Rational. Rhapsody

IBM

## IBM® Rational® Rhapsody®

# IBM Rational Rhapsody Reference Workflow Guide

**Version 1.11**

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 structural 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 DO-178B (1) or DO-178C (2), DO-331 (10), DO-332 (11), DO-333 (12). DO-178B was released in 1992 and is a commonly used safety standard in the aerospace industry for aircrafts. Recently, DO-178C and its supplements DO-330 (Tool Qualification), DO-331 (Model-Based Development), DO-332 (Object-Oriented Technology), and DO-333 (Formal Methods) have been released which are updated versions of DO-178B and additionally regard technologies that have become popular mainly after releasing DO-178B.
These standards describe proven processes and methods for the development of safety related software, 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. In this document, we focus on safety standards DO-178B, DO-178C and mainly on its supplement DO-331 (Model-Based Development). The different process activities and objectives that are defined in these standards and supplement respectively are very similar. Whenever there are relevant differences with respect to the workflows and activities described in this document we explicitly mention such differences.
While the above mentioned safety standards cover all aspects of planning, development, release, and maintenance of safety related software 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 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 DO-178B, DO-178C and DO-331.

Besides the information in this document users can find more information about IBM Rational Aerospace solutions, IBM Rational Method Composer for process definition and management including DO-178B/C process templates under:
“IBM Rational solutions for Aerospace”
“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 objectives in DO-178B, DO-178C and DO-331 respectively.

# 3 IBM Rational Rhapsody Reference Workflow

## 3.1 General Considerations

DO-331 introduces two types of models: specification models and design models. A specification model represents high-level requirements that provide an abstract representation of functional, performance, interface, or safety characteristics of software components. Specification models do not define software design details such as internal data structures, internal data flow or internal control flow. Therefore, a Specification Model may express high-level requirements but neither low-level requirements nor software architecture. A Design Model prescribes software component internal data structures, data flow, and/or control flow. A Design Model includes low-level requirements and/or architecture.

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:

- Creation of a design model based on the given high-level requirements. The model is created with respect to modeling guidelines. Traceability from requirements to model elements is realized. The high-level requirements can be described as text or as a Specification Model or as a mixture of both.

- The design model is translated into source code by applying traditional software development methods or by applying an automatic code generator. Traceability from the requirements to the source code is realized.

- The source code is compiled (on the host system or on the target system) and can be executed.

- Test Cases are created and traceability from test cases to requirements (and vice versa) is realized.

- Test cases are executed on the compiled generated software and test results are computed.

- Requirements coverage and structural coverage is measured.

The presented workflow is very similar to traditional workflows that directly derive the source code from the low-level requirements. However, in the presented workflow design models are used to represent low-level requirements. In order to create such design models in a safe and consistent way, modeling guidelines are adopted. It is crucial that bi-directional traceability information between textual requirements and model elements is available. After creation of the design model it is translated into source code by applying an automatic code generator and/or traditional software development methods. In order to derive source code that meets safety standards, coding guidelines needs to be adopted. Correctness of the generated

source code regarding given requirements is verified by requirements based testing of the compiled source code. Test case execution comes along with structural coverage measurement to assess the completeness of the tests. Structural coverage metrics give evidence that the generated code does not contain untested code and the generated code is fully tested.

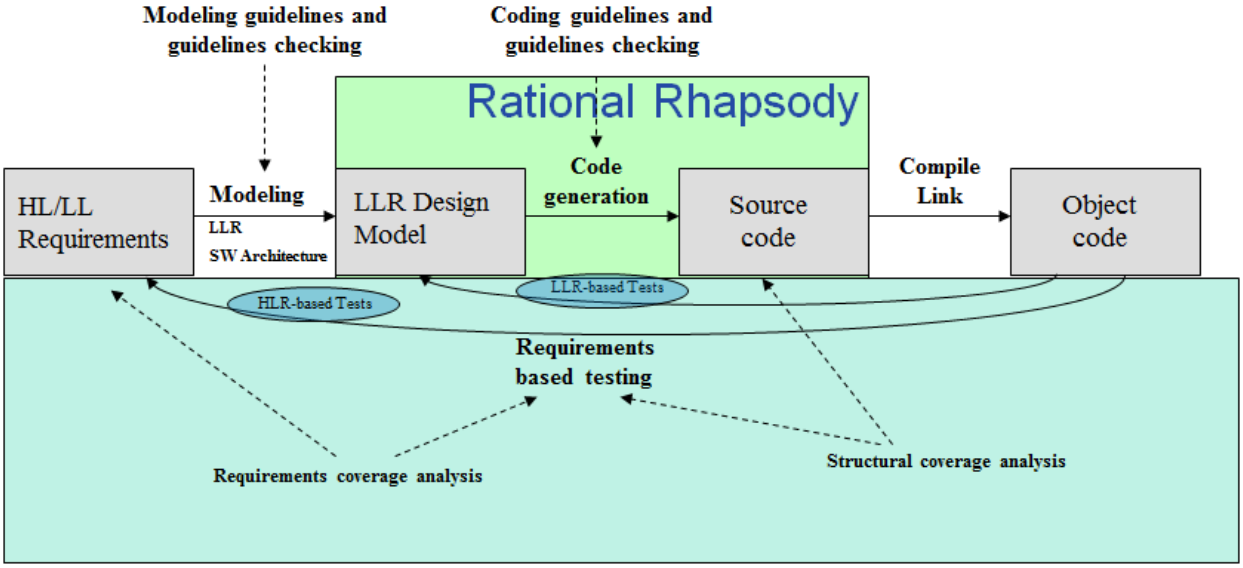


**Figure 1: Activities of the IBM Rational Rhapsody Reference Workflow**

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 section 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 assess if qualification of the tools or individual features of tools or tool chains have to be performed. 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:

*"Qualification of a tool is needed when processes of this document are eliminated, reduced, or automated by the use of a software tool without its output being verified as specified in section 6. The purpose of the tool qualification process is to ensure that the tool provides confidence at least equivalent to that of the process(es) eliminated, reduced, or automated."*
(DO-178C, section 12.2)

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 software level respectively.

DO-178C, section 12.2, requires that an assessment shall be carried out for tools to determine the appropriate tool qualification level (TQL). Tools are now classified according to three different criteria:

- Criteria 1: A tool whose output is part of the airborne software and thus could insert an error
- Criteria 2: A tool that automates verification process(es) and thus could fail to detect an error, and whose output is used to justify the elimination or reduction of:
    - Verification process(es) other than that automated by the tool, or
    - Development process(es) that could have an impact on the airborne software.
- Criteria 3: A tool that, within the scope of its intended use, could fail to detect an error.

Note: in DO-178B a "Criteria 1 tool" is a "development tool", while a "Criteria 3 tool" is a "verification tool". A "Criteria 2 tool" does not have a correspondence in DO-178B.

The appropriate TQL is as shown in Figure 2. Five levels of tool qualification, TQL-1 to TQL-5, are identified based on the tool use and its potential impact in the software life cycle processes. TQL-1 is the most rigorous level and TQL-5 is the least rigorous level.

| Software Level | Criteria 1 | Criteria 2 | Criteria 3 |
|---|---|---|---|
| A | TQL-1 | TQL-4 | TQL-5 |
| B | TQL-2 | TQL-4 | TQL-5 |
| C | TQL-3 | TQL-5 | TQL-5 |
| D | TQL-4 | TQL-5 | TQL-5 |

**Figure 2: Tool Qualification Level (TQL)**

Details about the tool qualification process, i.e. objectives, activities, guidance, and life cycle data required for each Tool Qualification Level are described in DO-330 “Software Tool Qualification Considerations” (9). In particular, section 11.3 in this document describes an approach to qualify COTS (Commercially of the shelf) tools.

As a consequence of the discussion above, IBM Rational Rhapsody code generation has to be classified according to Criteria 1. Hence, either appropriate risk mitigation measures are implemented in the process, or evidence must be created that the tool conforms to its specification by performing appropriate qualification activities needed for software level A-D. 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.

Nevertheless, users have to demonstrate that the input to the code generator is correct and complete with respect to the requirements from which the design model was developed. A qualified code generator does not reflect that the input to the code generator is correct. Hence, the qualification of the code generator would be of limited certification value anyway.

IBM Rational Rhapsody TestConductor Add On is a product to perform automated requirements-based testing in order to verify the IBM Rational Rhapsody generated source code with respect to the requirements. IBM Rational Rhapsody TestConductor Add On is a Criteria 3 tool. Hence, qualification for IBM Rational Rhapsody TestConductor Add On must be performed, but just according to the TQL-5 level. Hence, it can be performed with less many objectives which have to be met.

Document *IBM Rational Rhapsody TestConductor Add On Qualification Kit for DO-178B/C Overview (13)* discusses in more detail how TestConductor can be qualified according to DO-178B/C and DO-330.

From end user point of view the investment into IBM Rational Rhapsody TestConductor Add On brings two serious immediate returns:

- Rhapsody code generator does not need qualification
- Rhapsody TestConductor supports verification of the design model against the requirements, and also verification of the code against the requirements.

## 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 3 is applied. The difference between the workflow in Figure 1 and Figure 3 is that there are explicit verification steps of the model (model simulation using IBM Rational Rhapsody animation) regarding the given requirements. The approach can be summarized as follows:

- Creation of a design model based on the given high-level requirements. The model is created with respect to modeling guidelines. Traceability from requirements to model elements is realized. The high-level requirements can be described as text or as a Specification Model or as a mixture of both.

- Test cases are created and traceability from test cases to requirements (and vice versa) is realized. Test cases are executed on the design model level leveraging model simulation using IBM Rational Rhapsody's animation (Model Simulation).
  Note: DO-331 defines the notion of simulation cases for design model simulation as an equivalent to test cases for software testing.

- Requirements coverage and design model coverage are measured during the model based verification process in order to ensure completeness of the simulation based verification process.

- The design model is translated into source code by applying traditional software development methods or an automatic code generator. Traceability from the requirements to the source code is realized.

- The source code is compiled (on the host system or the target system) and can be executed.

- The simulation cases used for design model verification are used as test cases and executed on the compiled generated software and test results are computed. If simulation cases and test cases produce identical test verdicts it demonstrates equivalent behavior of design model and code regarding high-level requirements.

    - **Note**: in document DO-331, appendix MB.B (FAQs), it is discussed under FAQ #16 (MB.B.16) how design model simulation can support the assessment of test coverage of the low-level requirements contained in a design model.
        - If high-level requirements-based tests are developed, and
        - if these tests are run on the code to verify compliance of the code to the high-level requirements, and
        - if these same tests are used for the simulation of the design model to verify compliance of the design model to the high-level requirements.
    - In this case, simulation in combination with model coverage analysis, can support the assessment of test coverage of the low-level requirements

contained in the Design Model. When this approach is used, the high-level requirements-based tests are run on the executable object code.

- Structural coverage is measured in order to ensure completeness of software verification process.

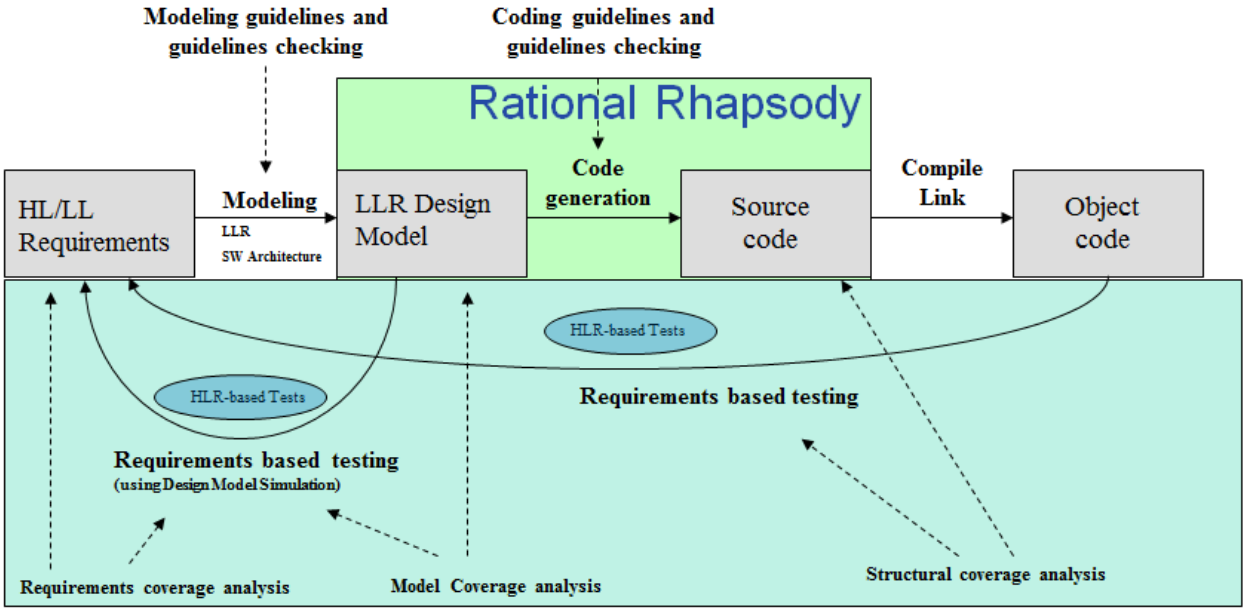


**Figure 3: Variation of the IBM Rational Rhapsody Reference Workflow with explicit Model Verification**

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. Requirements based testing of the code constitutes the key element for code verification. Running a test suite on both levels verifies that the model and code show the same behavior. Structural coverage metrics are used in order to ensure completeness of the test suite with regard to the predefined structural coverage criteria.

# 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 3. 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 standards 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 DO-178B, DO-178C or DO-331 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 Aerospace solutions, IBM Rational Method Composer for process definition and management including DO-178B/C process templates under:
“IBM Rational solutions for Aerospace”
“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 structural 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 Standards 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 design model verification, the created IBM Rational Rhapsody design model is verified against the high-level requirements from which the design model was developed. The goal of this activity is to make sure that the model behaves as it is specified in the requirements.

### 4.5.1 Model Simulation

Technically, design model verification is typically achieved by using IBM Rational Rhapsody animation, i.e. Model In the Loop (MiL) Simulation. Simulation for verification of the model is used to check the functionality of the design model against the high-level requirements from which it was developed. 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 4.

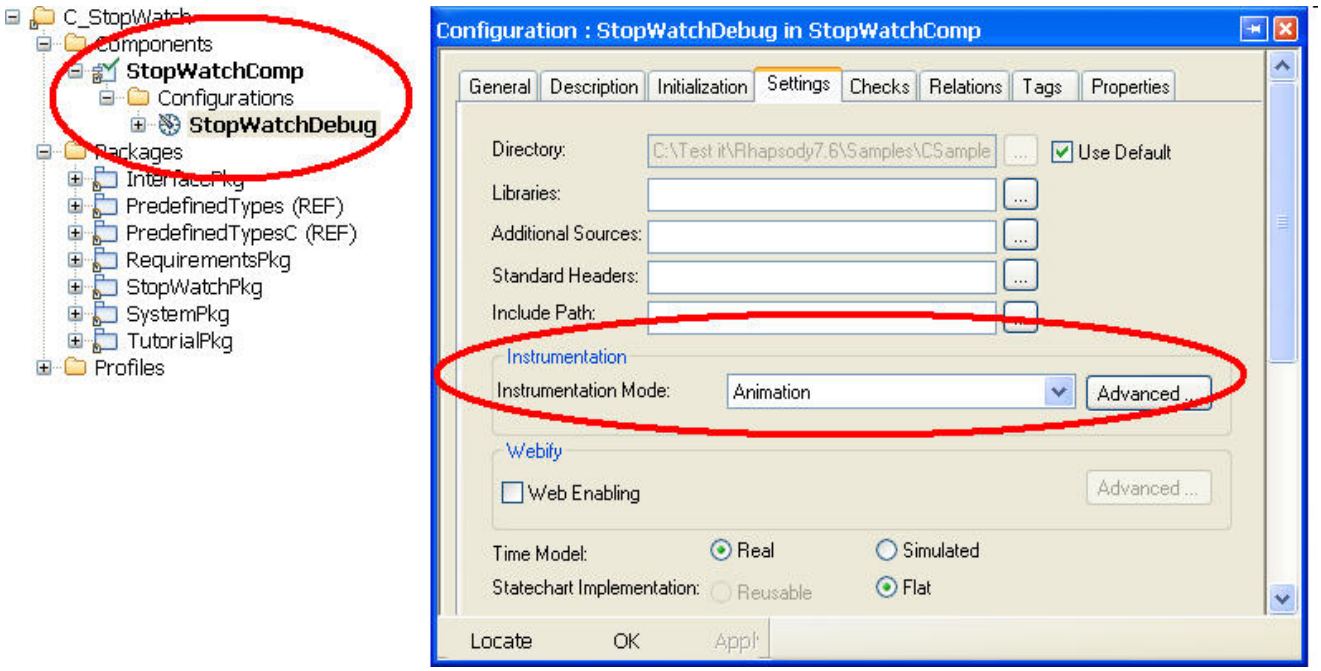


**Figure 4: IBM Rational Rhapsody Configuration with instrumentation mode set to "Animation". Such a configuration can be used in order to simulate the model.**

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 animate 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 5.

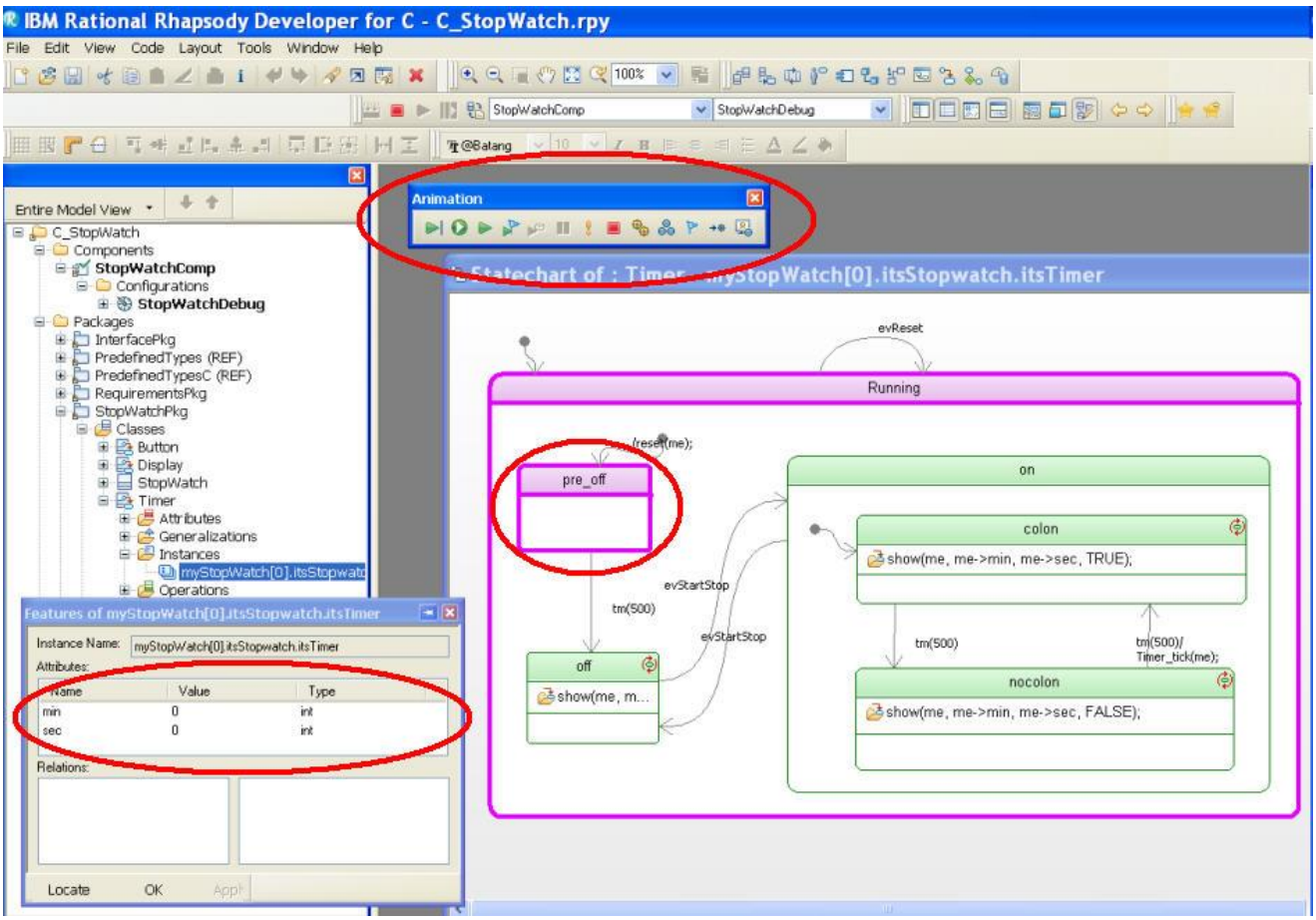


**Figure 5: By simulating the model, one can step through the behavior of the model, and one can inspect values of model variables (e.g. values of attributes) during the simulation run.**

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

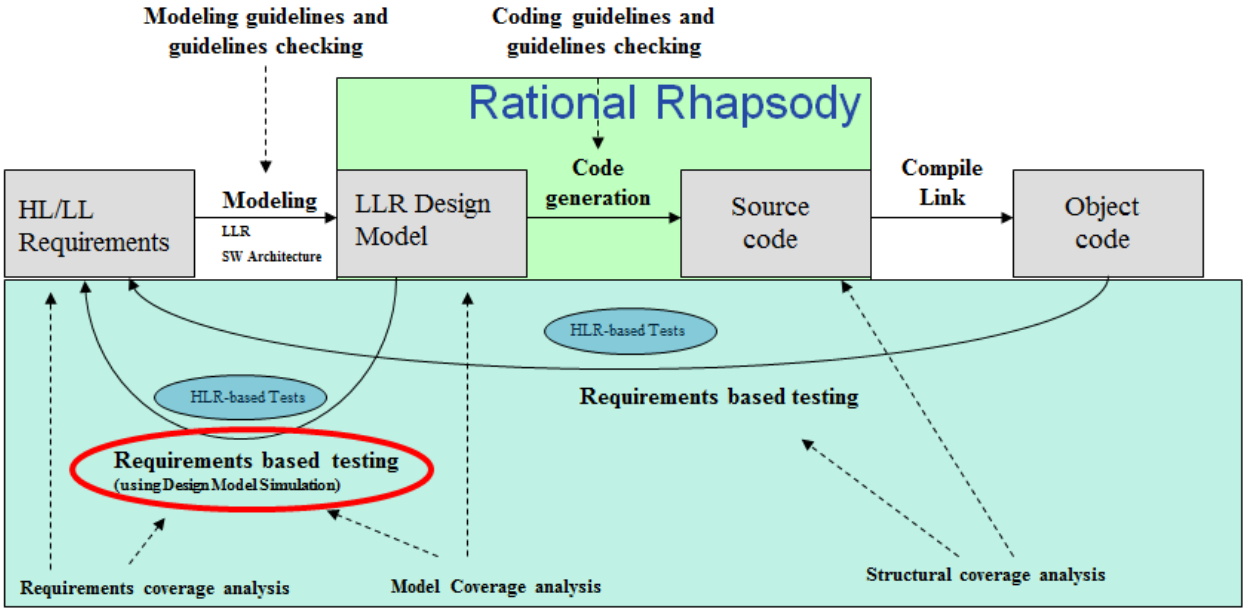


**Figure 6: Requirements based testing**

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 runs for each requirement as sketched in Figure 6.

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 modeling of the high-level requirements. For that purpose, the IBM Rational Rhapsody TestConductor Add On allows creating simulation cases for each requirement. By automatically executing the created simulation cases, IBM Rational Rhapsody TestConductor Add On can check the correctness of the behavior of the model with respect to the given requirements. DO-331 uses the term “simulation cases” for test cases that are executed on a model by means of model simulation. The behavior of the simulation cases can be described by means of different UML diagrams or code. Additionally, in case of changes in the model all simulation 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"].

### 4.5.3 Requirements Coverage

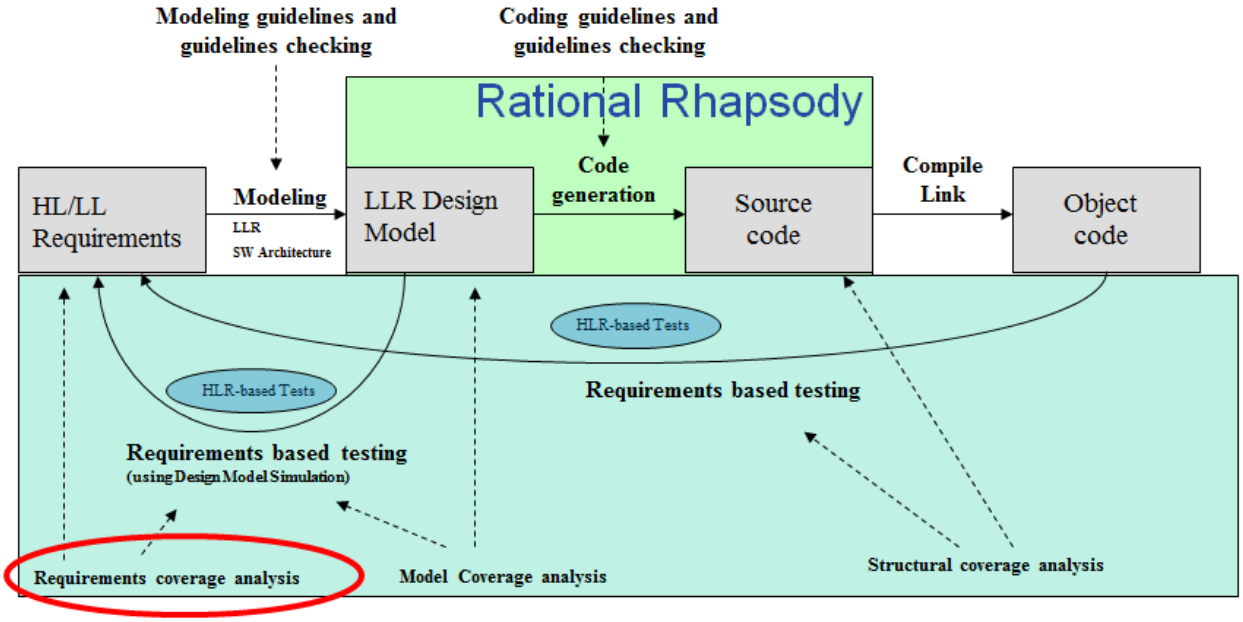


**Figure 7: Requirements coverage**

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 (simulation 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 7). 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

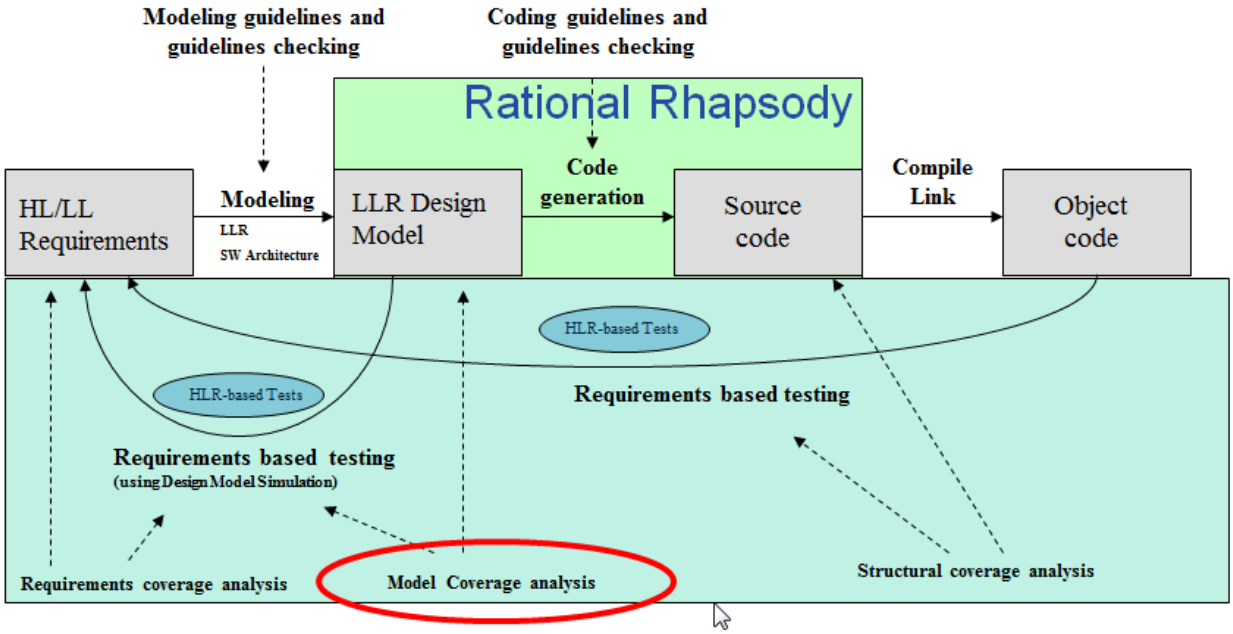


**Figure 8: Model coverage**

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 8).

In contrast to requirements coverage, model coverage measures which parts of the model have been executed during simulation. IBM Rational Rhapsody TestConductor Add On provides capabilities in order to generate a model coverage report after simulation. With this capability one can check if indeed all model elements have been executed by the model based simulation 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 9. 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".

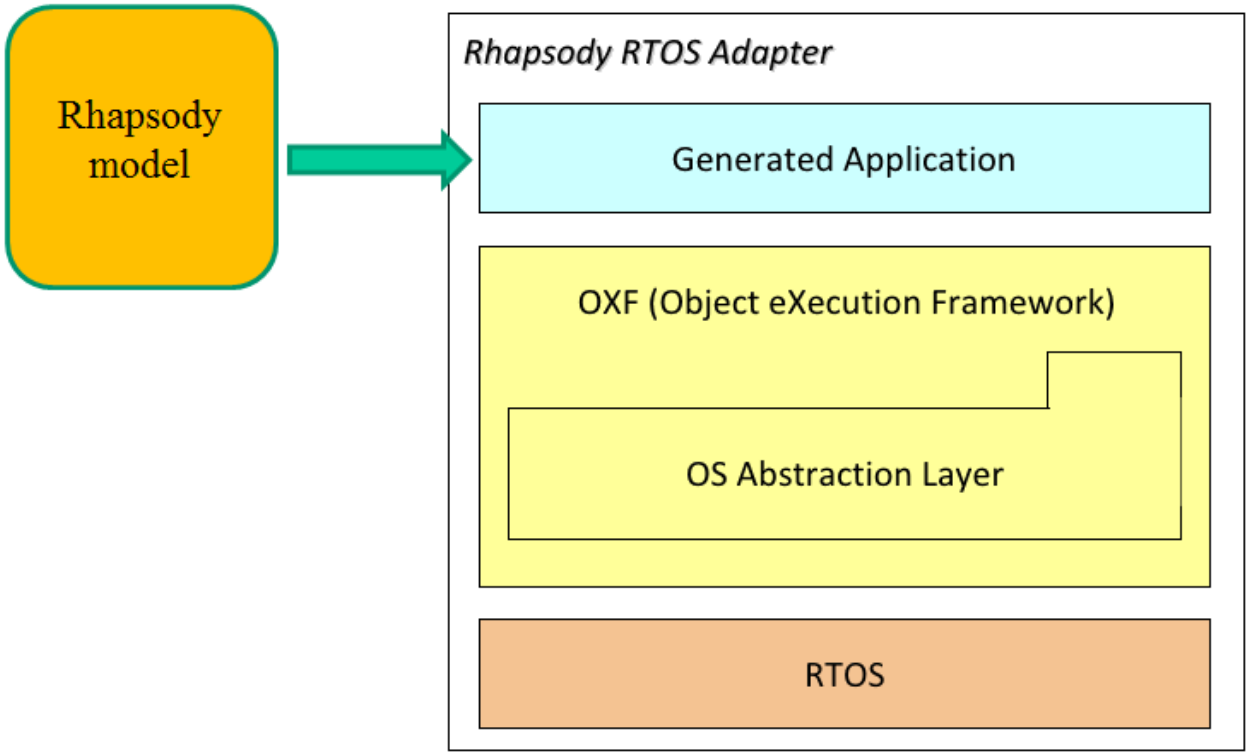


**Figure 9: IBM Rational Rhapsody generated code uses the IBM Rational Rhapsody framework library**

In Figure 10 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 10: 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++" as well as packages "AutoGeneratedCppBehavioralCodeRequirements" and "AutoGeneratedCppCodeRequirements" 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 artifacts to the new project. This avoids adding the settings manually to an existing project.

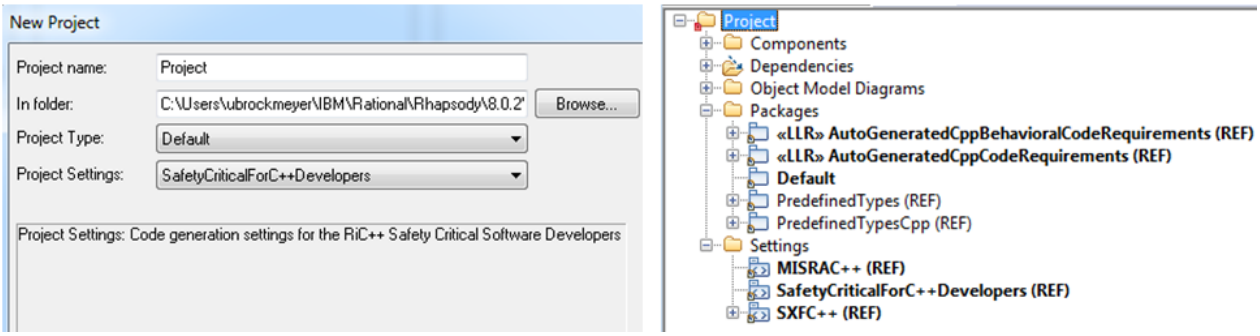


**Figure 11: Creating a new C++ model with safety related settings**

2. Add AppliedProfile dependency on MISRA C++ setting to the project manually.
3. Set the SXF stereotype configuration to use when generating C++ code.

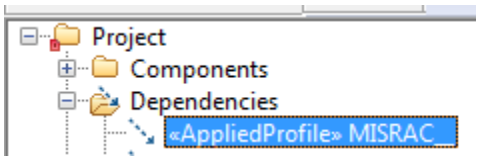


**Figure 12: Adding AppliedProfile MISRAC++**

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", and also packages AutoGeneratedCBehavioralCodeRequirements as well as AutoGeneratedCCodeRequirements to the model. Note, when a new project is created it is possible to pre-select "SafetyCriticalForCDevelopers" as default setting. It adds settings and packages to the new project. This avoids adding the setting manually to an existing project. A precondition is to select "MicroC" as project type before the setting can be selected.

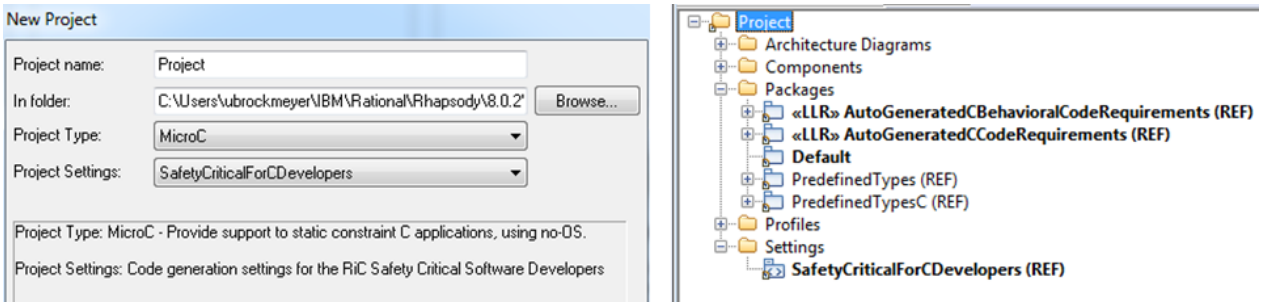


**Figure 13: Creating a new C model with safety related settings**

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 verification of the simplified frameworks. The SXF and SMXF come equipped with test suites containing:

- Test cases to verify functional correctness of the SXF/SXMF functionality
- Structural 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

The SXF or SMXF framework code will be eventually certified as part of the whole airborne software certification process.

## 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

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 design model against the high-elvel requirements including model coverage computation. Since the IBM Rational Rhapsody code generation translates a design model into source code it has to be verified that the translation is correct. Requirements based testing is the technique used to demonstrate the executable object code correctly implements the low-level and high-level requirements. Structural coverage metrics give evidence that the generated code does not contain untested code and the generated code is fully tested.

### 4.8.1 Requirements based testing of model and code

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 14).


**Figure 14: Different code generation configurations (MiL (Model in the Loop), SiL (Software in the Loop), and PiL (Processor in the Loop))**

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. Hence, SiL does not take into account the final hardware. PiL (Processor in the loop) is a code generation configuration that is used in order to generate source code for the target processor. Testing

on the target processor is a mandatory activity.  Hence, tt is mandatory to make sure that design model verification results from simulation runs are preserved when executing the source code generated for SiL or PiL configurations. If there are significant deviations in the behavior observed for MiL compared to e.g. PiL it means that the model does not behave as the source code on the target processor. If these deviations are not detected, errors might show up in the final production code although they are not observable during model simulation.

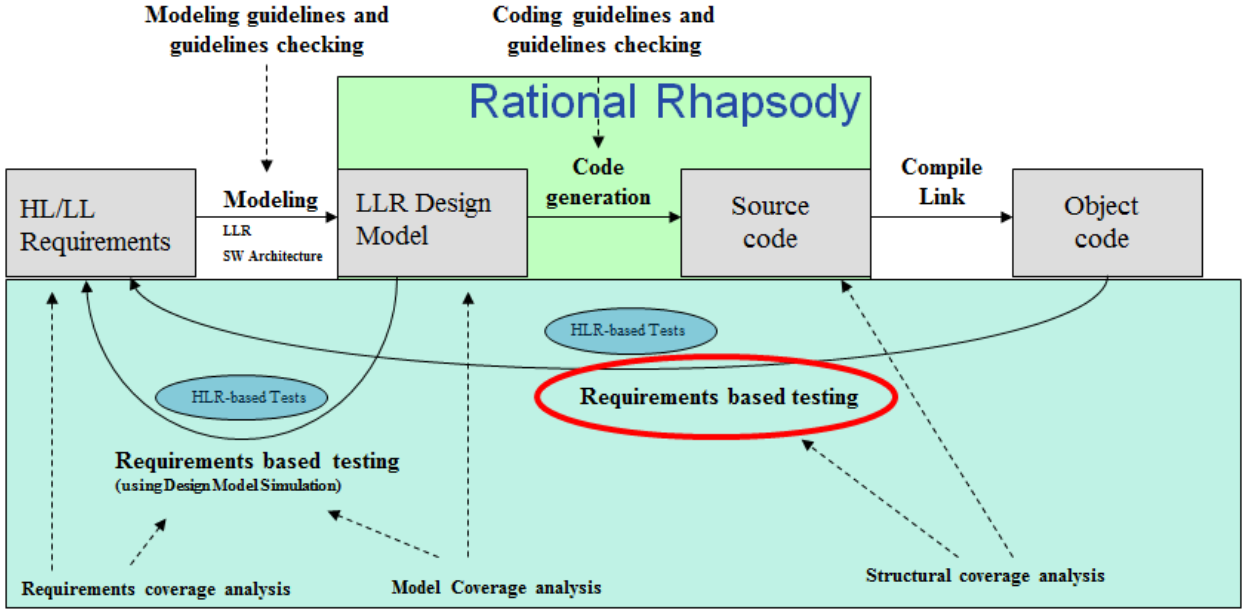


**Figure 15: Requirements based testing of model and executable object code**

In order to detect such deviations, the behavior of MiL configurations must be compared with the behavior of SiL or PiL configurations (cf. Figure 15). 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 requirements based testing activities (cf. Figure 15). More information about requirements based testing with IBM Rational Rhapsody TestConductor Add On can be found in (6).

### 4.8.2 Structural coverage

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

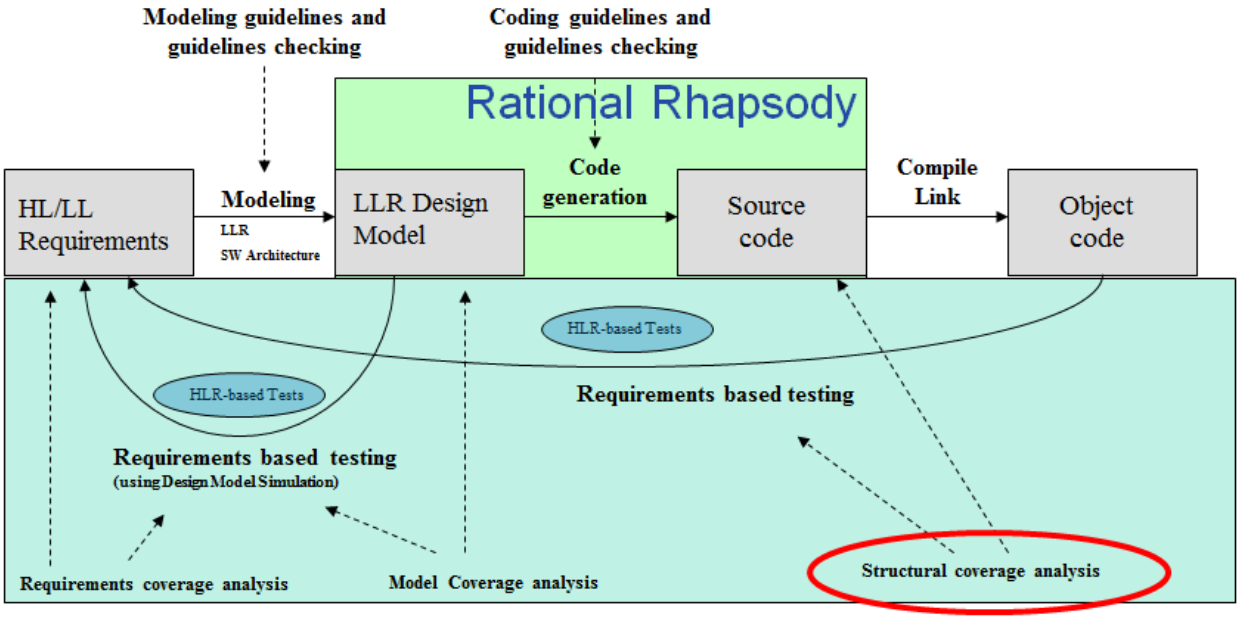


**Figure 16: Structural coverage**

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 16). In order to do that usually structural 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 structural coverage statistics for code execution runs. When using IBM Rational Rhapsody TestConductor Add On, structural coverage measurement (Statement-, Decision-, Condition-, MC/DC-, Function-Coverage)can be easily combined with the requirements based testing approach described in section 4.8.1. More information about structural coverage measurement with IBM Rational Rhapsody TestConductor Add On can be found in (6).

In DO-178C it is important to perform an analysis to confirm that requirements-based testing activities have exercised the data and control coupling between code components. 3rd party tools are available which can be integrated with Rhapsody to perform such automated analysis.

# 5 Mapping Reference Workflow Activities to Safety Standards

Figure 17 below shows the DO-178B/C software life cycle processes.

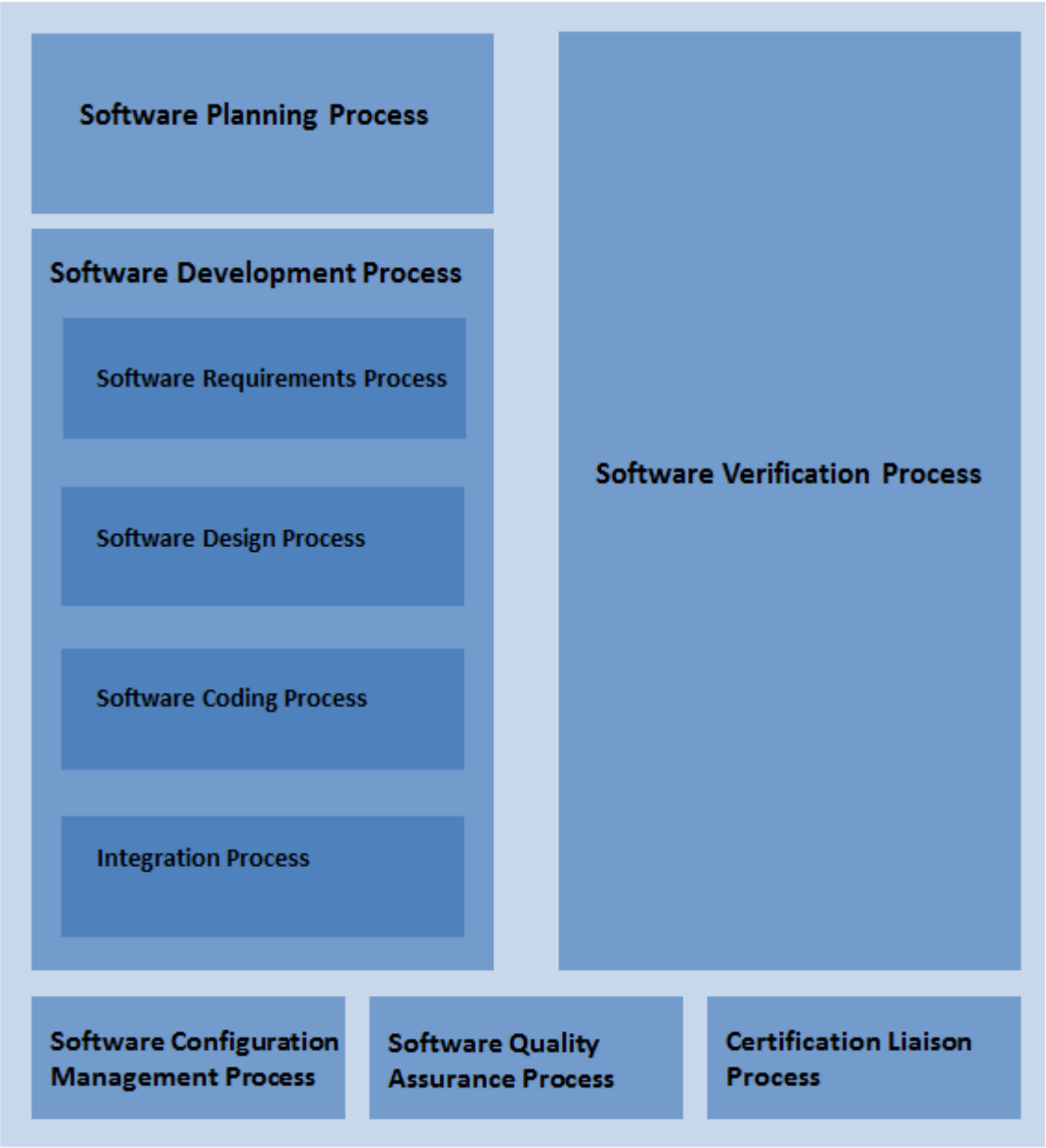


**Figure 17: Overview about DO-178B/C software life cycle processes**

## 5.1 DO-178C and DO-331

Figure 18 below provides an overview about the mapping between the DO-178C and DO-331 software development reference process phases to the workflow activities of the IBM Rational Rhapsody Reference Workflow. DO-178B is also covered by these tables. Objectives introduced with DO-331 are prefixed with 'MB'. For instance in Table A-2, objective MB-8.

Note: "n.A." means "not applicable"

| | Objective | | Applicability by Software Level | | | | | | |
|---|---|---|---|---|---|---|---|---|---|
| Software Development Subphase | Description | Ref. | A | B | C | D | Workflow Reference | Model Level | Code Level |
| Table A-1 Software Planning Process | 1) The activities of the software life cycle processes are defined. | 4.1a | ○ | ○ | ○ | ○ | n.A. | n.A. | n.A. |
| | 2) The software life cycle(s), including the inter-relationships between the processes, their sequencing, feedback mechanisms, and transition criteria, is | 4.1b | ○ | ○ | ○ | | n.A. | n.A. | n.A. |
| | 3) Software life cycle environment is selected and defined. | 4.1c | ○ | ○ | ○ | | n.A. | n.A. | n.A. |
| | 4) Additional considerations are addressed. | 4.1d | ○ | ○ | ○ | ○ | n.A. | n.A. | n.A. |
| | 5) Software development standards are defined. | 4.1e | ○ | ○ | ○ | | n.A. | n.A. | n.A. |
| | 6) Software plans comply with this document. | 4.1f | ○ | ○ | ○ | | n.A. | n.A. | n.A. |
| | 7) Development and revision of software plans are coordinated. | 4.1g | ○ | ○ | ○ | | n.A. | n.A. | n.A. |

| | Objective | | Applicability by Software Level | | | | | | |
|---|---|---|---|---|---|---|---|---|---|
| Software Development Subphase | Description | Ref. | A | B | C | D | Workflow Reference | Model Level | Code Level |
| Table A-2 Software Development Processes | 1) High-level requirements are developed. | 5.1.1a | ○ | ○ | ○ | ○ | n.A. | n.A. | n.A. |
| | 2) Derived high-level requirements are defined and provided to the system processes, including the system safety assessment process. | 5.1.1b | ○ | ○ | ○ | ○ | n.A. | n.A. | n.A. |
| | 3) Software architecture is developed | 5.2.1a | ○ | ○ | ○ | ○ | Modelling (Software architectural design; Section | • Using IBM Rational Rhapsody for creating a software architectural design model | n.A. |
| | 4) Low-level requirements are developed | 5.2.1a | ○ | ○ | ○ | | Modelling (Software architectural design; Section 4.3) | • Using IBM Rational Rhapsody for creating a software design model and/or software behaviour model | n.A. |
| | 5) Derived low-level requirements are defined and provided to the system processes, including the system safety assessment process. | 5.2.1b | ○ | ○ | ○ | | Modelling (Software architectural design; Section 4.3) | • Using IBM Rational Rhapsody for creating a software design model and/or software behaviour model | n.A. |
| | 6) Source Code is developed. | 5.3.1a | ○ | ○ | ○ | | Code Generation (Section 4.6) | • Using IBM Rational Rhapsody for creating a software design model and/or software behaviour model • Using Rhapsody code generation capabilities to generate code from low level | • IBM Rational Rhapsody provides many features supporting manual and automatic code generation including requirements into code capabilities |
| | 7) Executable Object Code and Adaptation Data Item Files, if any, are produced and loaded in the target computer. | 5.4.1a | ○ | ○ | ○ | ○ | n.A. | n.A. | n.A. |
| | MB 8) Specification Model elements that do not contribute to implementation or realization of any high-level requirement are identified. | MB.5.1.1.c | ○ | ○ | ○ | ○ | Modelling (Software architectural design; Section 4.3) | • Using IBM Rational Rhapsody for creating a specification model | n.A. |
| | MB 9) Design Model elements that do not contribute to implementation or realization of any software architecture are identified. | MB.5.2.1.c | ○ | ○ | ○ | ○ | Modelling (Software architectural design; Section 4.3) | • Using IBM Rational Rhapsody for creating a software design model and/or software behaviour model | n.A. |
| | MB 10) Design Model elements that do not contribute to implementation or realization of any low-level requirement are identified. | MB.5.2.1.c | ○ | ○ | ○ | | Modelling (Software architectural design; Section 4.3) | • Using IBM Rational Rhapsody for creating a software design model and/or software behaviour model | n.A. |

| | Objective | | Applicability by Software Level | | | | | | |
|---|---|---|---|---|---|---|---|---|---|
| **Software Development Subphase** | **Description** | **Ref.** | **A** | **B** | **C** | **D** | **Workflow Reference** | **Model Level** | **Code Level** |
| **Table A-3 Verification of Outputs of Software Requirements Process** | 1) High-level requirements comply with system requirements. | 6.3.1a | ● | ● | ○ | ○ | n.A. | n.A. | n.A. |
| | 2) High-level requirements are accurate and | 6.3.1b | ● | ● | ○ | ○ | n.A. | n.A. | n.A. |
| | 3) High-level requirements are compatible with target computer. | 6.3.1c | ○ | ○ | | | n.A. | n.A. | n.A. |
| | 4) High-level requirements are verifiable. | 6.3.1d | ○ | ○ | ○ | | n.A. | n.A. | n.A. |
| | 5) High-level requirements conform to standards. | 6.3.1e | ○ | ○ | ○ | | n.A. | n.A. | n.A. |
| | 6) High-level requirements are traceable to system requirements. | 6.3.1f | ○ | ○ | ○ | ○ | n.A. | n.A. | n.A. |
| | 7) Algorithms are accurate | 6.3.1g | ● | ● | ○ | | n.A. | n.A. | n.A. |
| | MB 8) Simulation cases are correct. | MB.6.8.3.2.a | ● | ○ | ○ | ○ | Requirements-based testing (Section 4.5.2) | • IBM Rational Rhapsody TestConductor AddOn provides all needed concepts for test case specification, execution and test management | n.A. |
| | MB 9) Simulation procedures are correct. | MB.6.8.3.2.b | ● | ○ | ○ | ○ | Requirements-based testing (Section 4.5.2) | • IBM Rational Rhapsody TestConductor AddOn provides all needed concepts for test case specification, execution and test management | n.A. |
| | MB 10) Simulation results are correct and discrepancies explained. | MB.6.8.3.2.c | ● | ○ | ○ | ○ | Requirements-based testing (Section 4.5.2) | • IBM Rational Rhapsody TestConductor AddOn provides all needed concepts for test case specification, execution and test management | n.A. |

| | Objective | | Applicability by Software Level | | | | | | |
|---|---|---|---|---|---|---|---|---|---|
| Software Development Subphase | Description | Ref. | A | B | C | D | Workflow Reference | Model Level | Code Level |
| **Table A-4 Verification of Outputs of Software Design Process** | 1) Low-level requirements comply with high-level requirements. | 6.3.2a | ● | ● | ○ | | n.A. | n.A. | n.A. |
| | 2) Low-level requirements are accurate and | 6.3.2b | ● | ● | ○ | | n.A. | n.A. | n.A. |
| | 3) Low-level requirements are compatible with target computer. | 6.3.2c | ○ | ○ | | | n.A. | n.A. | n.A. |
| | 4) Low-level requirements are verifiable. | 6.3.2d | ○ | ○ | | | n.A. | n.A. | n.A. |
| | 5) Low-level requirements conform to standards. | 6.3.2e | ○ | ○ | ○ | | Guidelines for modelling and coding and guideline checking (Section 4.4, 4.7) | • MISRA C: 2004 guidelines<br>• MISRA C++: 2008 guidelines<br>• IBM Rational Rhapsody: Enabling the generation of MISRA compliant code<br>• IBM Rational Rhapsody: SafetyCriticalForCDevelopers setting<br>• IBM Rational Rhapsody: SafetyCriticalForC++Developers setting<br>• IBM Rational Rhapsody: Checking the model | • MISRA C: 2004 guidelines • MISRA C++: 2008 guidelines<br>• IBM Rational Rhapsody: Simplified C execution framework (SMXF)<br>• IBM Rational Rhapsody: Simplified C++ execution framework (SXF)<br>• 3rd party tools for guideline checking on code level |
| | 6) Low-level requirements are traceable to high-level requirements. | 6.3.2f | ○ | ○ | ○ | | 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<br>• IBM Rational Rhapsody provides features supporting the process of verification and validation including traceability from requirements to model to code to test cases | n.A. |
| | 7) Algorithms are accurate. | 6.3.2g | ● | ● | ○ | | n.A. | n.A. | n.A. |
| | 8) Software architecture is compatible with high- | 6.3.3a | ● | ○ | ○ | | n.A. | n.A. | n.A. |
| | 9) Software architecture is consistent. | 6.3.3b | ● | ○ | ○ | | n.A. | n.A. | n.A. |
| | 10) Software architecture is compatible with target | 6.3.3c | ○ | ○ | | | n.A. | n.A. | n.A. |
| | 11) Software architecture is verifiable. | 6.3.3d | ○ | ○ | | | n.A. | n.A. | n.A. |
| | 12) Software architecture conforms to standards. | 6.3.3e | ○ | ○ | ○ | | n.A. | n.A. | n.A. |
| | 13) Software partitioning integrity is confirmed. | 6.3.3f | ● | ○ | ○ | ○ | n.A. | n.A. | n.A. |
| | MB 14) Simulation cases are correct. | MB.6.8.3.2.a | ● | ○ | ○ | | Requirements-based testing (Section 4.5.2) | • IBM Rational Rhapsody TestConductor AddOn provides all needed concepts for test case specification, execution and test management | n.A. |
| | MB 15) Simulation procedures are correct. | MB.6.8.3.2.b | ● | ○ | ○ | | Requirements-based testing (Section 4.5.2) | • IBM Rational Rhapsody TestConductor AddOn provides all needed concepts for test case specification, execution and test management | n.A. |
| | MB 16) Simulation results are correct and discrepancies explained. | MB.6.8.3.2.c | ● | ○ | ○ | | Requirements-based testing (Section 4.5.2) | • IBM Rational Rhapsody TestConductor AddOn provides all needed concepts for test case specification, execution and test management | n.A. |

| | Objective | | Applicability by Software Level | | | | | | |
|---|---|---|---|---|---|---|---|---|---|
| **Software Development Subphase** | **Description** | **Ref.** | **A** | **B** | **C** | **D** | **Workflow Reference** | **Model Level** | **Code Level** |
| **Table A-5 Verification of Outputs of Software Coding & Integration Processes** | 1) Source Code complies with low-level requirements. | 6.3.4a | ● | ● | ○ | | 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 |
| | 2) Source Code complies with software architecture. | 6.3.4b | ● | ○ | ○ | | 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 |
| | 3) Source Code is verifiable. | 6.3.4c | ○ | ○ | | | | | |
| | 4) Source Code conforms to standards. | 6.3.4d | ○ | ○ | ○ | | Guidelines for modelling and coding and guideline checking (Section 4.4, 4.7) | • MISRA C: 2004 guidelines<br>• MISRA C++: 2008 guidelines<br>• IBM Rational Rhapsody: Enabling the generation of MISRA compliant code<br>• IBM Rational Rhapsody: SafetyCriticalForCDevelopers setting<br>• IBM Rational Rhapsody: SafetyCriticalForC++Developers setting<br>• IBM Rational Rhapsody: Checking the model | • MISRA C: 2004 guidelines • MISRA C++: 2008 guidelines<br>• IBM Rational Rhapsody: Simplified C execution framework (SMXF)<br>• IBM Rational Rhapsody: Simplified C++ execution framework (SXF)<br>• 3rd party tools for guideline checking on code level |
| | 5) Source Code is traceable to low-level requirements. | 6.3.4e | ○ | ○ | ○ | | 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 |
| | 6) Source Code is accurate and consistent. | 6.3.4f | ● | ○ | ○ | | n.A. | n.A. | n.A. |
| | 7) Output of software integration process is complete and correct. | 6.3.5 | ○ | ○ | ○ | | n.A. | n.A. | n.A. |
| | 8) Adaptation Data Item File is correct and complete. | 6.6.a | ● | ● | ○ | ○ | n.A. | n.A. | n.A. |
| | 9) Verification of Adaptation Data Item File is achieved. | 6.6.b | ● | ● | ○ | | n.A. | n.A. | n.A. |

| | Objective | | Applicability by Software Level | | | | | | |
|---|---|---|---|---|---|---|---|---|---|
| **Software Development Subphase** | **Description** | **Ref.** | **A** | **B** | **C** | **D** | **Workflow Reference** | **Model Level** | **Code Level** |
| **Table A-6 Testing of Outputs of Integration Process** | 1) Executable Object Code complies with high-level requirements. | 6.4.5.b | ○ | ○ | ○ | ○ | Requirements-based testing (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 |
| | 2) Executable Object Code is robust with high-level requirements. | 6.4.5.c | ○ | ○ | ○ | ○ | | | |
| | 3) Executable Object Code complies with low-level requirements. | 6.4.4.a | ● | ● | ○ | | | | |
| | 4) Executable Object Code is robust with low-level requirements. | 6.4.4.b | ● | ○ | ○ | | | | |
| | 5) Executable Object Code is compatible with target computer. | 6.4.4.c | ○ | ○ | ○ | ○ | | | |

| | Objective | | Applicability by Software Level | | | | | | |
|---|---|---|---|---|---|---|---|---|---|
| Software Development Subphase | Description | Ref. | A | B | C | D | Workflow Reference | Model Level | Code Level |
| Table A-7 Verification of Verification Process Results | 1) Test procedures are correct. | 6.4.5.b | ● | ○ | ○ | | 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 |
| | 2) Test results are correct and discrepancies | 6.4.5.c | ● | ○ | ○ | | | | |
| | 3) Test coverage of high-level requirements is | 6.4.4.a | ● | ○ | ○ | ○ | | | |
| | 4) Test coverage of low-level requirements is achieved. | 6.4.4.b | ● | ○ | ○ | | | | |
| | 5) Test coverage of software structure (modified condition/decision) is achieved. | 6.4.4.c | ● | | | | 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 |
| | 6) Test coverage of software structure (decision coverage) is achieved. | 6.4.4.c | ● | ● | | | | | |
| | 7) Test coverage of software structure (statement coverage) is achieved. | 6.4.4.c | ● | ● | ○ | | | | |
| | 8) Test coverage of software structure (data coupling and control coupling) is | 6.4.4.d | ● | ● | ○ | | | | |
| | 9) Verification of additional code, that cannot be traced to Source Code, is achieved. | 6.4.4.c | ● | | | | n.A. | n.A. | n.A. |
| | MB 10) Simulation cases are correct | MB.6.8.3.2.a | ● | ○ | ○ | | Requirements-based testing (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 model level testing and code level testing |
| | MB 11) Simulation procedures are correct | MB.6.8.3.2.b | ● | ○ | ○ | | Requirements-based testing (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 model level testing and code level testing |
| | MB 12) Simulation results are correct and discrepancies are explained. | MB.6.8.3.2.c | ● | ○ | ○ | | Requirements-based testing (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 model level testing and code level testing |

| | Objective | | Applicability by Software Level | | | | | | |
|---|---|---|---|---|---|---|---|---|---|
| Software Development Subphase | Description | Ref. | A | B | C | D | Workflow Reference | Model Level | Code Level |
| Table A-8 Software Configuration Management Process | 1) Configuration items are identified. | 7.2.1 | ○ | ○ | ○ | ○ | n.A. | n.A. | n.A. |
| | 2) Baselines and traceability are established. | 7.2.2 | ○ | ○ | ○ | ○ | n.A. | n.A. | n.A. |
| | 3) Problem reporting, change control, change review, and configuration status accounting are established. | 7.2.3, 7.2.4, 7.2.5, 7.2.6 | ○ | ○ | ○ | ○ | n.A. | n.A. | n.A. |
| | 4) Archive, retrieval, and release are established. | 7.2.7 | ○ | ○ | ○ | ○ | n.A. | n.A. | n.A. |
| | 5) Software load control is established. | 7.2.8 | ○ | ○ | ○ | ○ | n.A. | n.A. | n.A. |
| | 6) Software life cycle environment control is established. | 7.2.9 | ○ | ○ | ○ | ○ | n.A. | n.A. | n.A. |

| | Objective | | Applicability by Software Level | | | | | | |
|---|---|---|---|---|---|---|---|---|---|
| Software Development Subphase | Description | Ref. | A | B | C | D | Workflow Reference | Model Level | Code Level |
| Table A-9 Software Quality Assurance Process | 1) Assurance is obtained that software plans and standards are developed and reviewed for compliance with DO-278A and this Supplement and for | 8.1.a | ● | ● | ● | | n.A. | n.A. | n.A. |
| | 2) Assurance is obtained that software life cycle processes comply with approved software plans. | 8.1.b | ● | ● | ● | ● | n.A. | n.A. | n.A. |
| | 3) Assurance is obtained that software life cycle processes comply with approved software standards. | 8.1.b | ● | ● | ● | | n.A. | n.A. | n.A. |
| | 4) Assurance is obtained that transition criteria for the software life cycle processes are | 8.1.c | ● | ● | ● | | n.A. | n.A. | n.A. |
| | Assurance is obtained that software conformity review is conducted. | 8.1.d | ● | ● | ● | ● | n.A. | n.A. | n.A. |

| | Objective | | Applicability by Software Level | | | | | | |
|---|---|---|---|---|---|---|---|---|---|
| Software Development Subphase | Description | Ref. | A | B | C | D | Workflow Reference | Model Level | Code Level |
| Table A-10 Software Approval Process | 1) Communication and understanding between the applicant and the approval authority is established. | 9.a | ○ | ○ | ○ | ○ | n.A. | n.A. | n.A. |
| | 2) The means of compliance is proposed and agreement with the Plan for Software Aspects of Approval is | 9.b | ○ | ○ | ○ | ○ | n.A. | n.A. | n.A. |
| | 3) Compliance substantiation is provided. | 9.c | ○ | ○ | ○ | ○ | n.A. | n.A. | n.A. |

**Figure 18: DO-178C mapping to the Rhapsody Reference Workflow**

# Appendix A: List of Figures

## Appendix B: List of References


1. Software Considerations in Airborne Systems and Equipment Certification*, RTCA Inc., RTCA DO-178B.* 1992.

2. Software Considerations in Airborne Systems and Equipment Certification*, RTCA Inc., RTCA DO-178C.* 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 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. Software Tool Qualification Considerations*, RTCA Inc., RTCA DO-330.* 2011.

10. Model-Based Development and Verification – Supplement to DO-178C and DO-278A*, RTCA Inc., RTCA DO-331.* 2011.

11. Oject-Oriented Technology and Related Techniques – Supplement to DO-178C and DO-278A*, RTCA Inc., RTCA DO-332.* 2011.

12. Formal Methods – Supplement to DO-178C and DO-278A*, RTCA Inc., RTCA DO-333.* 2011.

13*. IBM Rational Rhapsody TestConductor Add On Qualification Kit for DO-178B/C Overview.*