Model-Driven Development for Safety Critical Software

Giulio Santoli (giulio_santoli@it.ibm.com)
Client Technical Professional, IBM Rational

Agenda

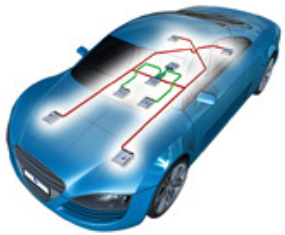
1. Safety Critical Application Standards
2. Integrated Software Development Process
3. Model-Driven Development for Safety Critical
4. Rational Rhapsody Enhancements for Safety Critical
5. Rhapsody TestConductor AddOd Qualification Kit
6. Summary

Standards for Safety Critical Applications

A **Safety Critical System** is a system whose failure or malfunction may result in serious injury or even death to people.

Some Safety Critical Standards:

- **IEC 61508,** Functional Safety Standard
- **DO-178B/C**, Aerospace and Defense
- **ISO 26262**, Automotive
- **EN 50128**, Rail
- **IEC 60601 & 62304**, Medical



Increase of Software in Aerospace & Defence

- Complexity in modern systems requires **more software**
- Technology enhancements and project contraints make aerospace Industries to adopt **new processes** and **programming languages** (F-35 uses C++)

F-35 Lightning

First adopting "relaxed static stability"

F-4 Phantom

F-16 Falcon

F-22 Raptor

1960'

Only 8% of requirements required software (assembly) in the F-4

1970'

45% of requirements required software (JOVIAL) in the F-16

1980'

80% of specification requirements required software (Ada83) in the F-22

2000'

F-35 has 24 million lines of code (C/C++), vs 1.7 million lines of code for F-22

Example: RTCA DO-178B

- RTCA DO-178B is an **objective-based** standard applied by FAA (Federal Aviation Administration) for the certification of software in avionics systems.
- Published in 1992, it covers the **5 main processes** concerning Planning, Development, Verification, Configuration Management and Quality Assurance.
- DO-178B outlines the **objectives** to be met, the work **activities** to be performed for each objective, and the **evidence** (output documents) to be supplied for each objective (based on criticality level A-E)

Software Criticality Level	Failure Condition Category	Failure Condition Description	Objectives
Level A	Catastrophic	Conditions which would prevent continued safe flight and landing.	66
Level B	Haazardous/ Sever-Major	Conditions which would reduce aircraft safety margin/functional capabilities, produce a higher workload to the flight crew or have serious adverse effencts on occupants	65
Level C	Major	Conditions which would not significantly reduce aircraft safety, crew ability to work under adveser operation or produce discomfort to occupants.	57
Level D	Minor	Conditions which would not significantly reduce aircraft safety, slight increas in crew workload or produce some inconvenience to occupants	28
Level E	No Effect	Conditions which do not affect the aircraft operations or crew workload.	0

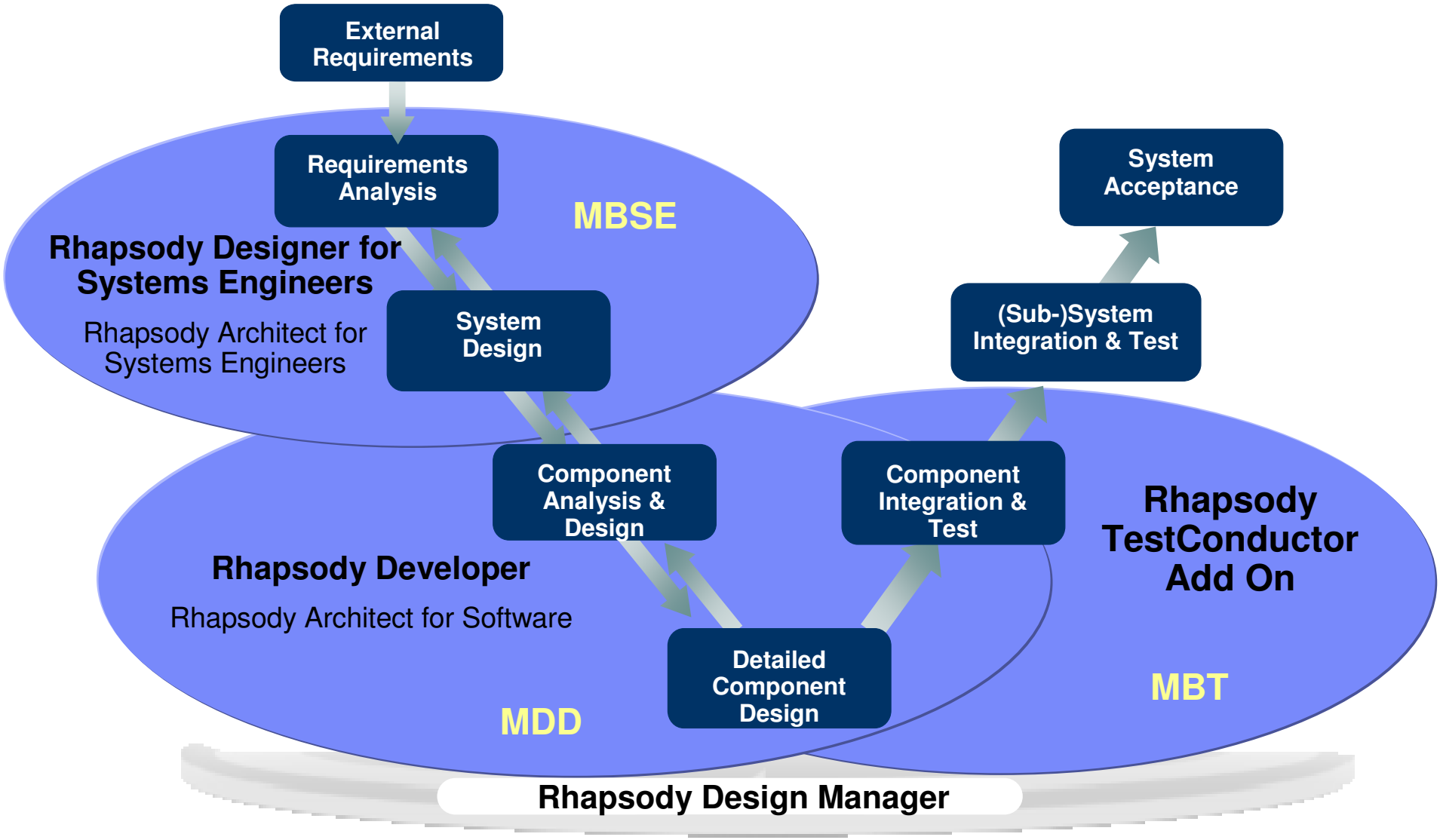
Integrated Software Process for DO-178B

- The **Integrated Software Development Process for DO-178B (ISDP-178)** is a set of practices to help organizations developing products for certification under DO-178B
  - Specifies a large number of modern sw engineering best practices, including MDD and MBT
- The process may be applied to any appropriate development tooling but is specifically optimized for the Rational System Accelerator consisting of tools
  - **Rational Team Concert** for project planning, enactment, and tracking, incl. CM
  - **Rational DOORS** for requirements management
  - **Rational Rhapsody** for system engineering, safety analysis, software design & development
  - **Rational Quality Manager** for test specification, execution, and analysis
  - **Rational Method Composer** for process customization
- The ISDP-178 address three primary needs
  - Process specification
  - Process enactment
  - Specific links from the DO-178B standard to process content to aid in ensuring compliance
    - By Objective
    - By Certification Level
    - By Work Product

ISDP-178 Process Description

More Processes Available:
Harmony/SE
Harmony/ESW
ISO 26262

Model-Driven Development positioning in the V-Process



Model-Driven Development in Safety Critical Development

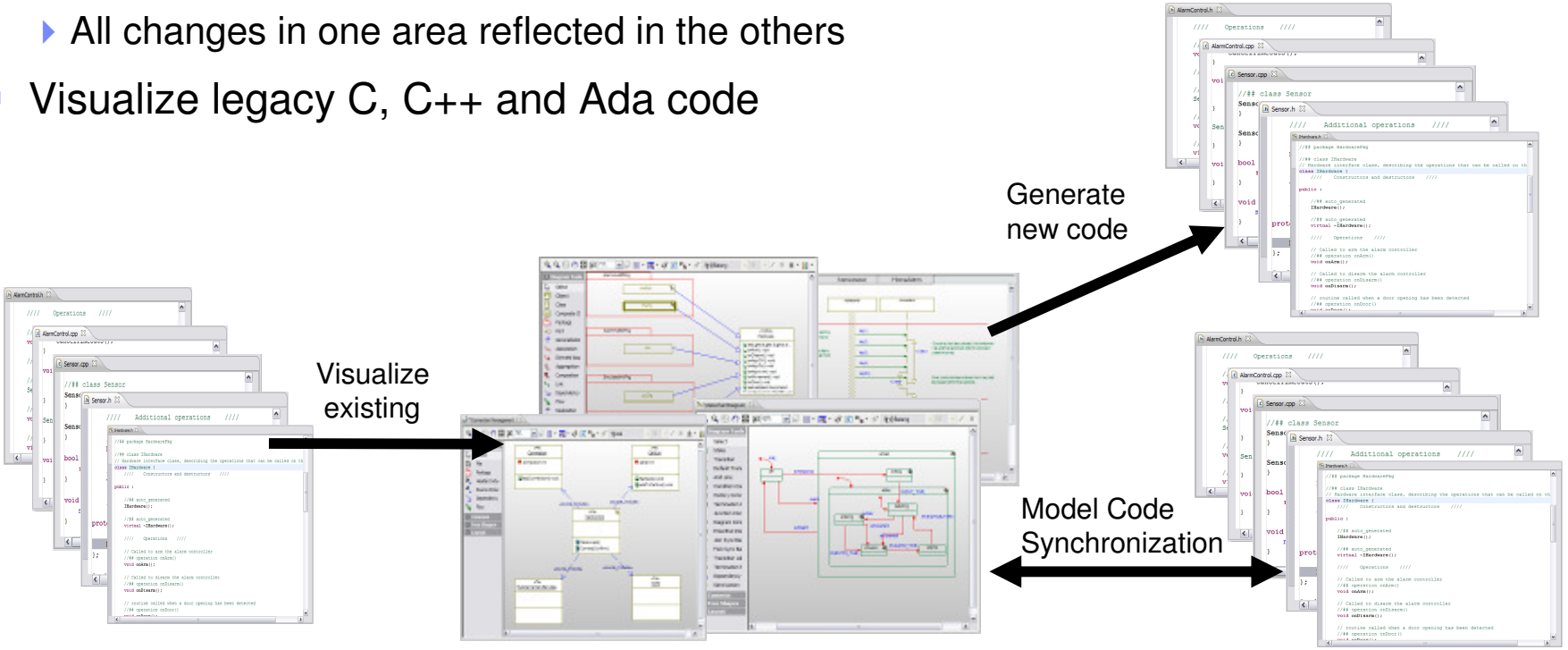
- Adopting MDD you can increase productivity and code quality
- Rhapsody provides many MDD technoligies:
  - Production Code Generation (80%-90%)
  - Model/Code Associativity (aka Roundtripping)
  - Model Checking
  - Model Helpers and Transformations via Rhapsody API and Rhapsody RulesComposer

Average Developer time lost to delays in design completion

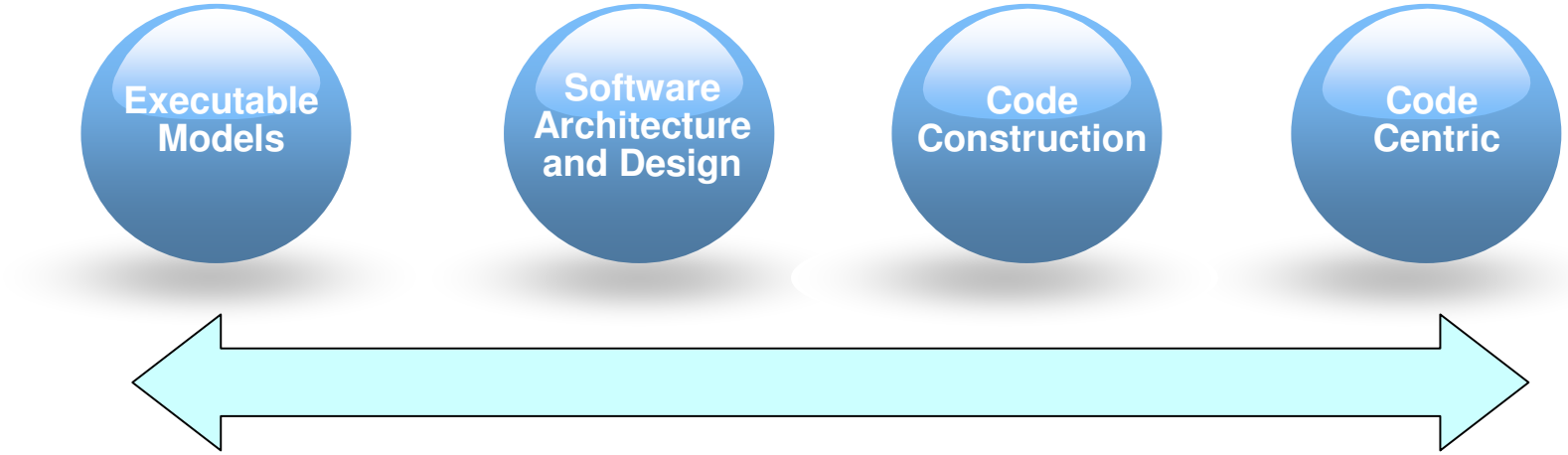
Source: *2011 EMF (Embedded Market Forecasters) Study*

Model-Driven Development Approaches

- Generate new code from the model
  - Develop MISRA-C, MISRA-C++ and Ada applications
- Maintain automated synchronization between model and code
  - Work simultaneously with architecture, software and target
  - All changes in one area reflected in the others
- Visualize legacy C, C++ and Ada code



Different Modeling Paradigms: Code-Centric or Model-Centric



Model-is-code

From the model

Code is “black box”

One-way development flow

Using implementation language

Generating readable code

Open framework

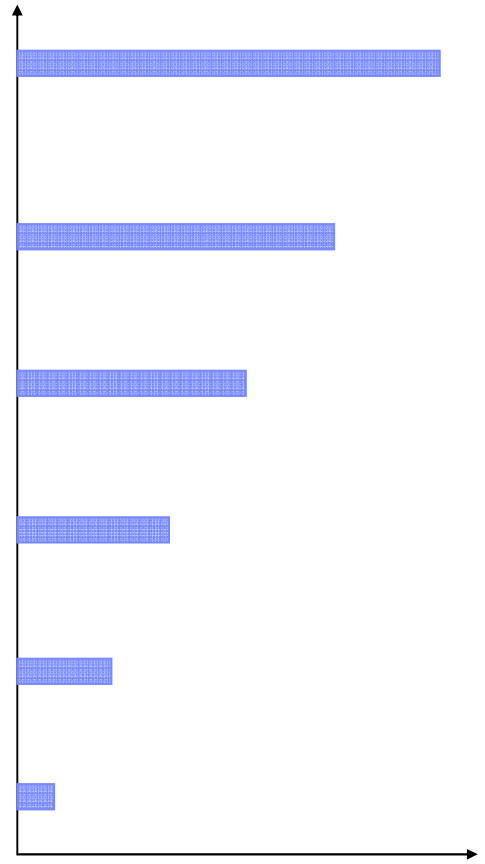
Model-Code Associativty

Model-Code co-Debugging

Code is the master

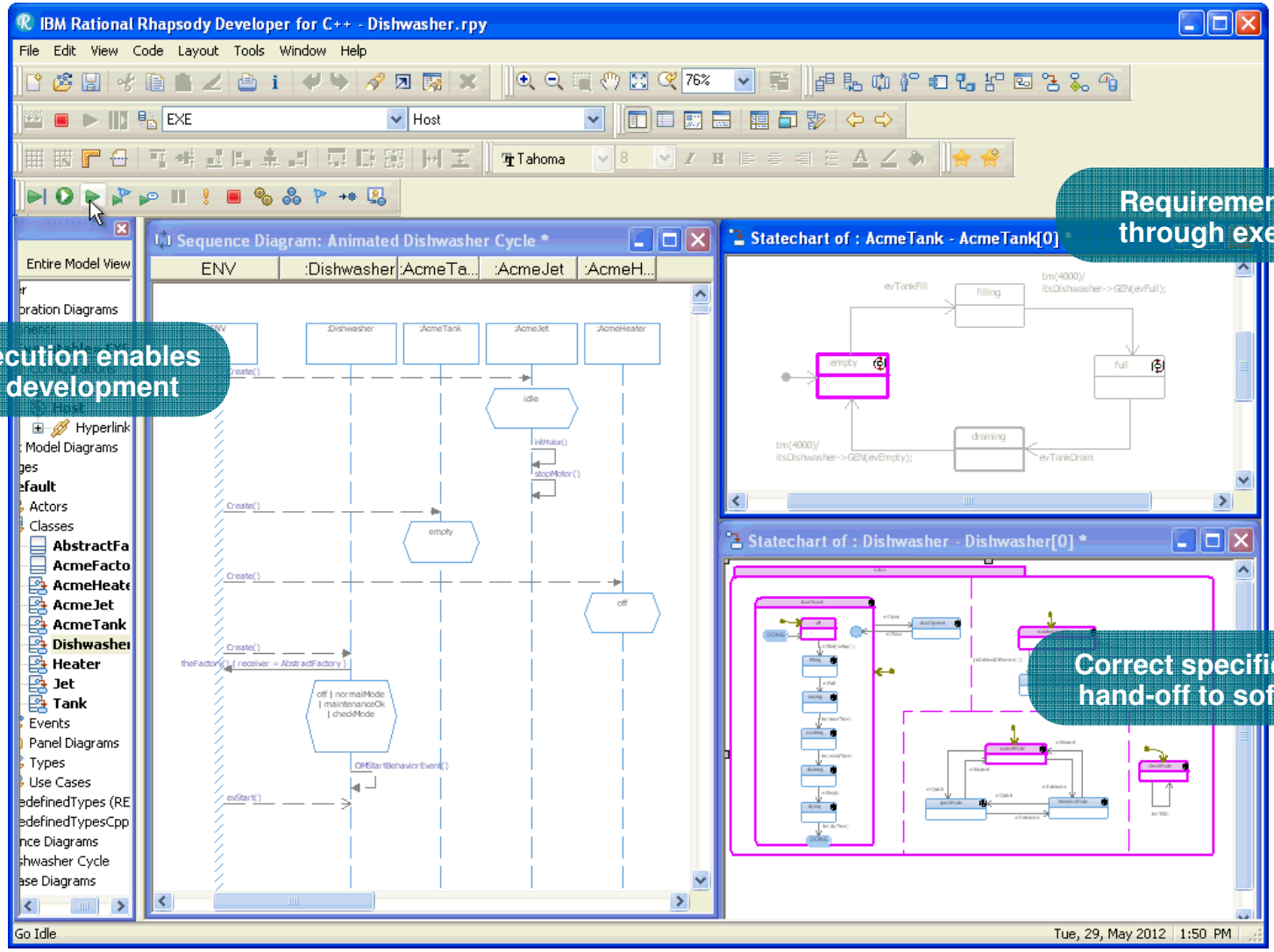
Everything is done in the code and should stay exactly as-is

UMMI – UML Modeling Maturity Index (By Bruce Douglass)

Level	Benefit	Focus	Technologies	Result
5 Optimizing	100%	Agile and Engineering Best Practices	Model-based testing, nanocycle execution, test driven development, continuous integration	Productivity and Quality 
4 Executing	70%	Model-based verification	Model execution, code generation, model-based debugging	
3 Behavioral Modeling	30%	State and algorithmic modeling	State, sequence and activity diagrams	
2 Structural Modeling	15%	Class and block modeling of structure	Class and block diagrams	
1 Visualization	5%	Visualizing code structures	Reverse engineering	
0 Code Based Development	0%	Manual, time intensive heroic development		

Model-Driven Development with IBM Rational Rhapsody

Find errors early in the process with advanced model execution



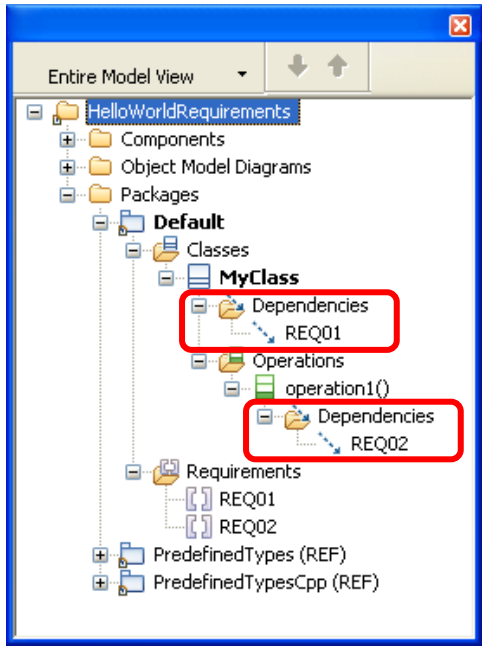
Requirements test through execution

Model execution enables iterative development

Correct specification hand-off to software

Requirements Traceability into Generated Code

- In Rhapsody, you have always been able to link classes and operations to requirements and have them included into the generated code:




```
//## class MyClass
// Realizes requirement REQ01 #:
// This is a requirement example number one
class MyClass {
    ////    Constructors and destructors    ////

public :

    //## auto_generated
    MyClass();

    //## auto_generated
    ~MyClass();

    ////    Operations    ////

    // Realizes requirement REQ02 #:
    // This is an example requirement number 2
    //## operation operation1()
    void operation1();
};
```

- But additional granularity was missing to link statechart elements (states and transitions) to requirements.

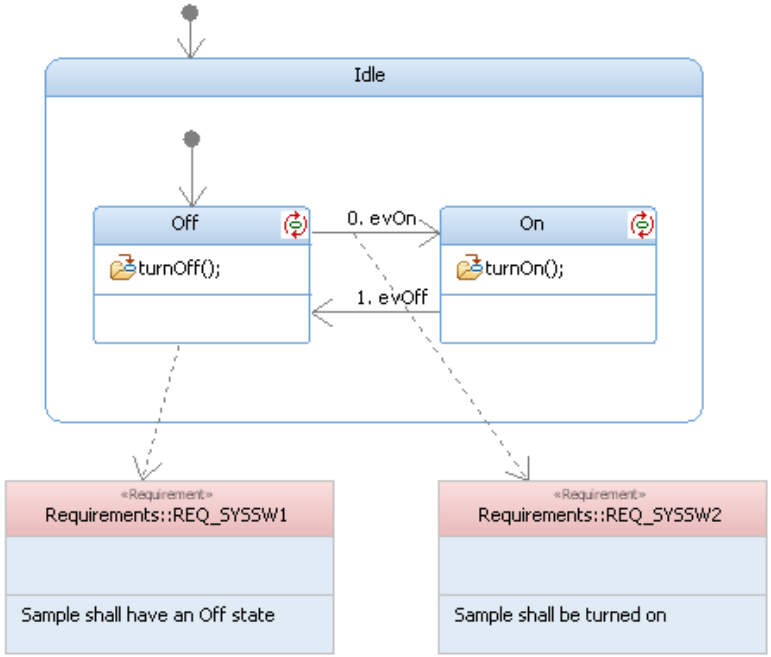
Main Enhancemnts for MDD in Rhapsody 8.0

- "High-Level" and "Low-Level" Requirements stereotypes
- Better Treacability from Statechart to Code
  - Distinguish between HLR and LLR with new stereotypes
  - Improved location in code for Transitions and States' Requirements
  - Improved mapping-back of Statechart code to the specific model element
  - Generate Requirements associated with Entry/Exit Action and Internal Transition
  - Associate Statechart's auto-generated code with its justification
  - Ability to generate Requirement on Operation to implementation file as well
- Safety-Critical Frameworks for Rational Rhapsody
  - C: SMXF (Simplified MicroC eXecution Framework)
  - C++: SXF (Simplified C++ eXecution Framework)
- Rhapsody TestConductor Qualification Kit for ISO 26262 and IEC 61508

Improved location of Requirements for Transitions/States

- If you have Requirements that are met by specific States or Transitions in a Statechart, they can be included as comments in the generated code.

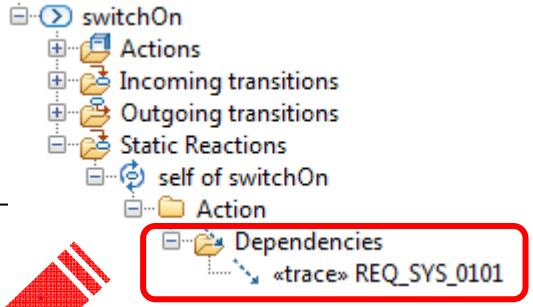
```
switch (rootState_active) {
    // State Off
    // Realizes requirement REQ_SYSSW1:
    // Sample shall have an Off state
    case Off:
    {
        // Realizes requirement REQ_SYSSW2:
        // Sample shall be turned on
        if(IS_EVENT_TYPE_OF_evOn_Default_id))
        {
            ...
        }
    }
    break;
    // State On
    case On:
    {
        ...
    }
    break;
    default:
        // Realizes requirement SwitchStatementDefaultClause #SCR1.36:
        // The default clause shall be generated for a switch statement;
        break;
}
return res;
```



This is a requirement for autogenerated code as discussed later

Generate code for Requirements of Internal Transition

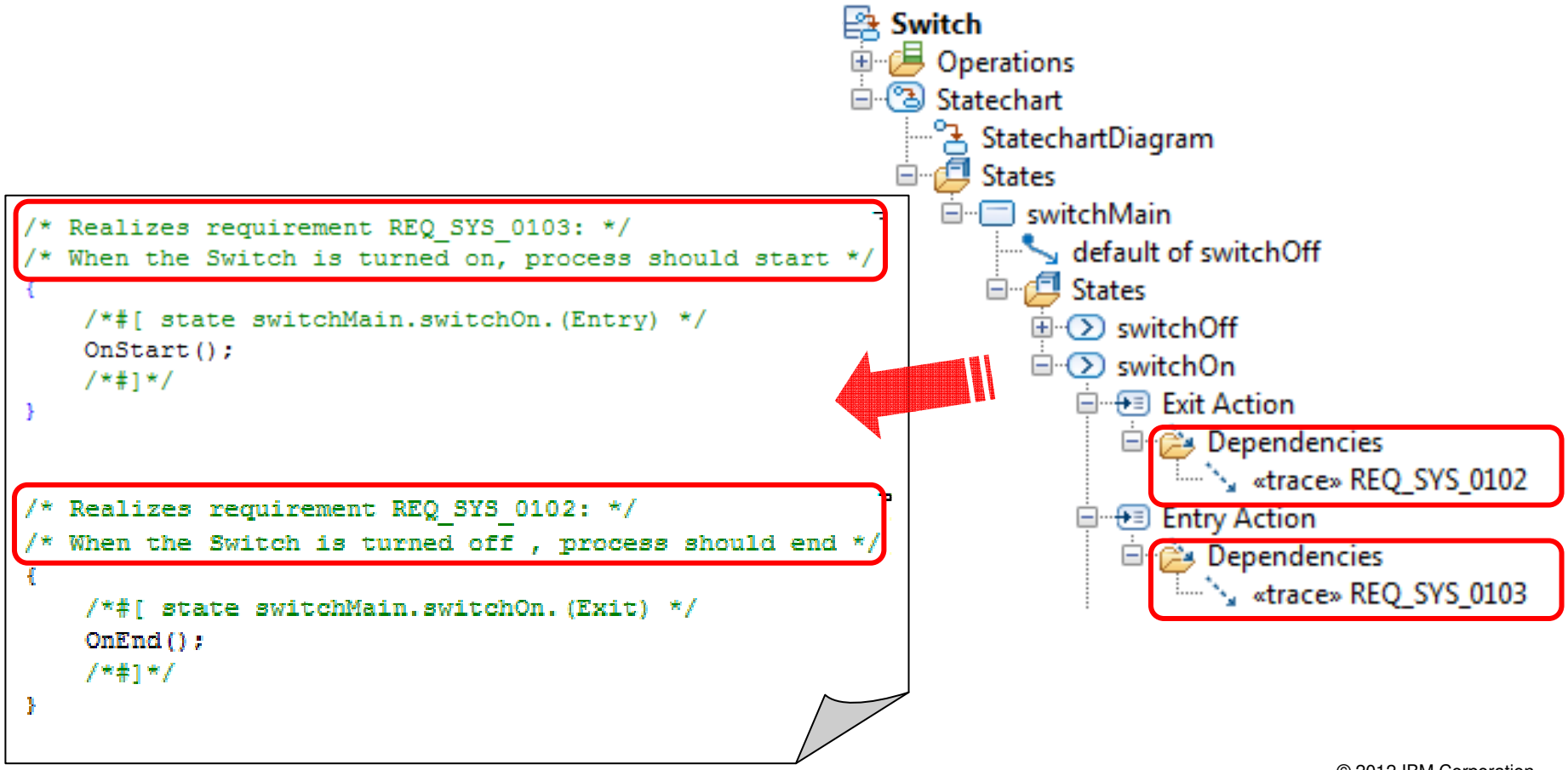
- Rhapsody supports Requirements on Internal Transition and on its Action (if any)
  - Generate Requirements into code, using the transition trigger for mapping back to model
  - You can associate Requirements in the Browser




```
/* State switchOn */
/* Description: myDescription */
case Switch_switchOn:
{
    switch (id) {
        case Timeout_id:
        {
            if(RiCTimeout_getTimeoutId(me->ric_reactive.current_event) == Switch_Timeout_switchOn_id)
                {
                    /* Realizes requirement REQ_SYS_0101: */
                    /* Indication of switch on should be green light blinking */
                    {
                        /*#[ transition 0 */
                        GreenBlink();
                        /*#]*/
                    }
                    res = eventConsumed;
                }
        }
        break;
```

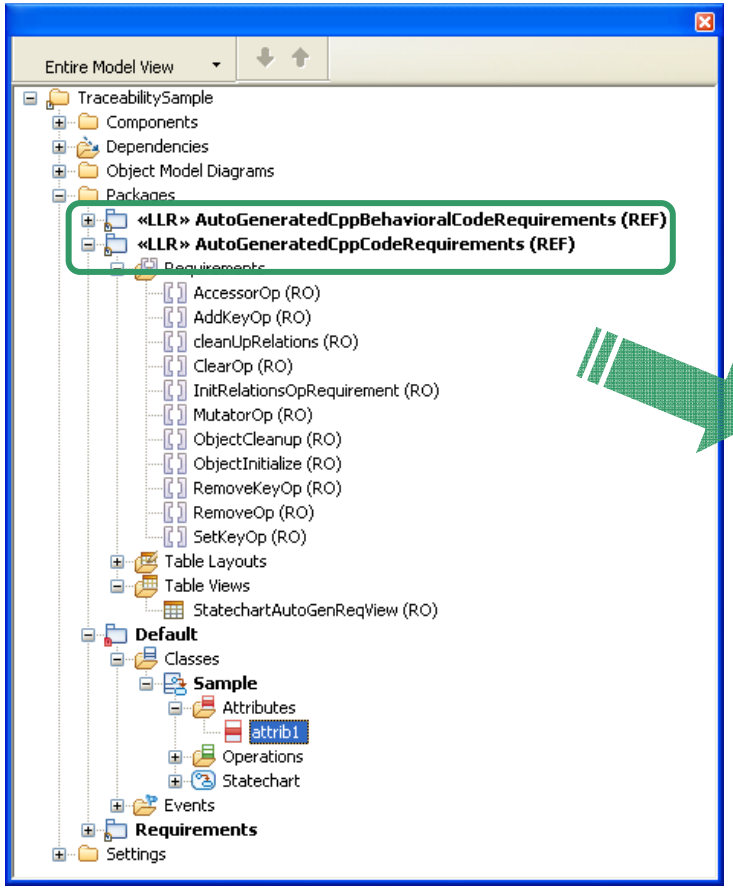
Requirements for Entry/Exit Actions

- Rhapsody supports Requirements on Entry Action / Exit Action
  - You can associate Requirements in the Browser
  - Rhapsody generates the Requirements in the Generated Code



Requirements Justification for Autogenerated Code

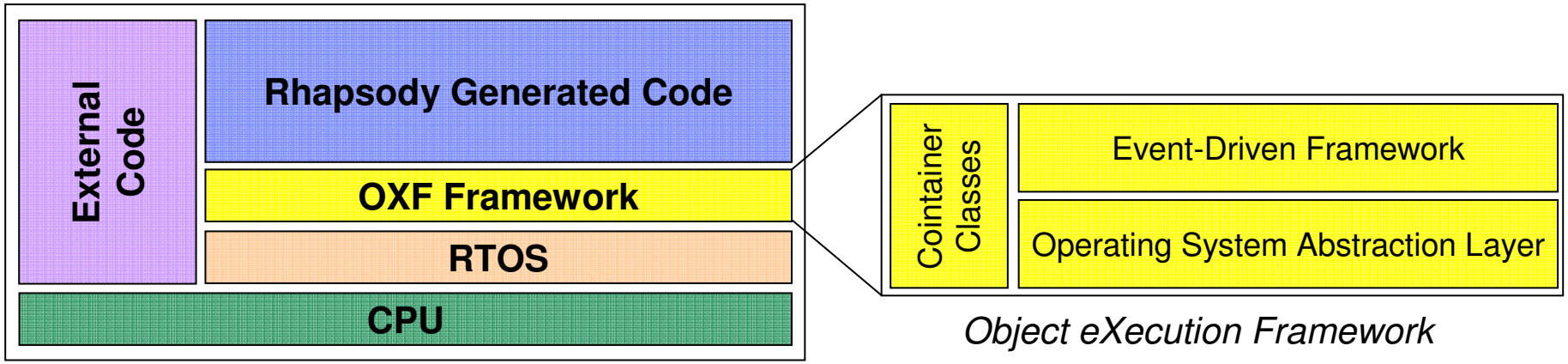
- Rhapsody can include Requirements to **justify autogenerated code**, such as accessor and mutator operations of an attribute.




```
// Realizes requirement ObjectInitialize #LR1.05
//## auto_generated
Sample(IOxfActive* theActiveContext = 0);

// Realizes requirement ObjectCleanup #LR1.04
//## auto_generated
~Sample();

// Realizes requirement AccessorOp #LR1.06
//## auto_generated
int getAttrib1() const;

// Realizes requirement ReactiveStartBehavior #SCR1.23
//## auto_generated
virtual bool startBehavior();
```

How Rhapsody implements Behavioral Diagrams

- Rhaposdy implements behavioral diagrams by leveraging a framework of base classes and interfaces.
- There are two main parts to this framework:
  - The **Object eXecution Framework** (OXF), which is the part of the framework that is always linked into the final generated code.
  - The **Animation** and **Tracing Framework**, which is only used when animating or tracing.
  - The OXF is provided for each supported language (**C, C++, Java, Ada, C#**), with different flavors (interrupted-driven, static memory only, etc..)



Simplified MicroC Framework (SMXF) Overview

- SMXF is an execution framework optimized for MISRA compliant real-time C applications generated from Rhapsody Models
- Full static/compile-time architecture
  - Support only compile-time initialization, Support segmented memory (allocation to memory banks)
- MISRA-C 1996/2004 compliance
- Supporting the Extended Execution Model
  - Periodic Execution
  - Execution Manager, Runnable Manager
  - Events (Asynchronous events, Synchronous events, Timeouts)
- Adapters
  - ARINC 653 (APEX API based)
  - “Mainloop” - self scheduling executive

Comparison of C eXecution Frameworks

Purpose	Standard C Object Execution Framework (OXF)	MicroC eXecution Framework (MXF)	Simplified MicroC eXecution Framework (SMXF)	Interrupt- Driven Framework (IDF)
Supports statecharts	Y	Y	Y	Y
Supports asynchronous messaging	Y	Y	Y	Y
Supports synchronous messaging	Y	Y	Y	Y
Timers (time events)	Y	Y	Y	Y
UML ports	Y	Flow ports	N	N
Deterministic	N	Y	Y	Y
Periodic scheduling	N	Y	Y	N
Multi-thread support	Y	Y	Y	N
Supports multiple event queues	Y	Y	Y	Y
Resource protection	Y	Y	Y	N
Requires an OS?	Y	N	N	N
Can be used with an OS?	Y	Y	Y	Y
Defines own Container Classes	Y	Configurable	N	Y
Defines a Memory Manager	Y	Configurable	N	N
Error manager / notifier	N	N	N	Y
Static memory allocation	N (property settings or user-defined)	Y	Y	Y
Animation	Y	Y	N	N
Tracing	Y	Y	N	N
Simulated time model	Y	N	N	N
Size	~15000 LOC	~10000 LOC	~5000 LOC	~2500 LOC
MISRA C compliance	N	Med-high compliance	High compliance	High compliance

Simplified C++ eXecution Framework (SXF)

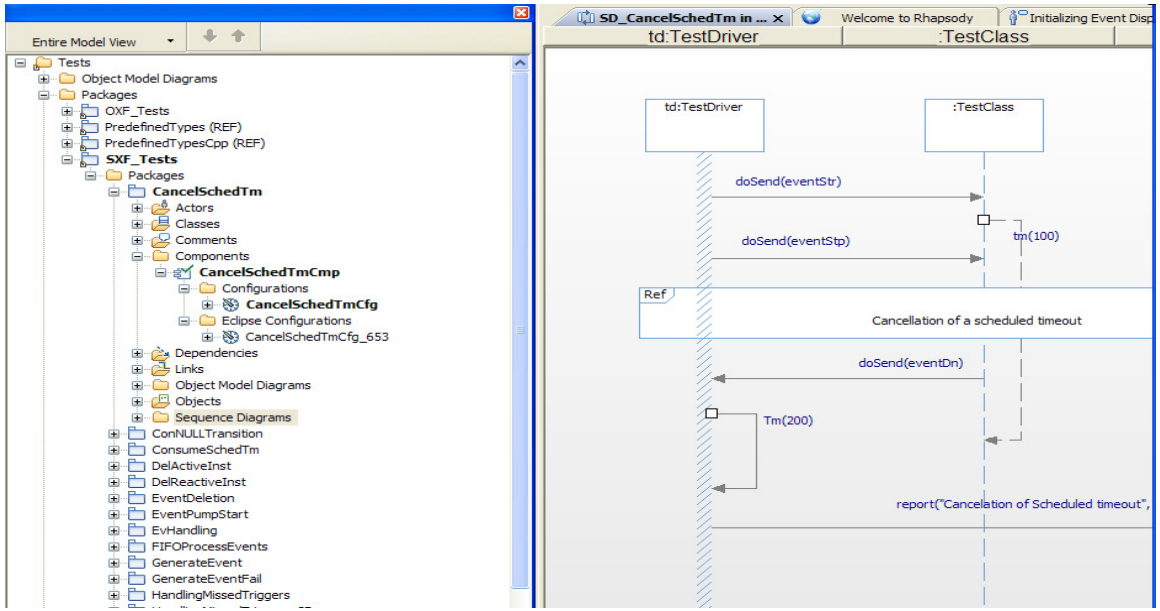
- Based on IDF with support for Active classes (multi threading)
- Static architecture
  - Static memory manager for events allocation
- MISRA C++ 2008 compliance
  - Safety critical C++ settings
  - Checks to support MISRA compliant modelling style
- Events
  - Asynchronous events, Synchronous events (triggered operations), Timeouts
- Adapters
  - Workbench Managed 653 (APEX API based)
  - Microsoft (VS 2008/2010)
- Constraints
  - Flat state charts
  - No Ports
  - No Animation/Tracing

Comparison of SXF and OXF for C++

SXF C++	OXF C++
Static architecture	Dynamic allocation
No animation/tracing	Animation/Tracing
Only Real Time	Real Time/Simulated Time
No containers (can be added)	Containers
Static memory manager (only BasedNumberOfInstances)	Static memory manager
Flat state charts	Flat/Reusable state charts
No Multi core in 7.6	Multi core
No Interfaces	Interface based
No Ports	Ports

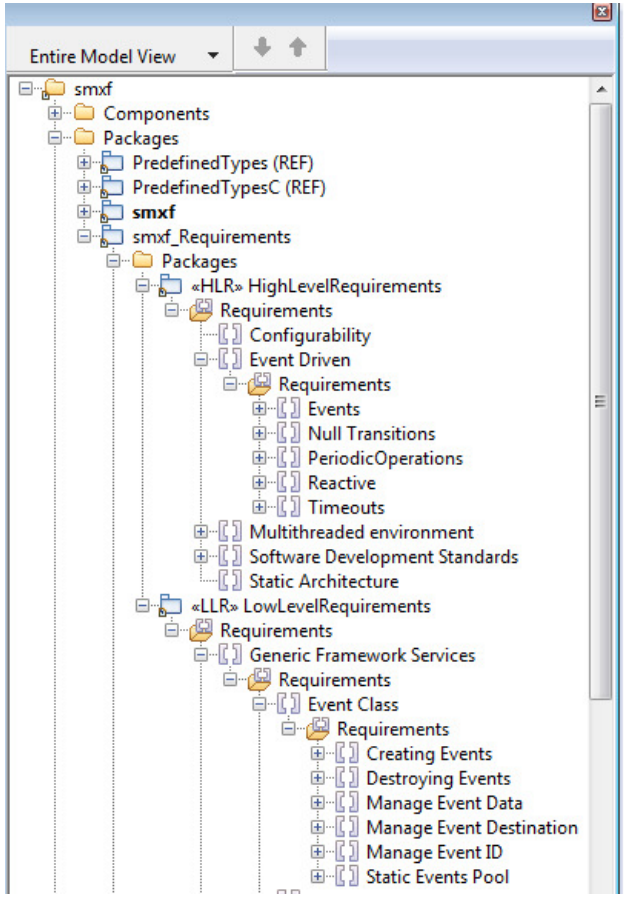
Certification Supporting Materials for SMXF and SXF

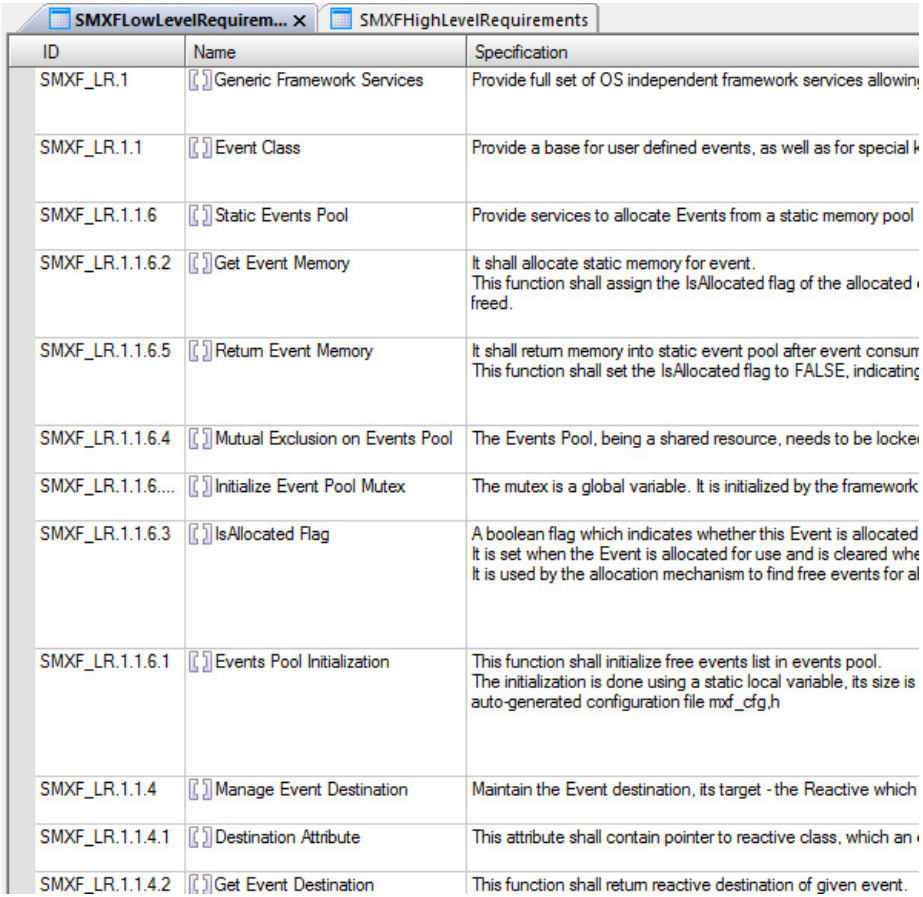
- Associated High Level (HLR) and Low Level (LLR) Requirements provided
- Trace back from code to requirements
  - Fully justified code
- Test Cases provided using Rational Rhapsody TestConductor Add On
  - Test Cases trace back to requirements
  - Statement coverage
  - Branch coverage



SMXF High Level (HLR) and Low Level (LLR) Requirements

- Both SXF and SMXF Models include coverage to High and Low Level Requirements





SMXF and SXF Test Reports

- Also the Test Reports generated from the SMXF and SXF frameworks are provided or can be re-gererated:
  - Code Coverage Report
  - Requirements Coverage Report
  - MISRA/MISRA-C++ Compliancy Statement
  - MISRA/MISRA-C++ LDRA Testbed Report

	Quality Result	Uniq Violations % in Function	No in Function	Breakdown of Violations
SXF (Set)	Pass	12 Files | 145 Functions		
OMOSSpecific.cpp	Pass	0		
OXF.cpp	Pass	0		
OMTimeoutPool.cpp	Pass	0		
OMTimeout.cpp	Pass	0		
OMThread.cpp	Pass	0		
OMStartBehaviorEvent.cpp	Pass	0		
OMReactive.cpp	Pass	0		
OMProtected.cpp	Pass	0		
OMMainThread.cpp	Pass	0		
OMGuard.cpp	Pass	0		
OMEventQueue.cpp	Pass	0		
OMEvent.cpp	Pass	0		

LDRA Static Analysis Report

Function Coverage

SMXF Coverage

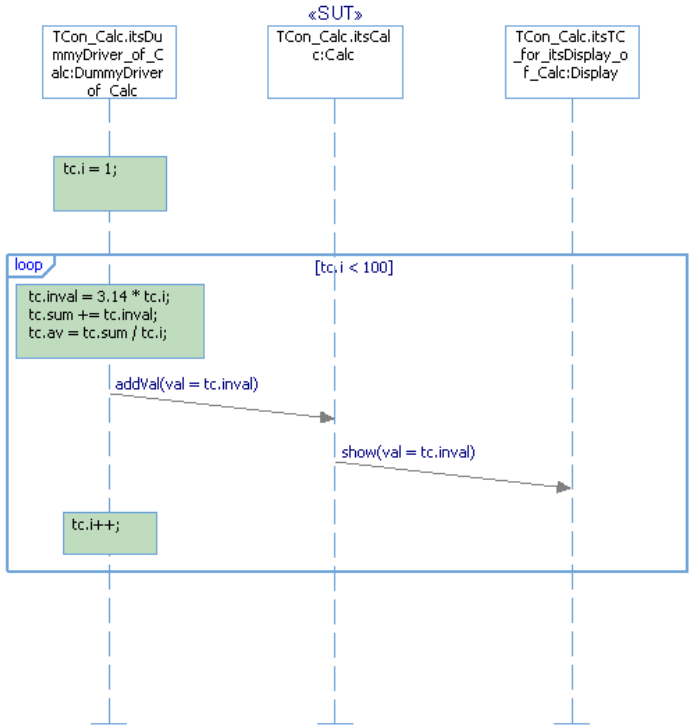
Coverage Goals		
total	136	
covered	136	100%
unknown	0	0%

Coverage Items (1 Goal)		
total	136	
covered (completely)	136	100%
covered (partially)	0	0%
uncovered	0	0%

Detailed Coverage Results

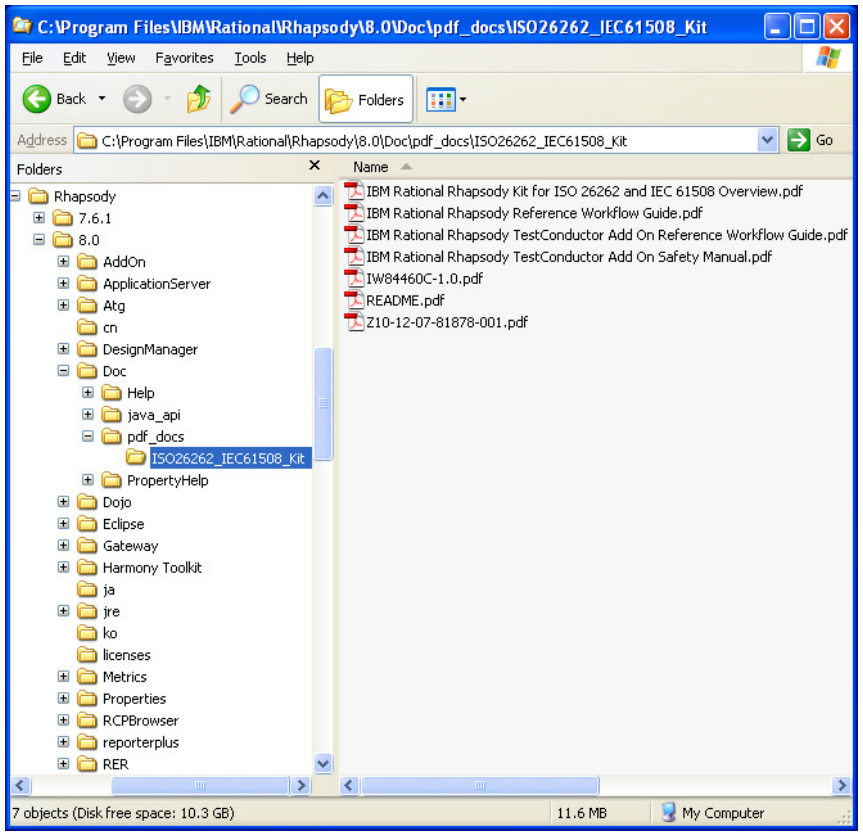
Rhapsody TestConductor Add-on for Model-Based Testing

- Define test cases with sequence diagrams, statecharts, flowcharts or even code
  - OMG UML Testing Profile
- Automate testing tasks
  - Create Test Architecture
  - Execute and monitor tests
    - Interactive for debugging,
    - Batch test suites for nightly regression
    - Include CUnit/CppUnit tests
- Traceability across lifecycle – from requirements to integration
- Host level and target based execution
  - White-Box, Black-Box for design validation
  - “Offline testing” mode:–for testing on target
  - C++, C, Java, Ada Supported
- Definition and management of regression tests
- Reporting of results, coverage and traceability



Rhapsody Kit for ISO 26262 and IEC 61508

- **Overview Doc**: describes the contents of the Rhapsody kit
- **Rhapsody Reference workflow**: provides an exemplary workflow for modelling, code generation and verification in safety critical
- **Rhapsody TestConductor Add On Workflow**: describes testing activities and objectives
- **Rhapsody TestConductor Safety Manual**: provides additional information for using TestConductor in safety related applications
- **TÜV SÜD Certificate for Rhapsody TestConductor Add On**
- **TÜV SÜD Report on Certificate for ISO 26262 and IEC 61508**
- **Rhapsody TestConductor Add On Validation Suite**: separately available test suite for Rhapsody TestConductor to help in qualification efforts
- **Certification kits** for the SXF and SMXF frameworks



IBM Rational Rhapsody TestConductor Add-On Certification

- The **Validation Suite** is an integral part of the IBM Rational Rhapsody TestConductor Add-On certification (ISO 26262 and IEC 61508)
  - IBM Rational Rhapsody TestConductor Add-On is a **qualified testing tool** for IBM Rational Rhapsody
  - Successful qualification has been acknowledged by **TÜV Süd** (independent German certification body)
  - TÜV Süd issued a certificate about **successful qualification**
  - Customers can immediately leverage from the certificate
  - Certificate will be also issued for IBM Rational Rhapsody TestConductor Add-On integrated into IBM Rational Rhapsody

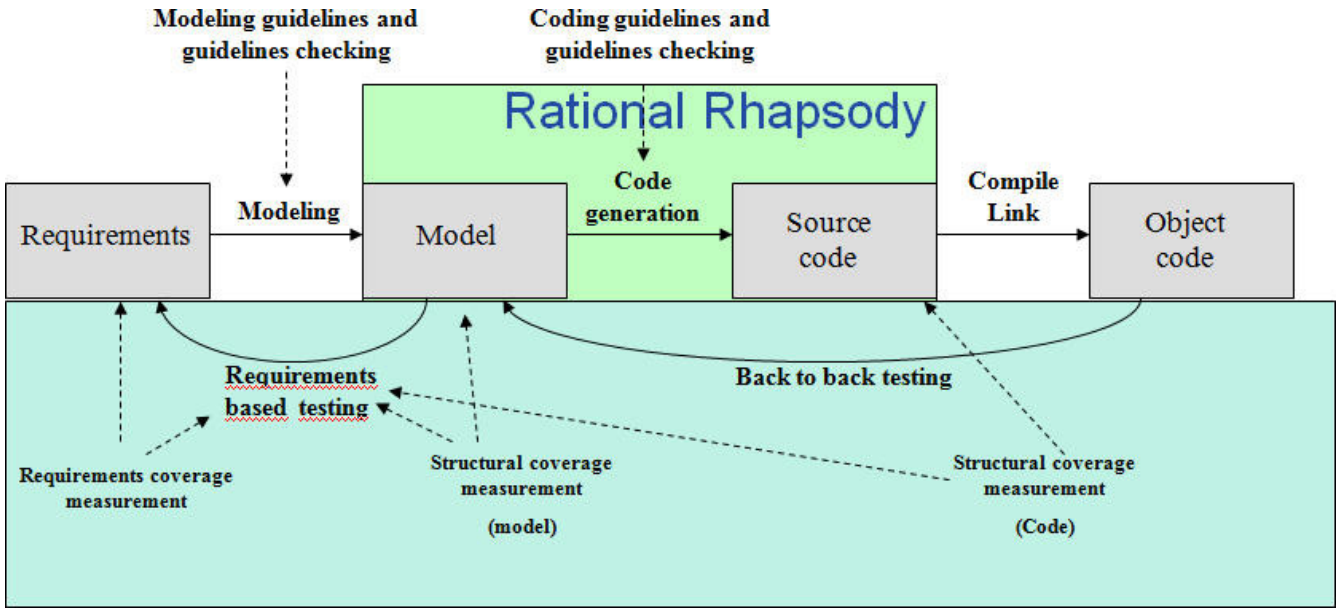
CERTIFICATE

No. Z10 12 02 73219 002

Holder of Certificate:	BTC Embedded Systems AG An der Schmiede 4 26135 Oldenburg GERMANY
Factory(ies):	73219
Certification Mark:	TÜV SÜD
Product:	Software Tool for Safety Related Development
Model(s):	BTC TestConductor
Parameters:	The model-based verification tool is classified as T2 offline support tool according to IEC 61508-4. It is qualified to be applied in safety related software development for all SIL levels according to IEC 61508, and for all ASIL-Levels according to ISO 26262. The report BO84164C is a mandatory part of this certificate.
Tested according to:	IEC 61508-3:2010 ISO 26262-8:2011

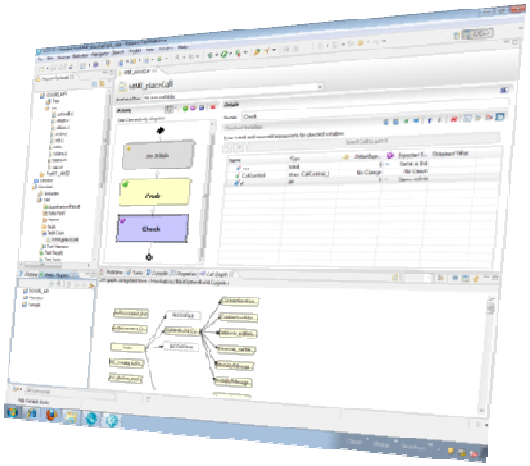
IBM Rational Rhapsody Reference Workflow

- Rhapsody Reference Workflow for the development of safety-related software
  - provides **guidance** on how to fulfill functional safety requirements with model-based development methods and tools;
  - is based on **best practices** for safety-related projects;
  - addresses various workflow **activities** relevant for the development of safety-related software with a special focus on verification and validation to develop safe software;
  - conforms to **IEC 61508** and **ISO 26262**.



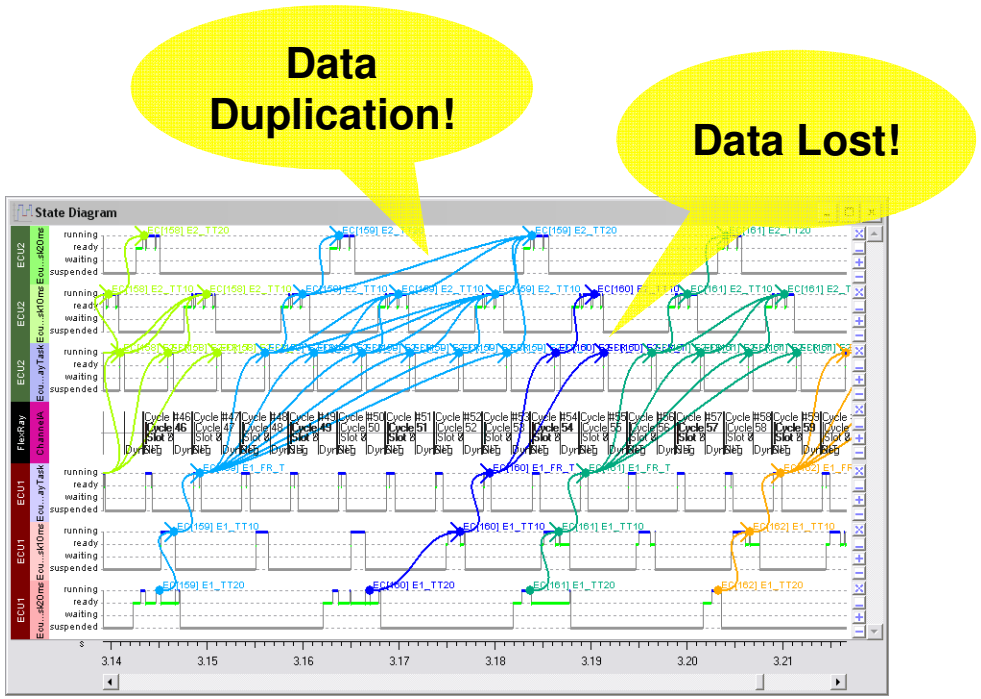
IBM Rational Test RealTime

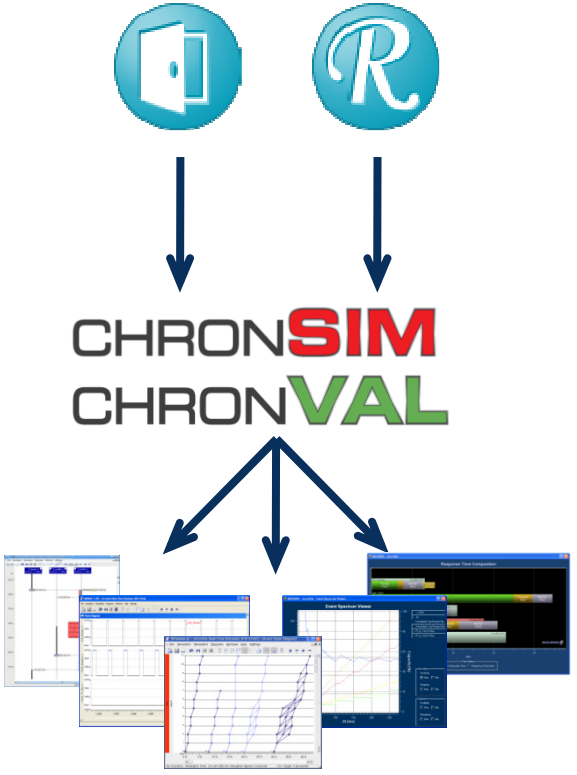
- **Rational Test RealTime** is a cross-platform solution for component testing and runtime analysis of embedded software for C,C++, Ada and Java.
  - Software Unit & Integration Testing
  - Electronic Control Unit (ECU) / Hardware in the Loop (HIL) Validation
  - Modified Condition/Decision Coverage (MC/DC)
  - Memory Profiling
  - Performance Profiling
  - Runtime Tracing
  - Static Code Analysis (MISRA-C)
  - Integrated with Rhapsody TestConductor
- Rational Test RealTime DO-178B Qualification Kit



Testing Ecosystems: Timing-Modeling

INCHRON, an IBM Business Patner, offers test tools for model based **real-time** simulation, **chronSIM**, and analysis/validation, **chronVAL**.
The INCHRON Tool Suite is ingetraged with IBM Rational products, cush as **Rhapsoty**, **DOORS**, **Rational Team Concert**, **Rational Quality Manager**.



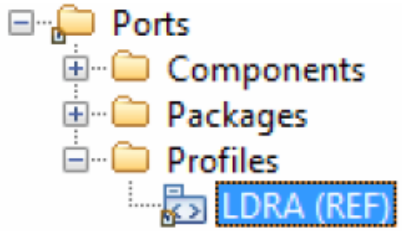


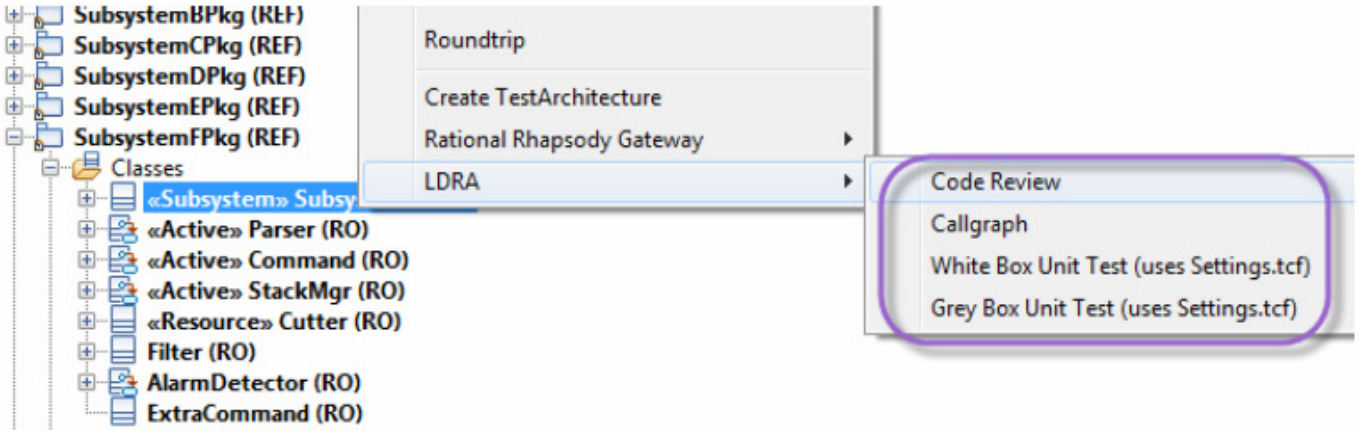
Testing Ecosystems: LDRA

LDRA offers automated analysis and testing tools for safety-critical software to ensure adherence to compliance to standards (i.e. MISRA-C, MISRA-C++, JSF++).

The LDRA Tools Suite for C/C++ provides a **Rhapsody plugin**:

- To instrument all the files generated by a Rhapsody configuration for Static and Dymanic Analysis
- To Analyze a single file and perform Unit Testing on it





An Example: Invensys Rail Dimetronic

Speeds innovation with a unified platform for multi-stage development processes

The need:

- Modernize development processes
- Ensure systems integration with other railways while meeting railway standards

The solution:

- Incorporated system intelligence into its development process
- Deployed an application development platform to:
  - Model system reliability
  - Highlight areas requiring improvement

"Innovation and process flexibility are important in allowing us to differentiate our offerings. We're now able to ensure that our design can be rapidly adapted, not only to customer needs, but to changing ERTMS requirements, at a reasonable cost."

Francisco Lozano
ERTMS Program Manager

Invensys Rail DIMETRONIC

The benefits:

- ✓ Reduced time-to-market for signaling systems products by 40%
- ✓ Facilitated 100% compliance rate with ERTMS standards for code traceability and safety
- ✓ Reduced cost and risks of development and documentation

Solution components:

- IBM® Rational® Rhapsody
- IBM Rational DOORS
- IBM Rational Synergy
- IBM Rational Change
- IBM Rational Publishing Engine
- IBM Software Services

An Example: Tata Consultancy Services Limited

Improving time-to-market with IBM Rational Rhapsody

The need:

- Achieve high-quality design/code
- Speed-up development and variants

The solution:

- Improved software development process by incorporating both MBDA (SysML) and MDD (UML) for embedded real-time:
  - Systems Engineering
  - Software Development
  - Software Testing

The benefits:

- ✓ Extracted 60% of a new design from reverse engineering of existing software
- ✓ Reduced 50% learning curve for new staff members
- ✓ Eliminated 90% of design errors with model simulation

"We used IBM Rational Rhapsody to aid and succeed in the model-driven development (MDD) methodology for the key product development of our customers. Behavioral modeling in Rhapsody is very powerful and we used it extensively to test our design and generate high-quality code."

Rampura Venkatachar Raman

Head – EIS Semiconductor & Consumer Electronics Vertical, Tata Consultancy Services

TATA

Solution components:

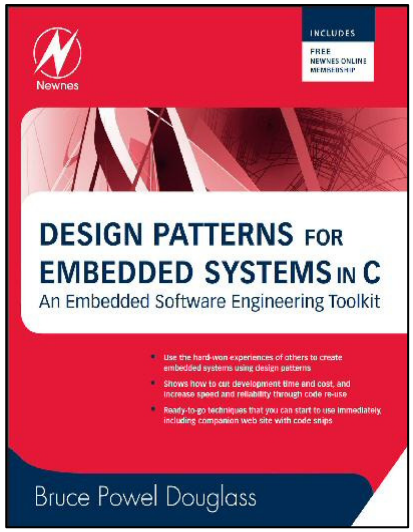
- IBM® Rational® Rhapsody

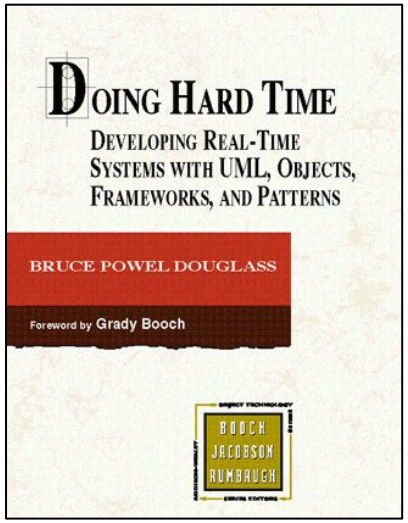
Summary

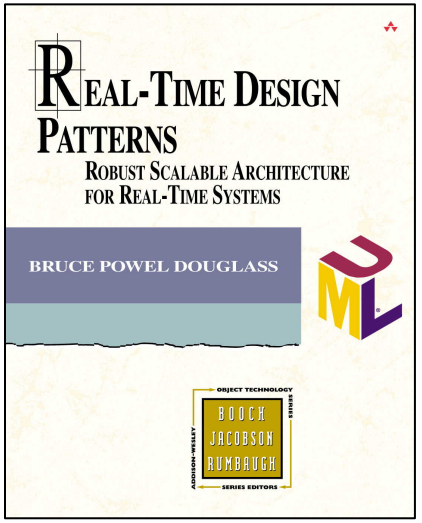
- **Safety Critical** software development is hard and expensive
- **Model-Driven Development** brings better quality and
- The **IBM Rational Solutions for Systems and Software Engineering enables** you focusing on what really matterns and reduce the certification effort.

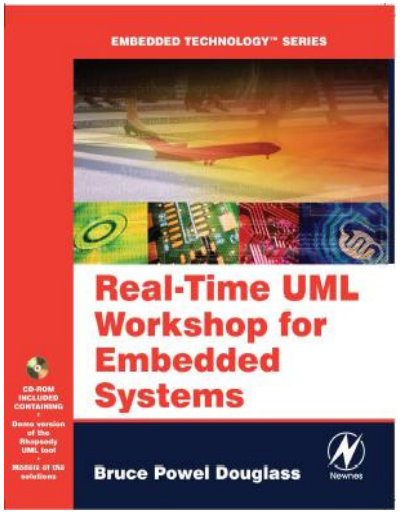
http://www-01.ibm.com/software/rational/workbench/systems/

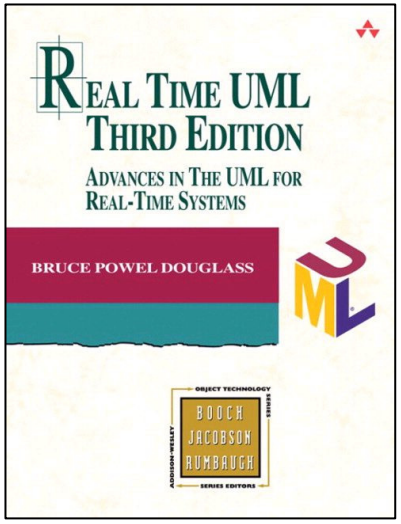
Some References

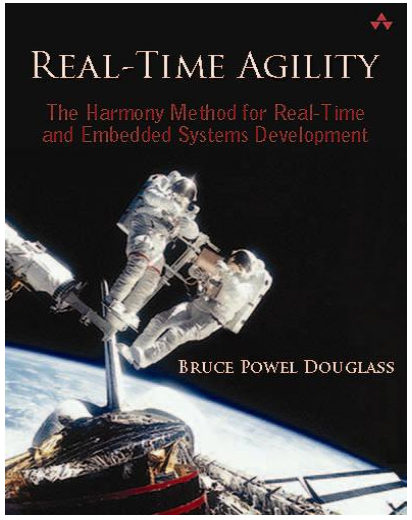


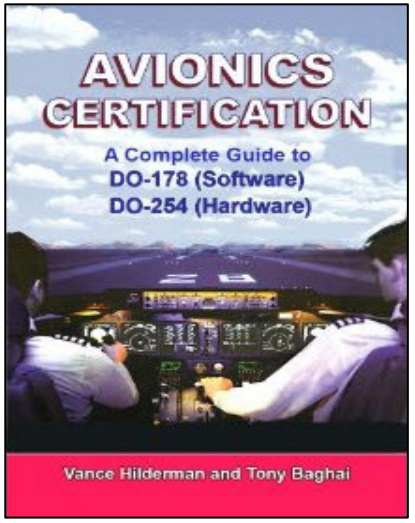


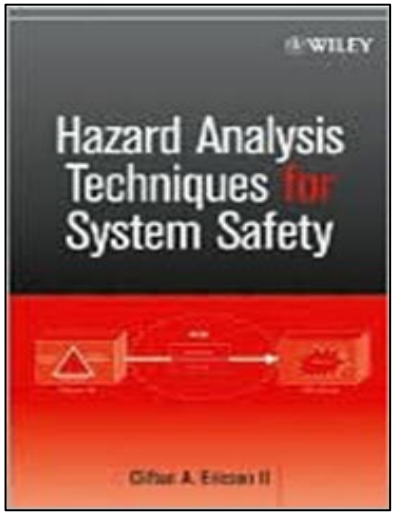












www.ibm.com/software/rational

www.ibm.com/software/rational

Acknowledgements and disclaimers

Availability: References in this presentation to IBM products, programs, or services do not imply that they will be available in all countries in which IBM operates.

The workshops, sessions and materials have been prepared by IBM or the session speakers and reflect their own views. They are provided for informational purposes only, and are neither intended to, nor shall have the effect of being, legal or other guidance or advice to any participant. While efforts were made to verify the completeness and accuracy of the information contained in this presentation, it is provided AS-IS without warranty of any kind, express or implied. IBM shall not be responsible for any damages arising out of the use of, or otherwise related to, this presentation or any other materials. Nothing contained in this presentation is intended to, nor shall have the effect of, creating any warranties or representations from IBM or its suppliers or licensors, or altering the terms and conditions of the applicable license agreement governing the use of IBM software.

All customer examples described are presented as illustrations of how those customers have used IBM products and the results they may have achieved. Actual environmental costs and performance characteristics may vary by customer. Nothing contained in these materials is intended to, nor shall have the effect of, stating or implying that any activities undertaken by you will result in any specific sales, revenue growth or other results.

C++ for Safety Critical Systems: JSF++

- Lockheed Martin decided to adopt C++ for the Joint Strike Fighter (F-35) Project
- **Bjarne Stroustrup** has been asked to define a C++ Safe Coding Standard
  - "*C++ can provide a **safer** subset of a C superset*"
- JSF++ AV (Air Vehicle) Coding Standard has formally released on 2005
- MISRA-C++ has been released on 2008