Rational. Rhapsody

IBM

IBM® Rational® Rhapsody®

IBM Rational Rhapsody Reference Workflow Guide

Version 1.10

IBM®

License Agreement

Trademarks

Table of Contents

1 Introduction

This document focuses on the model-based development (MBD) with IBM Rational Rhapsody in safety-related projects. Model-based development is widely accepted as a proven method to cope with the rapidly growing complexity of developing systems and software. MBD can improve delivery of products with higher quality by also incorporating complementary model-based testing (MBT) methods. MBD includes -- but is not limited to -- modeling, simulation, traceability information, automatic code generation, model testing, model-based code testing, model coverage and code coverage measurement, and report generation.

When using MBD and MBT for developing safety-related software additional quality objectives have to be met in order to produce and deliver “safe” systems. The mentioned additional quality objectives essentially depend on:

- a specific industrial domain where the product under development shall be deployed,
- an appropriate safety standard that must be applied for a particular domain.

The scope of this document covers software that is developed according to IEC 61508 (1), IEC 62304 (9), EN 50128 (10) or ISO 26262 (2). ISO 26262 was released in 2011 and is becoming a commonly used safety standard in the Automotive industry for passenger cars. Although ISO 26262 is mainly concerned with functional safety of Electrical/Electronic systems, it provides a framework within which safety-related systems based on other technologies can be considered. IEC 61508 Edition 2.0 was published in 2010 and is a commonly used standard for the development of electrical, electronic, programmable electronic safety-related systems. IEC 62304 was released in 2006 for the medical industry. An updated version of EN 50128 was published in 2012 and is a commonly used standard for the development of Software for Railway Control and Protection Systems. Such standards describe proven processes and methods for the development of safety-related systems, provide guidelines and recommendations when customizing process and methods to a specific customer process, describe how tools can help develop and testing of software, and what it means to qualify tools for their use that fulfills the additional requirements regarding functional safety. Although the safety standards show many similarities in general, they do differ across the detailed aspects they discuss. In particular they define different approaches to classify the product under development in certain criticality classes. For instance, SIL[1] levels 1 – 4 are used in IEC 61508 and IEC 62304, while it is named ASIL[2] levels A - D in ISO 26262. It mainly depends on such classifications to derive the concrete additional quality objectives that have to be met in order to produce and deliver “safe” systems. The quality objectives to be met become more demanding the stronger the criticality classification of the software and also applies for the qualification of tools that are used.

While the above mentioned safety standards cover all aspects of planning, development, release, and maintenance of safety-related products across life cycle phases, this document focuses on the UML/SysML model-based development and testing of safety-related software with IBM Rational Rhapsody including automatic code generation and IBM Rational Rhapsody TestConductor Add On (3). To discuss the requirements, available

1 **S**afety **I**ntegrity **L**evel
2 **A**utomotive **S**afety **I**ntegrity **L**evel

methods, solutions, and tools we use a so-called *IBM Rational Rhapsody Reference Workflow* that is described in detail in section 3. The document *IBM Rational Rhapsody TestConductor Add On Reference Workflow Guide* (4) describes in more detail the testing aspects of the workflow.

In section 2 the application of this document for the development and testing of safety-related software is described. Section 3 describes in detail the mentioned IBM Rational Rhapsody Reference Workflow. Section 4 makes a walk-through the activities of the IBM Rational Rhapsody Reference Workflow, from modeling to code generation to testing. Section 5 provides a mapping of the workflow activities to IEC 61508, IEC 62304, EN 50128 and ISO 26262.

Besides the information in this document users can find more information about IBM Rational Automotive and Medical solutions, IBM Rational Method Composer for process definition and management including ISO 26262 and IEC 62304 process templates under:

- "IBM Rational solutions for Medical"
- "IBM Rational solutions for Automotive"
- "IBM Rational Method Composer"

2 Application of this Document

This document provides a reference workflow when using IBM Rational Rhapsody for the development of safety-related software. The IBM Rational Rhapsody Reference Workflow describes a set of development and testing activities accompanied by some guidelines and recommendations. Users shall consider this reference workflow when documenting how they implement the different activities and methods described here in their project specific process. In particular they shall asses where and how their specific process deviates from the IBM Rational Rhapsody Reference Workflow. It is mandatory to justify and document any deviations, and how it is implemented in the customer process.

Section 5 contains a set of tables providing mappings from the IBM Rational Rhapsody Reference Workflow to the recommended methods in ISO 26262-6, IEC 61508, EN 50128 and IEC 62304.

3 IBM Rational Rhapsody Reference Workflow

3.1 General Considerations

The IBM Rational Rhapsody Reference Workflow describes an approach for model-based development including automatic code generation and model-based testing. Figure 1 shows the major activities in this reference workflow. The upper part of the workflow describes activities to design and implement the software. The lower part of the workflow describes activities to validate and verify the software. The approach addresses design and implementation together with appropriate test and verification:

- Textual requirements guide the development of a formal UML/SysML model, which then is translated to code using code generation. Both refinement steps are accompanied with appropriate guidelines and checks.
- The refinement step from textual requirements to a model ready for code generation is verified by performing systematic requirements based testing on the model level leveraging model simulation using IBM Rational Rhapsody's animation, also called *Model in the Loop* (MiL) testing. The generated code, either automatic or manual or a mixture of both, can be verified on a host computer by executing the same set of test cases used during MiL, but without including IBM Rational Rhapsody's animation, also called *Software in the Loop* (SiL) testing, and then performing an equivalence check of the test results (back-to-back testing) between MiL and SiL. This can be complemented by executing the set of tests on the target processor, also called *Processor in the Loop* (PiL) testing.
- Test execution on model and code comes along with structural coverage measurement to assess the completeness of the tests and to avoid including unintended functionality. Requirements coverage is measured during execution of the test cases.

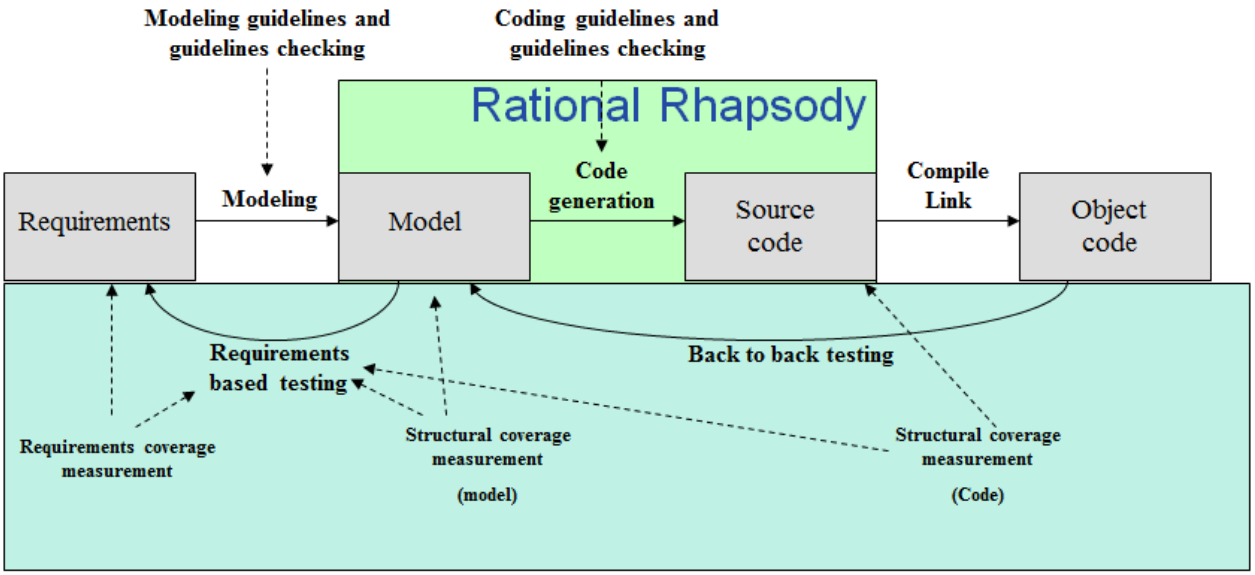

Figure 1: Activities of the IBM Rational Rhapsody Reference Workflow

The first step in the workflow is to translate given requirements into an executable model using appropriate modeling guidelines. Model-based tests are then added in order to ensure that the model indeed correctly captures the requirements. Coverage metrics (requirements coverage and model coverage) can measure the completeness of the model-based test suite. Code generation, either automatic or manual or a mixture of both, is used to generate an implementation from the model. Back-to-back testing between the model and code constitute the key element for code verification. Running a test suite on both levels verifies that the model and code show equivalent behavior. Code coverage metrics are used in order to ensure completeness of the test suite with regard to the predefined code coverage criteria.

In section (4) we make a walk-through the workflow diagram describing the construction and verification/validation of the software. Testing of models and software is discussed in even more detail in (4.5).

3.2 Tool Qualification Requirements for IBM Rational Rhapsody

When tools shall be used for the development and testing of safety-related software it is mandatory to qualify the tools or individual features of tools. The qualification depends on the concrete safety standard that is applied, the criticality level of the software under development, and how much risk is introduced into a process by using a tool or a feature

“The objective of the qualification of software tools is to provide evidence of software tool suitability for use when developing a safety-related item or element, such that confidence can be achieved in the correct execution of activities and tasks required by ISO 26262.“
(ISO 26262-8, section 11.1)

When going through the process of tool qualification several risk assessment steps have to be performed:

1. analyze how a Software Tool or a tool feature is used within a user process (“use case and tool impact”)
2. analyze if errors and malfunctions of the tool or feature would be detected in such process (“tool error detection mechanisms”)
3. choose an appropriate tool qualification method depending on (1), (2) and the ASIL or SIL level respectively.

3.2.1 ISO 26262: Tool Confidence Level and Tool Qualification

ISO 26262, part 8, chapter 11, “Confidence in the use of software tools”

- provides criteria to determine the required level of confidence in a software tool.
- provides means for the qualification of a software tool.

Confidence is needed that the software tool effectively achieves the following goals:

- The risk of systematic faults in the developed product due to malfunctions of the software tool leading to erroneous outputs is minimized.
- The development process is adequate with respect to compliance with ISO 26262 if activities or tasks required by ISO 26262 rely on the correct functioning of the software tool used.

To determine the required level of confidence in a software tool used within development under the conditions mentioned above, the following criteria are evaluated:

- The possibility that the malfunctioning software tool and its corresponding erroneous output can introduce or fail to detect errors in a safety-related item or element being developed, and
- The confidence in preventing or detecting such errors in its corresponding output

The Tool Confidence Level (TCL) is based upon

- Impact of tool failure (TI)
- Level of Tool error detection (TD)

Eventually the TCL, when combined with the customer product ASIL, leads to methods for tool qualification.

ISO 26262 describes the process as shown in Figure 2 below to determine the TCL.

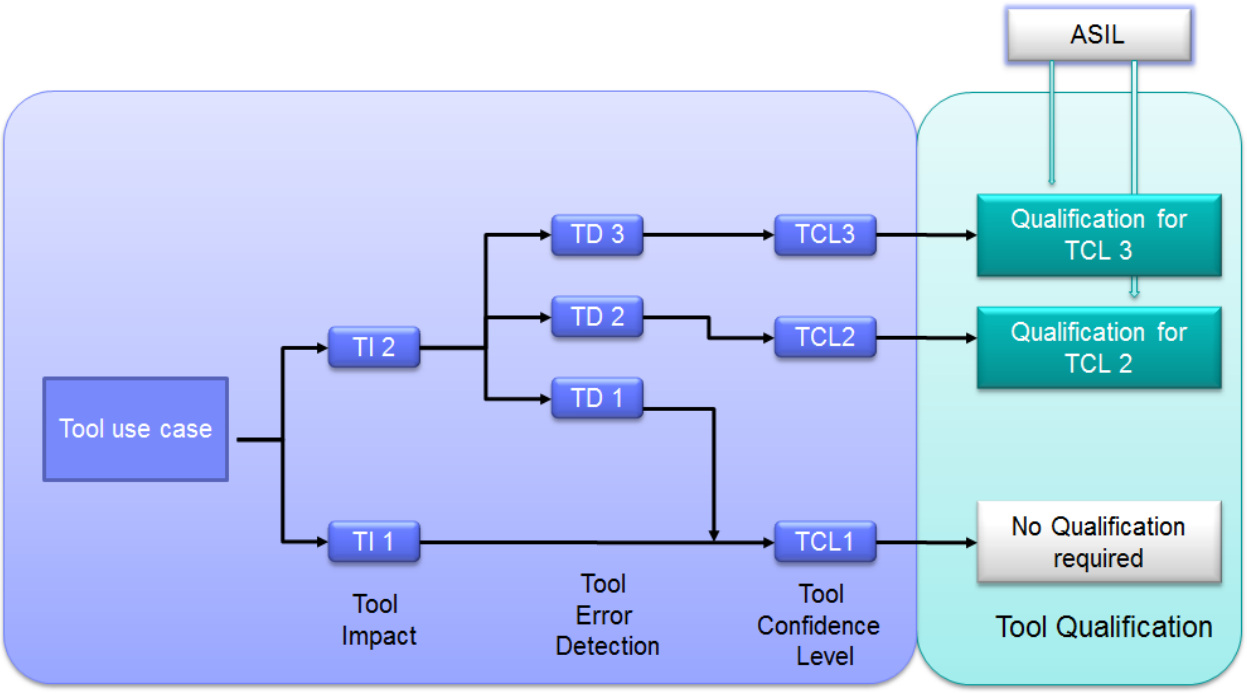

Figure 2: Process for Determining the Tool Confidence Level

- Tool Impact = 2: the tool might have an impact on safety
- Tool Error Detection = 2 or 3: errors and malfunctions are not detected with sufficient confidence in a given process
- Tool Confidence Level = 2 or 3: Qualification of a tool or feature is needed

The concrete tool qualification requirements depend on the ASIL level of the customer product under development.

The required software Tool Confidence Level shall be determined according to the following table.

		Tool Error Detection		
		TD1	TD2	TD3
Tool Impact	T1	TCL1	TCL1	TCL1
	T2	TCL1	TCL2	TCL3

Figure 3: Determining the Tool Confidence Level

Examples for tools and functions are:

- simulation
- automatic source code generation
- test specification
- test execution
- …

Prevention or detection of errors can be accomplished through

- process steps
- redundancy in tasks or software tools,
- rationality checks within the software tool itself.

- …

What does this mean for IBM Rational Rhapsody automatic code generation?

Since IBM Rational Rhapsody generated code will be part of the safety-related product it is clear that there is a Tool Impact on safety (TI=2). If the customer process is defined that it will prevent and/or detect errors and malfunctions with sufficient confidence then Tool Error Detection is 1 (TD=1). According to ISO 26262-8 prevention or detection can be accomplished through process steps, redundancy in tasks or software tools or by rationality checks within the software tool itself. As an example, TD1 can be chosen for the IBM Rational Rhapsody code generator functionality in case the produced source code is systematically verified in accordance with ISO 26262. As another example, usage guidelines can prevent malfunctions such as the incorrect or ambiguous interpretation of code constructs by a compiler.

For IBM Rational Rhapsody code generation the Tool Confidence Level can be set to TCL=1 if TD1 can be achieved. The IBM Rational Rhapsody Reference Workflow as described in this document can be used as a blue print to achieve TCL1. A software tool classified at TCL1 needs no qualification methods. Hence, IBM Rational Rhapsody code generation can be used without qualification.

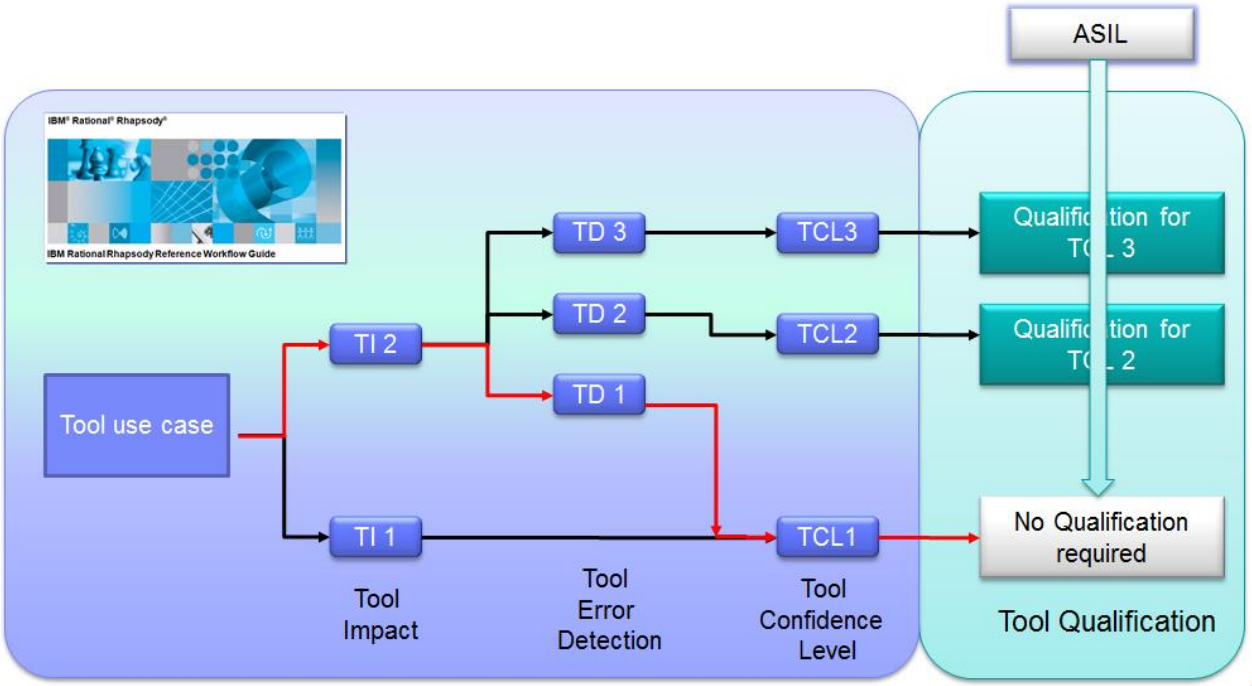

Figure 4: TCL1 for IBM Rational Rhapsody

ISO 26262 also provides the information that verification can be automated with tools. For instance the generated code can be verified with respect to the input model by applying the method of back-to-back testing. IBM Rational Rhapsody TestConductor Add On can be used to perform automated back-to-back testing in order to verify the generated source code with respect to the IBM Rational Rhapsody model. For IBM Rational Rhapsody TestConductor Add On the TCL must be set to TLC3 and qualification for IBM Rational Rhapsody TestConductor Add On must be performed.

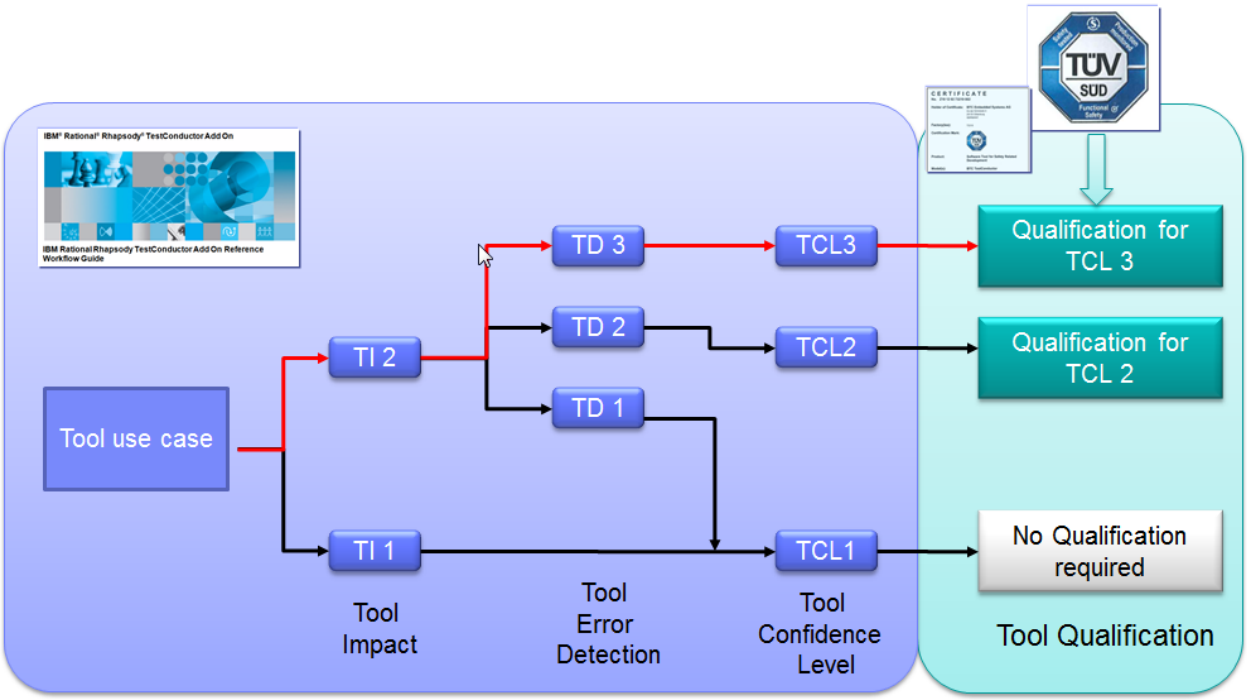

Figure 5: TCL3 for IBM Rational Rhapsody TestConductor Add On

3.2.2 IEC 61508 Edition 2.0: Tool Classification and Tool Qualification

IEC 61508-3:2010 (Edition 2.0) requires that an assessment shall be carried out for offline support tools in classes T2 and T3 to determine the level of reliance placed on the tools, and the potential failure mechanisms of the tools that may affect the executable software. Where such failure mechanisms are identified, appropriate mitigation measures shall be taken.

IBM Rational Rhapsody and also IBM Rational Rhapsody TestConductor Add On are offline support tools in the context of IEC 61508 Edition 2.0. Tools in class T3 generate outputs which can directly or indirectly contribute to the executable code of the safety-related system. IBM Rational Rhapsody is such an example. Tools in class T2 supports the test or verification of the design or executable code, where errors in the tool can fail to reveal defects but cannot directly create errors in the executable software. IBM Rational Rhapsody TestConductor Add On is a class T2 tool.
For tools in class T2, e.g. test and verification tools, it is recommended to perform a tool validation as described in IEC 61508-3:2010, paragraph 7.4.4.7, in order to get evidence that the tool conforms to its specification. When evidence is achieved the tool is qualified for being used regarding functional safety projects.
For class T3 tools it is also recommended to perform a tool validation unless appropriate risk mitigation measures are in place. Examples of such mitigation measures include: avoiding known bugs, restricted use of the tool functionality, using diverse tools for the same purpose, or checking the tool output. Checking the tool output can be a manual, interactive process activity or an automated activity as feasible with the T2 test tool IBM Rational Rhapsody TestConductor Add On.

As a consequence of the discussion above, IBM Rational Rhapsody code generation has to be classified as a T3 offline support tool. Hence, either appropriate risk mitigation

measures are implemented in the process, or evidence must be created that the tool conforms to its specification. The IBM Rational Rhapsody Reference Workflow as described in this document can be used as a blue print to implement a process providing appropriate risk mitigation measures for IBM Rational Rhapsody code generation. Hence, IBM Rational Rhapsody code generation can be used without qualification or validation respectively.

IBM Rational Rhapsody TestConductor Add On is a product to perform automated back-to-back testing in order to verify the IBM Rational Rhapsody generated source code with respect to the model. IBM Rational Rhapsody TestConductor Add On is a T2 test tool. Hence, qualification for IBM Rational Rhapsody TestConductor Add On must be performed, where validation of the software tool is a suitable method.

3.2.3 IEC 62304 Edition 1.0: Tool Qualification

IEC 62304 provides a framework of life cycle processes for the safe design and maintenance of medical device software. IEC 62304 does not place specific requirements on software tools, or on the qualification of tools. However, IEC 62304 advises that IEC 61508 can be looked to as a source of methods, tools and techniques that can be used to implement the requirements in IEC 62304.

3.2.4 EN 50128: Tool Qualification

The requirements for software tools in EN 50128 as well as the tool qualification scheme are the same as in IEC 61508 Edition 2.0.

3.3 Variation of the IBM Rational Rhapsody Reference Workflow

Beside the workflow in Figure 1 in practice sometimes the variation of the workflow in Figure 6 is applied. The difference between the workflow in Figure 1 and Figure 6 is that there is no explicit verification of the model (no MiL Simulation using IBM Rational Rhapsody animation) regarding the given requirements.

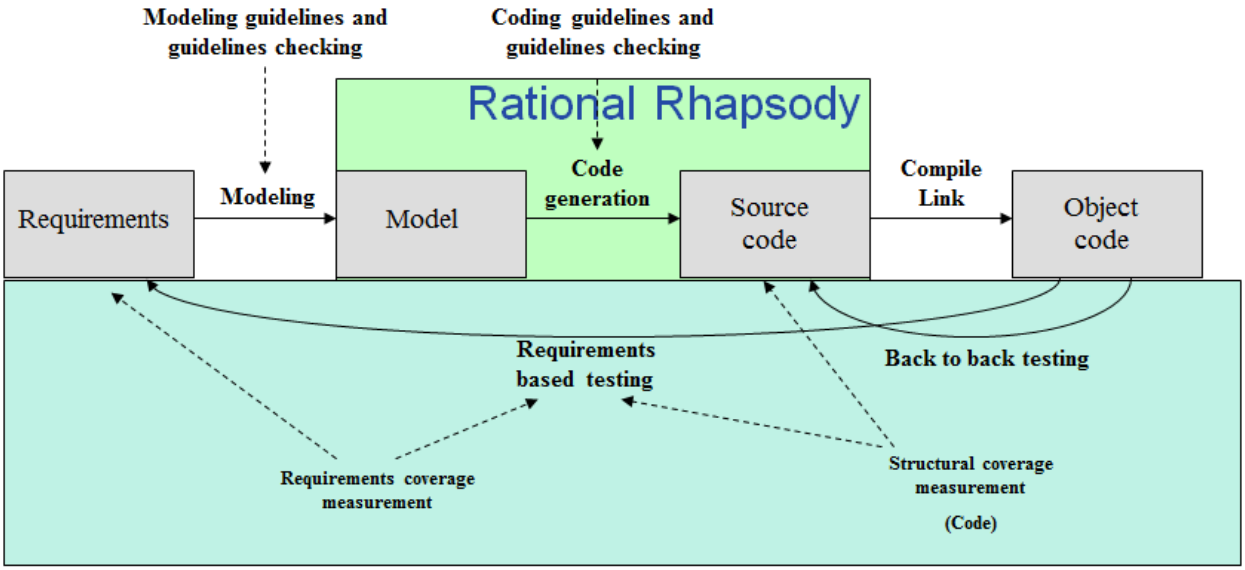

Figure 6: Variation of the IBM Rational Rhapsody Reference Workflow without Model Verification

Without explicit model verification, the simplified workflow contains the following activities:

- Creation of a model based on the given requirements. The model is created with respect to modeling guidelines. However, the *model is not simulated using IBM Rational Rhapsody animation or dynamically tested*. There might be multiple reasons for not performing animation based simulation or dynamic testing of the model. For instance, the model may contain some target hardware specific parts (e.g. some libraries only existing for the target hardware) that cannot be simulated at all on the model level.

- The model is translated into source code, either by applying an automatic code generator or manual or a mixture of both.

- The source code is compiled for SiL and/or PiL execution.

- Test Cases are created and executed on SiL and PiL level respectively. Back-to-back testing can be performed regarding SiL and PiL to ensure correct functioning even with target hardware, drivers, and operating system.

- Requirements coverage and code coverage is measured.

Although this variation of the IBM Rational Rhapsody Reference Workflow does not contain an explicit verification of the model, the correctness of the model is still verified indirectly by verifying the output of the automatic code generator on the code level by running requirement based test cases. The drawback of such an indirect verification on the code level is the fact that in case of errors the error analysis must be performed on the code level and cannot be done on the model level directly. After the source of a problem is identified on the code level, appropriate changes on the model level must be performed that will eventually correct the problem on the code level. Lifting such a problem resolution from the code level to the model level is sometimes not trivial and time consuming. Nevertheless, also with such an indirect verification on the code level the generated code can be thoroughly tested by performing a complete requirement based test. Code

coverage metrics give evidence that the generated code does not contain untested code and the generated code is fully tested.

4 IBM Rational Rhapsody Reference Workflow Activities in more Detail

In this section we describe the IBM Rational Rhapsody Reference Workflow activities captured in Figure 1 and the variant captured in Figure 6. For each explicitly shown workflow activity, how these activities can be realized with IBM Rational Rhapsody is described. The following activities are considered:

- Requirements traceability: This topic is described in detail in section 4.2.
- Modeling: General Modeling with UML and SysML is out of scope of this document. Section 4.3 points to other sources of information.
- Modeling guidelines and guideline checking: This topic is described in detail in section 4.4.
- Model verification: This topic is described in detail in section 4.5.
- Code generation and IBM Rational Rhapsody frameworks: This topic is described in detail in section 4.6.
- Coding guidelines and guideline checking: This topic is described in detail in section 4.7.
- Code verification: This topic is described in detail in section 4.8.

4.1 General Considerations

In order to develop safety-related software according to IEC 61508 Edition 2.0, IEC 62304 Edition 1.0, EN 50128 or ISO 26262 a strict process should be followed. Such processes demand many planning, construction, and verification activities during the specification, architectural design, implementation, testing, and release phases. In the subsequent sections we focus on the activities when doing modeling, code generation and unit/integration testing with IBM Rational Rhapsody. IBM Rational Rhapsody is likely to be used for many other activities as well, for instance requirement engineering, system design, software architectural design, documentation, etc. Guidance for those activities is beyond of the scope of this document. Guidance and best practices for those other features and activities are described in the IBM Rational Rhapsody Help under "IBM Rational Rhapsody 8.1". More information for using Rhapsody for safety-related development can be found in the IBM Rational Rhapsody Help under "Getting started: Designing safety-critical applications with Rational Rhapsody".

Besides the information in this document users can find more information about IBM Rational Automotive and Medical solutions, IBM Rational Method Composer for process definition and management including ISO 26262 and IEC 62304 process templates under:

"IBM Rational solutions for Medical"
"IBM Rational solutions for Automotive"
"IBM Rational Method Composer"

4.2 Requirements Traceability

Requirements traceability means that requirements can be traced to derived elements like modeling elements and finally into source code and also to test cases. Requirements traceability is a key concept that shall ensure that

- Each requirement can be traced to one or more derived artifact like model elements and/or source code and test cases. This shall ensure that all requirements are considered in subsequent development phases.
- Each model artifact, the source code and test case can be traced back to one or more requirement. This shall ensure that no unintended functionality is developed for which no requirement exists.

Within IBM Rational Rhapsody, requirements traceability can be realized as follows:

1. Create or import requirements into IBM Rational Rhapsody: In order to be able to link requirements to model elements and later to source code and to test cases, the underlying requirements must exist in the IBM Rational Rhapsody model. Requirements are usually created and managed outside of a IBM Rational Rhapsody model, e.g. in requirements management tools like IBM Rational DOORS or simply in text documents. In order to ensure requirements traceability to IBM Rational Rhapsody elements and later to source code, these requirements must be imported into IBM Rational Rhapsody. Importing requirements can either be done manually or automatically. Manually importing requirements means that requirements are created directly in IBM Rational Rhapsody, and traceability to the requirements outside of IBM Rational Rhapsody is realized by specifying a requirement ID that uniquely identifies one of the requirements. Alternatively, requirements can also be created and linked automatically by using requirements importing capabilities of IBM Rational Rhapsody. How to import requirements from other tools is described in the IBM Rational Rhapsody Help under "Integrating IBM Rational Rhapsody and Rational DOORS" and "Integrating IBM Rational Rhapsody Gateway".
2. After having created requirement elements in IBM Rational Rhapsody, one can link requirements to model elements (system model, design model, test model, ..) by using dependencies. Usually, the dependency is added to a model element that was created due to a certain requirement, and the target of the dependency is that requirement. Additonally, in order to specify that the dependency is added because of traceability reasons, usually the stereotype <<trace>> is added to the dependency.
3. Traceability from requirements to model elements: In order to verify that all requirements can be traced to a model element and vice versa, one can use e.g. the IBM Rational Rhapsody Gateway Add-On. How to use it in order to ensure complete traceability from requirements to model elements and vice versa is described in the IBM Rational Rhapsody Help under “Integrating IBM Rational Rhapsody Gateway”.
4. Traceability from requirements to source code: in order to ensure traceability from requirements to source code, IBM Rational Rhapsody provides a code generation option allowing the generation of requirements as comments into the generated source code. How to enable and use this code generation option is described in the IBM Rational Rhapsody Help under “Including requirements as comments in generated code" and under “Including requirements as comments in statechart code".

5. Users can use the requirements as comments in code capability to perform systematic manual verification if all the generated source code can be traced back to one or more requirements. The verification if all requirements are indeed implemented into source code can be verified by performing requirements based testing together with structural code coverage computation.
6. Traceability from test cases to requirements: the UML Testing Profile (5) provides an element *TestObjective* that is essentially a dependency. It allows to link test cases to requirements (6). TestObjective can also be used to link test cases to design model elements.

4.3 Modeling

UML and SysML provide many concepts for modeling software architectures, software designs and also the software behavior. Using these concepts is out of scope this document. General information about modeling software architectures and software designs with IBM Rational Rhapsody is described in the IBM Rational Rhapsody Help under “Designing and modeling”.

4.4 Modeling Guidelines and Guideline Checking

For safety-related projects it is necessary to constrain the usage of available modeling elements to those elements for which certifiable safety-related code can be generated. In general IBM Rational Rhapsody provides many modeling elements for which source code is generated (It is described in IBM Rational Rhapsody Help under “Generating code from a IBM Rational Rhapsody model”. In some cases the generated source code is not suitable to be used in safety-related projects, e.g. because the generated code is not MISRA-C (7) or MISRA-C++ (8) compliant. Thus, if it is necessary that source code can be generated that complies for instance to MISRA C/C++, such constructs should not be used. Information about how to ensure that MISRA compliant code can be generated from IBM Rational Rhapsody models can be found in IBM Rational Rhapsody Help under “Enabling the generation of MISRA compliant code”.

In order to verify that no modeling elements are used for which generated source code would not be compliant to MISRA and other guidelines, the IBM Rational Rhapsody check model feature can be used. Information about how to use IBM Rational Rhapsody’s check model feature for such purpose can be found in the IBM Rational Rhapsody Help under “Checking the model“.

4.5 Model Verification

During model verification, the created IBM Rational Rhapsody model is verified against the underlying requirements that form the basis of the model. The goal of this activity is to make sure that the model behaves as it is specified in the underlying requirements.

4.5.1 Model Simulation (MiL Simulation)

Technically, model verification is typically achieved by model simulation using IBM Rational Rhapsody animation, i.e. MiL Simulation. Model simulation can be done in IBM Rational Rhapsody by defining a configuration that has instrumentation mode set to “animation”. An example of such a configuration can be found in Figure 7.

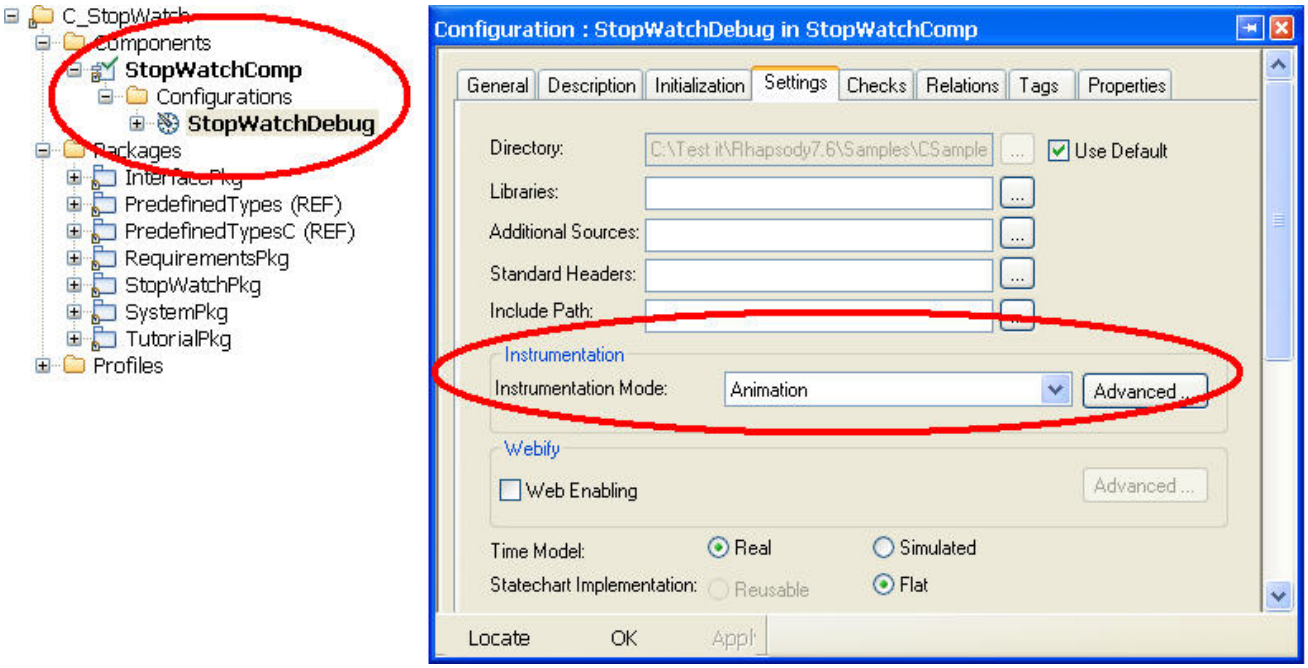

Figure 7: IBM Rational Rhapsody Configuration with instrumentation mode set to "Animation". Such a configuration can be used in order to simulate the model.

When having a configuration that can be used for simulation, one can use IBM Rational Rhapsody's simulation and animation capabilities in order to simulate and animated the model. During simulation, one can stimulate the model with inputs and one can monitor the reaction of the model to the provided inputs. IBM Rational Rhapsody provides different simulation views that can be used in order to understand and check the behavior of the model. For instance, one can use animated statecharts or animated sequence diagrams in order to verify the model's behavior, and one can inspect the values of model variables during simulation. An example of such a simulation run can be seen in Figure 8.

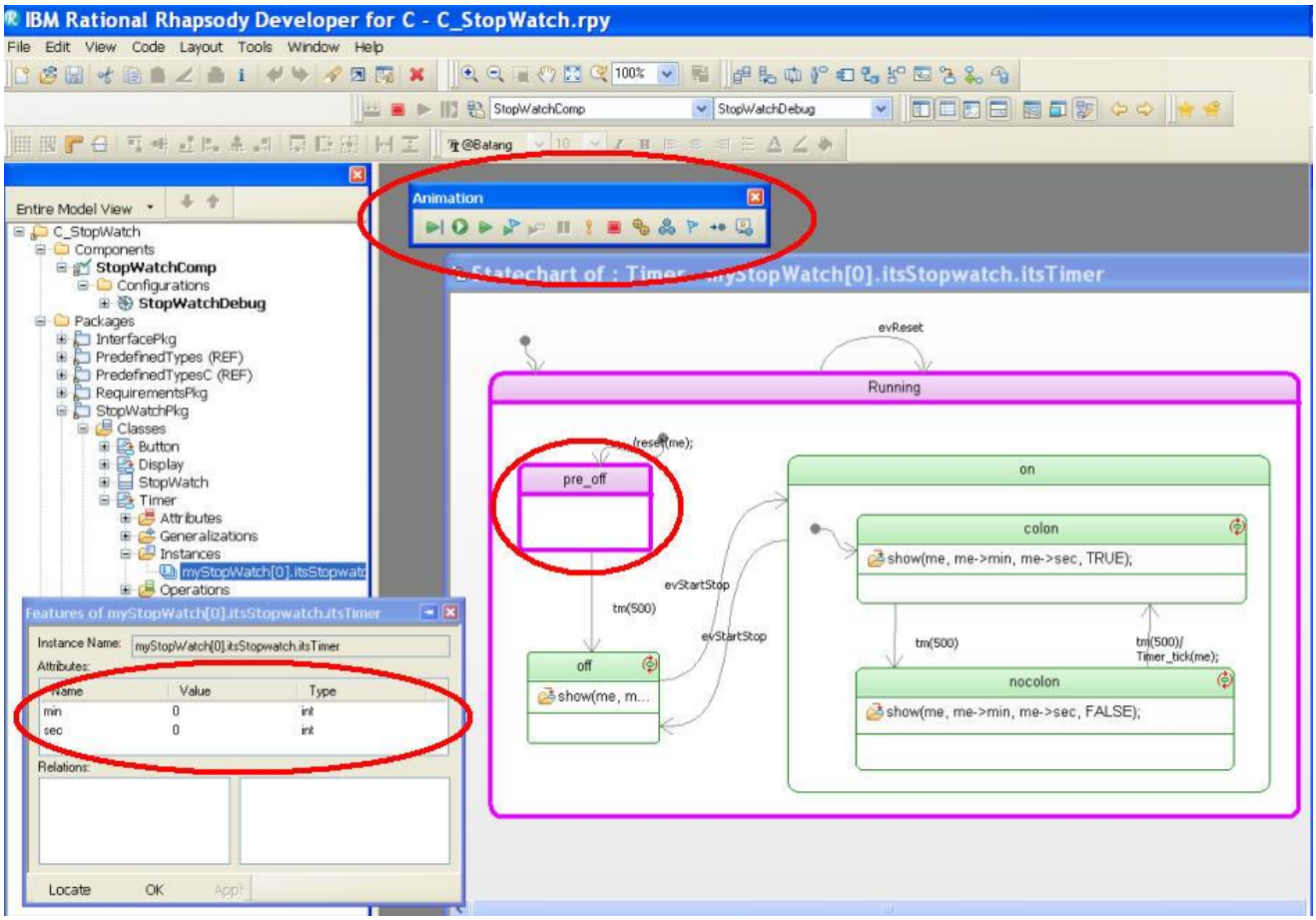

Figure 8: By simulating the model, one can step through the behavior of the model, and one can inspect values of model variables (e.g. values of attributes) during the simulation run.

Information about how IBM Rational Rhapsody models can be simulated by using animated configurations can be found in the IBM Rational Rhapsody Help under "Running animated models"

4.5.2 Requirements Based Testing

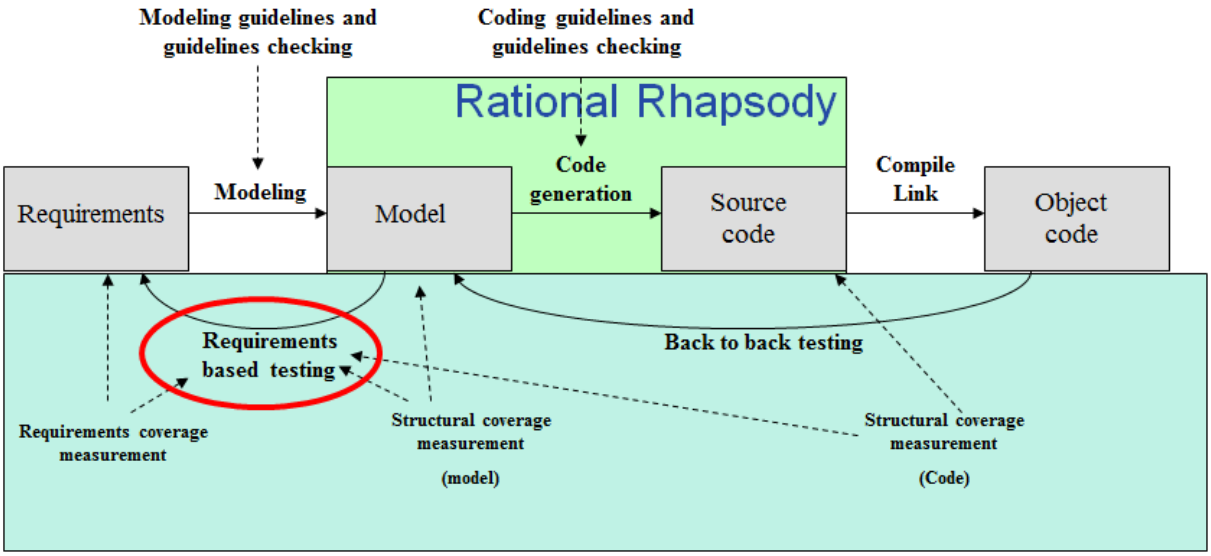

Figure 9: Requirements based testing

In the previous section we described that model simulation can be used in order to verify the correctness of the model. However, the user has to make sure that indeed each underlying requirement has been tested through model simulation. This can be done e.g. by systematically performing simulation MiL runs for each requirement as sketched in Figure 9.

Another alternative is to use the IBM Rational Rhapsody TestConductor Add On. The IBM Rational Rhapsody TestConductor Add On can be used to systematically test the correct implementation of the underlying requirements. For that purpose, the IBM Rational Rhapsody TestConductor Add On allows creating test cases for each requirement. By automatically executing the created test cases, IBM Rational Rhapsody TestConductor Add On can check the correctness of the behavior of the model with respect to the given requirements. The behavior of the test cases can be described by means of different UML diagrams or code. Additionally, in case of changes in the model all test cases can be executed automatically in order to perform a complete regression test to check that no errors were introduced by the changes. Details about how IBM Rational Rhapsody TestConductor Add On can be used in order to perform requirements based testing of an IBM Rational Rhapsody UML model is described in "IBM Rational Rhapsody TestConductor Add On User Guide". Also, document *IBM Rational Rhapsody TestConductor Add On Reference Workflow Guide* (4) describes in more detail the testing aspects of the workflow.

4.5.3 Requirements Coverage

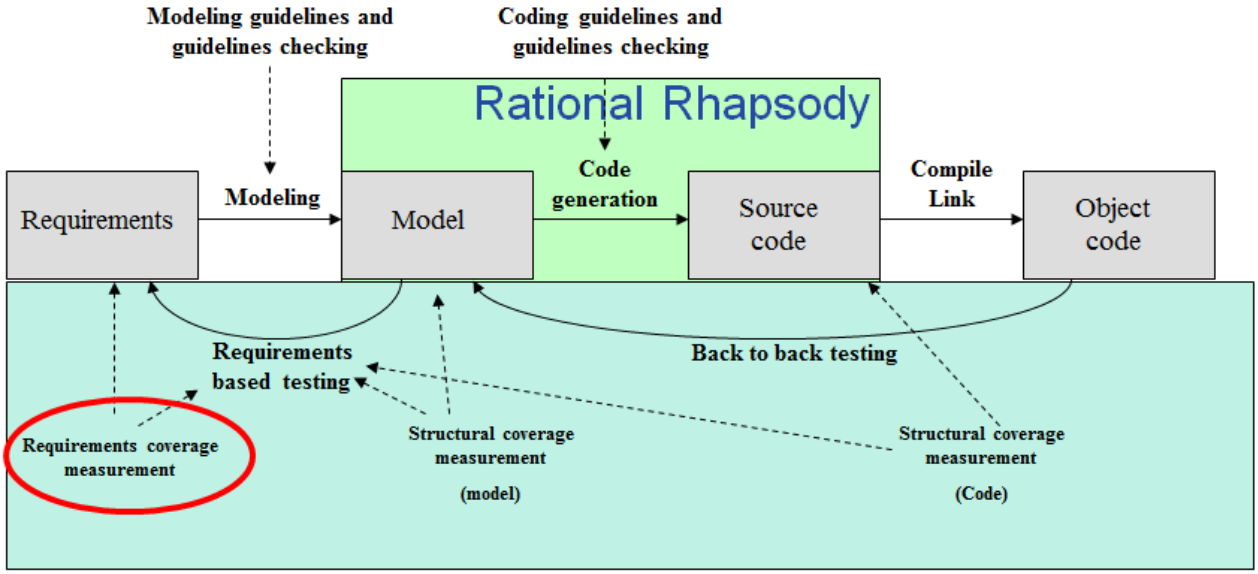

Figure 10: Requirements coverage

In order to make sure that indeed all underlying requirements have been tested properly, either by manual simulation or by specifying model based test cases with IBM Rational Rhapsody TestConductor Add On, one needs to keep track which requirements have been tested and which have not been tested so far (cf. Figure 10). If requirements are tested by manual simulation, a simple protocol can be used that keeps track of which and when certain requirements have been tested. If requirements are tested by model based test cases with IBM Rational Rhapsody TestConductor, one can use for instance predefined testing reports or testing matrices that are provided by TestConductor Add On in order to get an overview about which requirements were tested by which test cases. Information about how this can be achieved with IBM Rational Rhapsody TestConductor Add On is described in "IBM Rational Rhapsody TestConductor Add On User Guide".

.

4.5.4 Model Coverage

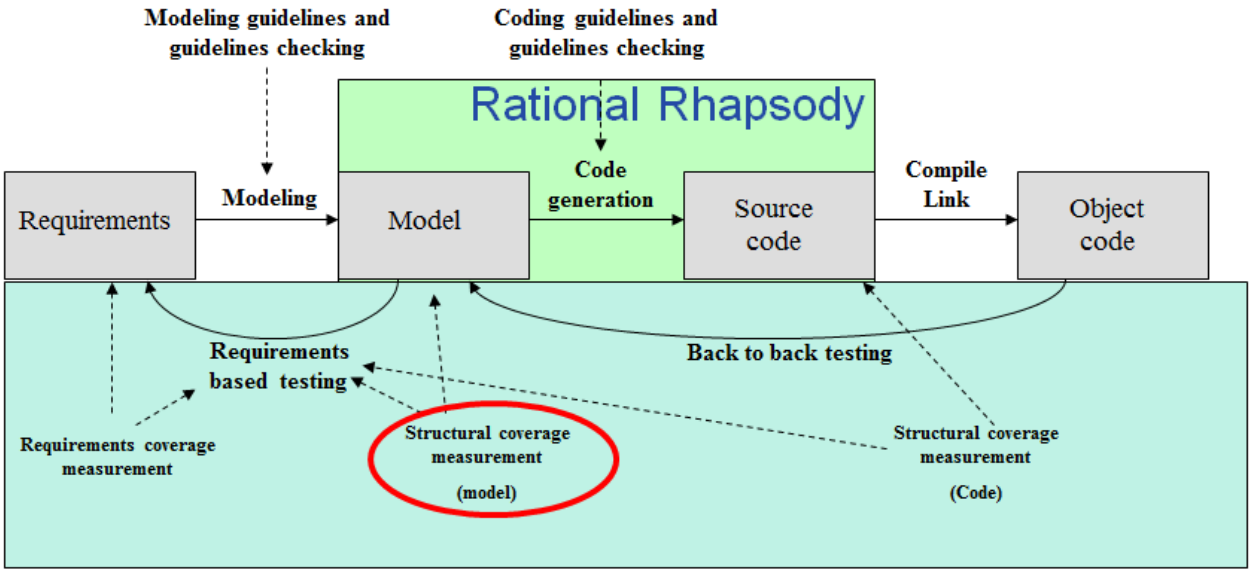

Figure 11: Model coverage

In section 4.5.3 we described how one can make sure that all underlying requirements are indeed tested on the developed IBM Rational Rhapsody UML model, either by manual simulation or by model based test cases. However, in order to make sure that all parts of the model have been tested properly, one should augment the requirements coverage information with model coverage information (cf. Figure 11).

In contrast to requirements coverage, model coverage measures which parts of the model have been executed during simulation or testing. IBM Rational Rhapsody TestConductor Add On provides capabilities in order to generate a model coverage report after test case execution. With this capability one can check if indeed all model elements have been executed by the model based test cases. Information about how to use this capability is described in "IBM Rational Rhapsody TestConductor Add On User Guide".
.

4.6 Code Generation and IBM Rational Rhapsody Frameworks

UML and SysML provide many concepts for modeling software architectures, software designs and also software behavior. With IBM Rational Rhapsody models can be translated into executable code. Using the behavioral modeling concepts and the automatic code generator is out of scope this document. General information about software development with IBM Rational Rhapsody and especially about generating code automatically from a software design model with IBM Rational Rhapsody is described in the IBM Rational Rhapsody Help under “Developing”.

The generic code generation scheme of IBM Rational Rhapsody is depicted in Figure 12. As one can see, IBM Rational Rhapsody generates the application source code for a certain IBM Rational Rhapsody model. The generated source code itself uses a library providing an execution framework. This execution framework provides implementations for certain common functionality like timers, event handling, etc. By using this execution framework library including its abstraction layer instead of real-time operating system specific functions, the source code of the generated application is independent of a certain RTOS. IBM Rational Rhapsody comes along with different implementations of this execution framework for the various existing target architectures.

General information about IBM Rational Rhapsody code generation can be found in the IBM Rational Rhapsody Help under "Generating code from a IBM Rational Rhapsody model".

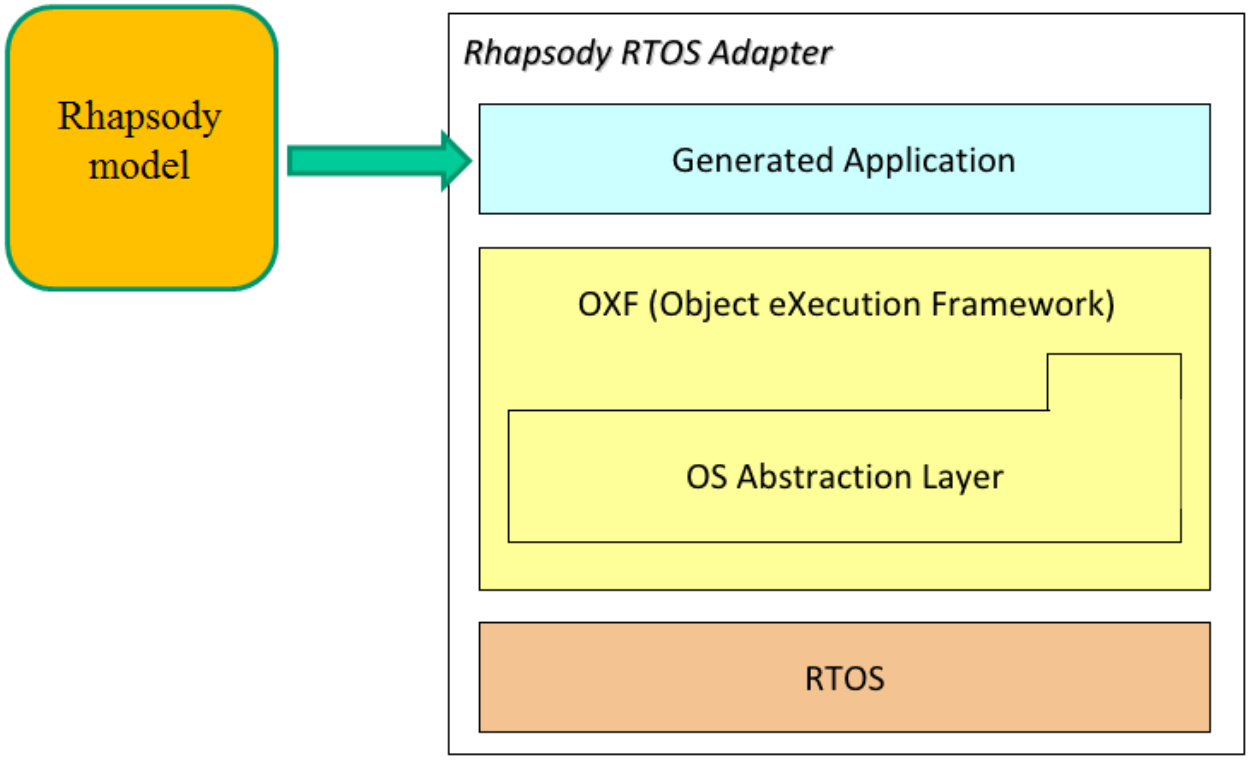

Figure 12: IBM Rational Rhapsody generated code uses the IBM Rational Rhapsody framework library

In Figure 13 three different variants of the framework library are listed. The reason why there are different versions of this framework library is that the different versions serve different purposes. The standard Object eXecution Framework (OXF) library is used for standard C and C++ code generation. When using this library, the IBM Rational Rhapsody model can even be simulated. However, the library is large with lots of different features that are not needed for safety-related production code. Thus, IBM Rational Rhapsody provides two alternative libraries called Simplified eXecution Framework (SXF) and Simplified MicroC eXecution Framework (SMXF).

OXF	SXF	SMXF
Standard C and C++ framework suitable for simulation	Safety critical C++ framework for production code	Safety critical C framework for production code

Figure 13: Different IBM Rational Rhapsody framework libraries

The SXF library is the safety-related C++ framework library. It's a comprehensive C++ library that is suitable to be used in safety-related production C++ code environments. The

C counterpart of the SXF library is the SMXF library. This is a comprehensive C library that is suitable to be used in safety-related production C code environments.

In order to be able to generate C++ safety-related production code from a IBM Rational Rhapsody model, the following setting needs to be defined:

1. The setting “SafetyCriticalForC++Developers” needs to be added to the model.

Setting “SafetyCriticalForC++Developers” also automatically loads the settings “MISRAC++” and “SXFC++” to the model. Note, when a new project is created it is possible to pre-select “SafetyCriticalForC++Developers” as default setting. It adds all three settings to the new project. This avoids adding the settings manually to an existing project.

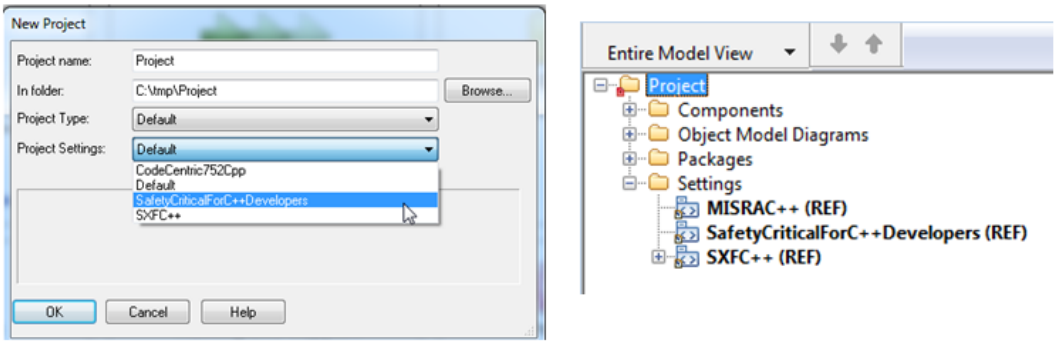

Figure 14: Creating a new model with safety-related settings

In order to be able to generate C safety-related production code from a IBM Rational Rhapsody model, the following setting need to be defined:

1. The setting “SafetyCriticalForCDevelopers” needs to be added to the model.

Setting “SafetyCriticalForCDevelopers” also automatically loads the setting “MicroC” to the model. Note, when a new project is created it is possible to pre-select “SafetyCriticalForCDevelopers” as default setting. It adds both settings to the new project. This avoids adding the settings manually to an existing project. A precondition is to select “MicroC” as project type before the setting can be selected.

Information about how settings can be added to a IBM Rational Rhapsody model can be found in the IBM Rational Rhapsody Help under “Project settings”.
More information about SXF framework and SMXF framework can be found under:

- "Simplified C++ execution framework (SXF)"
- "Simplified C execution framework (SMXF)"

Besides adding the right profiles and/or settings to the model, the code generation configurations that are used in order to generate code for the model must be attached with certain stereotypes. Details about which stereotypes must be used in order to use SXF framework or SMXF framework respectively can also be found in the IBM Rational Rhapsody SXF and SMXF help.

In order to be able using the SXF or SMXF for safety-related developments it is needed to do a systematic qualification of the simplified frameworks. The SXF and SMXF come equipped with validation suites containing:

- Test cases to verify functional correctness of the SXF/SXMF functionality
- Code coverage report after execution of the requirements based test suite
- Requirements coverage report using ReporterPlus. All framework classes and operations are traced to requirements
- MISRA compliance statements

By executing the proper validation suite it can be verified that the chosen framework is fit for its purpose.

4.7 Coding Guidelines and Guideline Checking

For safety-related applications, it is important that the generated code conforms to certain rules that are important for safety-related applications. For C, the MISRA standard is an important coding standard. In order to make sure that the IBM Rational Rhapsody generated code conforms to the MISRA standard, the setting "SafetyCriticalForCDevelopers" needs to be added to the IBM Rational Rhapsody model. This setting ensures that the design model can be refined into MISRA C compliant code.

For C++, the MISRAC++ standard is an important coding standard. In order to make sure that the code IBM Rational Rhapsody generates conforms to the MISRAC++ standard, the profile "MISRAC++" needs to be added to the IBM Rational Rhapsody model. This profile ensures that the design model can be refined into MISRA C++ compliant code.

Commercially off the shelf tools are available to automatically verify if MISRA or MISRAC++ rules are violated in the developed code.

Additionally, please follow the rules described in section 4.4.

4.8 Code Verification (SiL and PiL Verification)

For safety-related applications, it is important that the generated code is thoroughly verified. It must be verified that the code correctly implements the requirements. An essential activity in the context of the IBM Rational Rhapsody reference workflow is the verification of the model against the requirements including model coverage computation. Since the IBM Rational Rhapsody code generation translates a model into source code it has to be verified that the translation is correct. Back-to-back testing is the technique used to demonstrate the equivalence between the model behavior and the code running on the host (SiL testing) or even on the target hardware (PiL testing). Code coverage metrics give evidence that the generated code does not contain untested code and the generated code is fully tested.

4.8.1 Back-to-Back Testing

As described in section 4.6, IBM Rational Rhapsody provides different frameworks and code generation settings for different purposes. Usually, for simulating the model, an IBM Rational Rhapsody code generation configuration is used with settings appropriate for model simulation, among others

- OXF standard framework is used (cf. section 4.6)
- Animation instrumentation is enabled

The final production source code must not contain elements like animation instrumentation code. Thus, IBM Rational Rhapsody users usually create different code generation configurations for different purposes. In many cases, one distinguishes three different code generation settings called MiL, SiL, and PiL (cf. Figure 15).

Figure 15: Different code generation configurations (MiL, SiL, and PiL)

MiL (Model in the loop) is a code generation configuration that is used in order to simulate the model with animation. The MiL configuration contains settings suitable for simulating the model. SiL (Software in the loop) is a code generation configuration that is used for generating source code that shall be compiled and executed on the host system, but does not contain any instrumentation code. The intention is to generate code that can be executed and tested on the host system, e.g. by using a cross compiler and an emulator. PiL (Processor in the loop) is a code generation configuration that is used in order to generate source code for the target processor.

Now, an important verification step is to check if the code generated by these different configurations shows equivalent behavior. This is mandatory in order to make sure that model verification results from simulation runs are preserved when executing the source code generated for SiL and PiL configurations. If there are significant deviations in the behavior observed for MiL compared to e.g. SiL it means that the model does not behave as the source code generated for SiL. If these deviations are not detected, errors might show up in the final production code although they are not observable during model simulation.

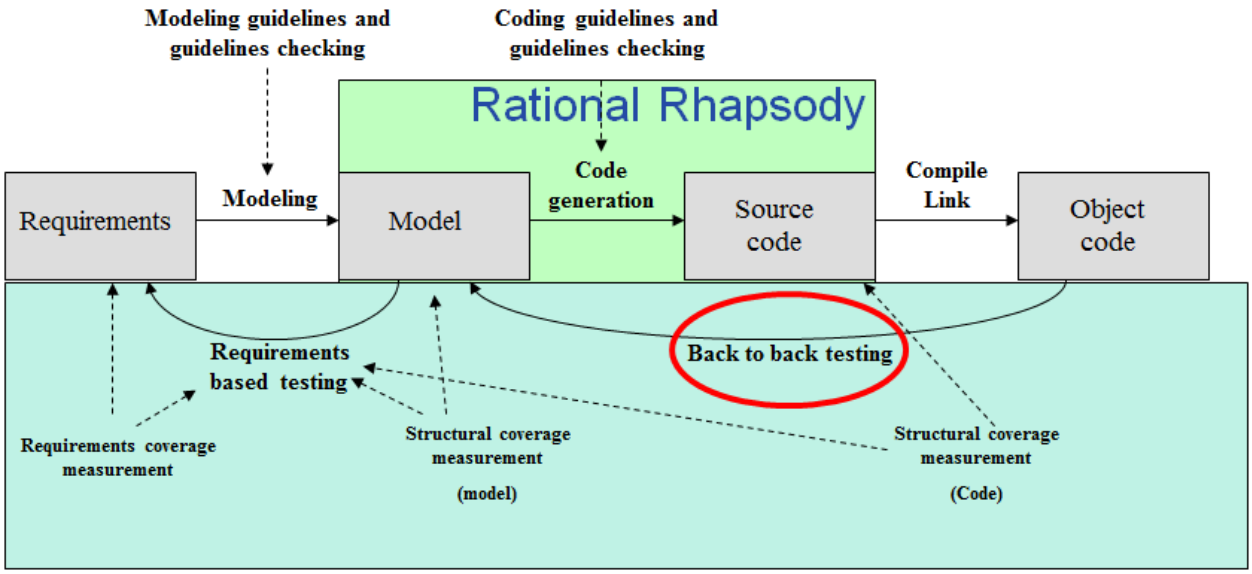

Figure 16: Back-to-back testing

In order to detect such deviations, the behavior of MiL configurations must be compared with the behavior of SiL and PiL configurations (cf. Figure 16). In order to perform such verification one can either do a manual testing and comparison or one can use a tool like IBM Rational Rhapsody TestConductor Add On that can automate back-to-back testing activities. More information about back-to-back testing with IBM Rational Rhapsody TestConductor Add On can be found in (6).

4.8.2 Code Coverage

In section 4.5.2 we have described that it is essential to systematically verify the correctness of the model with respect to the underlying requirements by using model simulation. Furthermore, in section 4.5.4 we have described that also exhaustive coverage of the model by using model simulation runs is needed. This provides evidence that all underlying requirements are correctly implemented by the model, and also that no unintended functionality is realized in the model without having a requirement for such a model part.

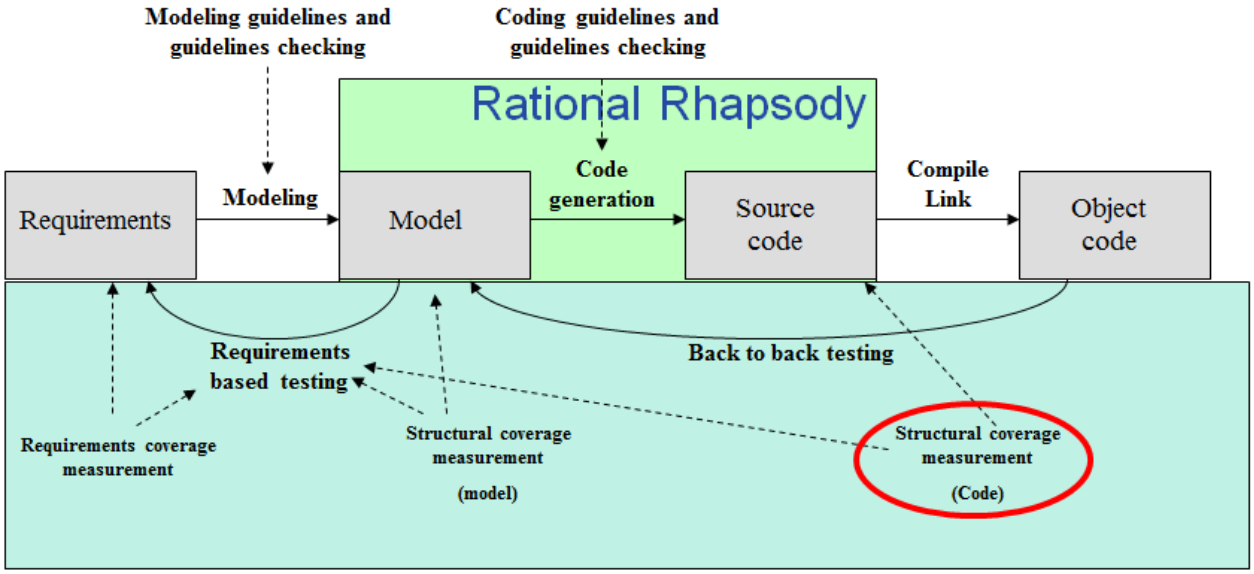

Figure 17: Code coverage

Now, if we look at the generated source code, an equivalent procedure is needed that checks if the generated source code does not contain unintended or untested functionality (cf. Figure 17). In order to do that usually source code coverage tools are applied that measure which part of the source code are executed during SiL and PiL test runs. Many tools exist that can compute and report code coverage statistics for code execution runs. When using IBM Rational Rhapsody TestConductor Add On, code coverage measurement can be easily combined with the Back-to-back testing approach described in section 4.8.1. More information about code coverage measurement with IBM Rational Rhapsody TestConductor Add On can be found in (6).

5 Mapping Reference Workflow Activities to Safety Standards

5.1 ISO 26262

Figure 18 below shows the ISO 262626-6 software development reference process.

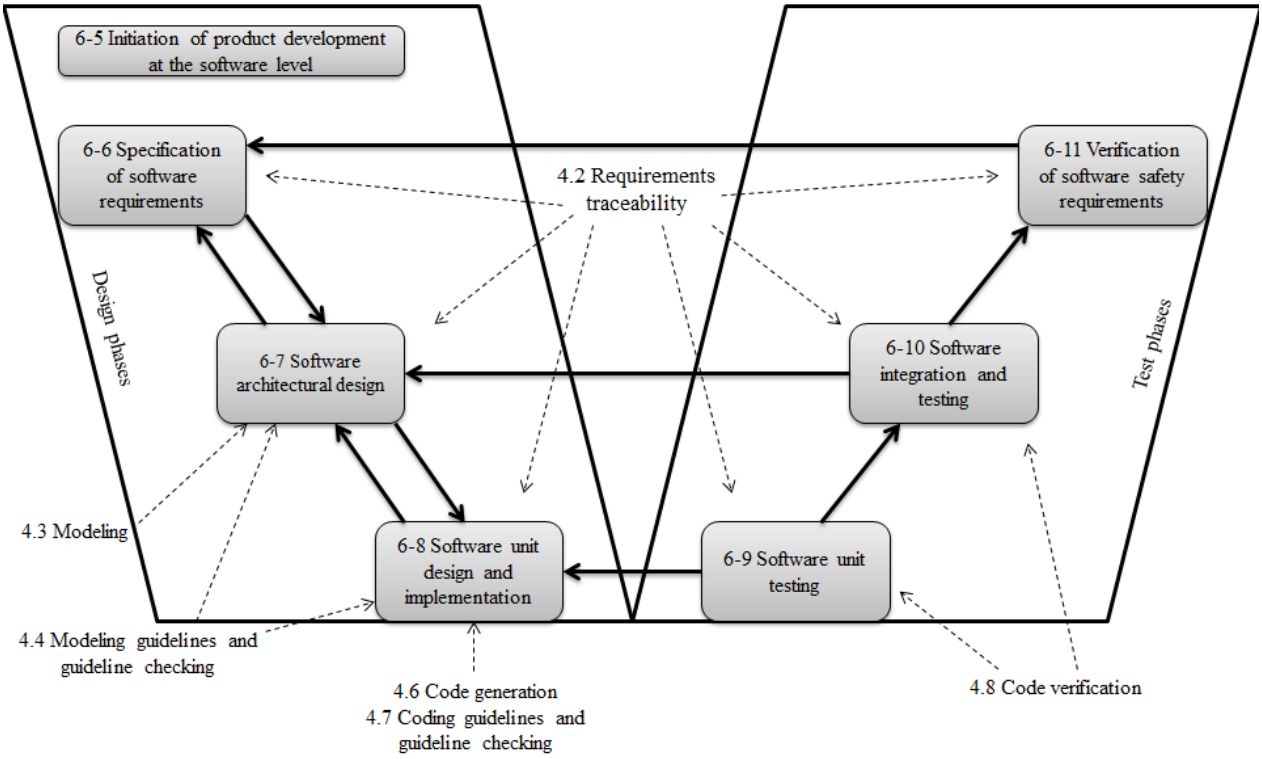

Figure 18: Overview of ISO 26262 software development reference process

Figure 19 below provides an overview about the mapping between the ISO 262626-6 software development reference process phases to the workflow activities of the IBM Rational Rhapsody Reference Workflow.

Software Development Subphase	ISO 26262-6	ASIL A	ASIL B	ASIL C	ASIL D	Workflow Reference	Model Level	Code Level
Initiation of product development at the software level	***Table 1 - Topics to be covered by modelling and coding guidelines*** 1a Enforcement of low complexity	++	++	++	++	Guidelines for modelling and coding and guideline checking (Section 4.4, 4.7)	• MISRA C: 2004 guidelines • MISRA C++: 2008 guidelines • IBM Rational Rhapsody: Enabling the generation of MISRA compliant code • IBM Rational Rhapsody: SafetyCriticalForCDevelopers setting • IBM Rational Rhapsody: SafetyCriticalForC++Developers setting • IBM Rational Rhapsody: Checking the model	• MISRA C: 2004 guidelines • MISRA C++: 2008 guidelines • IBM Rational Rhapsody: Simplified C execution framework (SMXF) • IBM Rational Rhapsody: Simplified C++ execution framework (SXF) • 3rd party tools for guideline checking on code level
	1b Use of language subsets	++	++	++	++			
	1c Enforcement of strong typing	++	++	++	++			
	1d Use of defensive implemantation techniques	o	+	++	++			
	1e Use of established design principles	+	+	+	++			
	1f Use of umambiguous graphical representation	+	++	++	++			
	1g Use of style guides	+	++	++	++			
	1h Use of naming conventions	++	++	++	++			

Software Architectural Design	***Table 2 Notations for software architectural design*** 1a Informal notations	++	++	+	+	Modelling (Software architectural design; Section 4.3)	• Using IBM Rational Rhapsody for creating a software architectural design model	
	1b Semi-formal notations	+	++	++	++			
	1c Formal notations	+	+	+	+			
	Table 3 - Principles for software architectural design 1a Hierarchical Structure of software components	++	++	++	++	Modelling (Software architectural design; Section 4.3)	• IBM Rational Rhapsody UML/SysML provides all needed concepts • IBM Rational Rhapsody provides a tool to measure UML/SysML model complexity	• IBM Rational Rhapsody UML/SysML provides all needed concepts • IBM Rational Rhapsody provide all needed concepts to generate code matching the principles
	1b Restricted Size of software components	++	++	++	++			
	1c Restricted Size of interfaces	+	+	+	+			
	1d High cohesion within each software component	+	++	++	++			
	1e Restricted coupling between software components	+	++	++	++			
	1f Appropriate scheduling properties	++	++	++	++			
	1g Restricted use of interrupts	+	+	+	++			
	Table 4 - Mechanisms for error detection at the software architectural level 1a Range checks of input and output data	++	++	++	++			
	1b Plausibility check	+	+	+	++			
	1c Detection of data errors	+	+	+	+			
	1d External monitoring facility	o	+	+	++			
	1e Control flow monitoring	o	+	++	++			
	1f Diverse software design	o	o	+	++			
	Table 5 - Mechanisms for error handling at the software architectural level 1a Static recovery mechanism	+	+	+	+			
	1b Graceful degradation	+	+	++	++			
	1c Independent parallel redundancy	o	o	+	++			
	1d Correcting codes for data	+	+	+	+			
	Table 6 - Methods for the verification of software architectural design 1a Walk-through of the design	++	+	o	o	Requirements traceability and requirements coverage measurement (Section 4.2, 4.5.3)	• IBM Rational Rhapsody UML/SysML provides all needed concepts to establish, report and verify requirements traceability • IBM Rational Rhapsody provides features supporting the process of verification and validation including traceability from requirements to model to code to test cases	
	1b Inspection of the design	+	++	++	++			
	1c Simulation of dynamic parts of the design	+	+	+	++	Requirements-based testing (Software architectural design; Section 4.5.2)	• IBM Rational Rhapsody MIL simulation • IBM Rational Rhapsody SIL simulation	
	1d Prototype generation	o	o	+	++			
	1e Formal verification	o	o	+	+			
	1f Control flow analysis	+	+	++	++			
	1g Data flow analysis	+	+	++	++			

Software Unit Design and Implementation	***Table 7 - Notations for software unit design*** 1a Natural language	++	++	++	++			
	1b Informal notations	++	++	+	+			
	1c Semi-formal notations	+	++	++	++	Modelling (Software unit; Section 4.6)	• IBM Rational Rhapsody software unit implementation model	
	1d Formal notations	+	+	+	+			
	Table 8 - Design principles for software unit design and implementation 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 9 - Methods for the verification of software unit design and implementation 1a Walk-through	++	+	o	o	Requirements traceability and requirements coverage measurement (Section 4.2, 4.5.3)	• IBM Rational Rhapsody UML/SysML provides all needed concepts to establish, report and verify requirements traceability • IBM Rational Rhapsody provides features supporting the process of verification and validation including traceability from requirements to model to code to test cases	
	1b Inspection	+	++	++	++			
	1c Semi-formal verification	+	+	++	++	Requirements-based testing (Software unit design; Section 4.5.2)	• IBM Rational Rhapsody MIL and/or SIL simulation	• IBM Rational Rhapsody SIL simulation
	1d Formal verification	o	o	+	+			
	1e Control flow analyses	+	+	++	++			
	1f Data flow analysis	+	+	++	++			
	1g Static code analysis	+	++	++	++			
	1h Semantic code analysis	+	+	+	+			
Software unit testing	***Table 10 - Methods for software unit testing*** 1a Requirements-based test	++	++	++	++	Requirements-based testing (Software unit implementation; Section 4.5.2)	• IBM Rational Rhapsody TestConductor AddOn provides all needed concepts for test case specification, execution and test management	• IBM Rational Rhapsody TestConductor AddOn supports SIL and PIL testing
	1b Interface test	++	++	++	++			
	1c Fault injection test	+	+	+	++			
	1d Ressource usage test	+	+	+	++			
	1e Back-to-back comparison test between model and code, if applicable	+	+	++	++	Back-to-back testing (Section 4.8.1)	• IBM Rational Rhapsody TestConductor AddOn supports back-to-back testing MIL <-> SIL	• IBM Rational Rhapsody TestConductor AddOn supports back-to-back testing SIL <-> PIL

	Table 11 – Methods for deriving test cases for software unit testing 1a Analysis of requirements	++	++	++	++	Requirements-based testing (Software unit implementation; Section 4.5.2)		
	1b Generation and analysis of equivalence classes	+	++	++	++			
	1c Analysis of boundary values	+	++	++	++			
	1d Error guessing	+	+	+	+			
	Table 12 – Structural coverage metrics at the software unit level 1a Statement coverage	++	++	+	+	Structural coverage measurement for model and/or code (Section 4.5.4, 4.8.2)	• IBM Rational Rhapsody TestConductor AddOn model coverage	• IBM Rational Rhapsody TestConductor AddOn code coverage
	1b Branch coverage	+	++	++	++			
	1c MC/DC (Modified Condition/Decision Coverage)	+	+	+	++			
Software integration and testing	***Table 13 - Methods for software integration testing*** 1a Requirements-based test	++	++	++	++	Requirements-based testing (Software integration; Section 4.5.2)	• IBM Rational Rhapsody TestConductor AddOn provides all needed concepts for test case specification, execution and test management	• IBM Rational Rhapsody TestConductor AddOn supports SIL and PIL testing
	1b Interface test	++	++	++	++			
	1c Fault injection test	+	+	++	++			
	1d Resource usage test	+	+	+	++			
	1e Back-to-back comparison test between model and code, if applicable	+	+	++	++	Back-to-back testing (Section 4.8.1)	• IBM Rational Rhapsody TestConductor AddOn supports back-to-back testing MIL <-> SIL	• IBM Rational Rhapsody TestConductor AddOn supports back-to-back testing SIL <-> PIL
	Table 14- Methods for deriving test cases for software integration testing 1a Analyses of requirements	++	++	++	++	Requirements-based testing (Software integration; Section 4.5.2)		
	1b Generation and analysis of equivalence classes	+	++	++	++			
	1c Analysis of boundary values	+	++	++	++			
	1d Error guessing	+	+	+	+			
	Table 15 – Structural coverage metrics at the software architectural level 1a Function coverage	+	+	++	++	Structural coverage measurement for model and/or code (Section 4.5.4, 4.8.2)	• IBM Rational Rhapsody TestConductor AddOn model coverage	• IBM Rational Rhapsody TestConductor AddOn code coverage
	1b Call coverage	+	+	++	++			

Figure 19: ISO 26262 mapping to the Rhapsody Reference Workflow

5.2 IEC 61508

The mapping between IEC 61508 and the IBM Rational Rhapsody Reference Workflow will be provided in future versions of this document.

5.3 IEC 62304

Figure 20 below shows the IEC 62304 software development reference process.

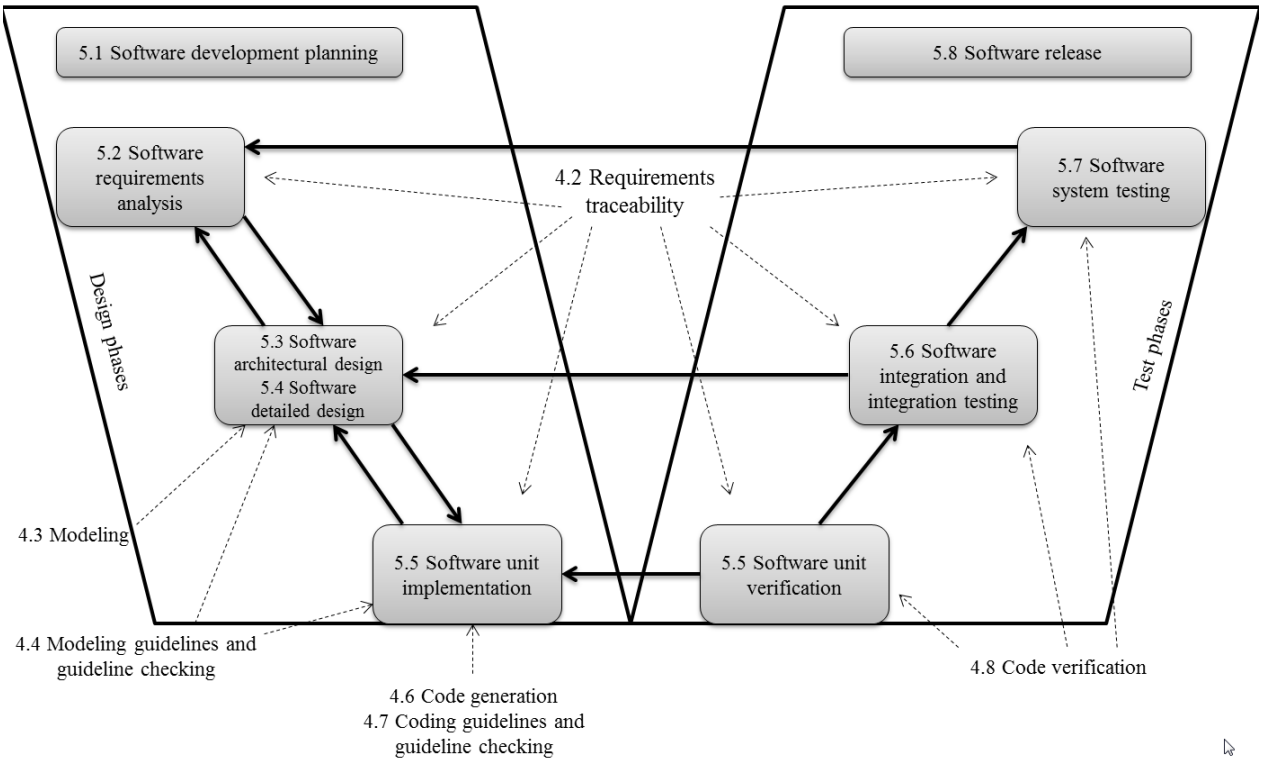

Figure 20: Overview of IEC 62304 software development reference process

Figure 21 below provides an overview about the mapping between the IEC 62304 software development reference process phases to the workflow activities of the IBM Rational Rhapsody Reference Workflow.

Software Development Subphase	IEC 62304		Class A	Class B	Class C	Workflow Reference	Model Level	Code Level
5.1 Software Development Planning	***5.1.1 Software Development Plan*** *The plan shall address the following:*	a) the PROCESSES to be used in the development of the SOFTWARE SYSTEM	X	X	X			
		b) the DELIVERABLES (includes documentation) of the ACTIVITIES and TASKS	X	X	X			
		c) TRACEABILITY between SYSTEM requirements, software requirements, SOFTWARE SYSTEM test, and RISK CONTROL measures implemented in software;	X	X	X			
		d) software configuration and change management, including SOUP CONFIGURATION ITEMS and	X	X	X			
		e)software problem resolution for handling problems detected in the SOFTWARE PRODUCTS, DELIVERABLES and ACTIVITIES at each stage	X	X	X			
	5.1.2 Keep software development plan updated	The MANUFACTURER shall update the plan as development proceeds as appropriate.	X	X	X			
	5.1.3 Software development plan reference to SYSTEM design and development	a) As inputs for software development, SYSTEM requirements shall be referenced in the software development plan by the MANUFACTURER.	X	X	X			
		b) The MANUFACTURER shall include or reference in the software development plan procedures for coordinating the software development and the design and development validation necessary to satisfy 4.1.	X	X	X			
	5.1.4 Software development standards, methods and tools planning	The MANUFACTURER shall include or reference in the software development plan: a) standards, b) methods, and c) tools associated with the development of SOFTWARE ITEMS of class C	--	--	X			
	5.1.5 Software integration and integration testing planning	The MANUFACTURER shall include or reference in the software development plan, a plan to integrate the SOFTWARE ITEMS (including SOUP) and perform testing during integration	--	X	X			
	5.1.6 Software VERIFICATION planning *The MANUFACTURER shall include or reference in the software development plan the following VERIFICATION information:*	a) DELIVERABLES requiring VERIFICATION;	X	X	X			
		b) the required VERIFICATION TASKS for each life cycle ACTIVITY;	X	X	X			
		c) milestones at which the DELIVERABLES are VERIFIED; and	X	X	X			
		d) the acceptance criteria for VERIFICATION of the DELIVERABLES. software used to support development; and	X	X	X			
	5.1.7 Software RISK MANAGEMENT planning	The MANUFACTURER shall include or reference in the software development plan, a plan to conduct the ACTIVITIES and TASKS of the software RISK MANAGEMENT PROCESS, including the management of RISKS relating to SOUP.	X	X	X			
	5.1.8 Document planning *For each identified document or type of document the following information shall be included or referenced:*	a) title, name or naming convention;	X	X	X			
		b) purpose;	X	X	X			
		c) intended audience of document; and	X	X	X			
		d) procedures and responsibilities for development, review, approval and modification.	X	X	X			
	5.1.9 Software configuration management planning *The software configuration management information shall*	a) the classes, types, categories or lists of items to be controlled;	X	X	X			
		b) the software configuration management ACTIVITIES and TASKS;	X	X	X			
		c) the organization(s) responsible for performing software configuration management and ACTIVITIES;	X	X	X			
		d) their relationship with other organizations, such as software development or maintenance;	X	X	X			
		e) when the items are to be placed under configuration control; and	X	X	X			
		f) when the problem resolution PROCESS is to be used.	X	X	X			
	5.1.10 Supporting items to be controlled	The items to be controlled shall include tools, items or settings, used to develop the MEDICAL DEVICE SOFTWARE, which could impact the MEDICAL DEVICE SOFTWARE.	--	X	X			
	5.1.11 Software CONFIGURATION ITEM control before VERIFICATION	The MANUFACTURER shall plan to place CONFIGURATION ITEMS under documented configuration management control before they are VERIFIED.	--	X	X			
5.2 Software requirements analysis	***5.2.1 Define and document software requirements from SYSTEM requirements***	For each SOFTWARE SYSTEM of the MEDICAL DEVICE, the MANUFACTURER shall define and document SOFTWARE SYSTEM requirements from the SYSTEM level requirements.	X	X	X	Requirements (Requirements traceability; Section 4.2)	Using IBM Rational Rhapsody for specifying the software requirements and to establish traceability	
	5.2.2 Software requirements content *As appropriate to the MEDICAL DEVICE SOFTWARE, the MANUFACTURER shall include in the software requirements:*	a) functional and capability requirements;	X	X	X			
		b) SOFTWARE SYSTEM inputs and outputs;	X	X	X			
		c) interfaces between the SOFTWARE SYSTEM and other SYSTEMS;	X	X	X			
		d) software-driven alarms, warnings, and operator messages;	X	X	X			
		e) SECURITY requirements;	X	X	X			
		f) usability engineering requirements that are sensitive to human errors and training;	X	X	X			
		g) data definition and database requirements;	X	X	X			
		h) installation and acceptance requirements of the delivered MEDICAL DEVICE SOFTWARE at the	X	X	X			
		i) requirements related to methods of operation and maintenance;	X	X	X			
		j) user documentation to be developed;	X	X	X			
		k) user maintenance requirements; and	X	X	X			
		l) regulatory requirements.	X	X	X			

	5.2.3 Include RISK CONTROL measures in software requirements	The MANUFACTURER shall include RISK CONTROL measures implemented in software for hardware failures and potential software defects in the requirements as appropriate to the	--	X	X			
	5.2.4 Re-EVALUATE MEDICAL DEVICE RISK ANALYSIS	The MANUFACTURER shall re-EVALUATE the MEDICAL DEVICE RISK ANALYSIS when software requirements are established and update it as appropriate.	X	X	X			
	5.2.5 Update SYSTEM requirements	The MANUFACTURER shall ensure that existing requirements, including SYSTEM requirements, are re-EVALUATED and updated as appropriate as a result of the software requirements analysis ACTIVITY.	X	X	X			
	5.2.6 Verify software requirements *The MANUFACTURER shall verify and document that the software requirements:*	a) implement SYSTEM requirements including those relating to RISK CONTROL;	X	X	X			
		b) do not contradict one another;	X	X	X			
		c) are expressed in terms that avoid ambiguity;	X	X	X			
		d) test criteria and performance of tests to determine whether the test criteria have been met;	X	X	X			
		e) can be uniquely identified; and	X	X	X			
		f) are traceable to SYSTEM requirements or other source.	X	X	X			
5.3 Software architectural design	***5.3.1 Transform software requirements into an ARCHITECTURE***	The MANUFACTURER shall transform the requirements for the MEDICAL DEVICE SOFTWARE into a documented ARCHITECTURE that describes the software's structure and identifies the SOFTWARE ITEMS.	--	X	X	Modelling (Software architectural design; Section 4.3)	Using IBM Rational Rhapsody for creating a software architectural design model	
	5.3.2 Develop an ARCHITECTURE for the interfaces of SOFTWARE ITEMS	The MANUFACTURER shall develop and document an ARCHITECTURE for the interfaces between the SOFTWARE ITEMS and the components external to the SOFTWARE ITEMS (both software and hardware), and between the SOFTWARE ITEMS.	--	X	X			
	5.3.3 Specify functional and performance requirements of SOUP item	If a SOFTWARE ITEM is identified as SOUP, the MANUFACTURER shall specify functional and performance requirements for the SOUP item that are necessary for its intended use.	--	X	X			
	5.3.4 Specify SYSTEM hardware and software required by SOUP item	If a SOFTWARE ITEM is identified as SOUP, the MANUFACTURER shall specify the SYSTEM hardware and software necessary to support the proper operation of the SOUP item.	--	X	X			
	5.3.5 Identify segregation necessary for RISK CONTROL	The MANUFACTURER shall identify the segregation between SOFTWARE ITEMS that is essential to RISK CONTROL, and state how to ensure that the segregation is effective.	--	--	X			
	5.3.6 Verify software ARCHITECTURE *The MANUFACTURER shall verify and document that:*	a) the ARCHITECTURE of the software implements SYSTEM and software requirements including those relating to RISK CONTROL;	--	--	X			
		b) the software ARCHITECTURE is able to support interfaces between SOFTWARE ITEMS and between SOFTWARE ITEMS and hardware; and	--	--	X			
		c) the MEDICAL DEVICE ARCHITECTURE supports proper operation of any SOUP items.	--	--	X			
5.4 Software detailed design	***5.4.1 Refine SOFTWARE ARCHITECTURE into SOFTWARE UNITS***	The MANUFACTURER shall refine the software ARCHITECTURE until it is represented by SOFTWARE UNITS.	--	X	X	• Modelling (Software unit; Section 4.6) • Requirements-based testing (Software unit design; Section 4.5.2) • Guidelines for modelling and coding and guideline checking (Section 4.4, 4.7)	• IBM Rational Rhapsody software unit implementation model • IBM Rational Rhapsody TestConductor AddOn provides all needed concepts for test case specification, execution and test management • MISRA C: 2004 guidelines • MISRA C++: 2008 guidelines • IBM Rational Rhapsody: Enabling the generation of MISRA compliant code • IBM Rational Rhapsody: SafetyCriticalForCDevelopers setting • IBM Rational Rhapsody: SafetyCriticalForC++Developers setting • IBM Rational Rhapsody:	• MISRA C: 2004 guidelines • MISRA C++: 2008 guidelines • IBM Rational Rhapsody: Simplified C execution framework (SMXF) • IBM Rational Rhapsody: Simplified C++ execution framework (SXF) • 3rd party tools for guideline checking on code level
	5.4.2 Develop detailed design for each SOFTWARE UNIT	The MANUFACTURER shall develop and document a detailed design for each SOFTWARE UNIT of the SOFTWARE ITEM.	--	--	X			
	5.4.3 Develop detailed design for interfaces	The MANUFACTURER shall develop and document a detailed design for any interfaces between the SOFTWARE UNIT and external components (hardware or software), as well as any interfaces between SOFTWARE UNITS.	--	--	X			
	5.4.4 Verify detailed design *The MANUFACTURER shall verify and document that the software detailed design:*	a) implements the software ARCHITECTURE;	--	--	X			
		b) is free from contradiction with the software ARCHITECTURE.	--	--	X			
5.5 SOFTWARE UNIT implementation and verification	***5.5.1 Implement each SOFTWARE UNIT***	The MANUFACTURER shall implement each SOFTWARE UNIT.	X	X	X	• Code Generation (Section 4.6) • Requirements-based testing (Software unit implementation; Section 4.5.2) • Requirements traceability and requirements coverage measurement (Section 4.2, 4.5.3) • Structural coverage measurement for model and/or code (Section 4.5.4, 4.8.2)	• IBM Rational Rhapsody UML/SysML provides all needed concepts to establish, report and verify requirements traceability and to develop source code • IBM Rational Rhapsody provides features supporting the process of verification and validation including traceability from requirements to model to code to test cases	• IBM Rational Rhapsody UML/SysML provides all needed concepts • IBM Rational Rhapsody provide all needed concepts to generate code matching the principles
	5.5.2 Establish SOFTWARE UNIT VERIFICATION PROCESS	The MANUFACTURER shall establish strategies, methods and procedures for verifying each SOFTWARE UNIT. Where VERIFICATION is done by testing, the test procedures shall be EVALUATED for correctness.	--	X	X			
	5.5.3 SOFTWARE UNIT acceptance criteria	The MANUFACTURER shall establish acceptance criteria for SOFTWARE UNITS prior to integration into larger SOFTWARE ITEMS as appropriate, and ensure that SOFTWARE UNITS meet acceptance	--	X	X			
	5.5.4 Additional SOFTWARE UNIT acceptance criteria *When present in the design, the MANUFACTURER shall include additional acceptance criteria as appropriate for:*	a) proper event sequence;	--	--	X			
		b) data and control flow;	--	--	X			
		c) planned resource allocation;	--	--	X			
		d) fault handling (error definition, isolation, and recovery);	--	--	X			
		e) initialisation of variables;	--	--	X			
		f) self-diagnostics;	--	--	X			
		g) memory management and memory overflows; and	--	--	X			
		h) boundary conditions.	--	--	X			
	5.5.5 SOFTWARE UNIT VERIFICATION	The MANUFACTURER shall perform the SOFTWARE UNIT VERIFICATION and document the results.	--	X	X			

5.6 Software integration and integration testing	***5.6.1 Integrate SOFTWARE UNITS***	The MANUFACTURER shall integrate the SOFTWARE UNITS in accordance with the integration plan.	--	X	X	• Code Generation (Section 4.6) • Requirements-based testing (Software unit implementation; Section 4.5.2) • Requirements traceability and requirements coverage measurement (Section 4.2, 4.5.3) • Structural coverage measurement for model and/or code (Section 4.5.4, 4.8.2)	• IBM Rational Rhapsody TestConductor AddOn provides all needed concepts for test case specification, execution and test management • IBM Rational Rhapsody TestConductor AddOn supports back-to-back testing MIL <-> SIL	• IBM Rational Rhapsody provide all needed concepts to integrate, compile, link, and test code • IBM Rational Rhapsody TestConductor AddOn supports SIL and PIL testing • IBM Rational Rhapsody TestConductor AddOn supports back-to-back testing SIL <-> PIL
	5.6.2 Verify software integration *The MANUFACTURER shall verify and record the following aspects of the software integration in accordance with the integration plan:*	a) the SOFTWARE UNITS have been integrated into SOFTWARE ITEMS and the SOFTWARE SYSTEM; and	--	X	X			
		b) the hardware items, SOFTWARE ITEMS, and support for manual operations (e.g., humanequipment interface, on-line help menus, speech recognition, voice control) of the SYSTEM	--	X	X			
	5.6.3 Test integrated software	The MANUFACTURER shall test the integrated SOFTWARE ITEMS in accordance with the integration plan and document the results.	--	X	X			
	5.6.4 Integration testing content	For software integration testing, the MANUFACTURER shall address whether the integrated SOFTWARE ITEM performs as	--	X	X			
	5.6.5 Verify integration test procedures	The MANUFACTURER shall EVALUATE the integration test procedures for correctness.	--	X	X			
	5.6.6 Conduct regression tests	When software items are integrated, the MANUFACTURER shall conduct REGRESSION TESTING appropriate to demonstrate that defects have not been introduced into previously integrated	--	X	X			
	5.6.7 Integration test record contents *The MANUFACTURER shall:*	a) document the test result (pass/fail and a list of ANOMALIES);	--	X	X			
		b) retain sufficient records to permit the test to be repeated; and	--	X	X			
		c) identify the tester.	--	X	X			
	5.6.8 Use software problem resolution PROCESS	The MANUFACTURER shall enter ANOMALIES found during software integration and integration testing into a software problem resolution	--	X	X			
5.7 SOFTWARE SYSTEM testing	***5.7.1 Establish tests for software requirements***	The MANUFACTURER shall establish and perform a set of tests, expressed as input stimuli, expected outcomes, pass/fail criteria and procedures, for conducting SOFTWARE SYSTEM testing, such that all software	--	X	X	• Requirements-based testing (Section 4.5.2) • Requirements traceability and requirements coverage measurement (Section 4.2, 4.5.3)	• IBM Rational Rhapsody TestConductor AddOn provides all needed concepts for test case specification, execution and test management	• IBM Rational Rhapsody TestConductor AddOn supports software system testing
	5.7.2 Use software problem resolution PROCESS	The MANUFACTURER shall enter ANOMALIES found during software system testing into a software	--	X	X			
	5.7.3 Retest after changes *When changes are made during SOFTWARE SYSTEM testing, the MANUFACTURER shall:*	a) repeat tests, perform modified tests or perform additional tests, as appropriate, to verify the effectiveness of the change in correcting the problem;	--	X	X			
		b) conduct testing appropriate to demonstrate that unintended side effects have not been introduced; and	--	X	X			
		c) perform relevant RISK MANAGEMENT ACTIVITIES as defined in 7.4.	--	X	X			
	5.7.4 Verify SOFTWARE SYSTEM testing *The MANUFACTURER shall verify that:*	a) the VERIFICATION strategies and the test procedures used are appropriate;	--	X	X			
		b) SOFTWARE SYSTEM test procedures trace to software requirements;	--	X	X			
		c) all software requirements have been tested or otherwise VERIFIED; and	--	X	X			
	5.7.5 SOFTWARE SYSTEM test record contents *The MANUFACTURER shall:*	a) document the test result (pass/fail and a list of ANOMALIES);	--	X	X			
		b) retain sufficient records to permit the test to be repeated; and	--	X	X			
		c) identify the tester.	--	X	X			
5.8 Software release	***5.8.1 Ensure software VERIFICATION is complete***	The MANUFACTURER shall ensure that software VERIFICATION has been completed and the results EVALUATED before the software is released.	--	X	X			
	5.8.2 Document known residual ANOMALIES	The MANUFACTURER shall document all known residual ANOMALIES.	--	X	X			
	5.8.3 EVALUATE known residual ANOMALIES	The MANUFACTURER shall ensure that all known residual ANOMALIES have been EVALUATED to ensure that they do not contribute to an	--	X	X			
	5.8.4 Document released VERSIONS	The MANUFACTURER shall document the VERSION of the SOFTWARE PRODUCT that is being released.	X	X	X			
	5.8.5 Document how released software was created	The MANUFACTURER shall document the procedure and environment used to create the released software.	--	X	X			
	5.8.6 Ensure activities and tasks are complete	The MANUFACTURER shall ensure that all ACTIVITIES and TASKS are complete along with all the associated documentation.	--	X	X			
	5.8.7 Archive software *The MANUFACTURER shall archive:*	a) the SOFTWARE PRODUCT and CONFIGURATION ITEMS; and	--	X	X			
		determined as the longer of: the life time of the device as defined by the MANUFACTURER or a time specified by relevant regulatory requirements.	--	X	X			
	5.8.8 Assure repeatability of software release	procedures to ensure that the released SOFTWARE PRODUCT can be reliably delivered to the point of use without corruption or unauthorised change. These procedures shall address the production and handling of media containing the SOFTWARE PRODUCT including as appropriate: - replication, - media labelling, - packaging, - protection, - storage, and - delivery.	--	X	X			

Figure 21: IEC 62304 mapping to the Rhapsody Reference Workflow

5.4 EN 50128

The mapping between EN 50128 and the IBM Rational Rhapsody Reference Workflow will be provided in future versions of this document.

Appendix A: List of Figures

Appendix B: List of References

1. *Functional safety of electrical/electronic/programmable electronic safety-related systems, IEC 61508, Edition 2.0.* 2010.

2. *Road vehicles – Functional Safety, International Organization for Standardization, ISO 26262.* 2011.

3. *IBM Rational Rhapsody TestConductor AddOn*. [Online] http://www-01.ibm.com/software/awdtools/IBM Rational Rhapsody/.

4. *IBM Rational Rhapsody TestConductor Add On Reference Workflow Guide*.

5. *UML Testing Profile, OMG, June 2011.* [Online] http://www.omg.org/spec/UTP/1.1/PDF/.

6. *IBM Rational IBM Rational Rhapsody TestConductor Add On User Guide.*

7. *MISRA-C: 2004 - Guidelines for the use of the C language in critical systems, MIRA Limited.* 2004.

8. *MISRA-C++: 2008 - Guidelines for the use of the C++ language in critical systems, MIRA Limited.* 2008.

9. *Medical device software – Software life cycle processes, IEC 62304 Edition 1.0*, 2006.

10. Railway Applications: Software for Railway Control and Protection Systems, EN 50128,
2011.