, and vice versa, is as a standard documented within the software design and construction Tool and kept-up-to-date during the project.

Also software units defined by the software design are analyzed for correctness and testability. (Analysis in terms of interoperability, interaction, criticality, technical complexity, risks and testability)

## Process Flow Chart



**Sub-Process 1: Initial SW Module Release**

The SW Architect decides if new software modules have to be created and integrated into the software build environment.

**Sub-Process 2: SW Module Configuration & Integration**

If a new SW module is added, the configuration and integration of this SW module into the build environment shall be done by the SW Integrator. Moreover the configuration for the static code analysis with

Static Code Analysis tool (QA-C) and the integration of Static Code Analysis tool into the build process shall be implemented. The definition of the sub-process can be found in chapter 3.3.1.

**Sub-Process 3: Software Construction**

During this sub-process, the construction of the SW shall be done (coding). The definition of the sub-process can be found in chapter 3.3.3. During the SW Construction phase, the SW Developer may decide, if the SW module configuration needs to be changed, e.g. as some source files have to be added or removed in comparison to the original SW design. In this case the SW Integrator has to repeat the SW Module Configuration & Integration task in order to integrate/remove these files and to rebuild the static code analysis configuration.

**Sub-Process 4: Static Code Analysis & Review**

Performing the static code analysis, the work results of the SW Construction phase must be checked against existing coding and programming guidelines (e.g. for the C programming language). The use of such guidelines pursues the following aims:

* Promotion of good programming techniques
* Avoidance of ambiguous language use in C
* Optimization of quality aims, e.g.
* Readability
* Comprehensibility
* Maintainability
* Testability

As the C programming language is partially imprecisely defined and therefore compiler manufactures have implemented different solutions, errors may easily occur. Therefore, a specified set of coding rules (e.g.

MISRA guidelines) have to be applied and observed during SW construction.

The static code analysis for checking those predefined coding rules is performed by means of the Static Code Analysis tool. It is important to start with the static code analysis from the beginning of the SW Construction phase. It is recommended to use first QA-C instead of running the compiler, because beyond checking coding rules, QA-C is capable of locating errors and warnings in a more efficient way than most compilers.

The SW Developer decides if the Static Code Analysis & Review step is completed. Otherwise a new step with SW Construction, Static Code Analysis & Review has to be started. A possible reason for the re-start of the SW construction process is e.g. errors/deviations detected during static code analysis and review.

The produced SW code must be verified. This is normally done using static code analysis methods and code review (see tasks 9 .. 13).

If the project is safety relevant, additional verification methods according to table 16 must be chosen, with the support of the Safety manager.

**Sub-Process 5: Baselining SW Module**

After completion of the SW Construction, the Static Code Analysis & Review phase the SW Developer creates a new SW module release.



### Task 1: Initial SW Module Release

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| Input | |  | | | | | |
| Output | |  | | | | | |
| D: | SW Architect | E: | SW Project Manager | S: |  | I: | SW Integrator  SW Development |
| D = Decision | | E = Execution | | S = Support | | I = Information | |

**Objective**

Objective of this task is to update the new SW modules in build environment.

**Description**

The SW Architect decides if new software modules have to be created and integrated into the software build environment. This decision also takes place, if external software modules (e.g. CAN, standard SW) are to be integrated in order to set up the environment required for the construction phase.

### Decision 1: Adapt Build Parameters

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| Input | | Configuration file (Compiler, Make, QAC, etc) | | | | | |
| Output | | Updated Configuration file (Compiler, Make, QAC, etc) | | | | | |
| D: | SW Project Manager | E: | SW Integrator | S: | SW Project Manager  SW Quality Planner  Test Manager | I: | SW Developer |
| D = Decision | | E = Execution | | S = Support | | I = Information | |

**Objective**

Objective of this task is to update the build environment for new modules or code analysis.

**Description**

Examples for these kinds of changes in the build environment are:

* Updates, patches or new releases of tools involved in the build and code analysis process (Compiler, Make, QA-C, etc.)
* Modifications in the test strategy (e.g. enforce tests on resource usage)
* Changes concerning quality aims for this software module or the whole project (e.g. strengthen reusability).

### Task 2: SW Module Integration (Build)

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| Input | | Make file Templates | | | | | |
| Output | | Make file and Configuration files | | | | | |
| D: | SW Project Manager | E: | SW Integration | S: | SW Architect | I: | SW Developer  CM Officer |
| D = Decision | | E = Execution | | S = Support | | I = Information | |

**Objective**

Objective of this task is to set up the build environment for SW construction.

**Description**

The SW Module Integration (Build) into the project build process has to be done by the SW Integrator.

Therefore he must use the Make file templates (based on default make tool “) provided by the department responsible for processes, methods and tools. These Make file templates ensure an automated build process.

The SW module integration includes activities like the definition of compiler/assembler settings and includes paths and adding these SW module and its linkable objects to the linker list of the Make files. As a result of this task the SW Integrator has created a set of Make files and different configuration files (e.g. SW module list, SW module configuration files), which allow to build the application including the new SW module.

### Task 3: SW Module Configuration & Integration (QA-C)

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| Input | | Code Metrics  SW QA Plan | | | | | |
| Output | | QAC Configuration (Analyzer / Message)  QAC Personalities (Analyzer / Message)  Make files and Configuration files | | | | | |
| D: | SW Project Manager | E: | SW Integrator | S: | SW Architect  SW Quality Planner | I: | SW Developer  CM Officer |
| D = Decision | | E = Execution | | S = Support | | I = Information | |

**Objective**

Objective of this task is to set up and to review QA-C configuration.

**Description**

Before starting with the SW construction, an initial configuration for the so-called message and analyzer personality subsets used by QA-C is done in cooperation with the department responsible for processes, methods and tools and according to the quality aims defined in the SW QA Plan and code metric limits defined in the MHE-PE-23, Chapter 9. For each C file an individual message and analyzer configuration is done.

During the Integration of the static code analysis the SW Integrator shall apply the QA-C configuration to the build process using the Make file templates provided by the department responsible for processes, methods and tools. Furthermore, the message and analyzer personality files are generated for each C file. The initial integration is done in cooperation with support of the department responsible for processes, methods and tools. It’s strongly recommended to use QA-C on the command line interface for configuration and static analysis instead of using the graphical interface (GUI) of QA-C.

After the first time the SW Module Configuration & Integration (Build & QA-C) has been finished, the resulting configuration must be reviewed. The SW Integrator must also decide if additional reviews are necessary for further SW releases.

### Task 4: Review Configuration & Integration

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| Input | |  | | | | | |
| Output | | Review Checklist Build and QA-C | | | | | |
| D: | SW Project Manager | E: | SW Quality Planner | S: | SW Integrator | I: | SW Developer |
| D = Decision | | E = Execution | | S = Support | | I = Information | |

**Objective**

Objective of this task is to review QA-C configuration by SW Quality Planner.

**Description**

After the initial configuration and integration of the SW Modules (Build & QA-C), the SW Quality Planer runs a review on the created documents and configurations.

The review checkpoints and the result of the review activities are documented in the Review Checklist Build & QA-C (8320).

### Task 5: Check Tool Bugs

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| Input | |  | | | | | |
| Output | | Complier Bugs Report | | | | | |
| D: | SW Project Manager | E: | SW Integrator | S: |  | I: | SW Developer |
| D = Decision | | E = Execution | | S = Support | | I = Information | |

**Objective**

Objective of this task is to check build environment.

**Description**

The SW Integrator regularly checks (e.g. monthly, or triggered by new compiler bug reports provided by the department responsible for processes, methods and tools), the existing compiler bug lists (e.g. bug report from the compiler manufacturer, bugs found by the project …), if there are known bugs for the used compiler. For each known bug the SW Integrator decides, if it affects the resulting executable.

In case code is generated bugs of the generator have to be regarded as well in the same manner.

In cooperation with the SW Developer, each bug which may affect the resulting executable is evaluated. How to handle the compiler bug is documented by creating a Review Master in the source code management SW or by filling in the “Compiler Bugs Report” (8501). It is checked if the build parameters need to be updated. In this case Task 2: SW Module Integration (Build) must be performed again.

In case of safety-relevant projects it is only allowed to use released compiler versions.

### Task 6: Create SW Units & Document Source Code

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| Input | | SW Module and Unit Design  C Style Guide  MISRA Coding Rules  Code Metrics  Code Analysis Findings | | | | | |
| Output | | Documented C Code File (ISO/IEC) | | | | | |
| D: |  | E: | SW Developer | S: | SW Quality Planner | I: | SW Project Manager |
| D = Decision | | E = Execution | | S = Support | | I = Information | |

**Objective**

Objective of this task is to create source code.

**Description**

In order to develop a single SW unit design both SW safety requirements as well as all non-safety related requirements are implemented.

Design principles for SW unit implementation at the source code level as listed in table below shall be applied to achieve the following properties:

1. Correct order of execution of subprograms and functions within the SW units, based on the SW architectural design;
2. Consistency of the interfaces between the SW units;
3. Correctness of data flow and control flow between and within the SW units;
4. Simplicity;
5. Readability and comprehensibility;
6. Robustness;
7. Suitability for SW modification; and
8. Testability.  
   - **O** The method has no recommendation for or against its usage for the identified ASIL.  
   - **++** The method is highly recommended for the identified ASIL.  
   - **+** The method is recommended for the identified ASIL.

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **Methods** | | **ASIL** | | | |
| **A** | **B** | **C** | **D** |
| 1a | One entry and one exit point in subprograms and functions a | ++ | ++ | ++ | ++ |
| 1b | No dynamic objects or variables, or else online test during their creation a, b | + | ++ | ++ | ++ |
| 1c | Initialization of variables | ++ | ++ | ++ | ++ |
| 1d | No multiple use of variable names a | + | ++ | ++ | ++ |
| 1e | Avoid global variables or else justify their usage a | + | + | ++ | ++ |
| 1f | Limited use of pointers a | O | + | + | ++ |
| 1g | No implicit type conversions a, b | + | ++ | ++ | ++ |
| 1h | No hidden data flow or control flow c | + | ++ | ++ | ++ |
| 1i | No unconditional jumps a, b, c | ++ | ++ | ++ | ++ |
| 1j | No recursions | + | + | ++ | ++ |
| 1. Methods 1a, 1b, 1e, 1f, 1g and 1i may not be applicable for graphical modelling notations used in model-based development. 2. Methods 1g and 1i are not applicable in assembler programming. 3. Methods 1h and 1i reduce the potential for modelling data flow and control flow through jumps or global variables. | | | | | |

Note: For the C language, MISRA C covers many of the methods listed in this table.

Based on the SW Module and Unit Design (8364), the MHE C Style Guide, the MISRA coding rules, and the Code Metric Guidelines, the individual SW units of this SW module are constructed and documented. At the same time, the static code analysis is started and performed in parallel to the SW construction. If there are code analysis findings, they need to be fixed during the SW construction process wherever possible. Code analysis findings can be deviations against MISRA coding rules, which can be checked automatically or manually, metrics, QA-C warnings and the C Style Guide.

### Task 7: Build & Debug SW Module

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| Input | | Documented C Code File (ISO/IEC)  Make file and Configuration files  Temporal Base System | | | | | |
| Output | | Documented C Code File (ISO/IEC) | | | | | |
| D: | SW Project Manager | E: | SW Developer | S: | SW Architect  SW Integrator | I: |  |
| D = Decision | | E = Execution | | S = Support | | I = Information | |

**Objective**

Objective of this task is to check the error during build. Also this task is to check the code analysis and code error.

**Description**

It is strongly recommended to run first the static code analysis with QA-C and especially the Cross Module Analysis (CMA) then remove or comment all QA-C findings, which may result in compiler/linker errors during this task.

The created SW modules must be compiled using the build settings and the build environment defined and provided by the SW Integrator (tools, Make files). As a result, the SW Developer has to check if compiler errors and warnings occur. In case of errors/warnings another SW Construction step has to be started in order to remove these findings.

To enable debugging on test hardware by means of a debugger or emulator, the created SW module has to be integrated into a temporal base system (provided by the SW Integrator or set up by the SW Developer himself for test purposes). All necessary SW modules shall be built and linked together. Integration errors (e.g. linker errors/warnings) are checked and removed by another SW Construction step.

During debugging the SW Developer determines if the SW module meets all functional and nonfunctional requirements.

### Task 8: Code Analysis (QA-C/CMA)/(MISRA/Style)

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| Input | | Documented C Code Files (ISO/IEC)  C Style Guide  MISRA Coding Rules | | | | | |
| Output | | Code Analysis Findings | | | | | |
| D: | SW Project Manager | E: | SW Developer | S: | SW Quality Planner | I: |  |
| D = Decision | | E = Execution | | S = Support | | I = Information | |

**Objective**

Objective of this task is to analysis code using tools.

**Description**

The static code analysis shall be run simultaneously to the SW Construction process. This task contains the automatic analysis of the executable SW modules using the Static Code Analysis tool QA-C. The C source code will be checked against coding rules (e.g. MISRA) defined in the used message subset. There are different message subsets for hand written, auto generated and third party code. Moreover, a Cross Module Analysis (CMA) with QA-C is executed to find deviations concerning the aggregation of different SW modules (e.g. namespace conflicts, recursions, etc.). At last, a predefined subset of file and function based source code metrics for each SW module is evaluated. The configuration for the automatic code analysis with the Static Code Analysis tool is defined by the SW Integrator during the SW Module Configuration & Integration phase (see Initial SW Module Release).

To avoid static code analysis warnings of any kind it is advised compliant code to be written from the beginning.

Criticality of static code analysis warning can only be assessed by projects not globally but warning above level 7 have to be removed.

For levels below warning level 7 where they cannot be removed, they shall be commented and documented with a given reason.

If the SW construction is completed, a manual code analysis concerning the MHE C Style Guide and

MISRA rules, which are not automatically enforceable by the Static Code Analysis tool, is done.

It shall be checked if the required MISRA rules have been violated which cannot be checked automatically. If there is a violation against this MISRA rules, the violation is eliminated by changes in the source code. If in fact a deviation is inevitable, this violation has to be commentated with a given reason within the document Manual Review of MISRA Rules (8395). For manually created source code it must be checked afterwards if the code meets the requirements defined in the MHE C Style Guide. All deviations against the C Style Guide must be fixed if possible. Remaining deviations have to be commented directly in the source code (i.e. by adding a C comment at the location affected, containing a reason for the deviation). This review can be done along with the code review for functionality described in this document.

### Task 9: Configuration Update & SW Module Integration (static code Analysis)

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| Input | |  | | | | | |
| Output | | QAC Message Configuration  QAC Analyzer Configuration | | | | | |
| D: | SW Project Manager | E: | SW Integrator | S: | SW Quality Planner | I: | SW Quality Planner |
| D = Decision | | E = Execution | | S = Support | | I = Information | |

**Objective**

Objective of this task is to update QA-C configuration.

**Description**

The configuration of the Static code Analysis can be updated by enabling/disabling additional messages. Each modification of the message configuration shall be documented with a reason by the SW Integrator.

Modifications concerning metric boundaries are also documented with a reason by the SW Integrator. It is not allowed to exceed the upper limits for the metrics given in MHE-PE-23.

For associated activities see Initial SW Module Release. Afterwards, the static code analysis for the SW module affected is re-started using the new configuration.

### Decision 2: Preventable Findings

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| Input | |  | | | | | |
| Output | |  | | | | | |
| D: | SW Project Manager | E: | SW Developer | S: |  | I: |  |
| D = Decision | | E = Execution | | S = Support | | I = Information | |

**Objective**

Objective of this task is to start another SW construction in case of preventable findings and metric deviation.

**Description**

In case of preventable findings and metric deviations (i.e. findings which can be removed by code changes without breaking constraints like resource usage, timings, etc.), another SW Construction step is started by the SW Developer. Otherwise, these findings remain unsolved and are reviewed and documented in case of a SW module baseline. In general, suppression of warnings by using “pragmas” within the source code is not allowed.

### Decision 3: SW Module Baseline

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| Input | |  | | | | | |
| Output | |  | | | | | |
| D: | SW Project Manager | E: | SW Developer | S: | CM Officer | I: | Test Manager  SW Module Tester |
| D = Decision | | E = Execution | | S = Support | | I = Information | |

**Objective**

Objective of this task is to review and to document all remaining findings of the static code analysis.

**Description**

When the SW construction of this SW module is finished and a SW module baseline is generated, all remaining findings of the static code analysis are reviewed and documented. This review and documentation is optional, if the SW module is still under construction and some more SW Construction steps have to be done.

### Task 10: Generate Static Code Analysis Reports & Annotate Warning/Metrics Deviations

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| Input | | QAC Configuration | | | | | |
| Output | | QAC Reports (Excel / HTML)  Manual Review of MISRA Rule | | | | | |
| D: | SW Project Manager | E: | SW Developer | S: | SW Quality Planner  Support from the  department  responsible for  processes, methods  and tools. | I: | SW Quality Planner |
| D = Decision | | E = Execution | | S = Support | | I = Information | |

**Objective**

Objective of this task is to record the code analysis result.

**Description**

If the SW Developer is going to create a SW module baseline, he has to generate an Excel sheet for each C file and HTML reports for the SW module. The Excel sheets called “File Report” contain all remaining static code analysis warning deviations and file/function metric values. The HTML reports contain a summary of all remaining static code analysis warning deviations and file/function metric values of this SW module. These Excel sheets and HTML reports are generated by using the “QacReportGenerator”

(Perl scripts “FileReport.pl” and “ProjectReport.pl” provided by the department responsible for processes, methods and tools).

The SW Developer reviews and comments all remaining static code analysis warnings and deviations against the metrics defined in MHE-PE-23 within the generated Excel sheets. Additionally all deviations against the MISRA coding rules given in the MHE-PE-23 are to be commented in the document Manual Review of MISRA rules (8395). Remaining deviations against the C Style Guide must be commented in the source code. Each comment must explain the reason why the deviation cannot be eliminated by source code changes. After review and annotation, all Excel sheets and the 8395 will be part of the CM (CM objects) and must be considered while creating the baseline for this SW module.

### Task 11: Code Review

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| Input | | Code | | | | | |
| Output | | Review Checklist Code  Review Checklist Code (Model-based) | | | | | |
| D: | SW Project Manager | E: | SW Developer | S: | SW Quality Planner | I: |  |
| D = Decision | | E = Execution | | S = Support | | I = Information | |

**Objective**

Objective of this task ensure the code quality.

**Description**

Depending on the software Module Class a Code Review shall be performed. The software Module Classes are determined by the Test Manager. A distinction between the following three types must be done:

|  |  |  |
| --- | --- | --- |
| **Module class** | **Review type** | **Reviewers involved** |
| Low | Review  Meeting | Colleague from own development team |
| Medium | Review  Meeting | Colleague from own development team,  additionally colleague from another development team |
| High | Review  Meeting | Colleague from own development team, additionally colleague from another development team; at least one reviewer is an 'expert' in the field to be reviewed |

The code review for hand-written source code is performed according to the Review Checklist Code (

8352). For model-based source code the Review Checklist Code (Model-based) (8388) shall be used.

On conclusion of the code review, the SW Module Class may be changed if required by arrangement with the SW Project Manager (Example: The Code Review showed that the SW Module contains properties which have a positive/negative effect on the criticality). The change has to be documented with a reason within the form MHE 8325.

### Task 12: Baseline SW Module

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| Input | |  | | | | | |
| Output | |  | | | | | |
| D: | SW Project Manager | E: | CM Officer | S: | SW Developer | I: | SW Project Manager  SW Integrator  Test Manager |
| D = Decision | | E = Execution | | S = Support | | I = Information | |

**Description**

After completion of the SW Construction, the Static Code Analysis & Review phase, the SW Developer creates a new SW module release. The MHE-PE-10 describes how to create a baseline.

# Changes, References, Appendix, Terms

## References

|  |  |  |
| --- | --- | --- |
| **Category** | **Document Name** | **Document Number** |
| Process | Construction of Product-related SW | AD-PE1-1-13 |

## Template

|  |  |  |
| --- | --- | --- |
| **Category** | **Document Name** | **Document Number** |
| Template | Manual Review of MISRA Rules | 8395 |
| Template | Review Checklist Build & QA-C | 9320 |
| Template | Review Checklist Code | 8352 |
| Template | Review Checklist Code (“Model-based”) | 8388 |
| Template | Compiler Bugs Report | 8501 |

# Supplement A: Assignment of the Failure Types

## Overview

This supplement describes the correlation between the static source code analysis, the manual code review and the SW module test.

Furthermore for the tools QA-C, Tessy/CTE and PolySpace an assignment of different error types has been made. As a rule of thumb, use

* QA-C, if the SW maintainability and portability shall be tested or enhanced (e.g. code readability, code complexity, less error prone source code statements)
* Tessy / CTE to test the SW for functional errors such as the calculation of false results
* Polyspace to detect runtime errors or spot locations where runtime errors may occur

The detailed error types which can be found with these tools are described below.

## Correlation of static analysis and test methods

For each SW module a static code analysis, a manual code review and a SW module test must be performed. It is important that these methods are not considered separately, but that the correlation of these methods is used to achieve optimal results.

First each SW module shall be classified into the SW module classes Low , Medium or High (see

MHE-PE-03, Supplement A). This SW module class determines the type of manual source code review, the involved reviewers and the testing method and depth for the SW module test. Additionally for each C file the scope of MISRA checks and metric boundaries must be configured (see AP-DE-1-13).

The static code analysis, which is done for a single SW module in parallel with the SW construction, gives an indication for complex SW parts which have a high probability for containing errors. These SW parts should be considered in more detail during a manual source code review.

Additionally it points out if it is useful to adapt test methods and depths for e.g. complex SW parts. The results of the manual source code review should also be taken into consideration for adapting the test method and depth.

As the SW module classification is needed for the manual source code review and the SW module test, the results of this analysis and test design techniques are also to be used to check and to adapt the SW module class. The results of the static source code analysis should also be taken into account for a possible adaption.

## Assignment of Metrics and Error Types to QA-C

The table below shows which metrics and error types can be found with the static code analysis tool QA-C.

The specified metric boundaries can be found in the supplement A of the MHE-PE-15. A list of MISRA rules and the information, which ones have to be checked and which ones can be checked automatically or manually, can be found in the supplement B of the AP-DE-1-13.

|  |  |
| --- | --- |
|  | **QA-C** |
| **Metrics (QA-C Identifier)** |  |
| Comment ratio (COMF) | X |
| Number of paths (STPTH) | X |
| Number of jump instructions (goto’s) (STGTO) | X |
| Cyclomatic complexity (STCYC) | X |
| Number of calling functions (STM29) | X |
| Number of called functions (STCAL) | X |
| Number of function parameters (STPAR) | X |
| Number of statements per function (STST3) | X |
| Number of call levels (STMIF) | X |
| Number of exit points (STM19) | X |
| Language scope (VOCF) | X |
| Stability index | - |
| Total number of violations of rules | - |
| Number of violations of rules, broken down by rules | - |
| Call graph for recursion | X |
| Number of instructions changed in current version compared to previous version | - |
| Number of instructions deleted in current version compared to previous version | - |
| Number of instructions added in current version compared to previous version | - |
| **MISRA rules** | X |
| **Syntax errors** | X |
| **Portability** | X |
| **Non-obvious run-time behavior** | X |
| **Implementation-dependent run-time behavior** | X |
| **Functional errors** | X |

## Assignment of Error Types to Tessy/CTE Tool

The table below shows which error types should be found with the Tessy/CTE tool at the module test level.

It depends on the test cases specified in the context of the dynamic module tests which errors are found.

Therefore, a recommendation of possible testing methods or test specification techniques for construction of the test cases is given in the right hand column of the table.

|  |  |  |  |
| --- | --- | --- | --- |
| **Error**  **category** | **Error type** | **Test method/test**  **specification technique** | **Realization in**  **Tessy/CTE** |
| Functional  errors | - Specified function not  implemented  - Specified function incorrectly  implemented  - Specified function implemented  over wrong area | Equivalence class formation  Boundary value analysis | CTE  CTE |
| Control floworiented  errors | - Missing instruction  - Unnecessary instruction  - Faulty instruction  - Instruction in wrong location in  code  - Missing branch  - Unexecutable branch  - Branching condition faulty  - Loop error | Statement coverage test  Branch coverage test  Path coverage test | n.a.  Tessy  n.a. |
| Constants  and operator  errors | - Wrong additive constant in a  calculation  - Wrong multiplicative constant in a  calculation  - Wrong coefficient in a calculation  - Wrong relational operator used  - Relational expression distorted by  constant  - Wrong boolean expression | Mutation of the input values | Partial in Tessy |

## Assignment of Use Cases and Error Types to Polyspace Tool

The table below shows which error types should be found with the Polyspace tool at the module and the integration test level. Unlike other test methods, no test cases need to be defined for a Polyspace analysis and the Polyspace analysis is recommended to be performed by the responsible department of GE-EP who will also prepare the source code for the analysis.

A Polyspace analysis first compiles the SW with a GNU C compiler and then calculates the possible values of all variables, structures and arrays for all possible execution contexts of the SW.

Typical use cases for a Polyspace analysis are

1. Test of a SW module prior to a code review

2. Test of the whole SW wrt. integration problems prior to a code review

3. Portage of the SW onto a different microcontroller

4. Runtime errors that cause a sporadic reset at any time during the SW development

Although a Polyspace analysis may deliver a list of certain errors (runtime exceptions, RTE), a Polyspace analysis does not necessarily list all possible errors due to imprecision during the analysis. The results of a

Polyspace analysis therefore can only provide the basis for a review of the SW following the analysis.

As a consequence, a Polyspace analysis provides the best results if it is applied repeatedly during the SW development process in conjunction with code reviews.

|  |  |
| --- | --- |
| **Typical runtime errors that are detected by a Polyspace analysis** | **PST** |
| Read access to non initialized variables (NIV, NIVL) | X |
| Null pointer or illegal pointer dereference (IDP, IRV, NIP) | X |
| Out of bounds array access (OBAI) | X |
| Illegal type conversion (IDP, COR) | X |
| Illegal arithmetic operations such as division by zero (ZDV, SHF) | X |
| Overflow or underflow (OVFL, UVFL) | X |
| Access problems with shared variables | X |
| Detection of unreachable code (UNR) | X |

|  |  |
| --- | --- |
| **Additional analysis benefits** |  |
| Creation of function call tree | X |
| List of access types and locations for each variable | X |
| List of shared variables and protection mechanism | X |
| Detection of type mismatch e.g. when calling a function | X |
| Detection of statically or dynamically not reachable functions (dead code) | X |
| List of recommended source code spots for manual reviews | X |