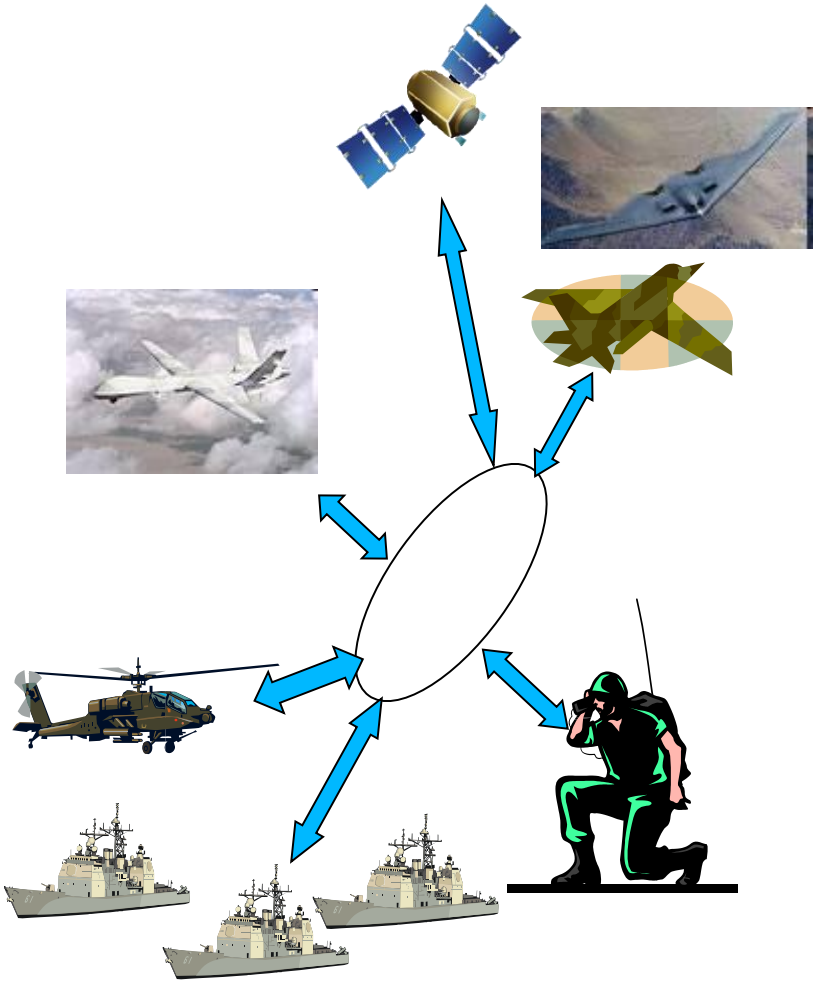
Mir Hidayathulla
IBM Rational Technical Sales Manager - Systems
Mir.Hidayat@in.ibm.com

IBM Software
Innovate2012
The Premier Event for Software and Systems Innovation

Next NOW!

Products are getting increasingly ***smarter*** to meet more demanding and unique needs of customers

- Sensor Fusion and Synthetic Vision allow helicopters to land safely in degraded visual environments (i.e., brownout)
- UAVs fly complex missions and provide realtime data, battlefield assistance, search and rescue, and other services to improve decision making and warfighter safety
- Fly-by-wire allows pilots to safely fly unstable aircraft or at high-speed
- The future is providing defense capability that can be rapidly changed and created through the complex orchestration of services offered by products and people such as the **Integrated Battlefield Command System (IBCS)**



What's possible by 2015?

Innovation for a smarter planet.

Increase of software in aerospace & defense

F-4	1960	8%
A-7	1964	10%
F-111	1970	20%
F-15	1975	35%
F-16	1982	45%
B-2	1990	65%
F-22	2000	80%

F-4

F-111

F-15

F-16

B-2

F-22

F-35

1960

Only 8% of specification requirements required software control in the F-4

80% of specification requirements required software control in the F-22

Today …

F35 will have 5.7 million lines of code, vs 1.7 million lines of code for F-22

Chevrolet Volt

GM leverages Rational solution to develop innovative products

What's smart?

- Innovative electric drive system
  10 million lines of code; Nearly 100 microprocessors

Smarter business outcomes

- Volt was delivered in <5 years
  Industry average is 10+ years

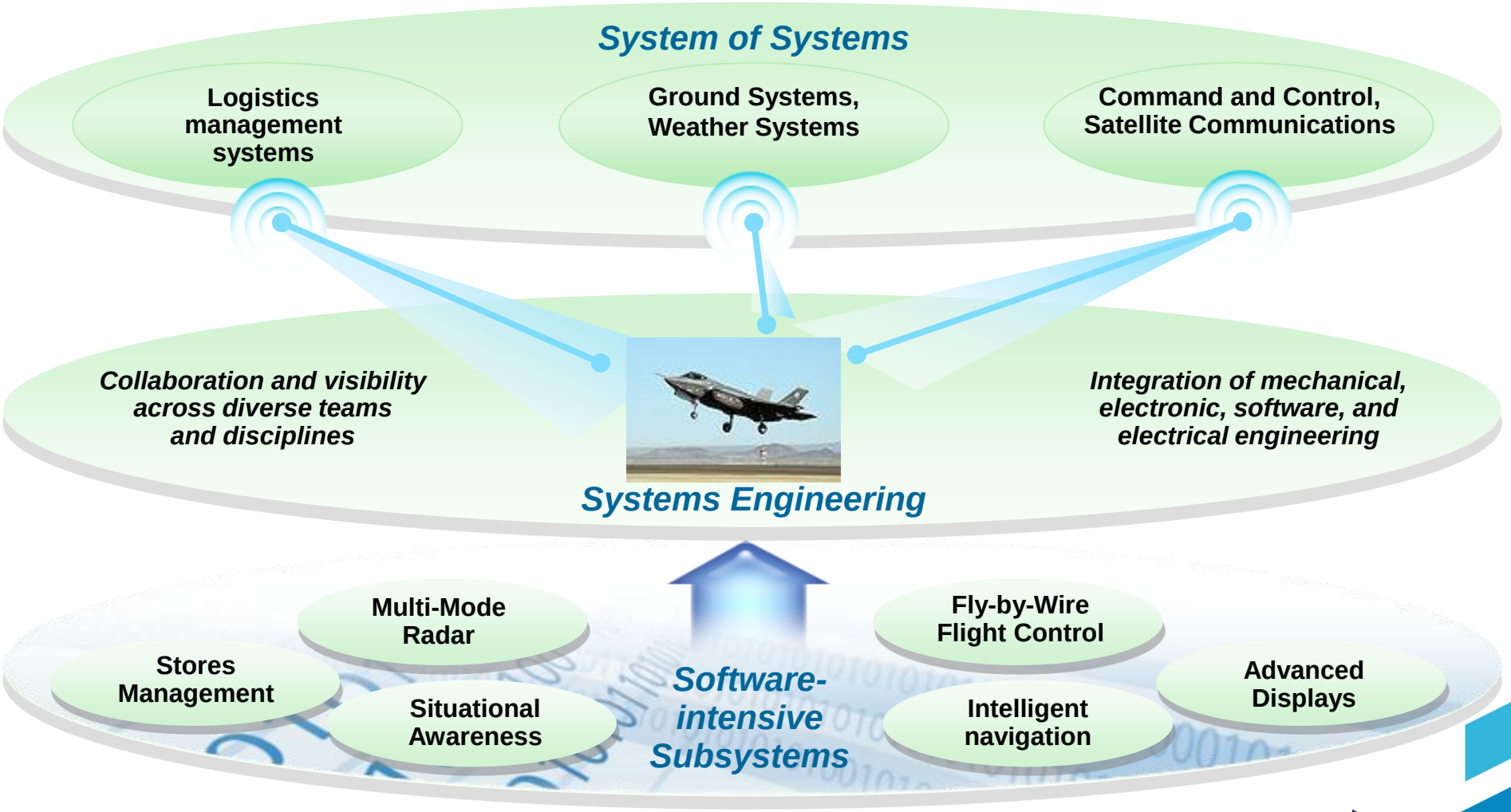
How IBM helps GM develop smarter products

- Requirements management
- Model-driven development
- Team collaboration
- Engineering asset management
- Technical services
- Business transformation services

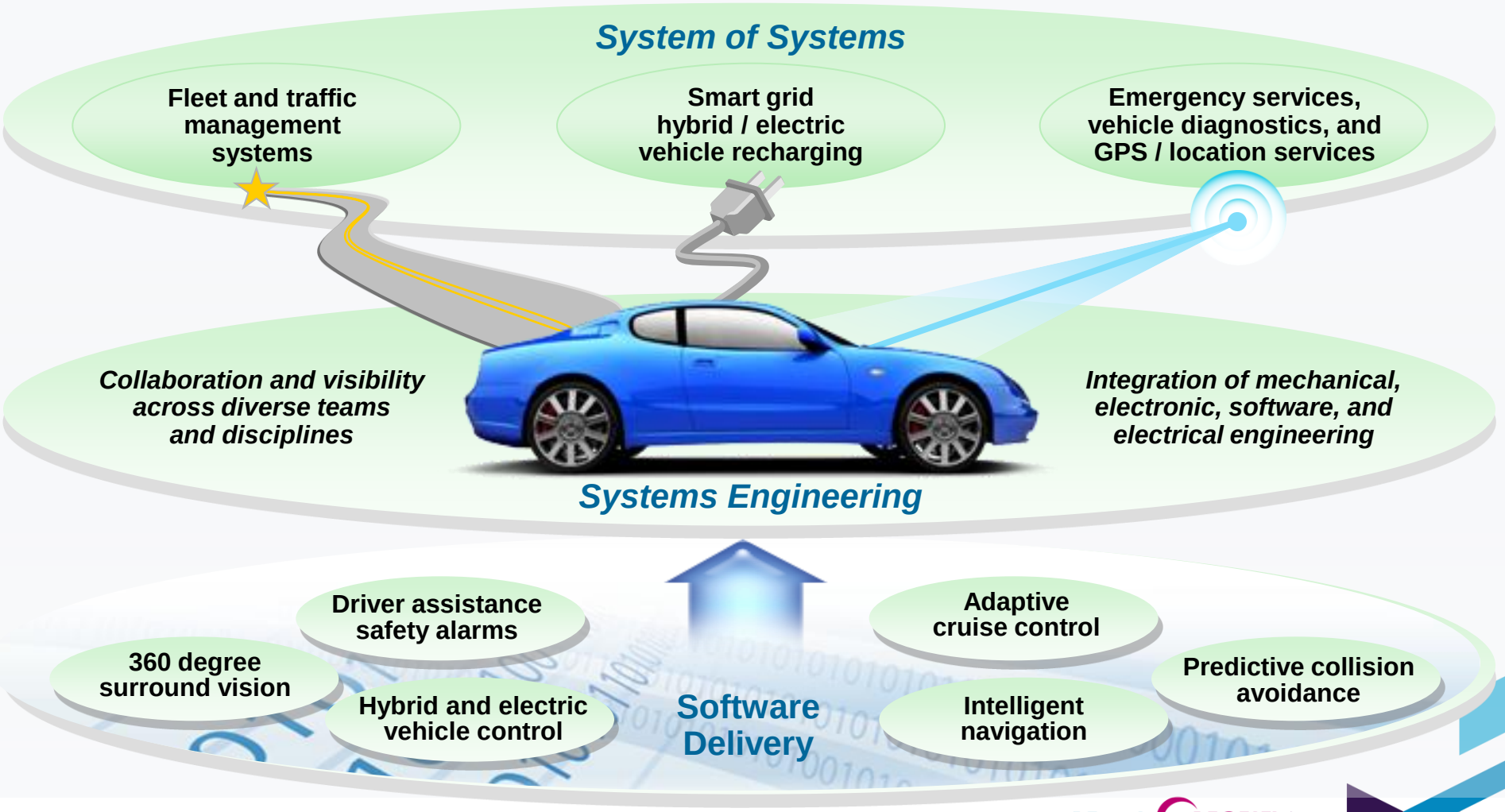
Watch the video

Tying it All Together: Smart Products and Services Example

From sophisticated in-device software, to complex "system of systems" ecosystems, products will continue to get smarter



From sophisticated in-vehicle software, to complex "system of systems" ecosystems, cars are getting ***smarter*** every day



Modern Approaches for Describing Systems Are Evolving

To Better Manage Complexity and Reduce Time-to-market

Past

Specifications

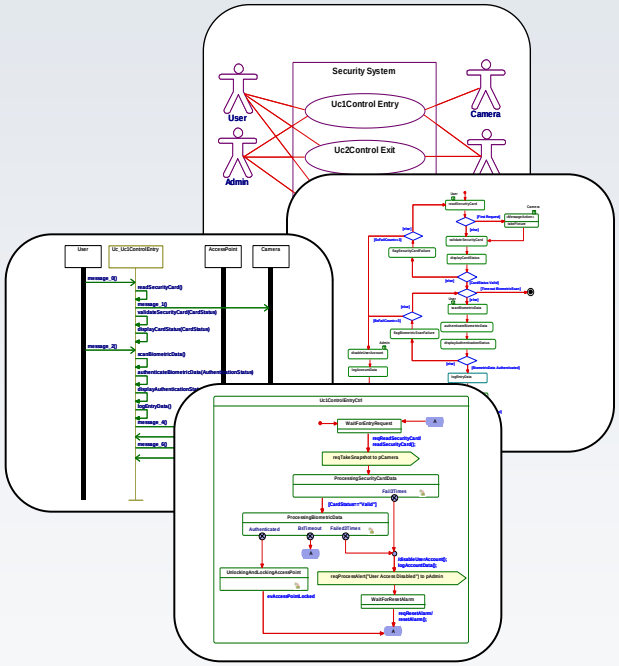
Interface requirements

System design

Analysis & trade-off

Test plans

Future



Moving from manual methods to an automated, visual approach

Aerospace and defense systems

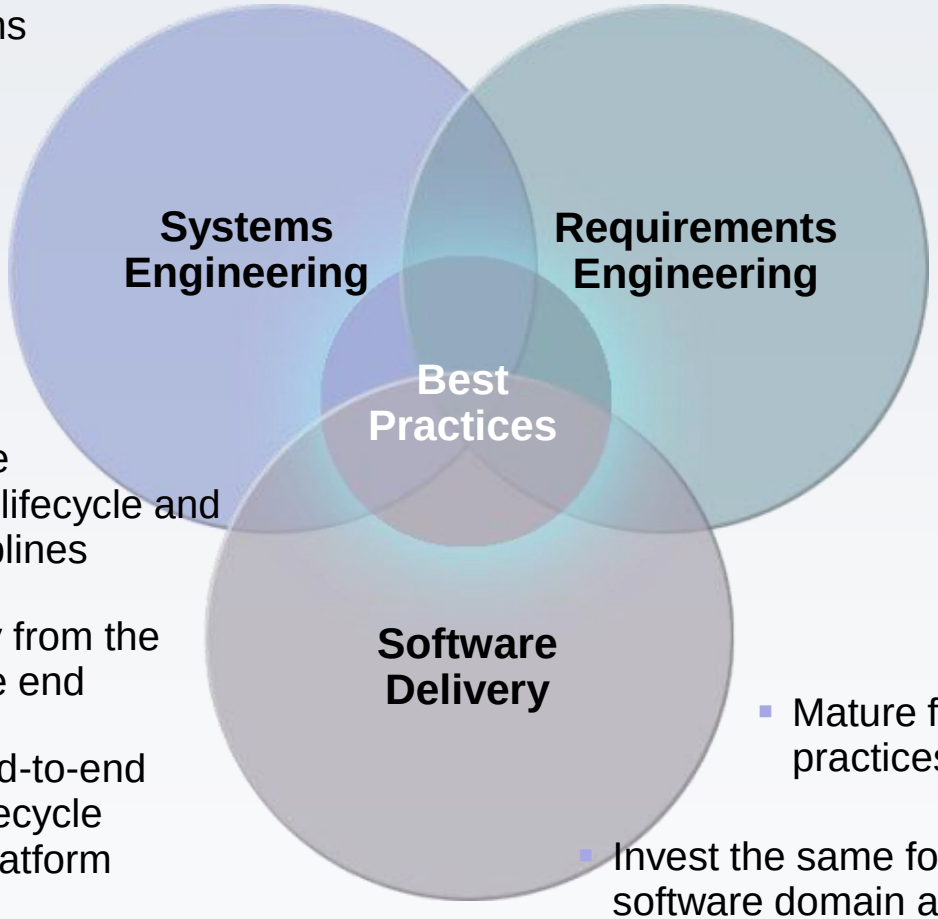
- Systems are being created that are almost incomprehensibly vast in scale
  - Thousands of systems needing to communicate
  - Millions of lines of code
  - Millions of pounds of steel, glass, titanium and copper
  - Billions of dollars
  - Thousands of engineers
  - Years of development time
  - Decades of useful product life

Examples of aerospace and defense applications using model-driven development solutions from IBM

- Air traffic control
- Avionics
  - Displays
  - Navigation
  - Fuel systems
  - Autopilot
- Cabin systems
- Civil aerospace
- Combat aircraft
- Command and control
- Communications
  - Wireless
  - Secure
- Engine controls
- Flight controls
- Hydraulic systems
  - Landing gear
  - Flight controls
- Homeland security
- Land military systems
- Lighting systems
- Maintenance systems
- Maritime/shipboard systems
- Military vehicles
  - Manned
  - Unmanned
- Missiles
- Network net-centric based systems
- Power management
- Radar
- Satellites
- Training systems
- Weapons systems
  - Stores management
  - Fire control

How we can help: Best Practices Adopted By Leading Companies *Establishing Discipline & Governance in Key Product Development Areas*

- Establish core discipline of systems engineering and mature into practices
- Transition from a paper-based to a model-based approach
  - Manage change through the full-lifecycle and across all disciplines
  - Manage Quality from the beginning to the end
  - Establish an end-to-end Engineering Lifecycle Management platform

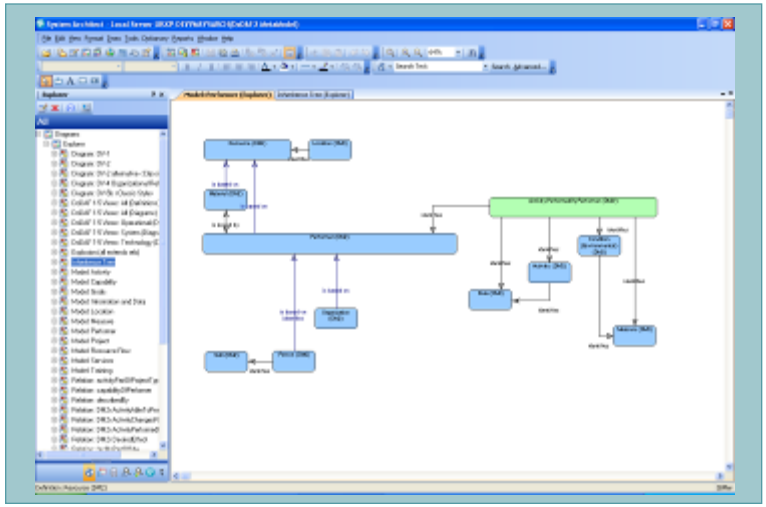


- Build the right product at the right time for the right market
- Mature from requirements engineering in isolated disciplines to requirement engineering across the whole product—software, mechanical, electronics
- Mature from processes to practices; tools to platforms
- Invest the same focus on the software domain as in mechanical

Architecture First

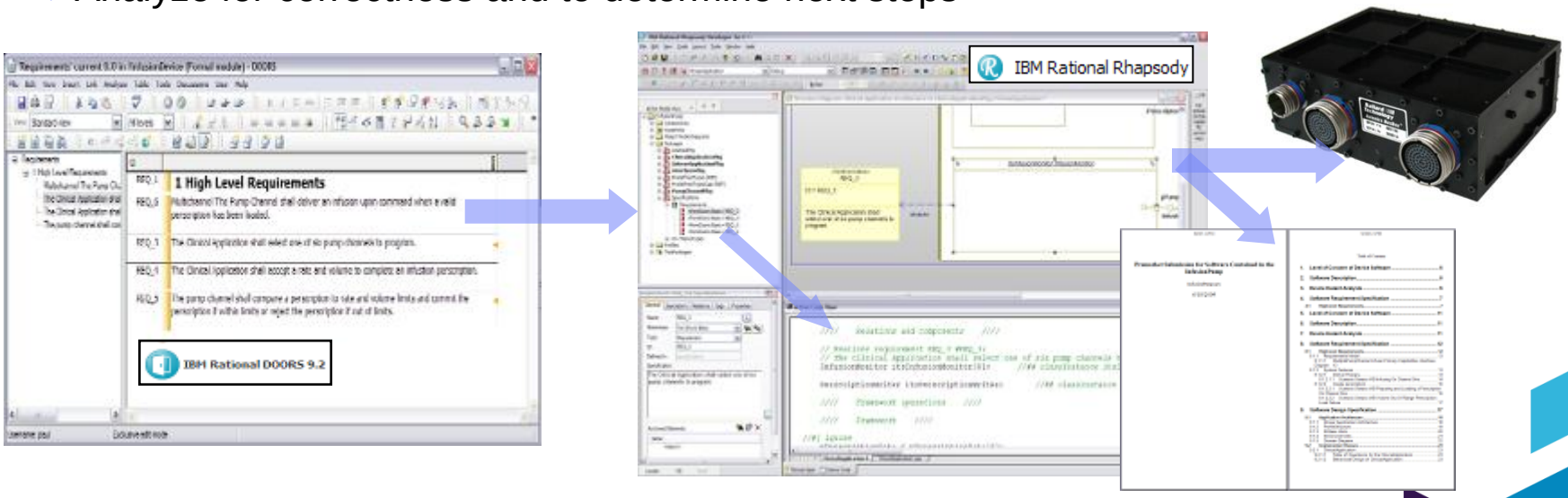
Specify Operational Capabilities, Enterprise Architectures, and Systems

- Specify operational capabilities needed for:
  - Warfighters
  - Network centric operation
- Analyze complex systems of systems
- Comply with DoDAF, MoDAF, FSAM architectural framework requirements

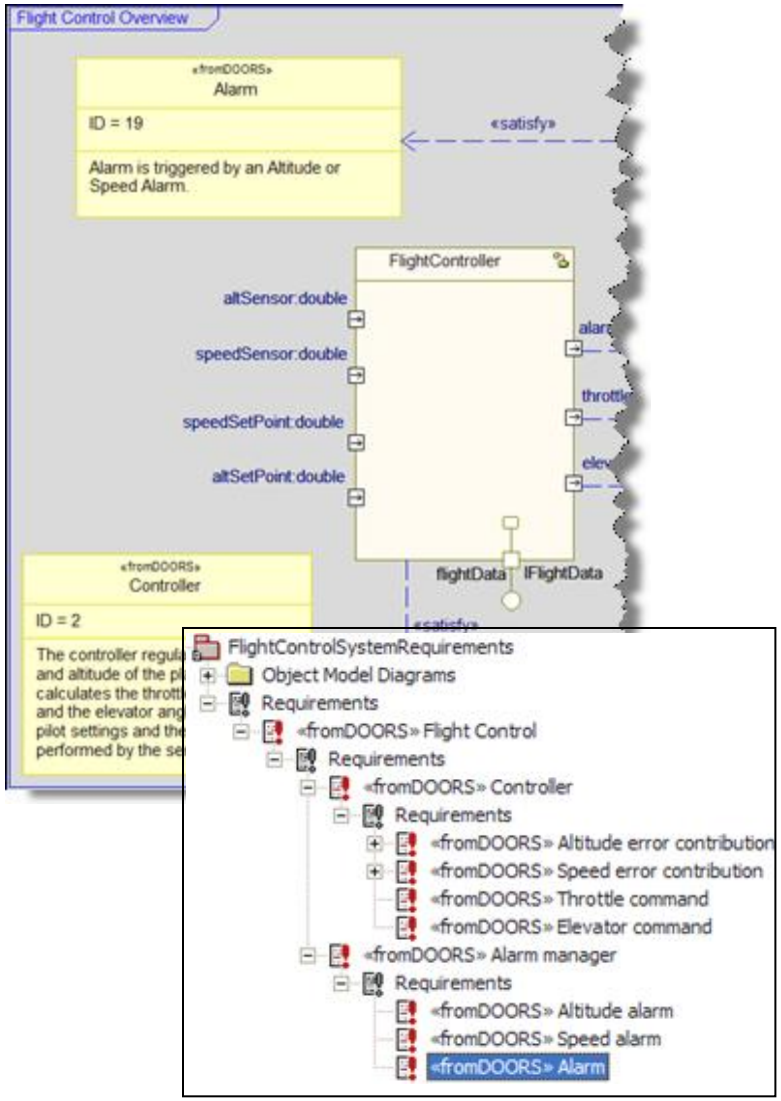


Manage Requirements across Lifecycle and Disciplines

- *Build the right product* because the requirements are visible at all times
  - Prove that all agency requirements (user, safety, regulatory, etc.) were fully satisfied
- Understand the requirements
  - Analyze stakeholder needs
  - Evaluate coverage and impact analysis
- Validate the requirements
  - Analyze for correctness and to determine next steps



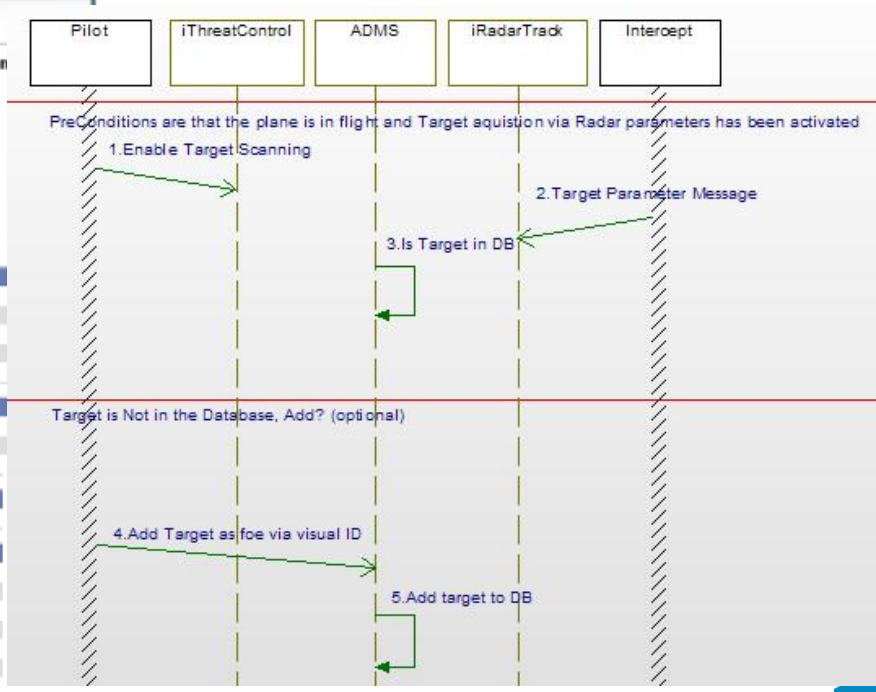
Translate Requirements into a System Design



- *Build the product right* with structural and behavioral analysis and design
- Visualize the system
  - Reduce confusion over requirements
  - Specify system functionality
  - Simulate to confirm functionality
- Analyze impact of changes
  - Whether in requirements or design
- Trace requirements in either direction
  - Provide full accountability and understanding
  - Comply with DO178B traceability
- Specify and develop software
  - Monitor and control the system

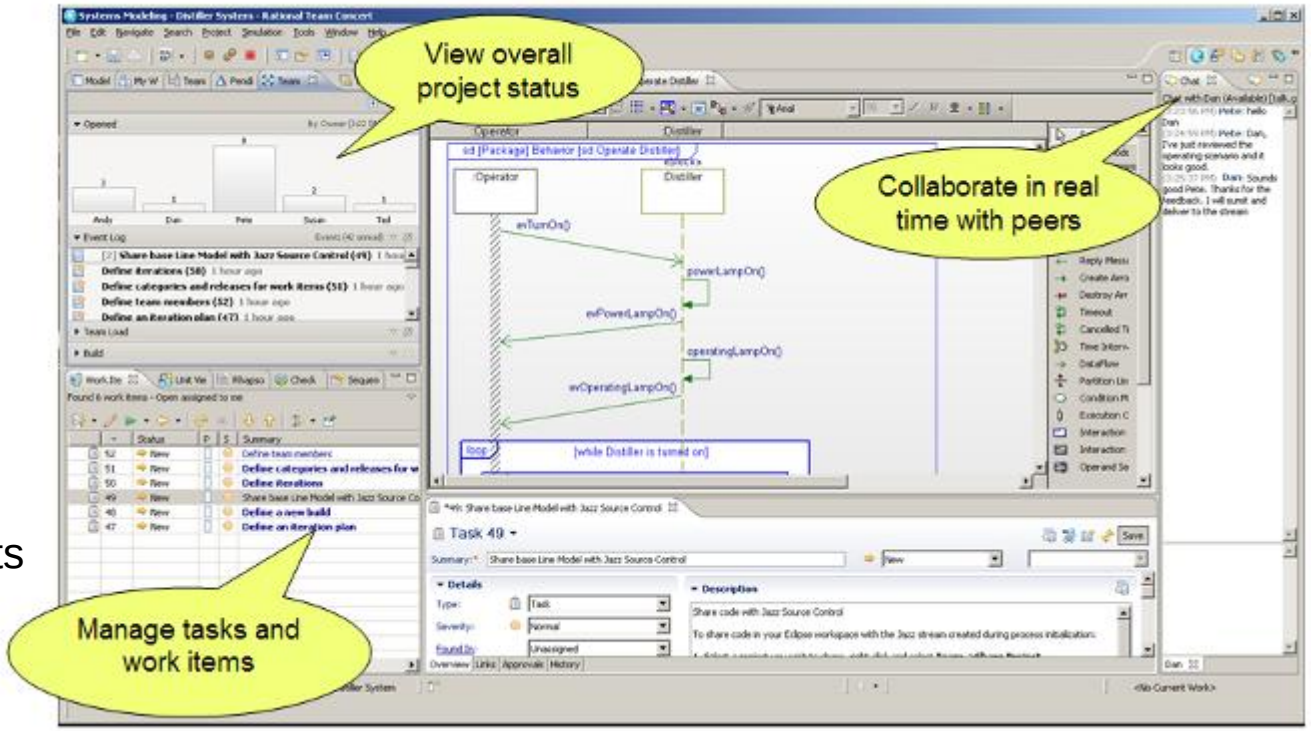
Build in Quality from Concept to Launch

- Simulate often to validate functionality and verify correctness
- Automatically create and execute tests from the design model or target platform
- Manage test cases, while prioritizing the features and functions to be tested



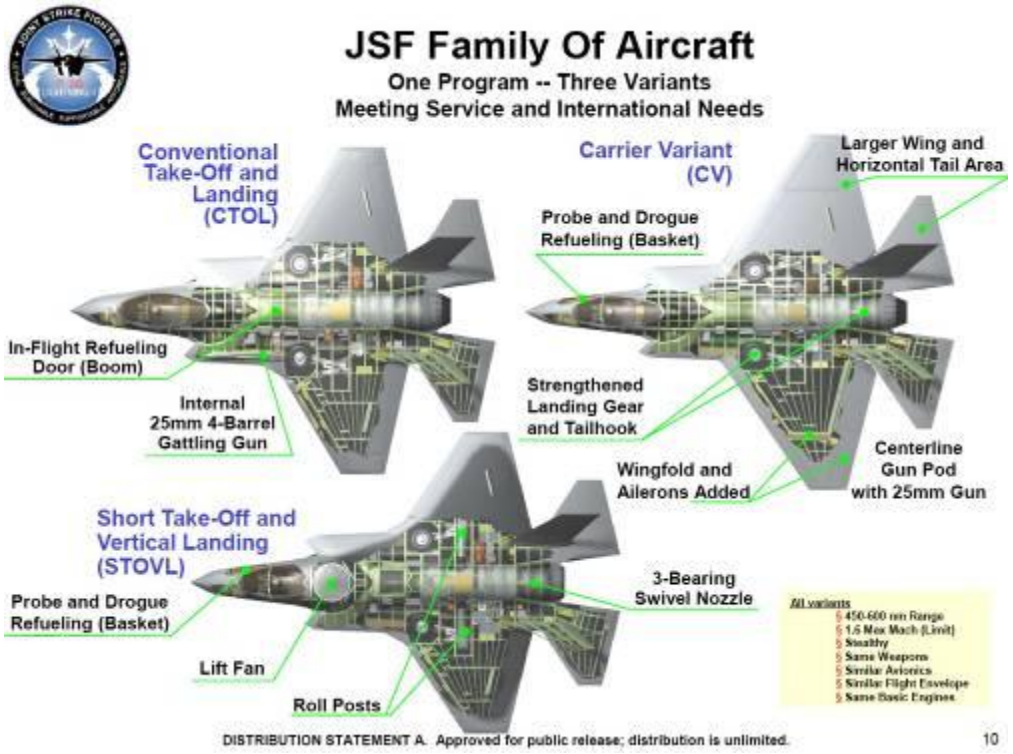
Collaborate and Communicate throughout Development

- Collaborate across teams and geographies
  - Reduce time and risk associated with parallel development
  - Enable integrated design, sharing and review across diverse engineering teams
- Enhance productivity
  - Share views
  - Collaboratively debug
  - Link work items
- Automatically generate reports and documentation directly from the design



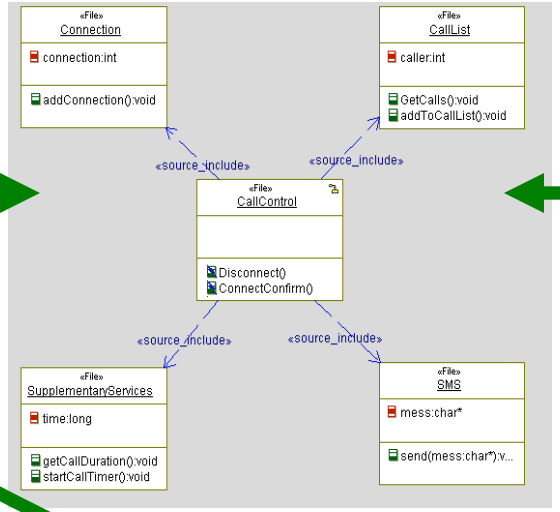
Recapture Intellectual Property

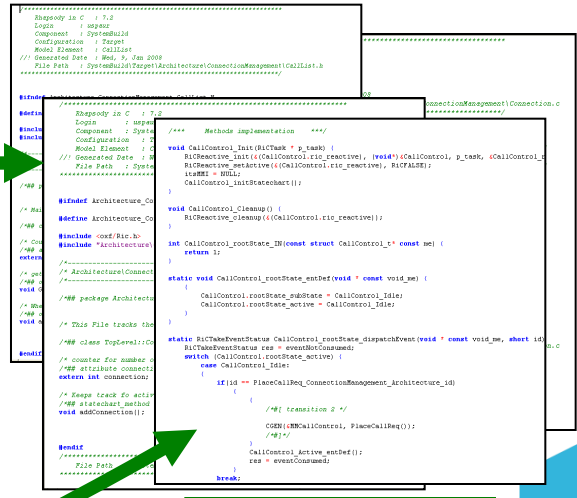
- Preserve and reuse designs and design data
  - Visualize and reverse-engineering existing software
  - Create a library of design assets
  - Analyze to best meet requirements
- Work with product lines
  - Expand product offerings
  - Exploit commonality across products
  - Focus efforts on unique product variants



Control the System with Optimized Software

- Design efficient embedded source code
  - Specify and create from the system requirements
  - Generate complete C, C++, Java, and Ada applications
- Unite the architecture and code
  - Simultaneously work with the system design, software and target platform
  - View how a change in any one area is reflected in the others





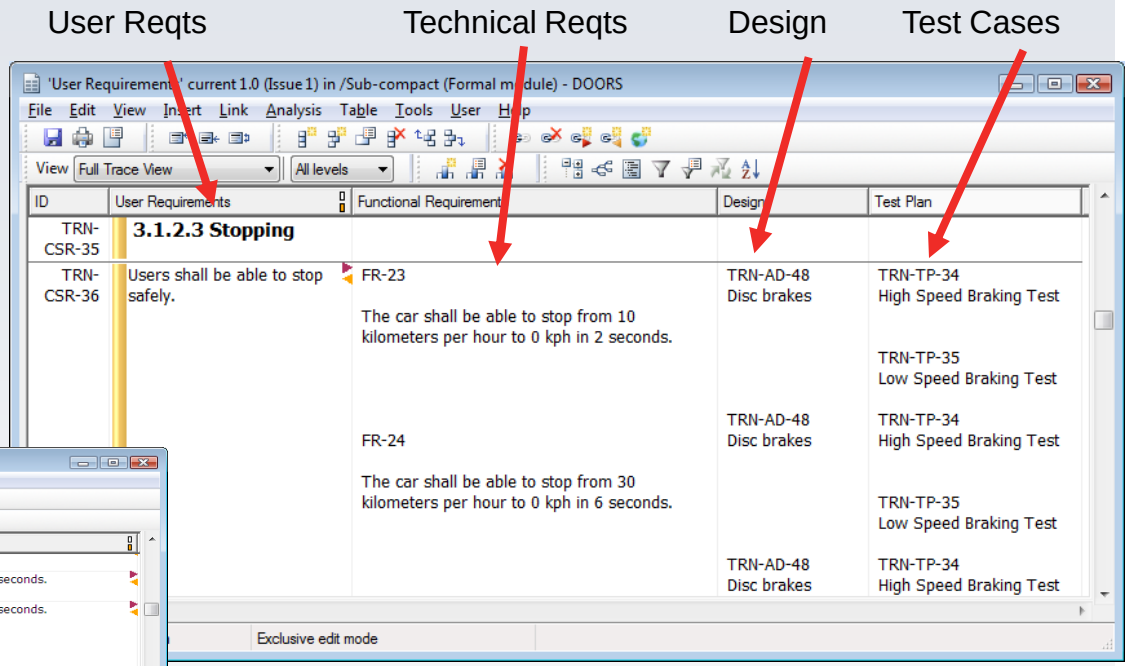
Architectural View

Source Code

Rational DOORS

Manage All Requirements Across the Lifecycle and Across Disciplines

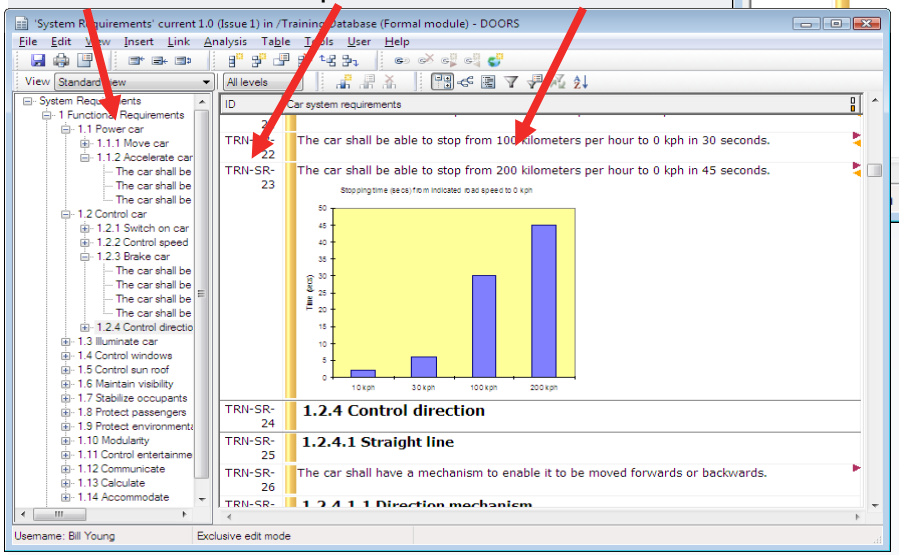
- Combined document and spreadsheet views
- Simple, intuitive interfaces for easy adoption
- History and baselines



End-to-end visual validation in a single view

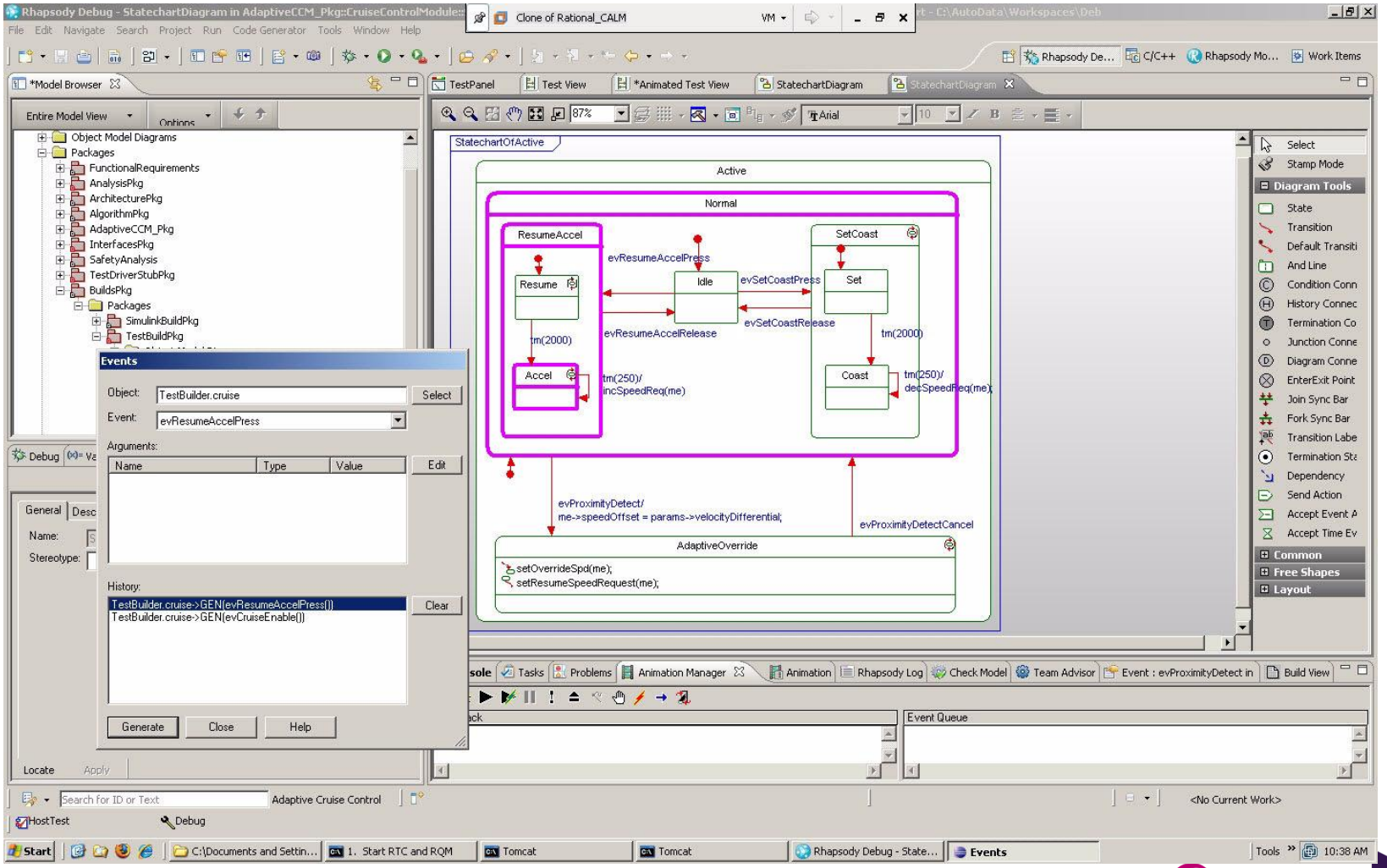
- Input and output from/to various common formats

Solve the right problem because the requirements are visible at all times



Writing Requirements within Context

Using the IBM Rational Rhapsody software to analyze and validate system requirements



IBM Rational Software

Rational. software

Requirements Definition & Management

Architecture Management & Construction

Project & Process Management

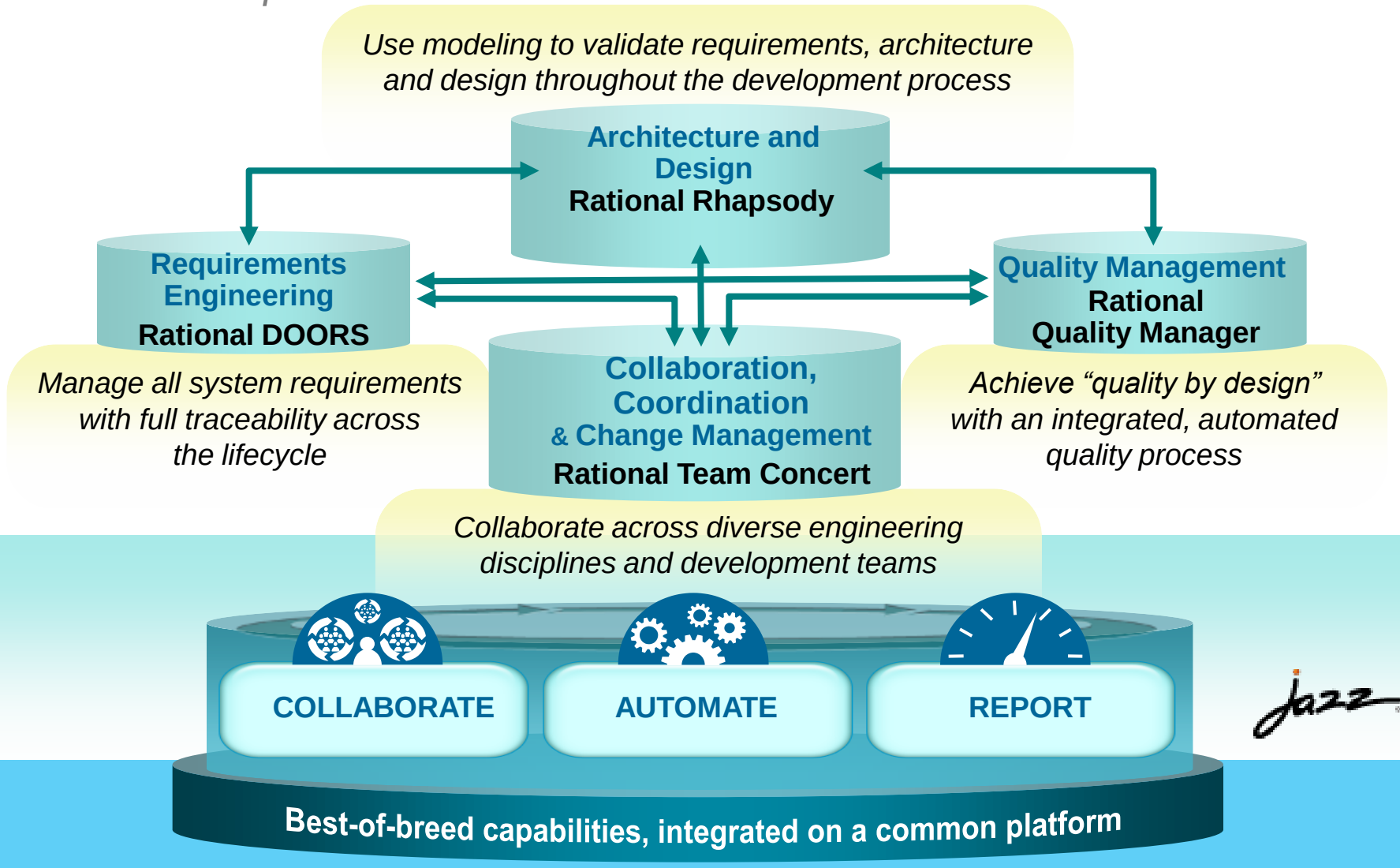
Quality & Compliance Management

Change & Release Management

Next NOW!

Rational Solutions for Systems and Software Engineering

Built on a core product set



Complexity Creates Development Challenges

Leading to cost overruns, schedule slips and quality issues

Poor requirements engineering = failed projects

Paper-based and manual processes hinder efficiency

Complex architecture is difficult to textually explain

Functionality is poorly distributed across components

Hardware/software integration is often late

Many organizations lack formalized practices

Silos of people, process, and projects

Geographic Barriers

- Poor communication
- Language, culture, time
- Process gaps resulting in rework

Organizational Barriers

- Weak collaboration
- Poor project governance and LOB oversight
- Security of IP

Infrastructure Barriers

- Incompatible tools
- Unreliable access
- Lengthy on-boarding
- Inflexible integration

Requirements Definition and Management

Two related dimensions

Elicit

- Engage stakeholders early and often to identify the need

Specify

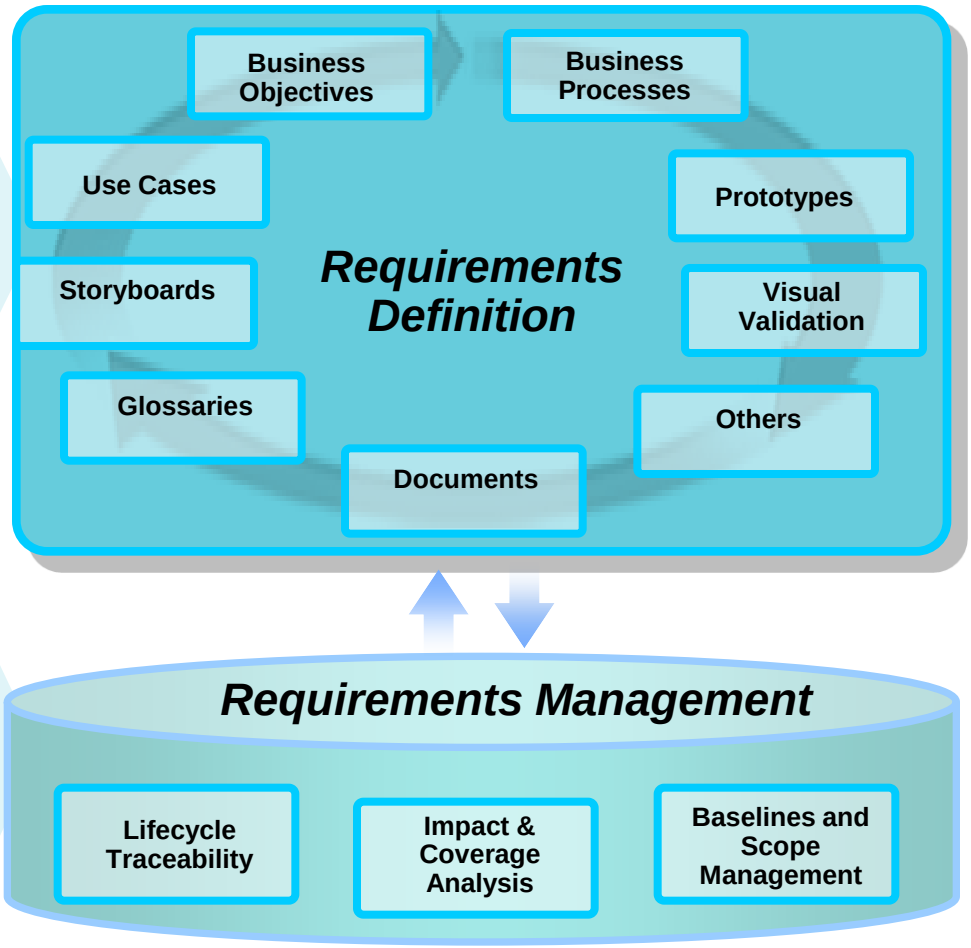
- Capture clear, unambiguous and actionable requirements

Validate

- Stakeholders review what is important and sign off with confidence

Control scope to stay on track as things change

- Which tests must be updated for this requirement change?
- Which requirements have been tested and delivered?
- Who approved this change to the requirements?
- Which requirements have changed since the project scope was originally approved?



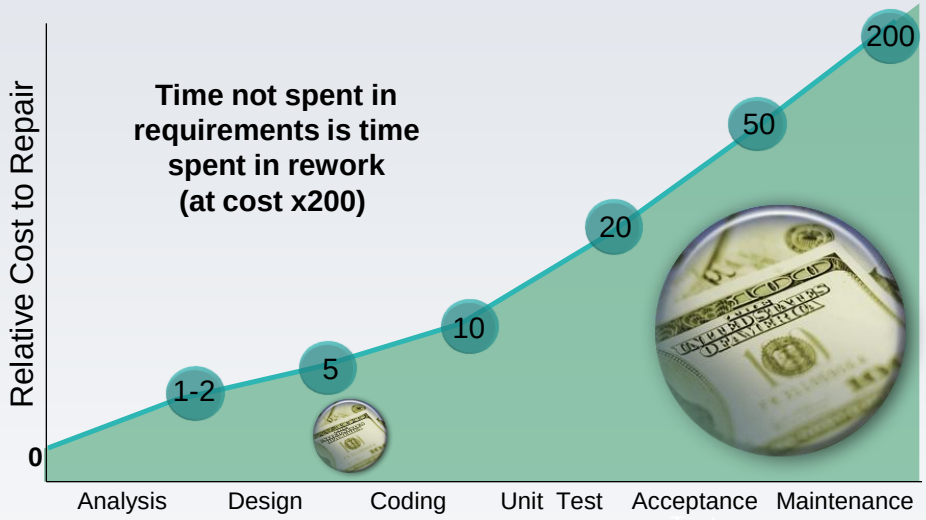
There is a Significant Impact to the Business
As a Result or a Poor Requirements Engineering Process

Requirements Rework
- Errors, late detected in the Maintenance phase can cost up to **200 times** more than detected early in Requirement Analysis phase**1**
- **More than 40%** of development budget can be consumed by poor requirements**2**

Project Impacts
- **41%** of projects fail to deliver the expected business value and ROI**3**
- **49%** of projects overrun original estimates**3**
- **28%** of projects on time and on budget**4**

Requirements Delays
- Being late to market by 6 months or more will cost organizations 33% of the 5-year ROI**5**

Requirements issues drive excessive rework, delays, poor quality, and project failures

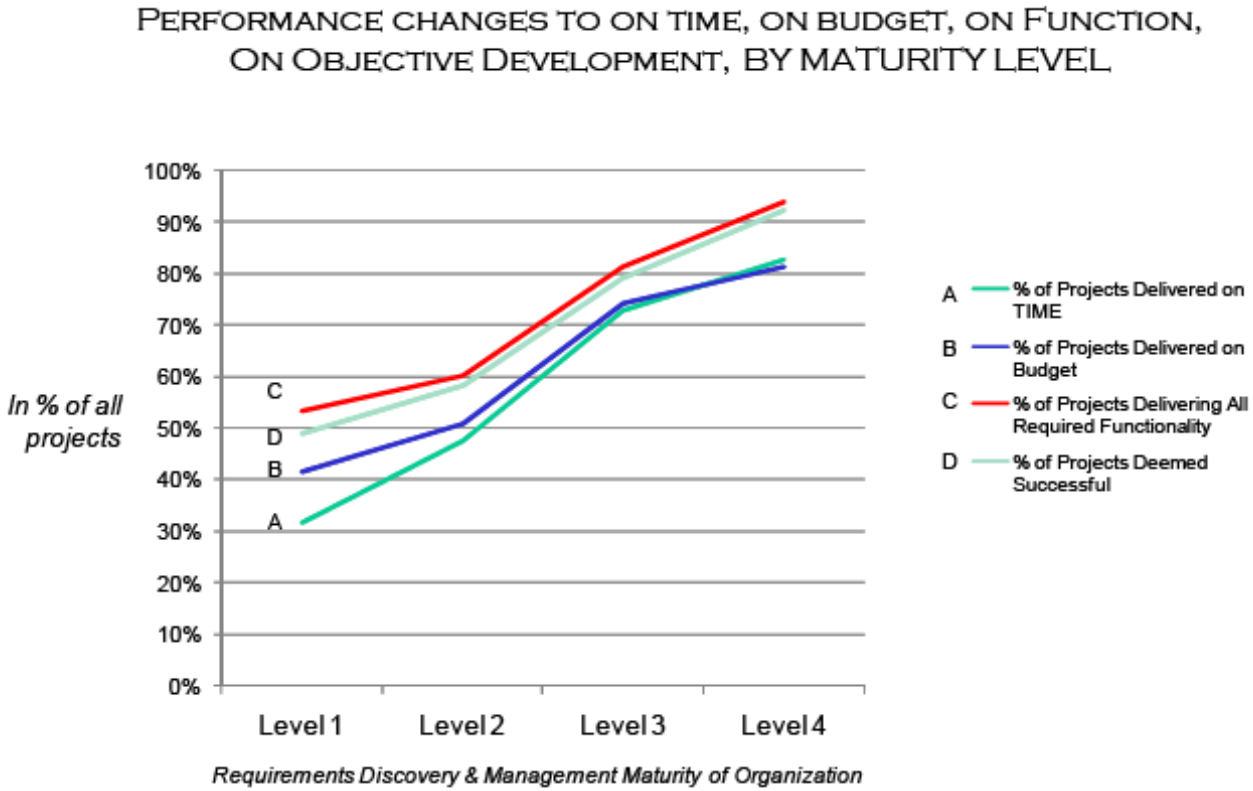


Stage in which Requirements Error Is Discovered

Sources: 1) Leffingwell & Widrig, "Managing Software Requirements," Addison Wesley, 1999 2) IAG Consulting, 2008 3) Dynamic Market Limited, 2007 4) Standish Group, 2001 5) Don Reinertsen, McKinsey, 1983

Good requirements are key to project success

Reclaim up to a third of project budget and schedule



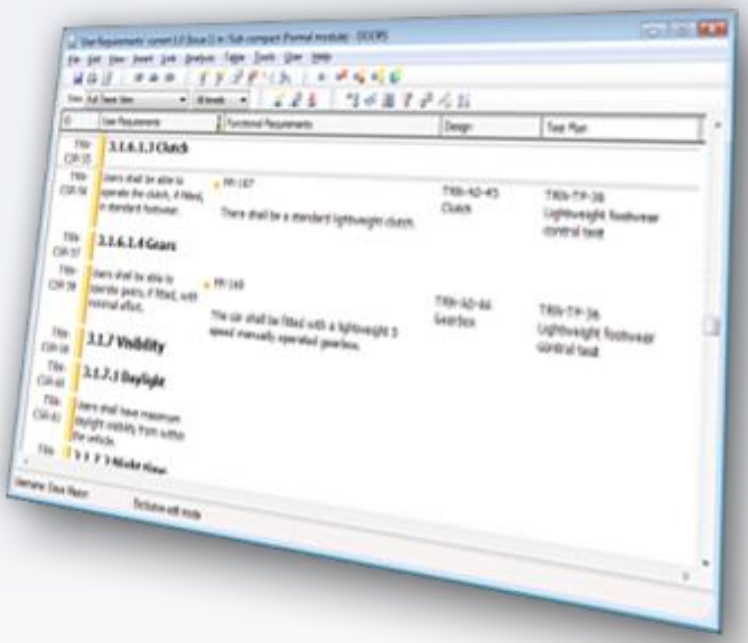
[Source: IAG Consulting, September 2009]

- Average on time performance increased 161%
- Time overruns were reduced 87%
- Average on budget performance improved over 95%
- Budget overruns reduced about 75%
- Percentage of projects that deliver the functionality needed by the business rose by about 75%

IBM Rational DOORS

Manage All Requirements Across the Lifecycle and Across Disciplines

- Provides end-to-end visibility of requirements
- Comprehensive support for recording, structuring, managing, and analyzing requirements and their traceability



"DOORS has helped Delphi improve development team communication, resulting in meeting customer requirements faster and more accurately."

Can manage requirements across multiple engineering disciplines - Software, Electric, Electronic & Mechanical

Index of pain points

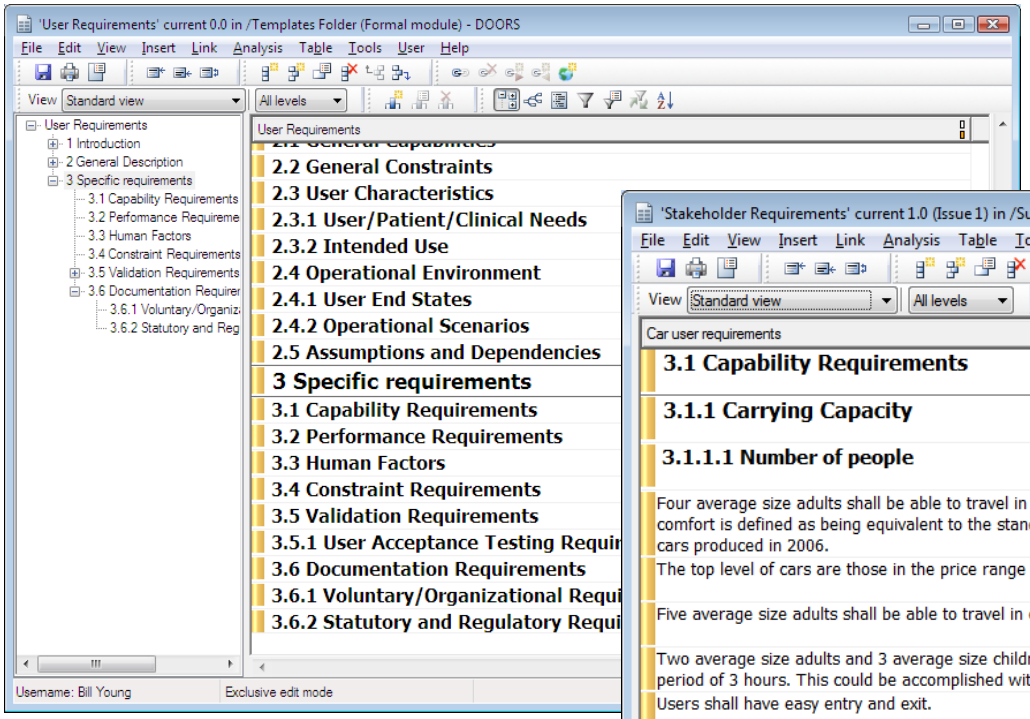
- How do we get started?
- Is DOORS easy to use?
- How can I manage traceability?
- But I can do this with standard document and spreadsheet applications; how is DOORS better?
- How can I find changes easily?
- How can I manage change?
- Can I communicate with non-DOORS users?
- How can I demonstrate compliance?

Index of pain points (continued)

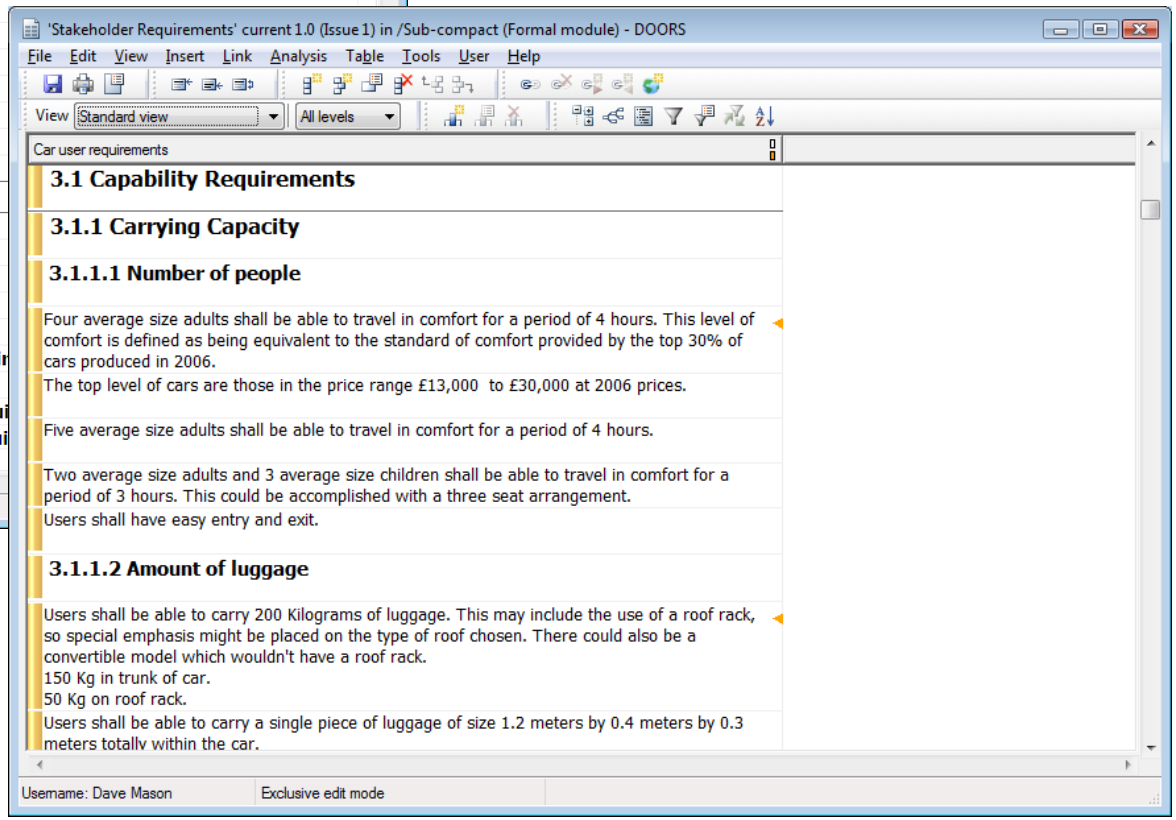
How do we get started?

- The project already has a lot of documentation
  - How can this be used without needing to start again?
- There are still a lot of requirements to write
  - Can they be written easily, right in DOORS?
- Printed documentation is very important
  - Is it necessary to spend time writing reports to get good documentation?

Getting started can be easy in DOORS

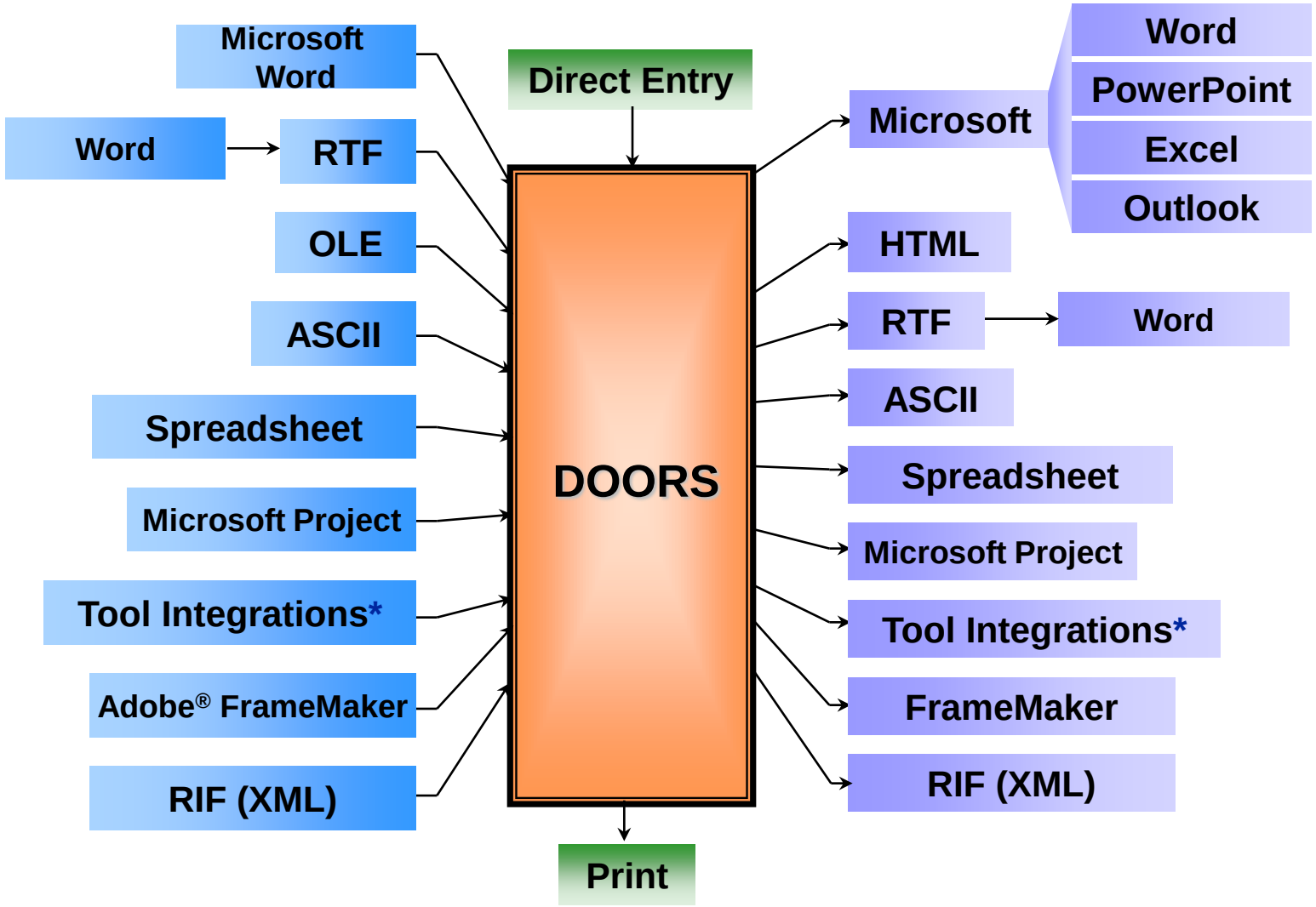


Start with an outline or template



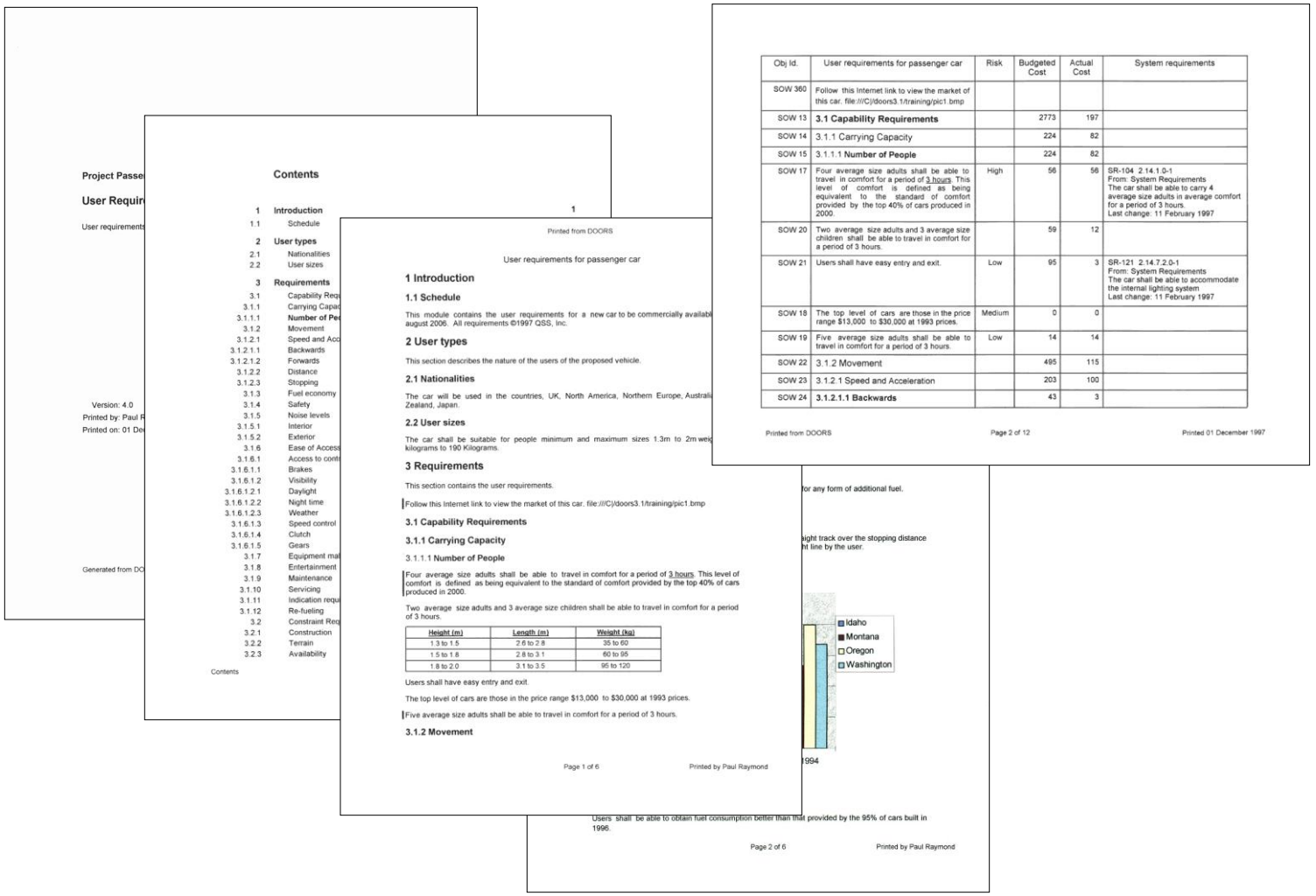
Simply type in your requirements

Import your data and create documents



* See later integrations slide

Printing with standard layouts



Contents

1 Introduction
1.1 Schedule
2 User types
2.1 Nationalities
2.2 User sizes
3 Requirements
3.1 Capability Req
3.1.1 Carrying Capac
3.1.1.1 Number of Pe
3.1.2 Movement
3.1.2.1 Speed and Acc
3.1.2.1.1 Backwards
3.1.2.1.2 Forwards
3.1.2.2 Distance
3.1.2.3 Stopping
3.1.3 Fuel economy
3.1.4 Safety
3.1.5 Noise levels
3.1.5.1 Interior
3.1.5.2 Exterior
3.1.6 Ease of Access
3.1.6.1 Access to cont
3.1.6.1.1 Brakes
3.1.6.1.2 Visibility
3.1.6.1.2.1 Daylight
3.1.6.1.2.2 Night time
3.1.6.1.2.3 Weather
3.1.6.1.3 Speed control
3.1.6.1.4 Clutch
3.1.6.1.5 Gears
3.1.7 Equipment mai
3.1.8 Entertainment
3.1.9 Maintenance
3.1.10 Servicing
3.1.11 Indication requ
3.1.12 Re-fueling
3.2 Constraint Req
3.2.1 Construction
3.2.2 Terrain
3.2.3 Availability

Contents

Printed from DOORS

User requirements for passenger car

1 Introduction

1.1 Schedule

This module contains the user requirements for a new car to be commercially availabl august 2006. All requirements ©1997 QSS, Inc.

2 User types

This section describes the nature of the users of the proposed vehicle.

2.1 Nationalities

The car will be used in the countries, UK, North America, Northern Europe, Australi Zealand, Japan.

2.2 User sizes

The car shall be suitable for people minimum and maximum sizes 1.3m to 2m weig kilograms to 190 Kilograms.

3 Requirements

This section contains the user requirements.

Follow this Internet link to view the market of this car. file:///C|/doors3.1/training/pic1.bmp

3.1 Capability Requirements

3.1.1 Carrying Capacity

3.1.1.1 **Number of People**

Four average size adults shall be able to travel in comfort for a period of 3 hours. This level of comfort is defined as being equivalent to the standard of comfort provided by the top 40% of cars produced in 2000.

Two average size adults and 3 average size children shall be able to travel in comfort for a period of 3 hours.

Height (m)	Length (m)	Weight (kg)
1.3 to 1.5	2.6 to 2.8	35 to 60
1.5 to 1.8	2.8 to 3.1	60 to 95
1.8 to 2.0	3.1 to 3.5	95 to 120

Users shall have easy entry and exit.

The top level of cars are those in the price range $13,000 to $30,000 at 1993 prices.

Five average size adults shall be able to travel in comfort for a period of 3 hours.

3.1.2 Movement

Page 1 of 6 Printed by Paul Raymond

for any form of additional fuel.

ight track over the stopping distance
nt line by the user.

Idaho
Montana
Oregon
Washington

1994

Users shall be able to obtain fuel consumption better than that provided by the 95% of cars built in 1996.

Page 2 of 6 Printed by Paul Raymond

Obj Id.	User requirements for passenger car	Risk	Budgeted Cost	Actual Cost	System requirements
SOW 360	Follow this Internet link to view the market of this car. file:///C|/doors3.1/training/pic1.bmp				
SOW 13	**3.1 Capability Requirements**		2773	197	
SOW 14	3.1.1 Carrying Capacity		224	82	
SOW 15	3.1.1.1 **Number of People**		224	82	
SOW 17	Four average size adults shall be able to travel in comfort for a period of 3 hours. This level of comfort is defined as being equivalent to the standard of comfort provided by the top 40% of cars produced in 2000.	High	56	56	SR-104 .2.14.1.0-1 From: System Requirements The car shall be able to carry 4 average size adults in average comfort for a period of 3 hours. Last change: 11 February 1997
SOW 20	Two average size adults and 3 average size children shall be able to travel in comfort for a period of 3 hours.		59	12	
SOW 21	Users shall have easy entry and exit.	Low	95	3	SR-121 .2.14.7.2.0-1 From: System Requirements The car shall be able to accommodate the internal lighting system Last change: 11 February 1997
SOW 18	The top level of cars are those in the price range $13,000 to $30,000 at 1993 prices.	Medium	0	0	
SOW 19	Five average size adults shall be able to travel in comfort for a period of 3 hours.	Low	14	14	
SOW 22	3.1.2 Movement		495	115	
SOW 23	3.1.2.1 Speed and Acceleration		203	100	
SOW 24	**3.1.2.1.1 Backwards**		43	3	

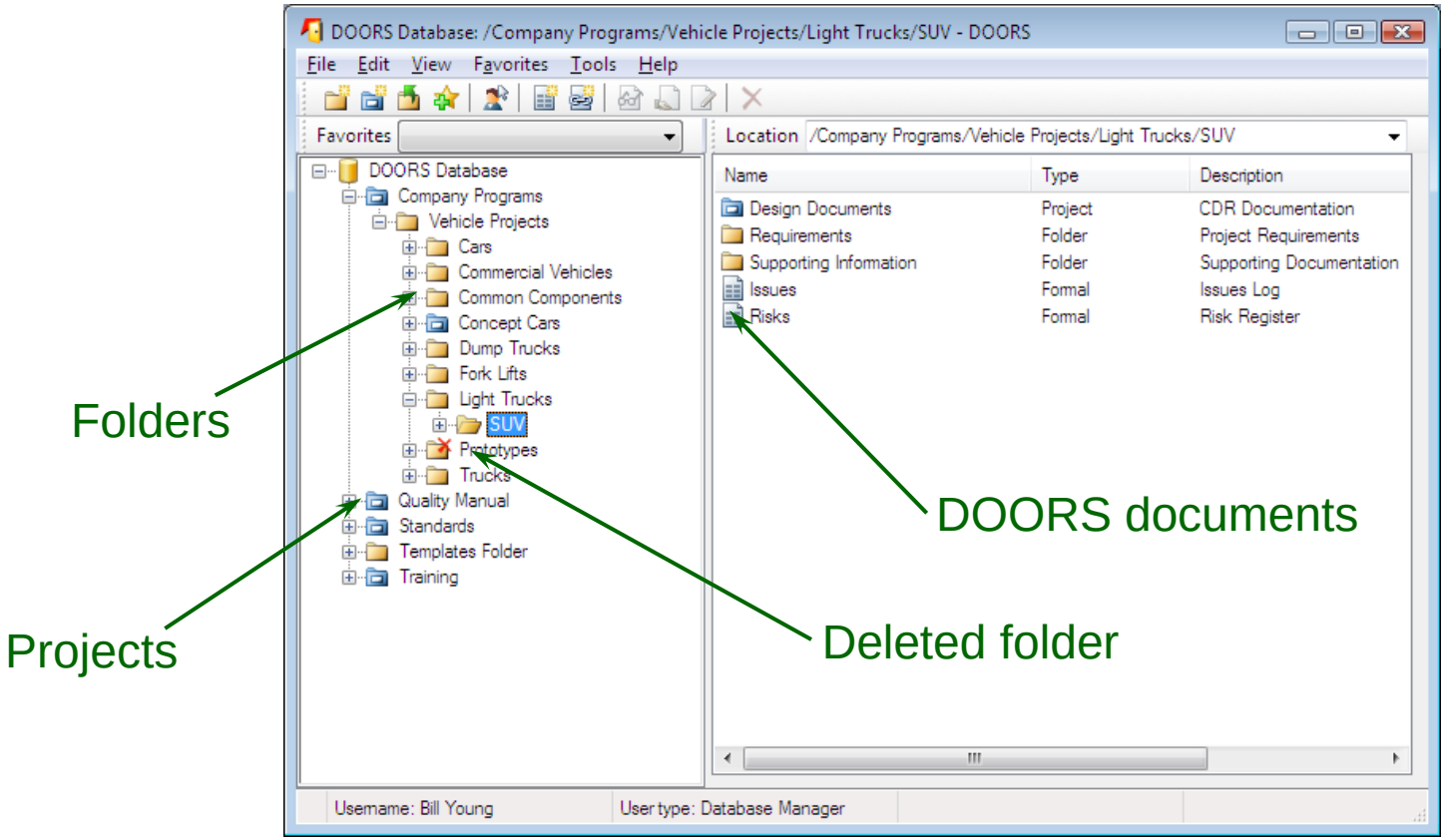
Printed from DOORS Page 2 of 12 Printed 01 December 1997

Is DOORS easy to use?

- The project is already in progress
  - How long will it take to get everybody using DOORS?
- People are used to other document and spreadsheet applications
  - Do users have to learn a totally new interface from the beginning?
- Documents are easy to understand
  - Do we have to understand databases to use DOORS?

DOORS database view

- Virtually unlimited hierarchy of projects/folders supports scalability



Help organize your projects

DOORS document views

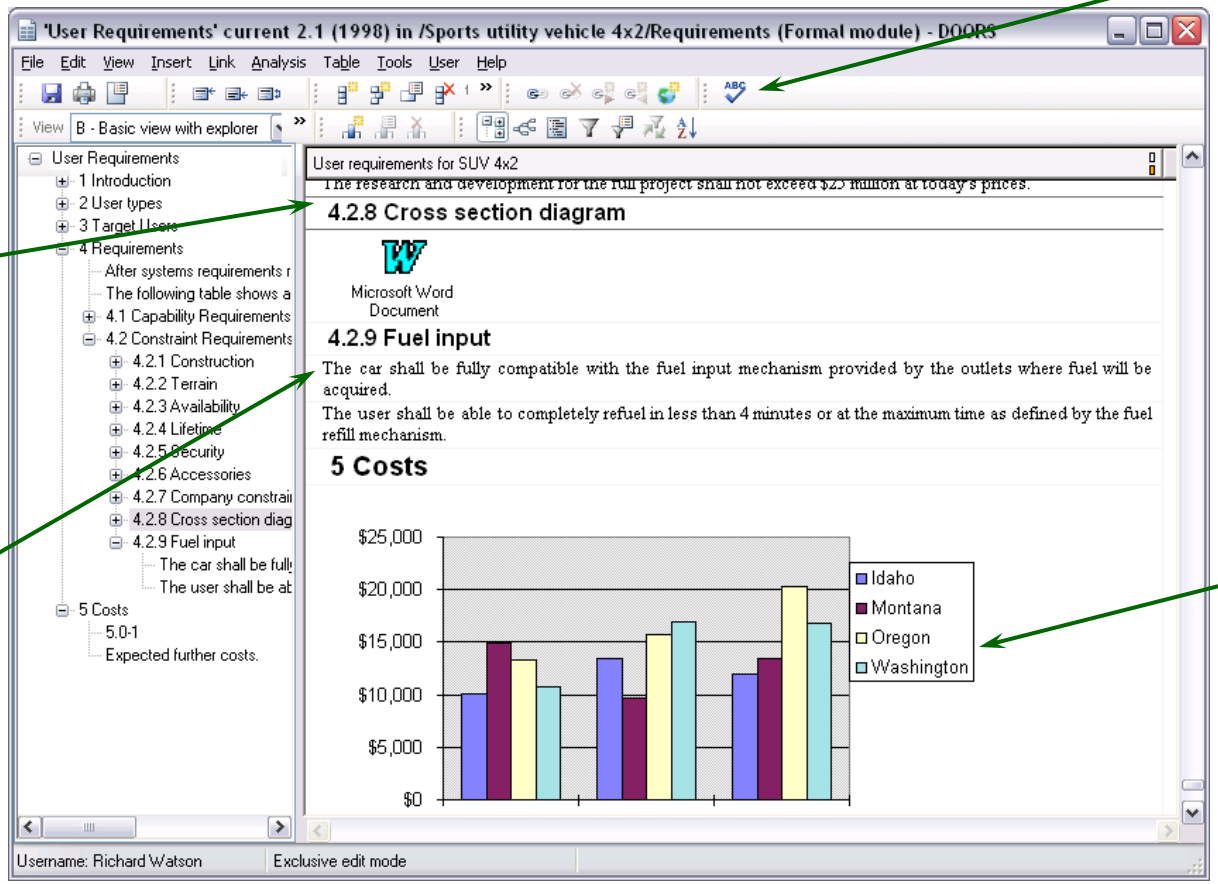
Virtually everything you need in one window!

Spell check

Context

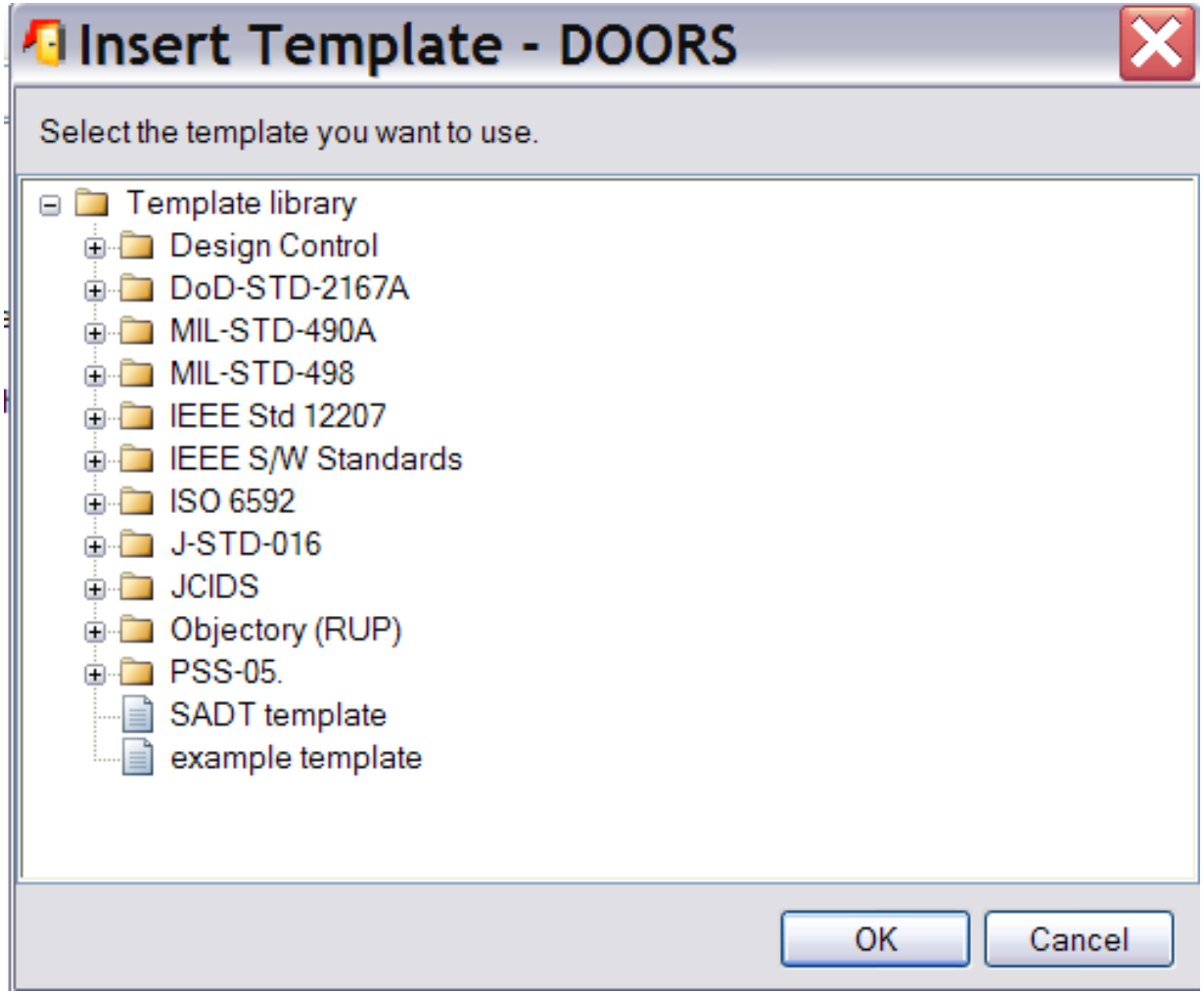
Requirements

Object Linking and Embedding (OLE)



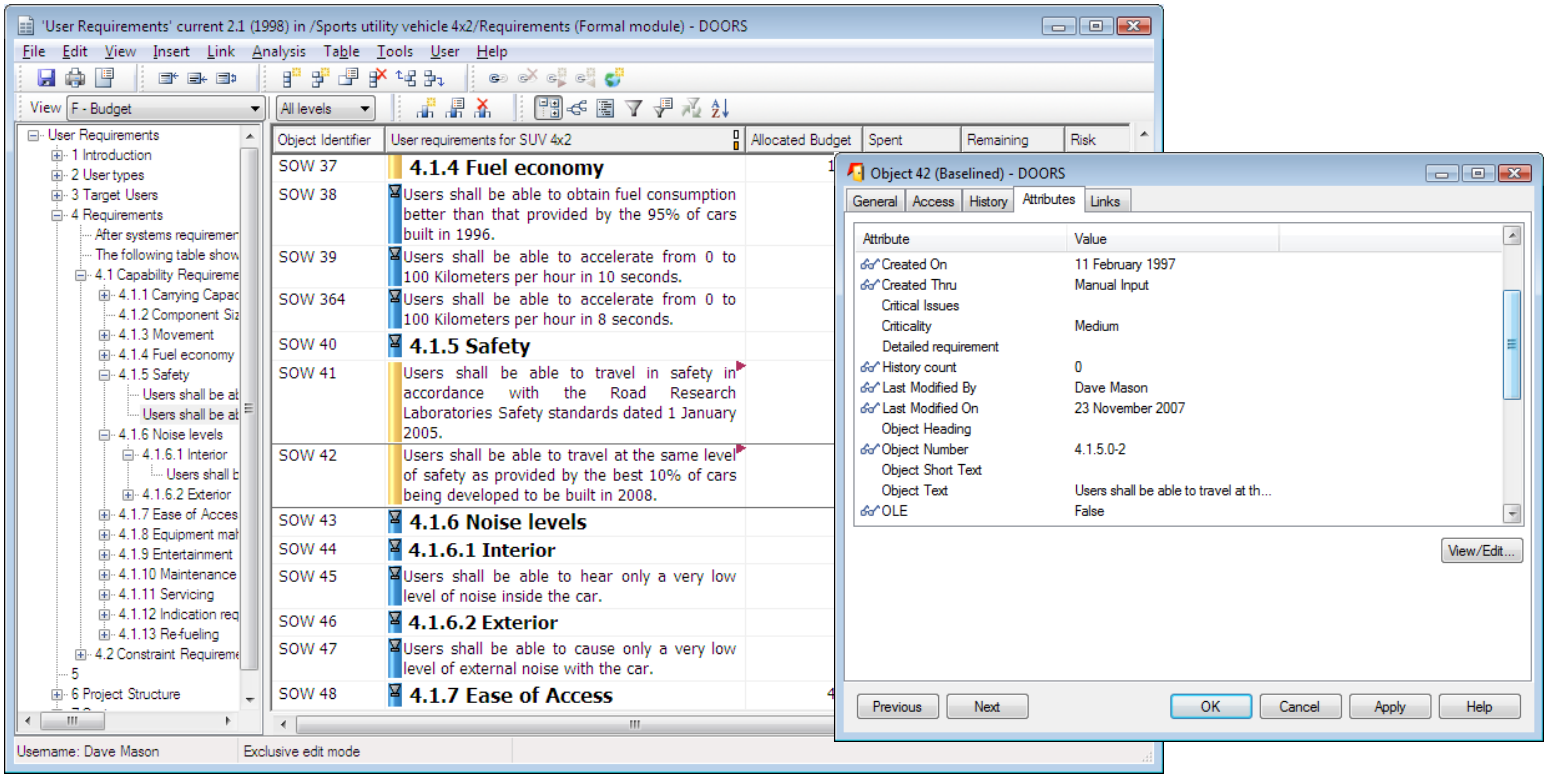
Help improve productivity, reduce errors and increase quality

Templates



Virtually unlimited user-defined attributes

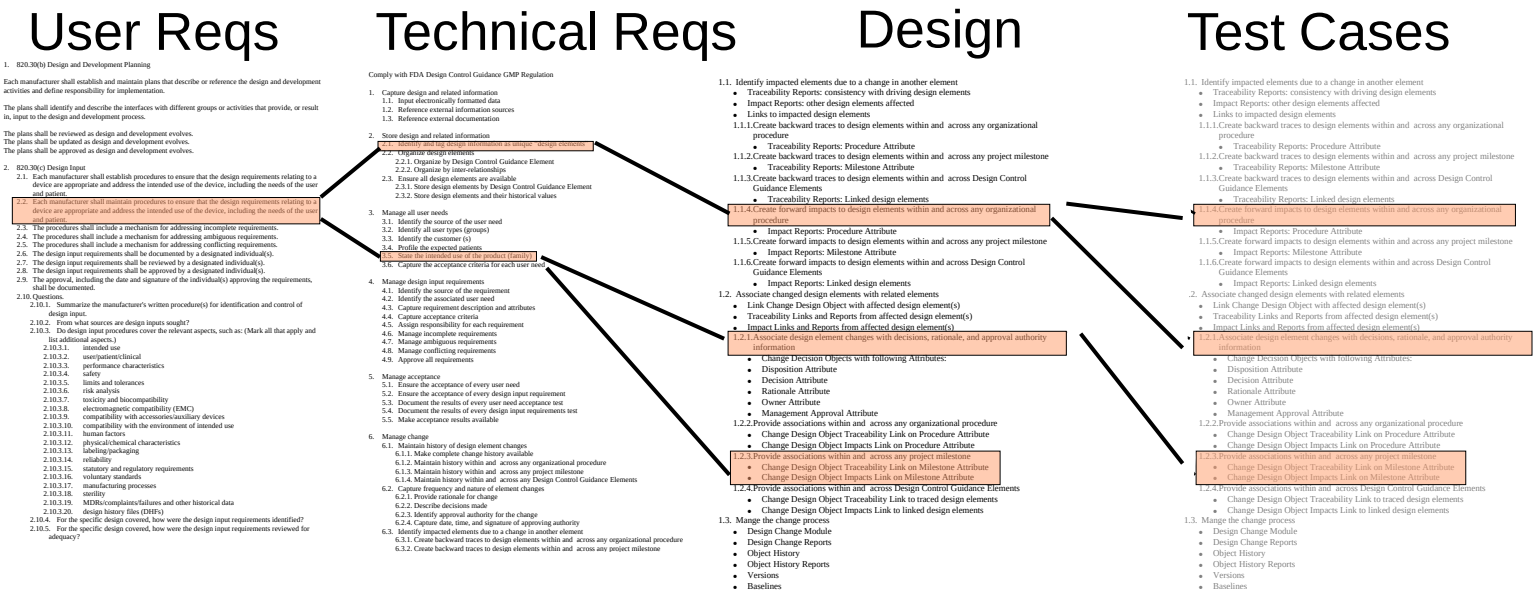
- Nearly unlimited number of attributes in a spreadsheet-like view
- Values can be calculated for metrics collection
- A value or attribute may be displayed in any column



How can I manage information traceability?

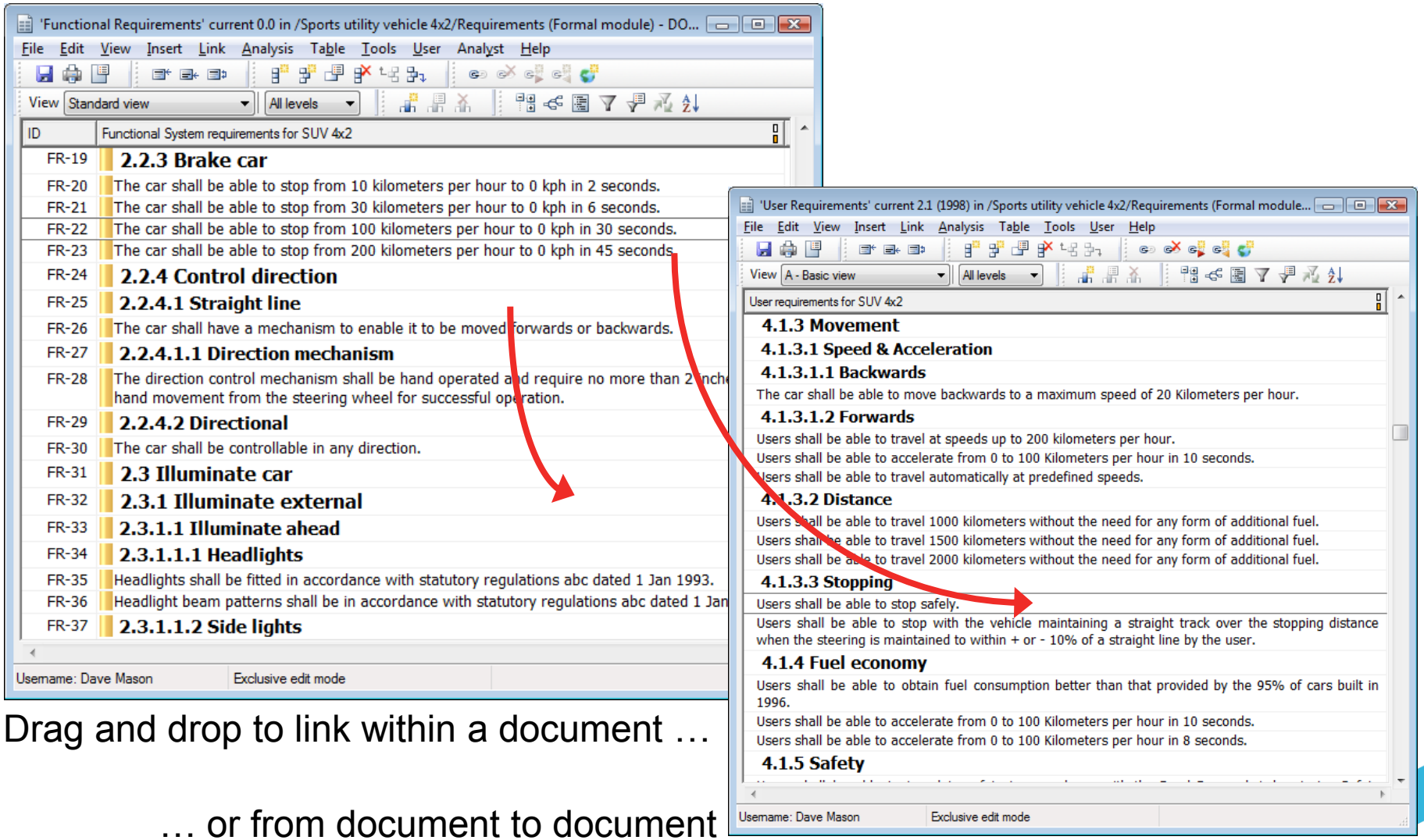
- We have never done traceability before
  - How much overhead is this going to add to a project?
- We must have detailed reports of impact
  - How comprehensive are the traceability reports?
- We need to see when requirements have been missed
  - Can we easily create queries to find “missing” links?
- We do incremental development with concurrent phases
  - How easy is it to keep traceability separate for each increment?

Traceability is key to compliance



- Initial user requirements should be decomposed to detailed requirements, and then to design, tests, etc.
- Decomposition creates traceability relationships
- Relationships define your traceability model
- Your traceability model is the basis for your process
- Enforce your traceability model; help improve your process

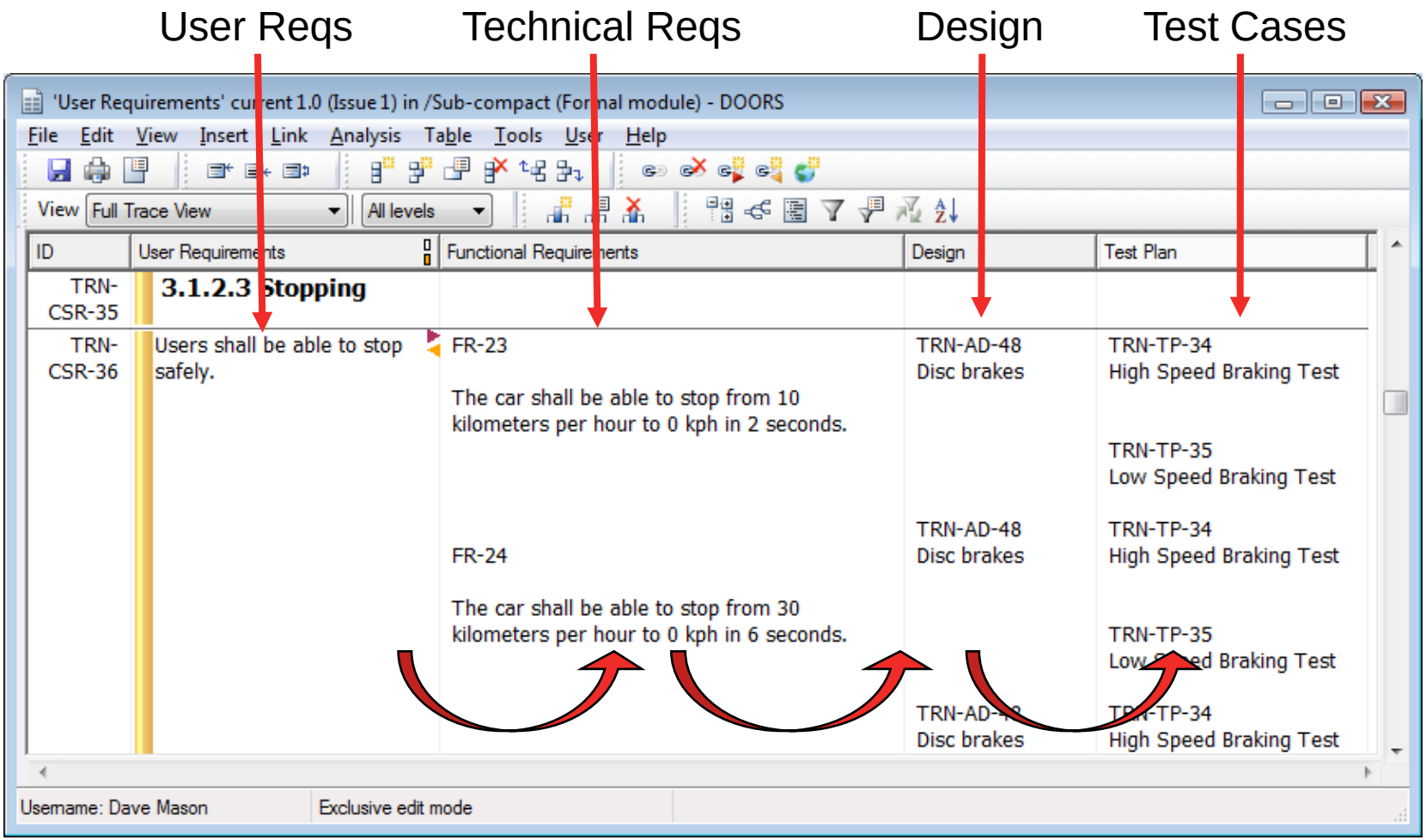
Traceability: drag-and-drop linking



Drag and drop to link within a document …

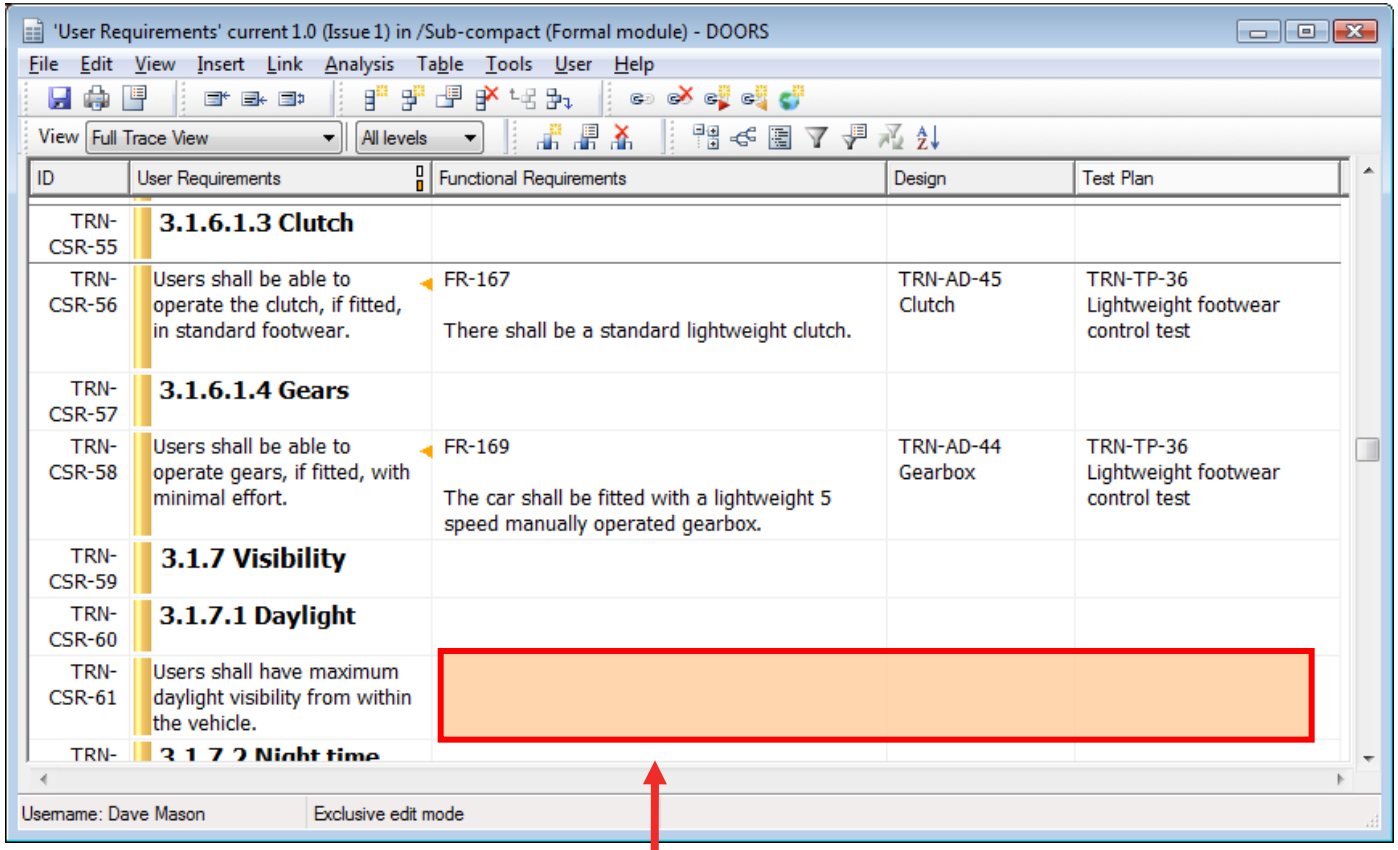
… or from document to document

Traceability view



End-to-end visual validation in a single view

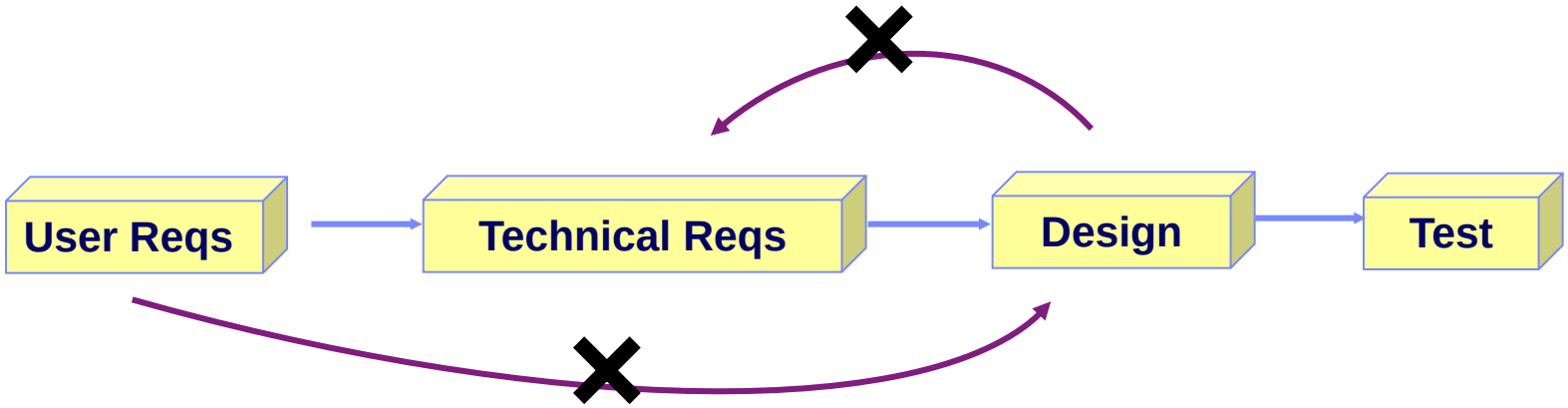
Traceability verification or “completeness”



- Helps increase customer confidence
- Detect missing links
- Creation and deletion of links are recorded in history

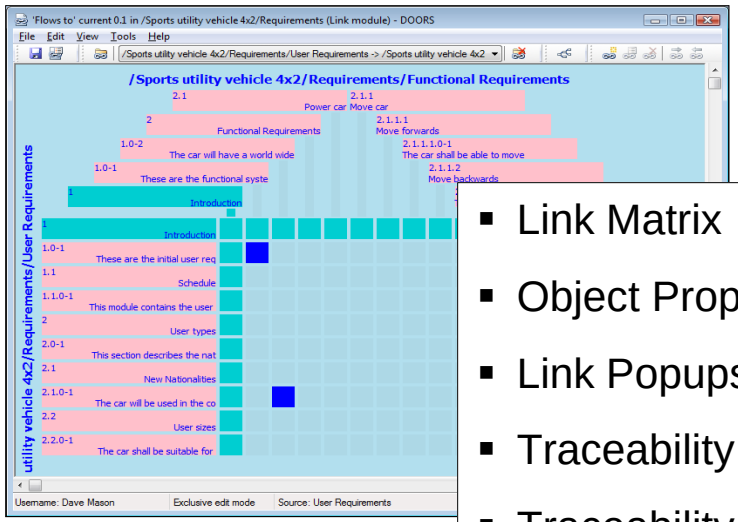
Traceability through an orphan report shows “missing” links

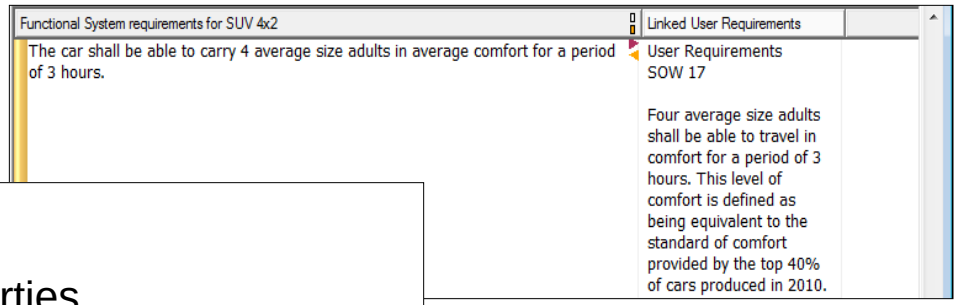
Define your process using enforced relationships



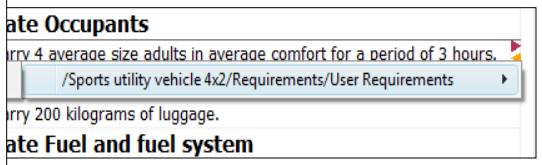
1. Define the legal relationships for your process
2. Make other links illegal; don't miss steps in the process
3. Help prevent tracing in the wrong direction
   - Enforce standards and help ensure consistency

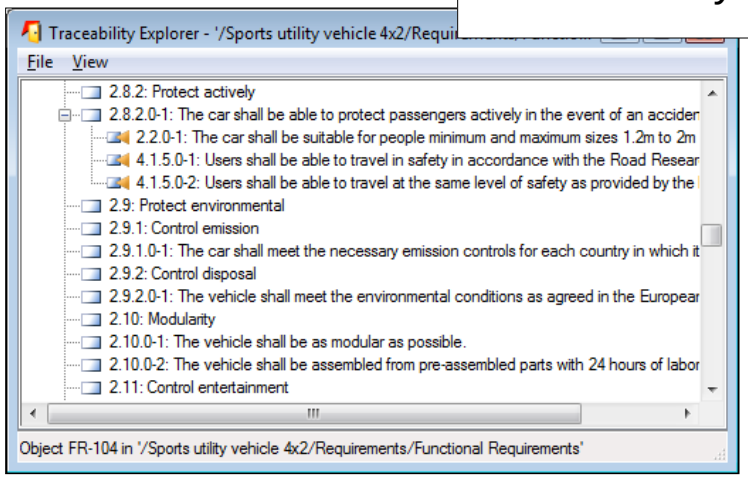
Standard DOORS traceability tools

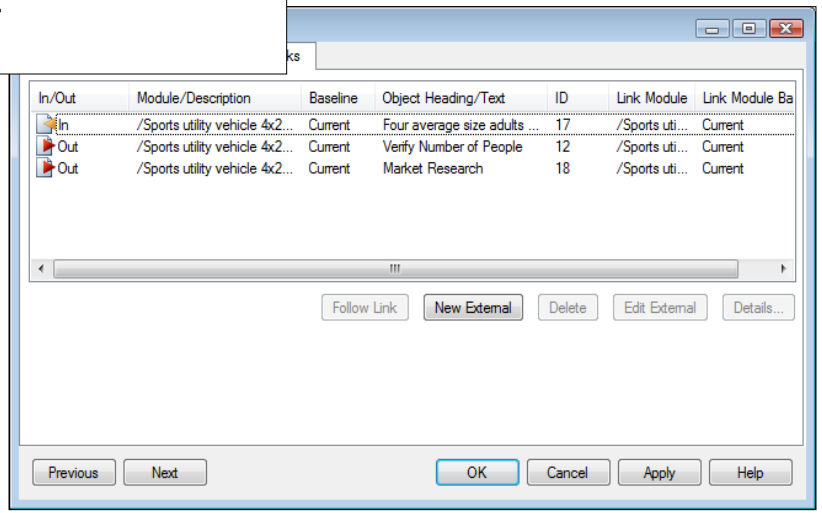




- Link Matrix
- Object Properties
- Link Popups
- Traceability Columns
- Traceability Explorer

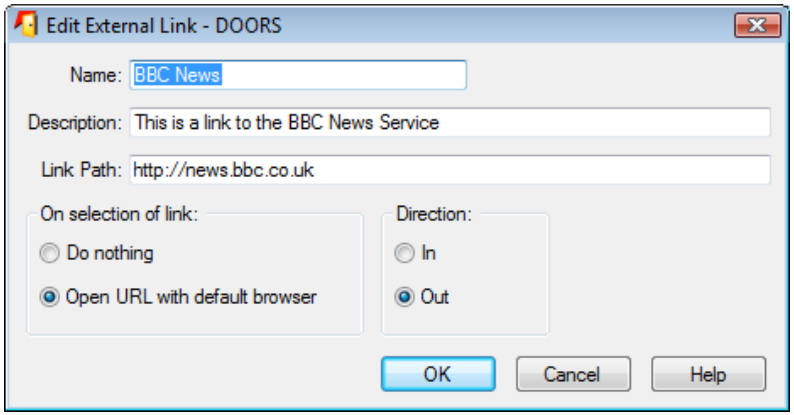


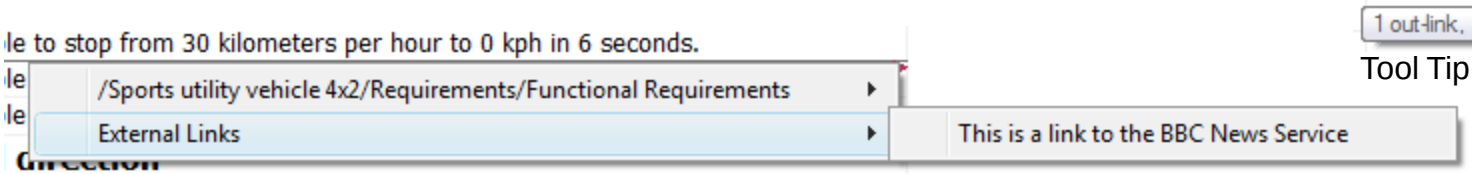




Traceability taking you outside of DOORS

- Everybody should understand the importance of requirements and be able to demonstrate that they meet requirements
- By extending traceability to go beyond the boundaries of DOORS, more people are encouraged to work against requirements
- Create links from DOORS to elements stored within applications outside of DOORS

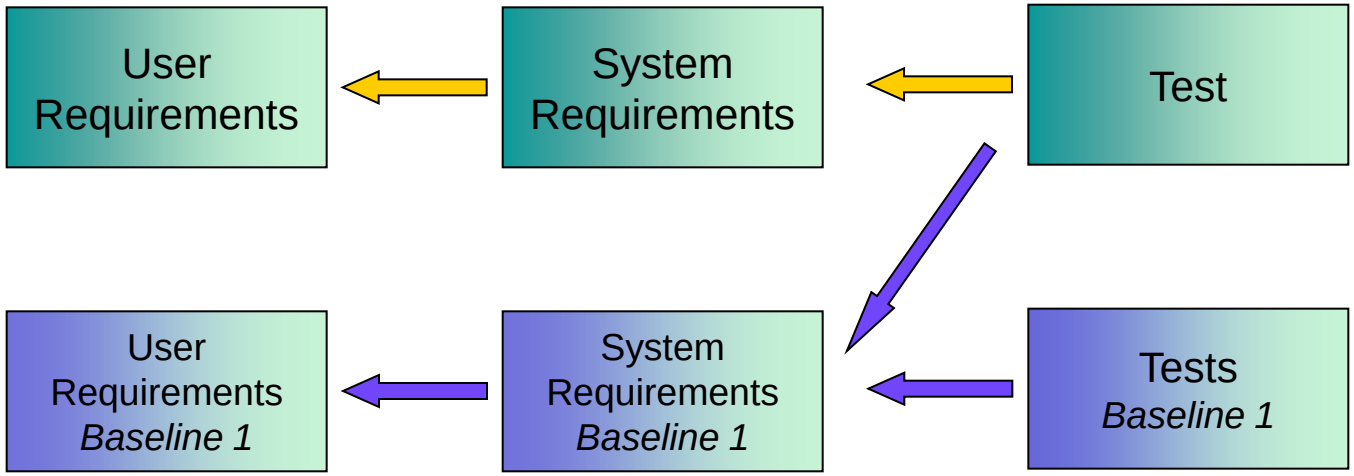




Right click the link indicator

Tool Tip on link indicator

Traceability to baselines



- Traceability is most often created between current information
- But some documents need to be baselined before other documents
- Then you need to link to the baseline for compliance with that phase
- When the final baseline is made, historical traceability is complete

Complex traceability in iterative development

Document 1

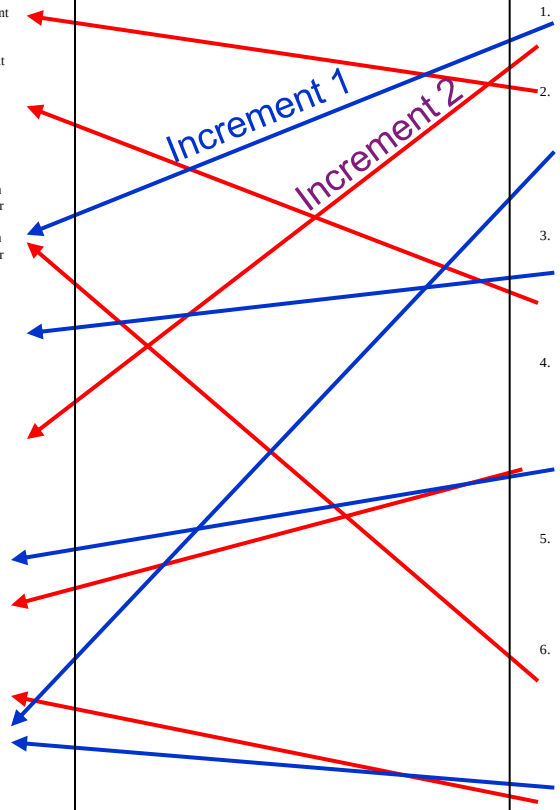
1. 820.30(b) Design and Development Planning

Each manufacturer shall establish and maintain plans that describe or reference the design and development activities and define responsibility for implementation.

The plans shall identify and describe the interfaces with different groups or activities that provide, or result in, input to the design and development process.

The plans shall be reviewed as design and development evolves.
The plans shall be updated as design and development evolves.
The plans shall be approved as design and development evolves.

2. 820.30(c) Design Input
   - 2.1. Each manufacturer shall establish procedures to ensure that the design requirements relating to a device are appropriate and address the intended use of the device, including the needs of the user and patient.
   - 2.2. Each manufacturer shall maintain procedures to ensure that the design requirements relating to a device are appropriate and address the intended use of the device, including the needs of the user and patient.
   - 2.3. The procedures shall include a mechanism for addressing incomplete requirements.
   - 2.4. The procedures shall include a mechanism for addressing ambiguous requirements.
   - 2.5. The procedures shall include a mechanism for addressing conflicting requirements.
   - 2.6. The design input requirements shall be documented by a designated individual(s).
   - 2.7. The design input requirements shall be reviewed by a designated individual(s).
   - 2.8. The design input requirements shall be approved by a designated individual(s).
   - 2.9. The approval, including the date and signature of the individual(s) approving the requirements, shall be documented.
   - 2.10. Questions.
     - 2.10.1. Summarize the manufacturer's written procedure(s) for identification and control of design input.
     - 2.10.2. From what sources are design inputs sought?
     - 2.10.3. Do design input procedures cover the relevant aspects, such as: (Mark all that apply and list additional aspects.)
       - 2.10.3.1. intended use
       - 2.10.3.2. user/patient/clinical
       - 2.10.3.3. performance characteristics
       - 2.10.3.4. safety
       - 2.10.3.5. limits and tolerances
       - 2.10.3.6. risk analysis
       - 2.10.3.7. toxicity and biocompatibility
       - 2.10.3.8. electromagnetic compatibility (EMC)
       - 2.10.3.9. compatibility with accessories/auxiliary devices
       - 2.10.3.10. compatibility with the environment of intended use
       - 2.10.3.11. human factors
       - 2.10.3.12. physical/chemical characteristics
       - 2.10.3.13. labeling/packaging
       - 2.10.3.14. reliability
       - 2.10.3.15. statutory and regulatory requirements
       - 2.10.3.16. voluntary standards
       - 2.10.3.17. manufacturing processes
       - 2.10.3.18. sterility
       - 2.10.3.19. MDRs/complaints/failures and other historical data
       - 2.10.3.20. design history files (DHFs)
     - 2.10.4. For the specific design covered, how were the design input requirements identified?
     - 2.10.5. For the specific design covered, how were the design input requirements reviewed for adequacy?

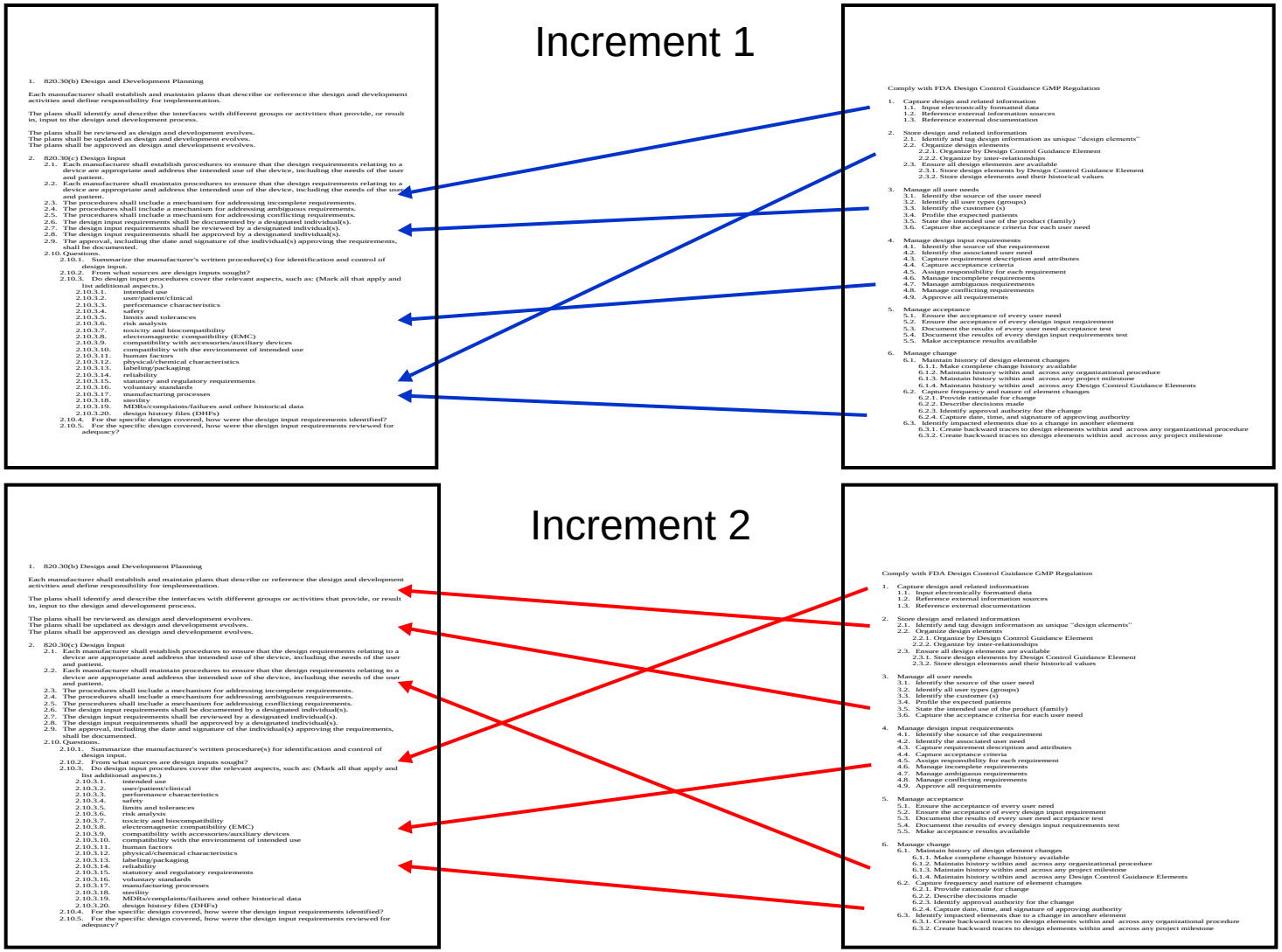


Document 2

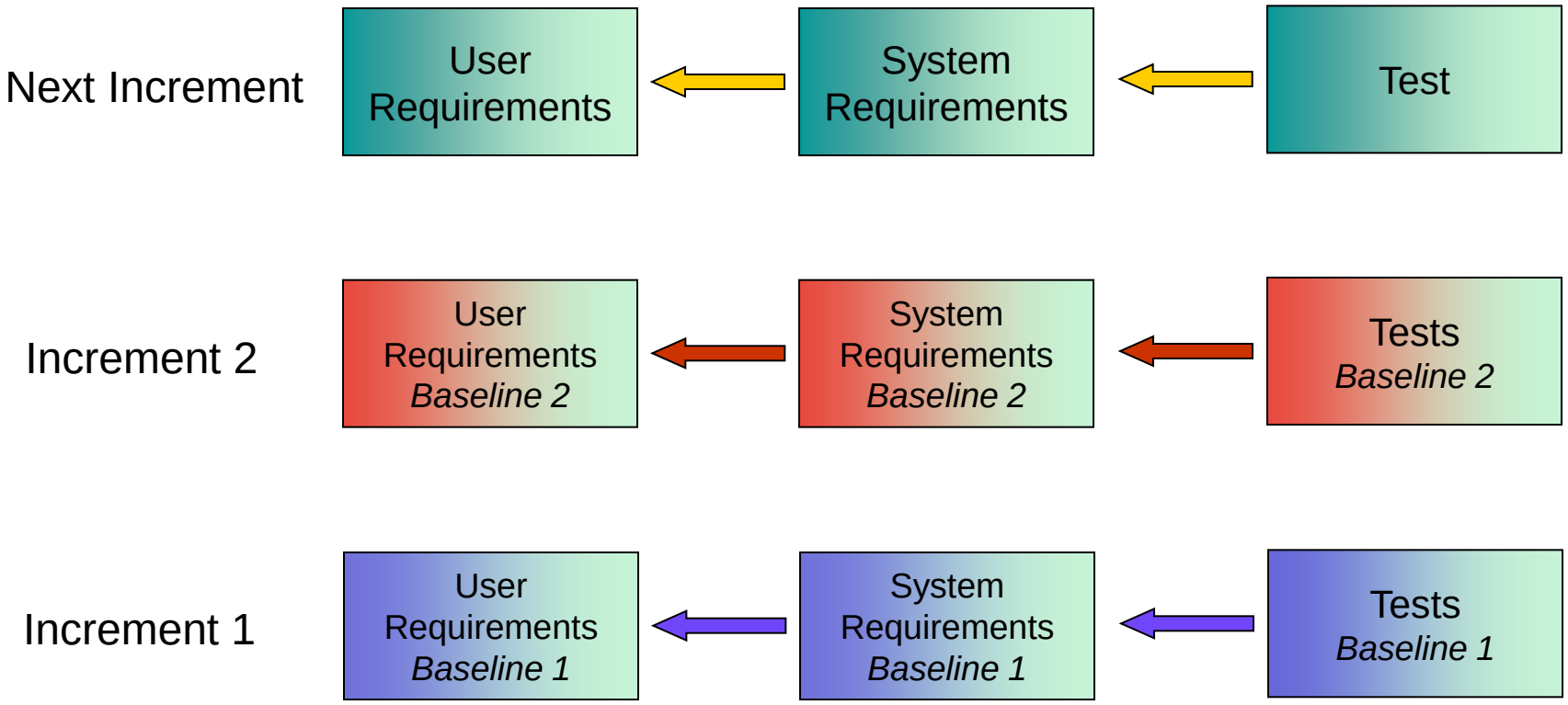
Comply with FDA Design Control Guidance GMP Regulation

1. Capture design and related information
   - 1.1. Input electronically formatted data
   - 1.2. Reference external information sources
   - 1.3. Reference external documentation
2. Store design and related information
   - 2.1. Identify and tag design information as unique "design elements"
   - 2.2. Organize design elements
     - 2.2.1. Organize by Design Control Guidance Element
     - 2.2.2. Organize by inter-relationships
   - 2.3. Ensure all design elements are available
     - 2.3.1. Store design elements by Design Control Guidance Element
     - 2.3.2. Store design elements and their historical values
3. Manage all user needs
   - 3.1. Identify the source of the user need
   - 3.2. Identify all user types (groups)
   - 3.3. Identify the customer (s)
   - 3.4. Profile the expected patients
   - 3.5. State the intended use of the product (family)
   - 3.6. Capture the acceptance criteria for each user need
4. Manage design input requirements
   - 4.1. Identify the source of the requirement
   - 4.2. Identify the associated user need
   - 4.3. Capture requirement description and attributes
   - 4.4. Capture acceptance criteria
   - 4.5. Assign responsibility for each requirement
   - 4.6. Manage incomplete requirements
   - 4.7. Manage ambiguous requirements
   - 4.8. Manage conflicting requirements
   - 4.9. Approve all requirements
5. Manage acceptance
   - 5.1. Ensure the acceptance of every user need
   - 5.2. Ensure the acceptance of every design input requirement
   - 5.3. Document the results of every user need acceptance test
   - 5.4. Document the results of every design input requirements test
   - 5.5. Make acceptance results available
6. Manage change
   - 6.1. Maintain history of design element changes
     - 6.1.1. Make complete change history available
     - 6.1.2. Maintain history within and across any organizational procedure
     - 6.1.3. Maintain history within and across any project milestone
     - 6.1.4. Maintain history within and across any Design Control Guidance Elements
   - 6.2. Capture frequency and nature of element changes
     - 6.2.1. Provide rationale for change
     - 6.2.2. Describe decisions made
     - 6.2.3. Identify approval authority for the change
     - 6.2.4. Capture date, time, and signature of approving authority
   - 6.3. Identify impacted elements due to a change in another element
     - 6.3.1. Create backward traces to design elements within and across any organizational procedure
     - 6.3.2. Create backward traces to design elements within and across any project milestone

Simplifying complexity

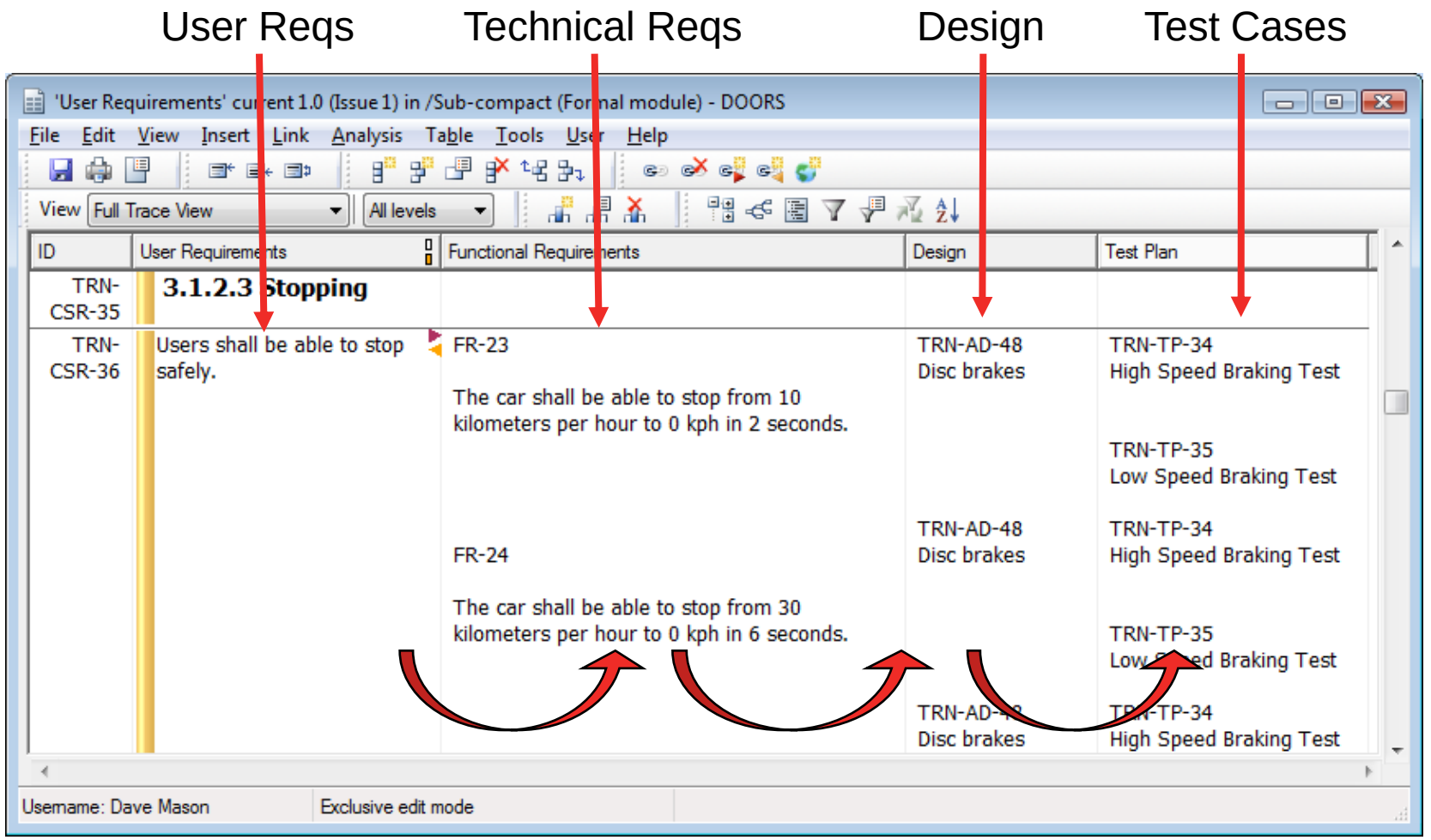


Intelligent traceability—managing multiple traceability streams



But I can do this with standard document and spreadsheet applications; how is DOORS better?

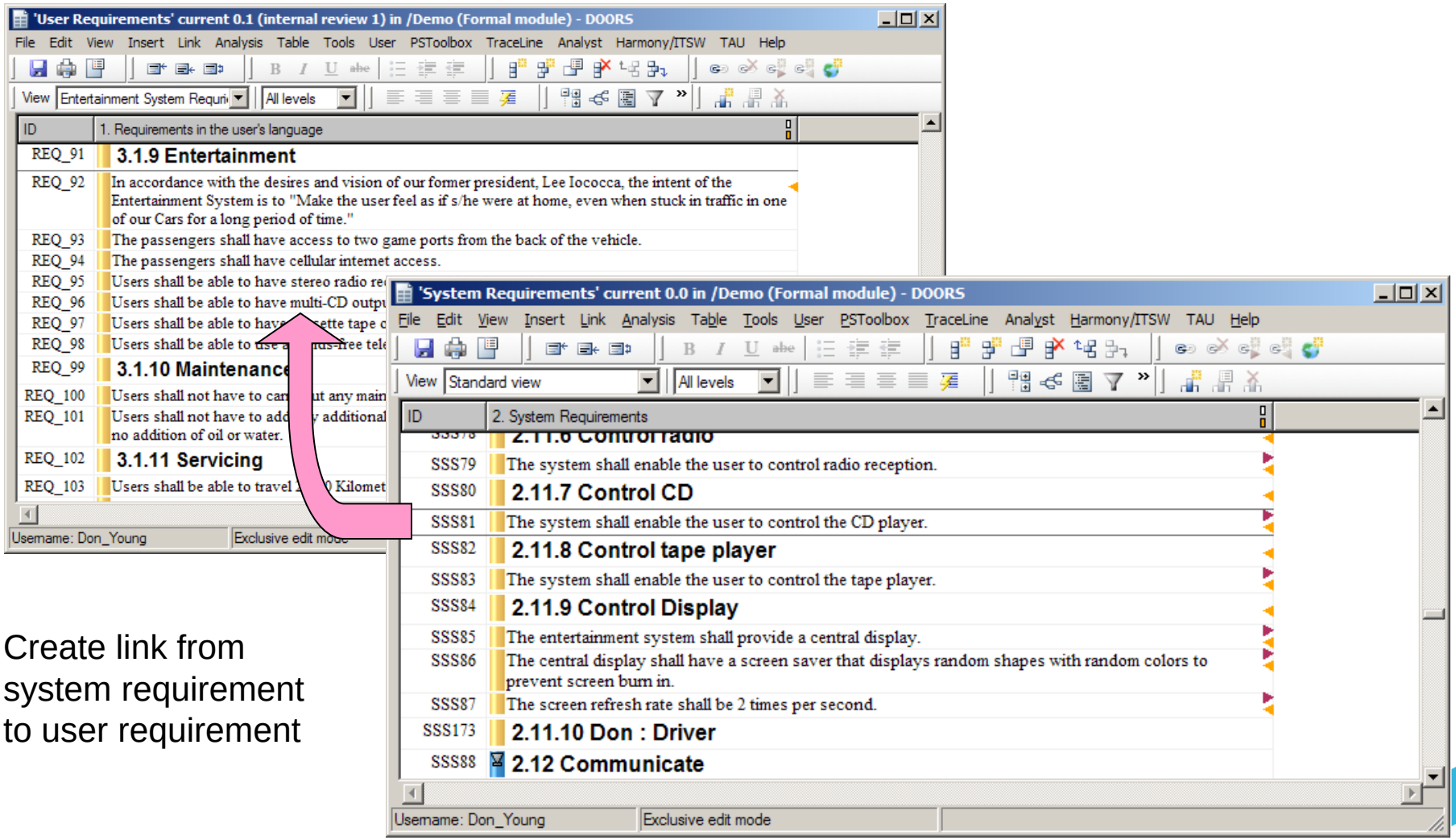
Traceability report



Comprehensive visual validation in a single view

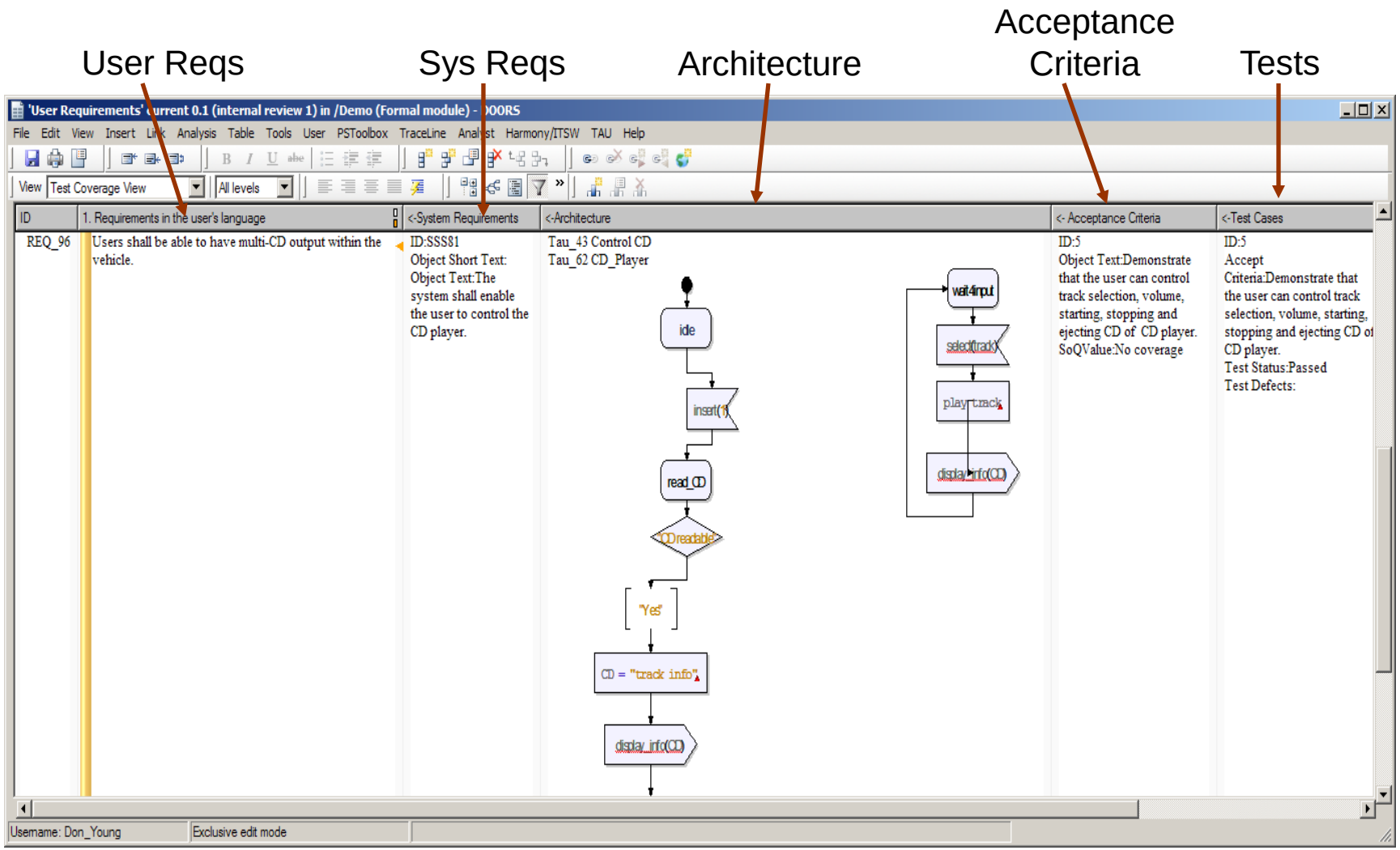
How do we do this quickly in DOORS?

Simply drag and drop to create links



Create link from system requirement to user requirement

Display comprehensive traceability in a single view



But I can do that with standard document and spreadsheet applications!

Yes, but at what cost in effort and maintenance?

Ask yourself this …

- What is the cost to the organization if:
  - We design and test against the wrong version of the requirement?
  - Completely miss a customer need or misinterpret due to incomplete or incorrect visibility to the information hierarchy?
  - The parent requirement is changed, and affected organizations do not get visibility to that change?

Potential cost/benefit factors

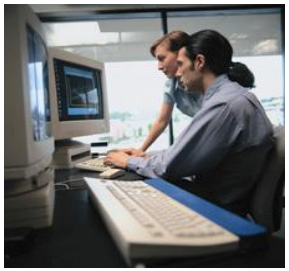
- Potential costs
  - Investments (software, training, consulting, etc.)
  - Tool use overhead
  - Added review and rigor
- Potential benefits
  - Savings from staff working more efficiently
  - Avoiding the cost of lost requirements
  - Avoiding the cost of unnecessary development and maintenance
  - Reducing the cost of requirements-related defects

How can I find changes easily?

- Changes can happen overnight
  - Can DOORS tell me if a change affects my work?
- Sometimes we need to look at older requirements
  - Can I see a history of each requirement?
- Milestones are very important to project progress
  - Can we take a snapshot of the requirements at any milestone?

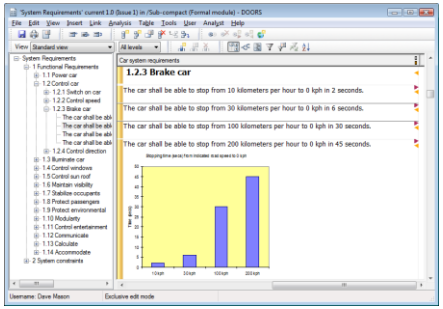
What are suspect links?

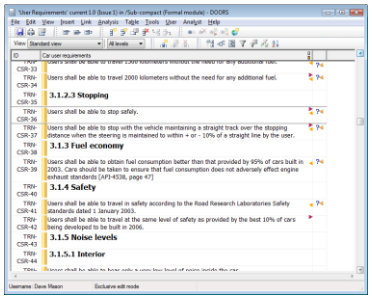
If documents are linked …



… a change by this user here …

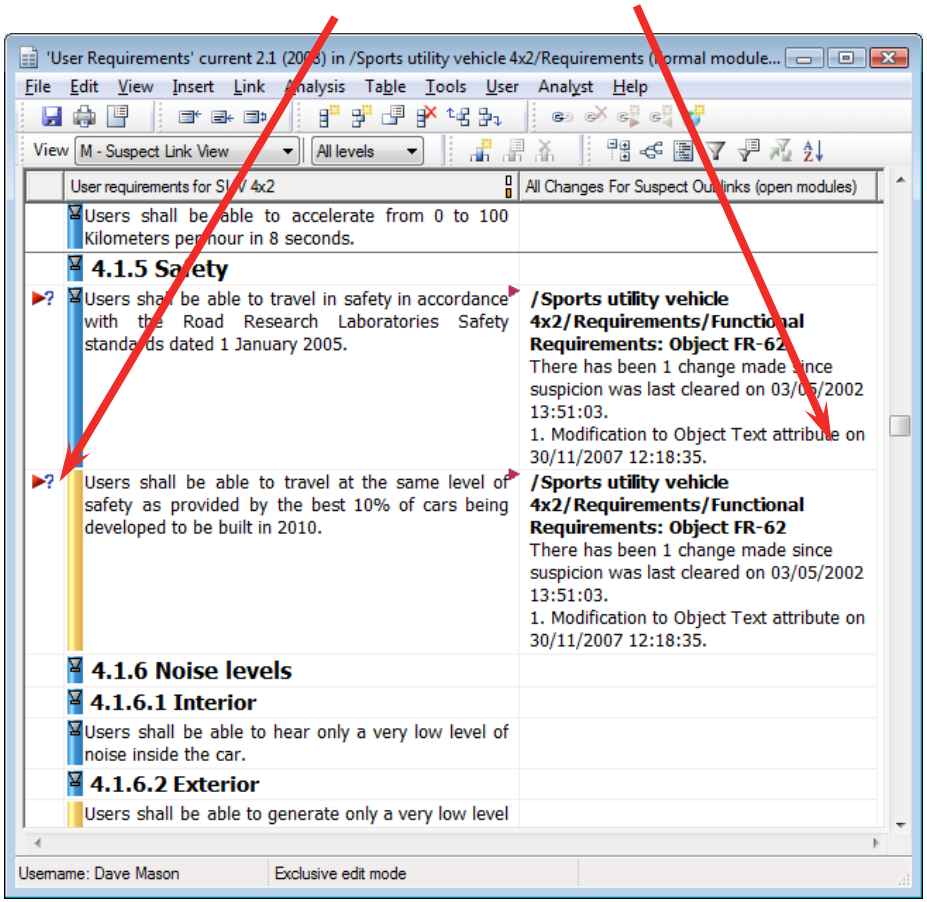
… shows up as a warning flag to this user here.

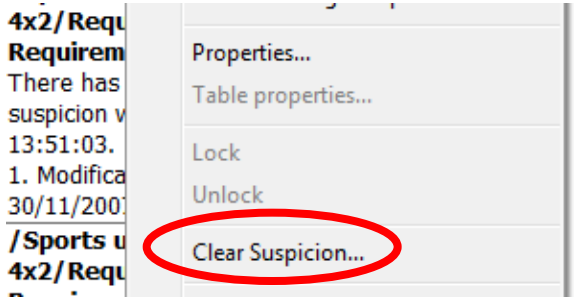




Suspect links

Suspect links are visible directly in the document—as indicators or as a description

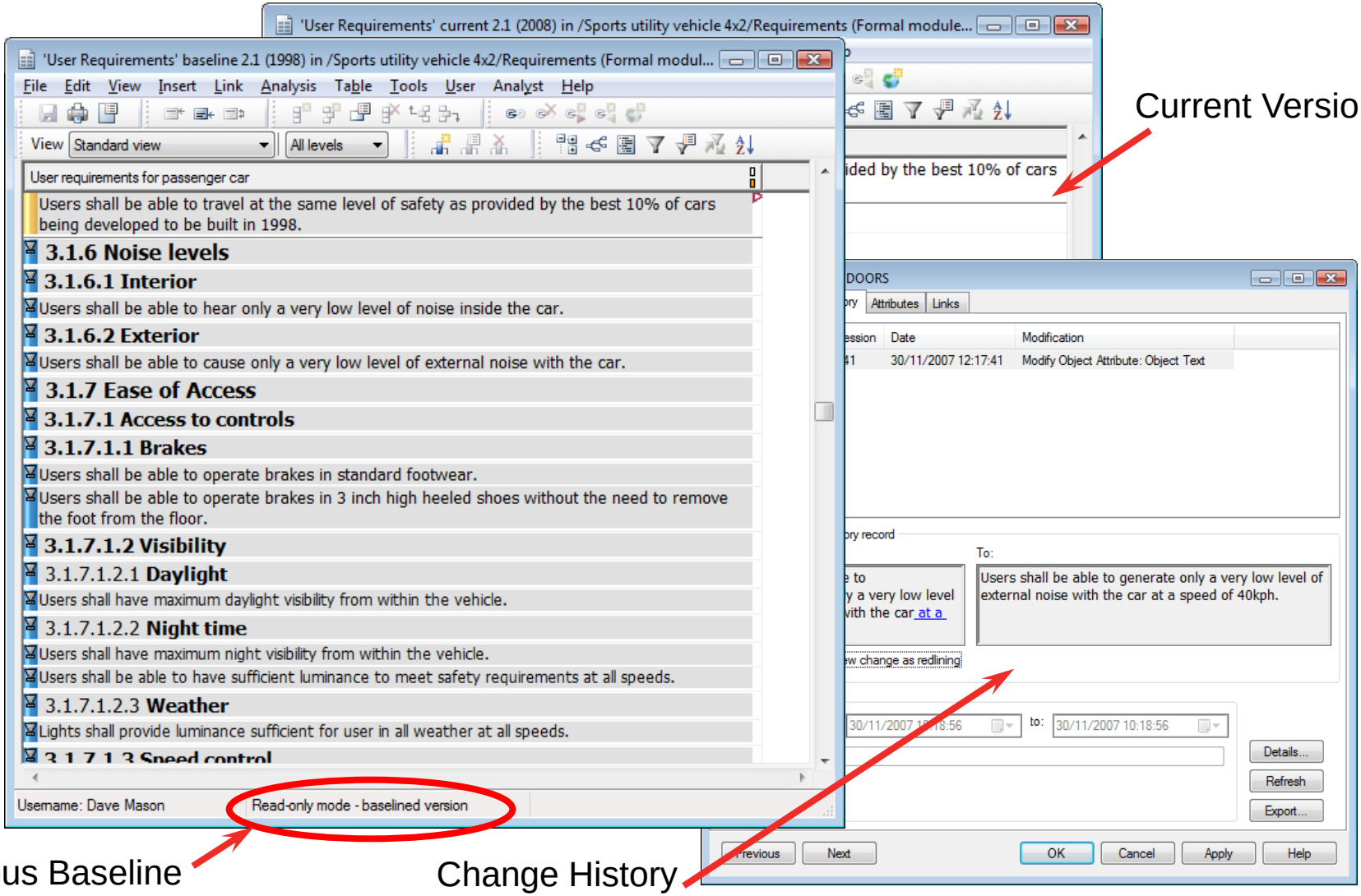




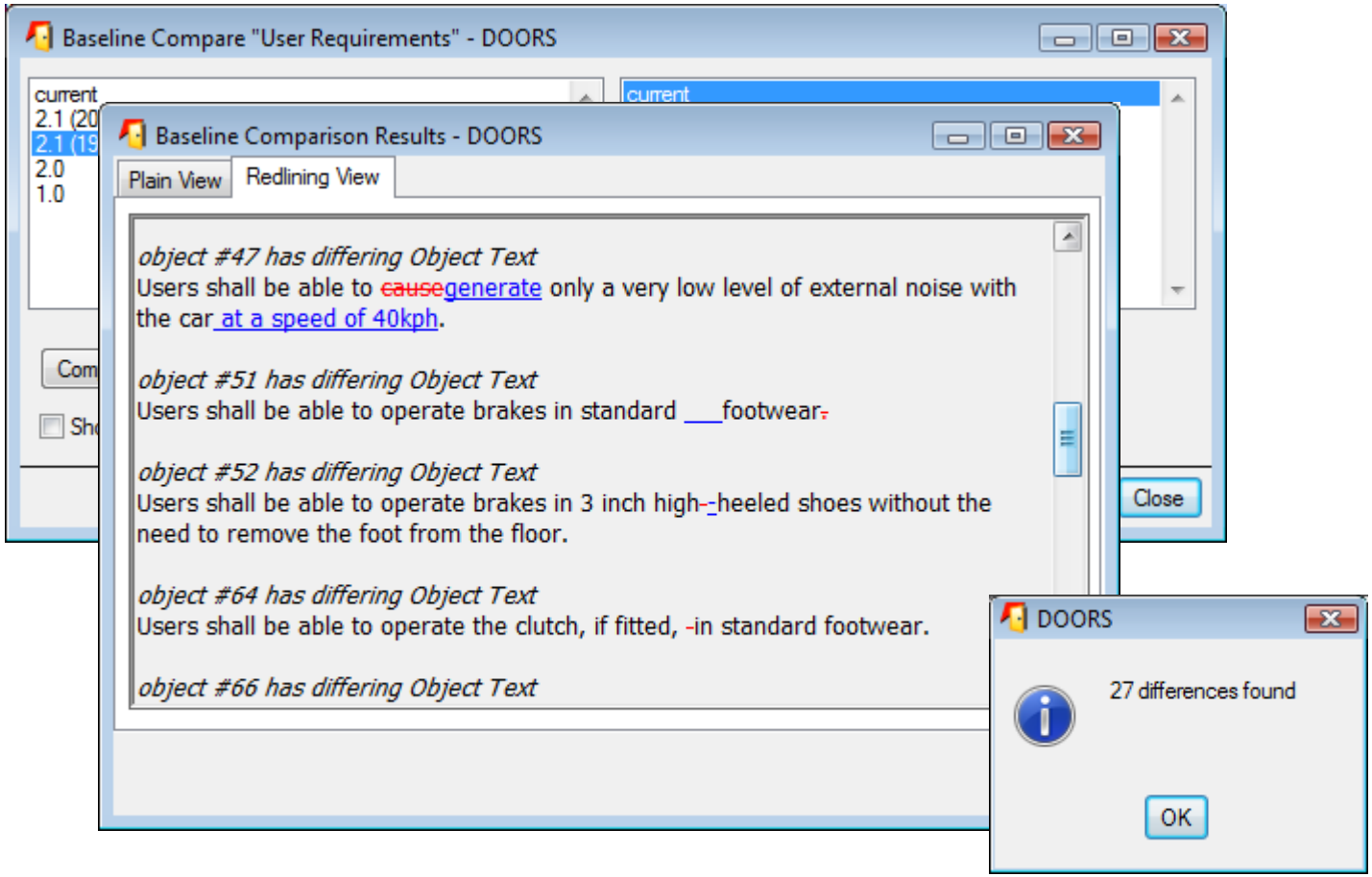
Clear on a right click

Help ensure that you never miss a change

History and baselines



Baseline compare

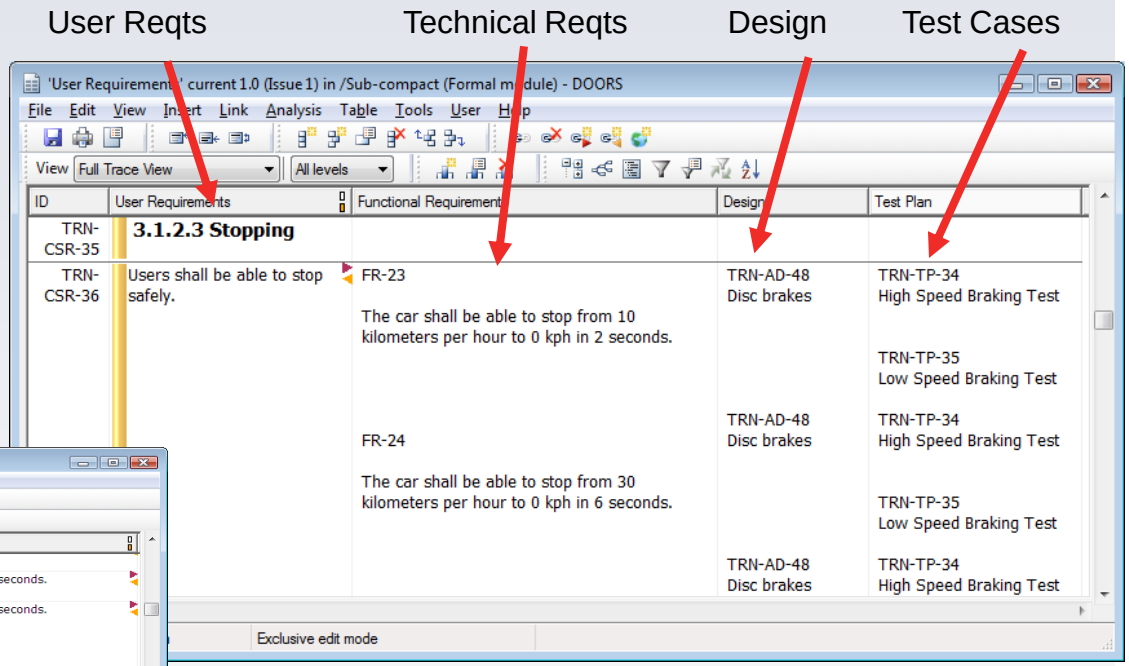


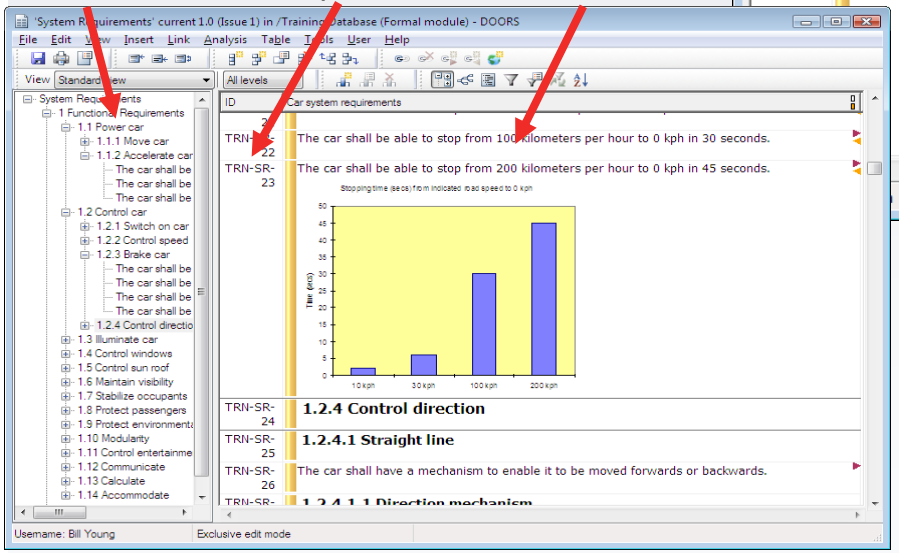
Provides a concise list of differences as a single report

Rational DOORS

Manage All Requirements Across the Lifecycle and Across Disciplines

- Combined document and spreadsheet views
- Simple, intuitive interfaces for easy adoption
- History and baselines





Writing Requirements within Context

End-to-end visual validation in a single view

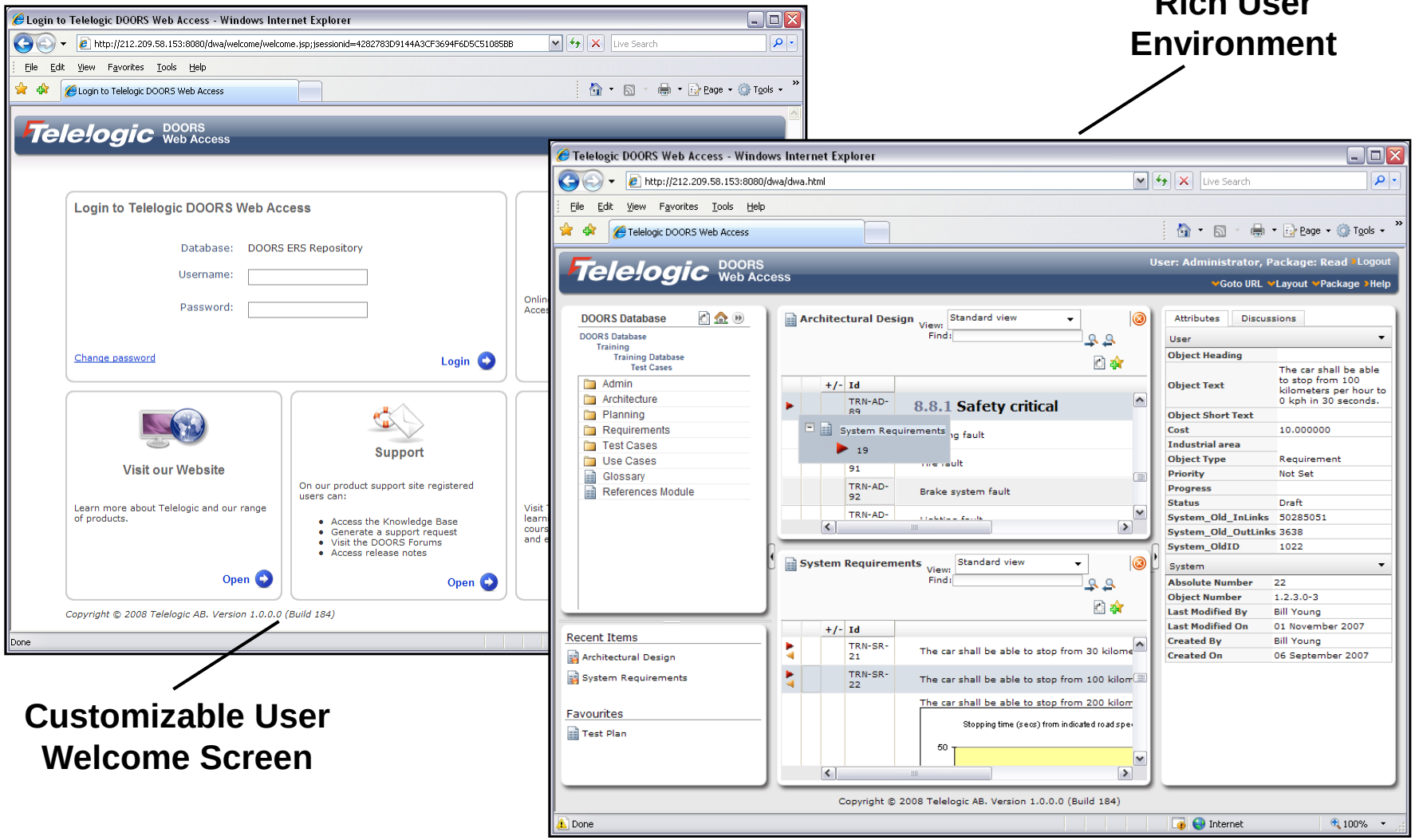
- Input and output from/to various common formats

Solve the right problem because the requirements are visible at all times

Good Requirements Management Allows Deep Analysis

- Query attributes to find specific properties
  - *"How many requirements are listed as high risk?"*
- Use traceability reports for checking dependencies
  - Before change is committed
- Find "missing" links
  - *"Which detailed requirements has no relation to a high-level user requirement?"*
- Coverage analysis
  - *"Which higher level requirement has no lower-level requirement?"*
- Impact analysis
  - *"What lower level requirements are affected if a high level requirement changes?"*
- Keep traceability
  - For each increment, if you develop incrementally with concurrent phases
  - For each variant, if you manage variants and product lines

The DOORS Web Access environment



What is Model Driven Systems Development (MDSD)?

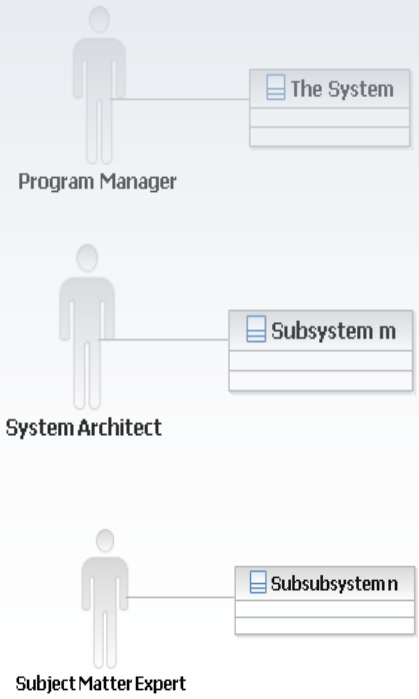
A structured approach for the development of complex systems across the mechanical, electronic and software disciplines

- Ensures that all requirements are fulfilled
- Employs models as the primary artifacts throughout systems development
- Facilitates improved communication among all stakeholders
- Provides a disciplined way to manage complexity through abstraction
- Improves quality through integration of testing with development
- Allows specification and development of software that controls the system and enables its use

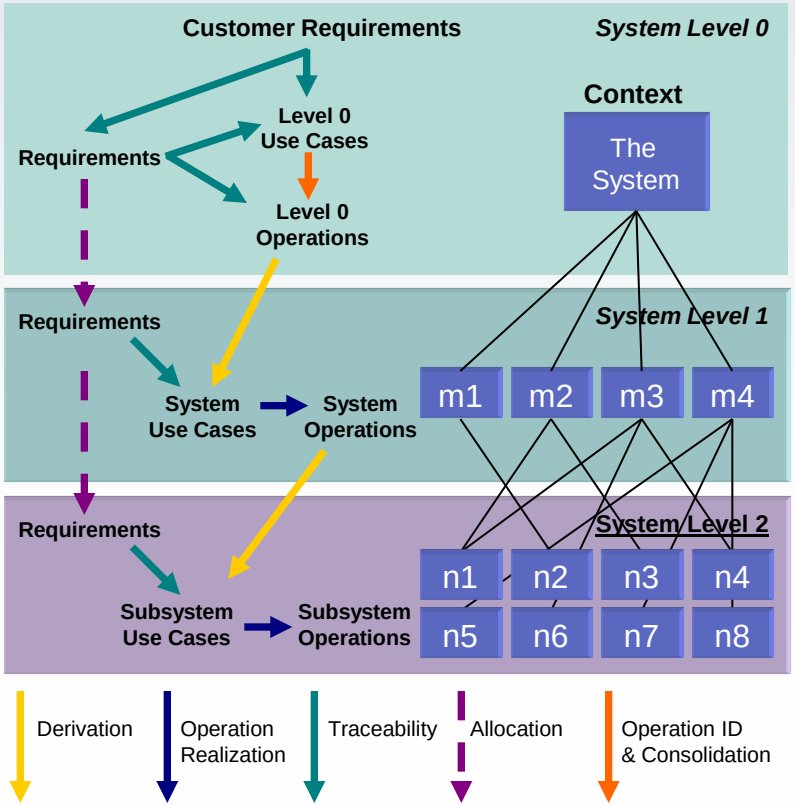
Model Driven Systems Development Approach

- Decompose the system
- Derive requirements and write specifications for the system and each subsystem in its own context

System Context



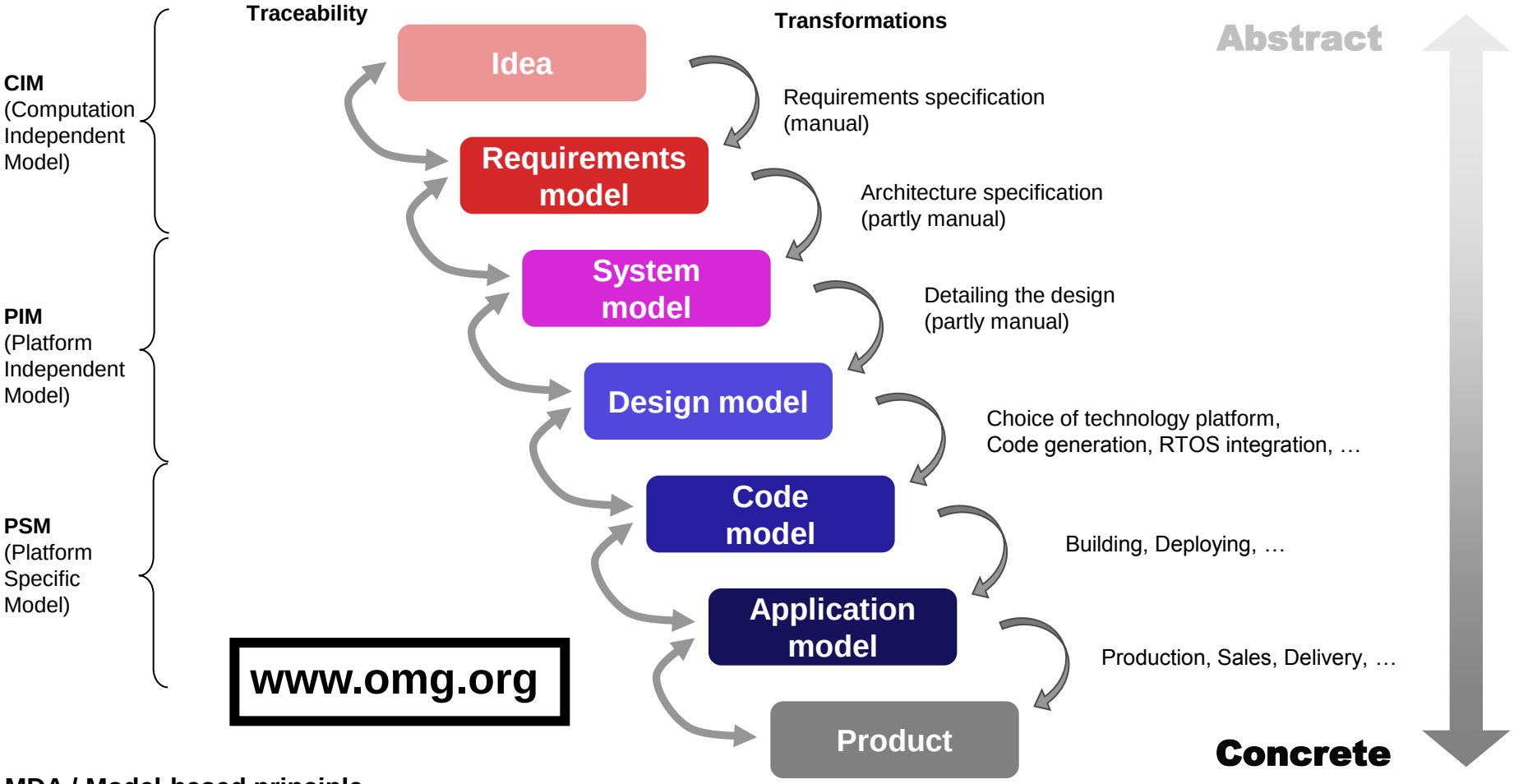
System Decomposition



Key MDSD Characteristics:

- Systems and subsystems are addressed one at a time, in their own context
- Requirements and specifications are done in context
- You are done when all subsystems are fully defined

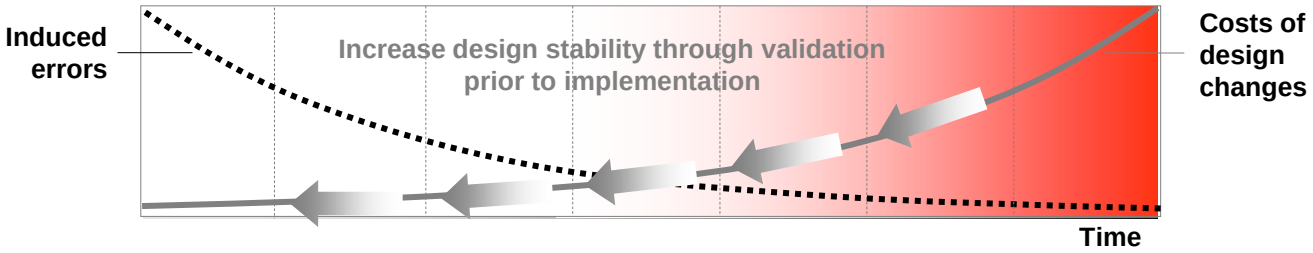
MDA

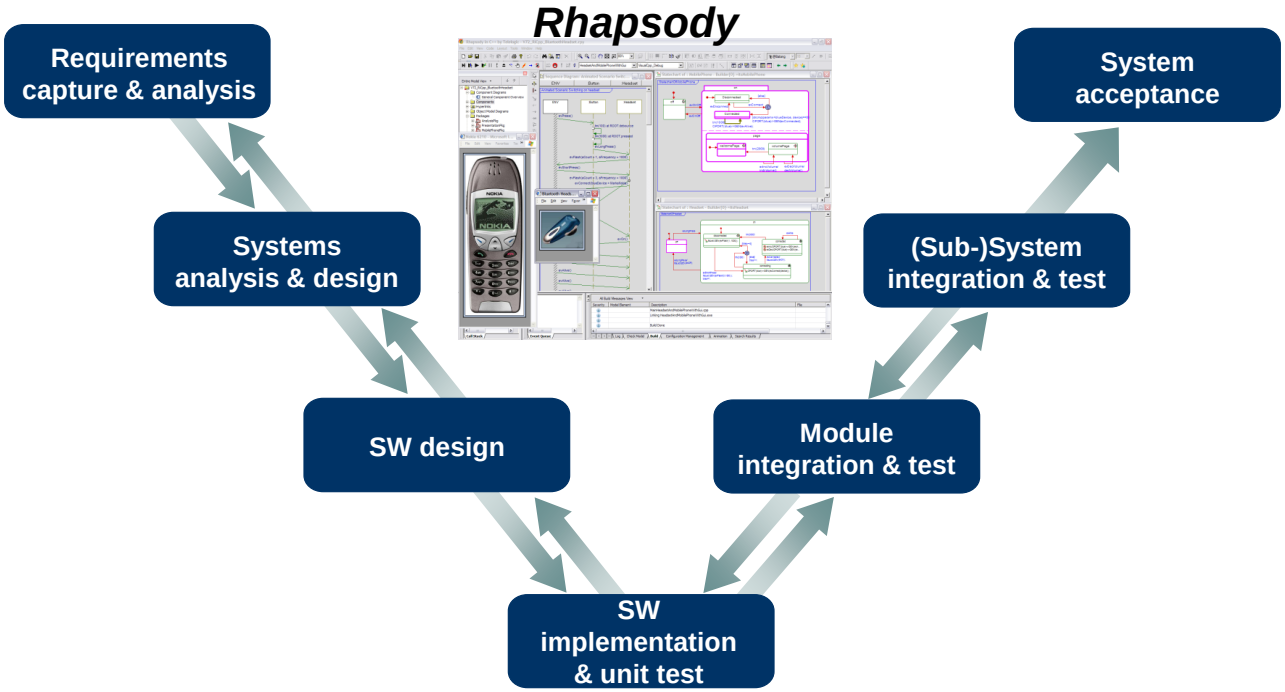


MDA / Model-based principle

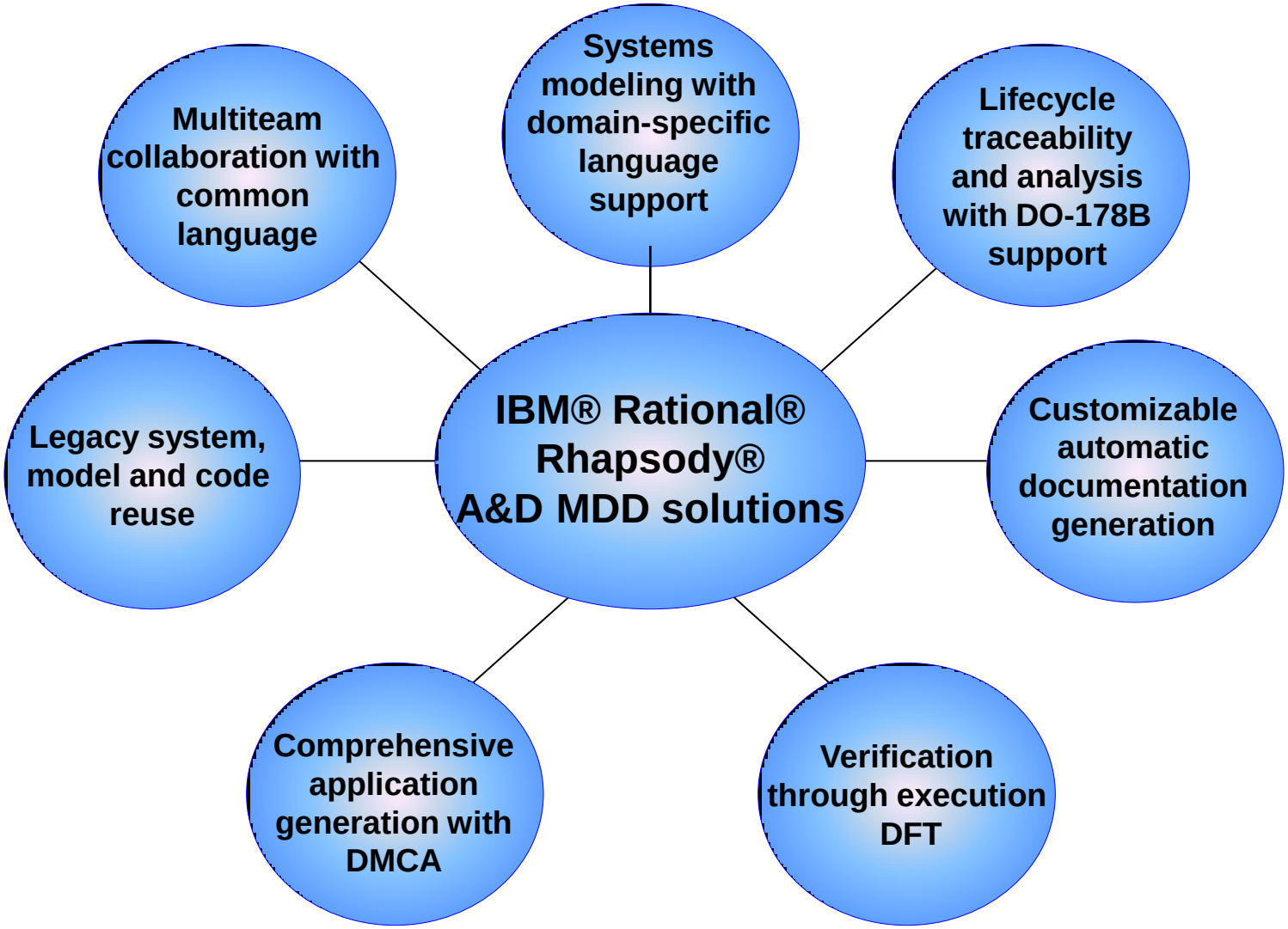
The models have so much in common that automation may be used with advantage.
Choices, options etc. has to be provided in each transformation step to supply the necessary additional new information.
The transformations could also be a partly manual activity.

Simulation and execution of models support elimination of errors early in the process





What we can do to help

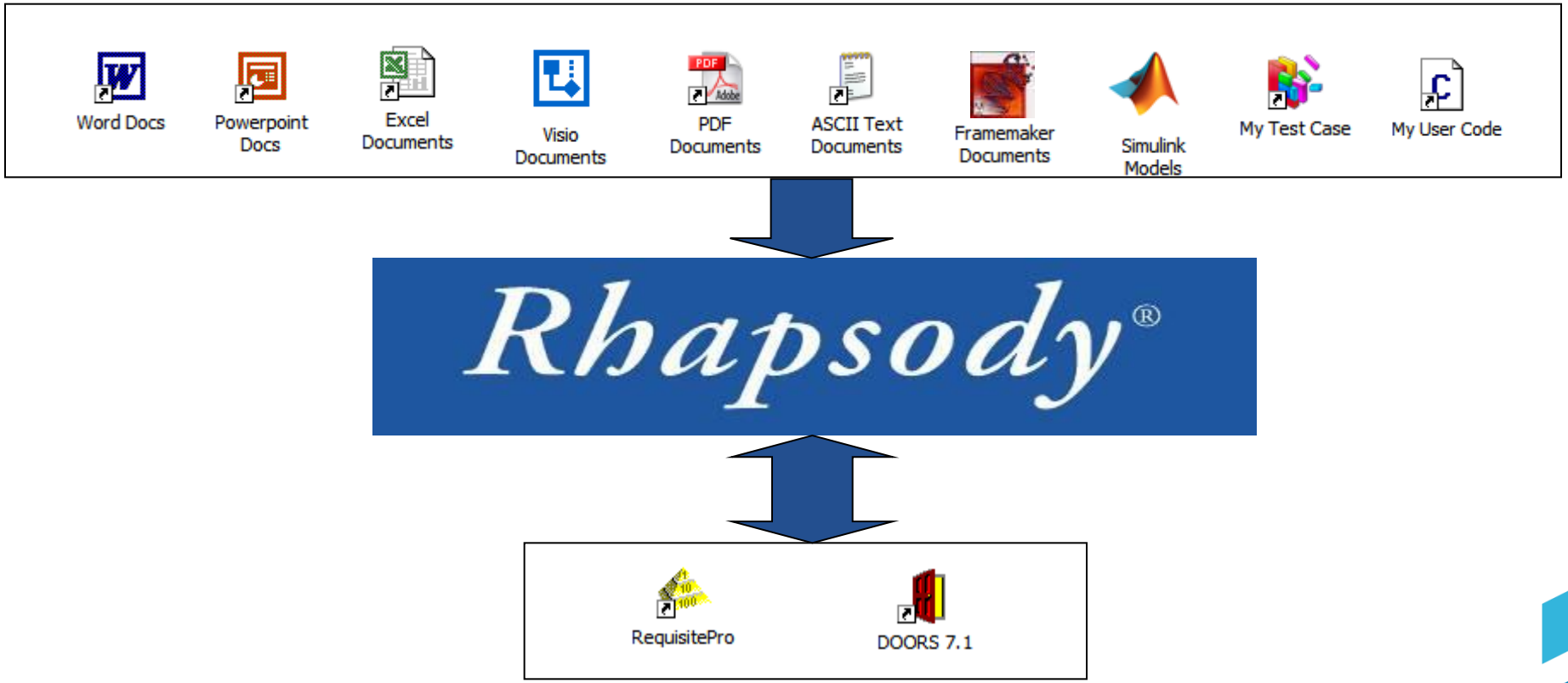


Rational Rhapsody MDD solutions for aerospace and defense applications

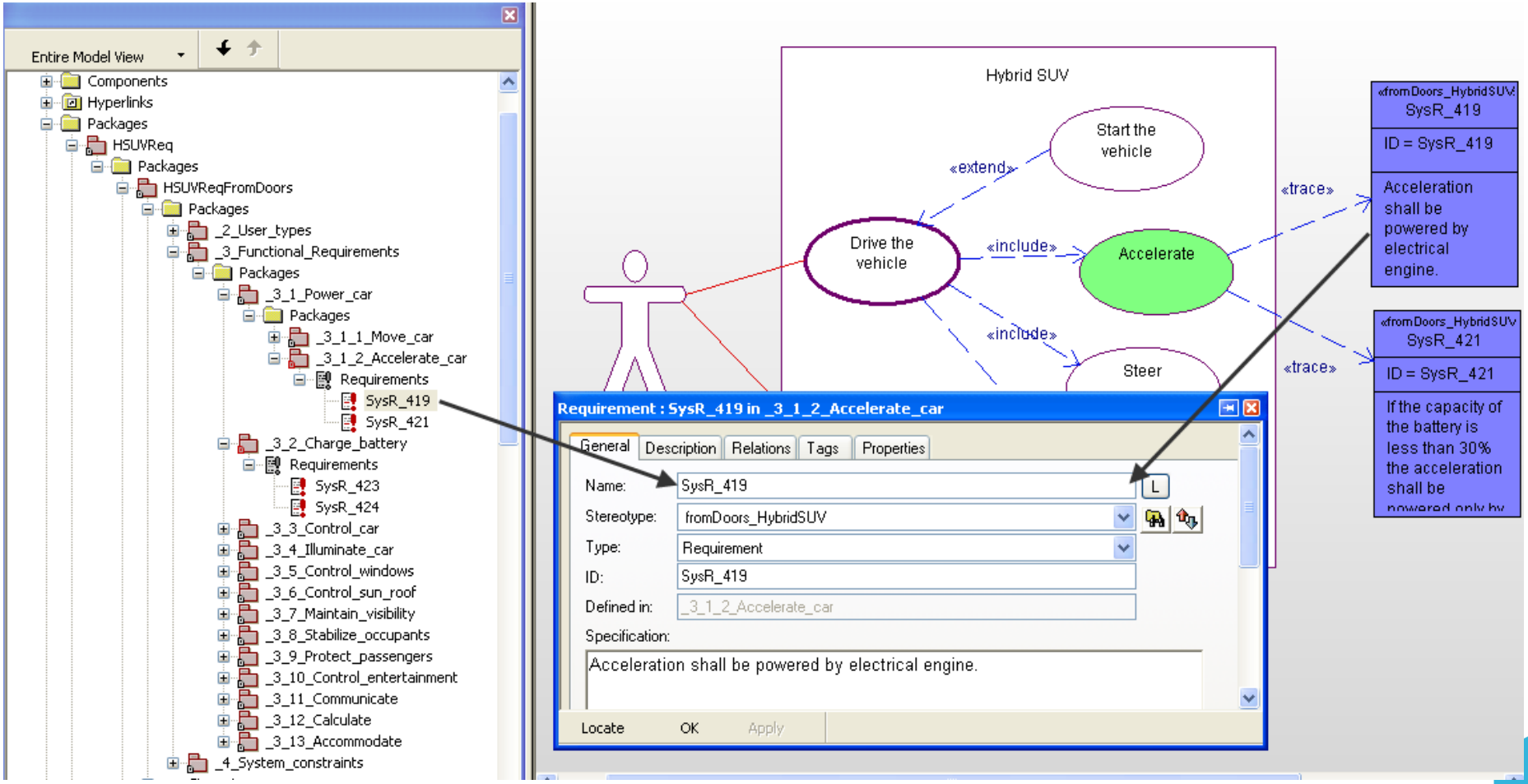
- **Support for SysML, UML 2.0 and DoDAF, our MDD solutions provide industry standard notations that are well suited for describing and communicating large, complex system (of system) requirements, design, architecture, behavior and implementation**
- **Integrated requirements traceability functionality with support for DO-178B level A allowing traceability within the model or within industry standard DOORS or other popular tools**
- **Find errors early and help save costs by applying our DFT productivity tools also supporting DO-178B**
- **Automatically produce design engineering documentation at the click of a button, again aiding communication**

Lifecycle traceability

- Create traceability links from model to requirements
- Produce automatic traceability documentation
- Import requirements from multiple sources

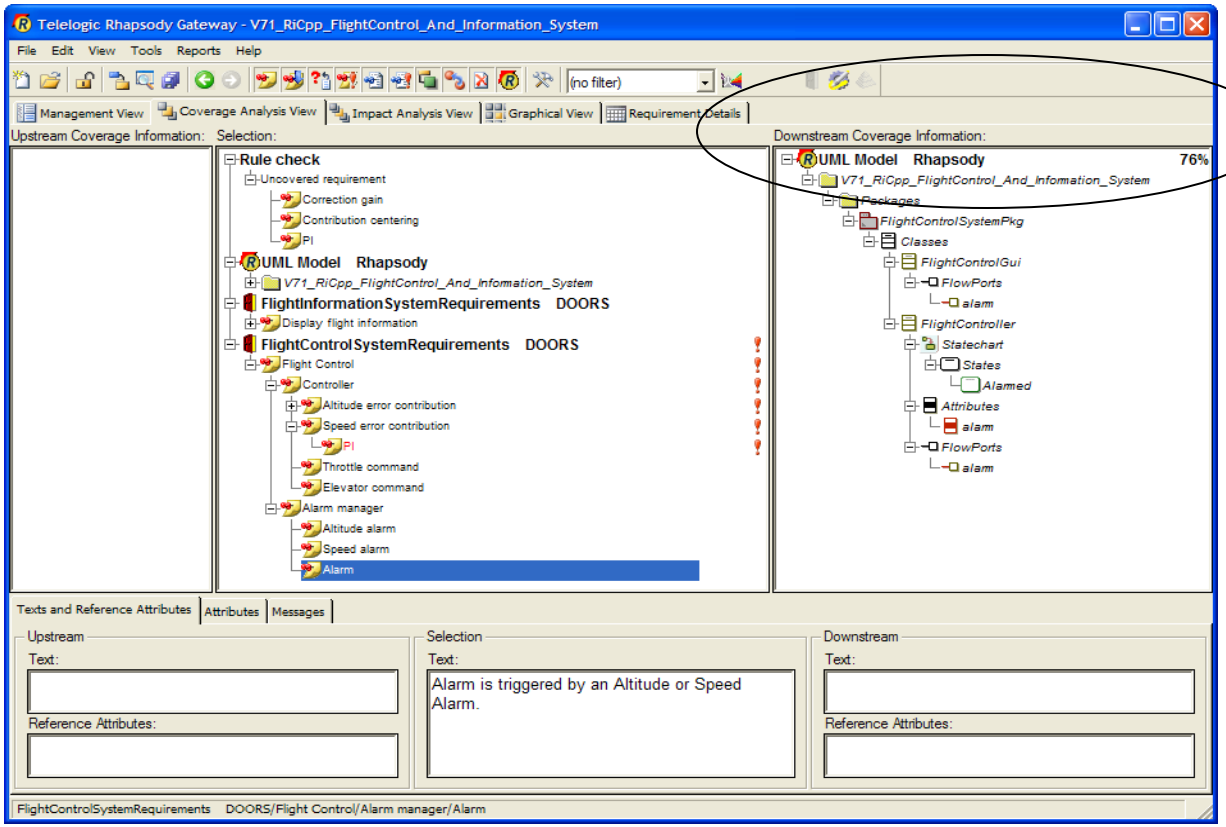


Requirements Capture and Trace



Requirements coverage analysis

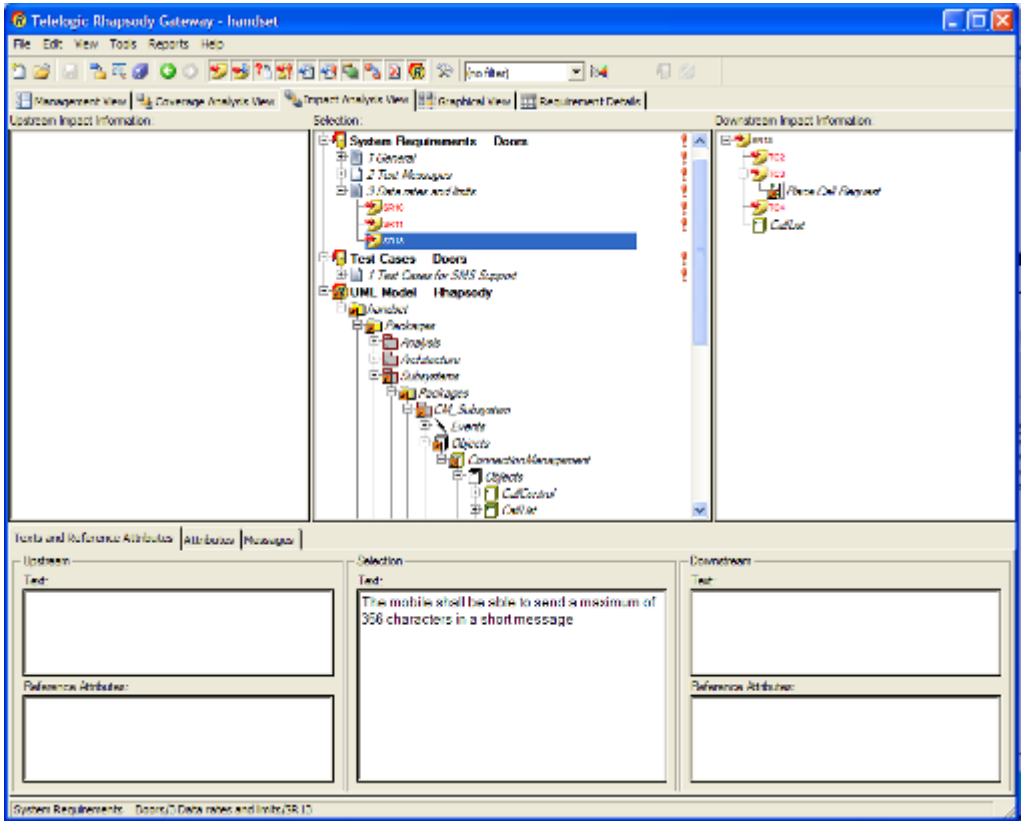
- Identify requirements that have not been addressed in the Rhapsody design
- Find Rhapsody design elements not justified by a requirement



Coverage

Change impact analysis

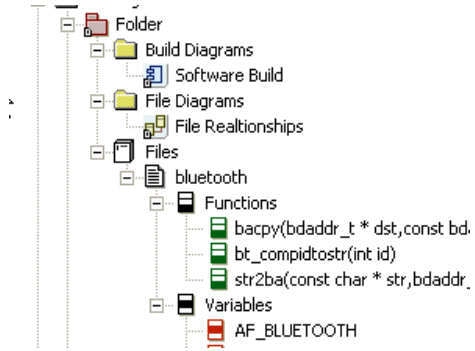
- Locate elements potentially impacted by a requirement change
- Determine requirements possibly impacted by a design change

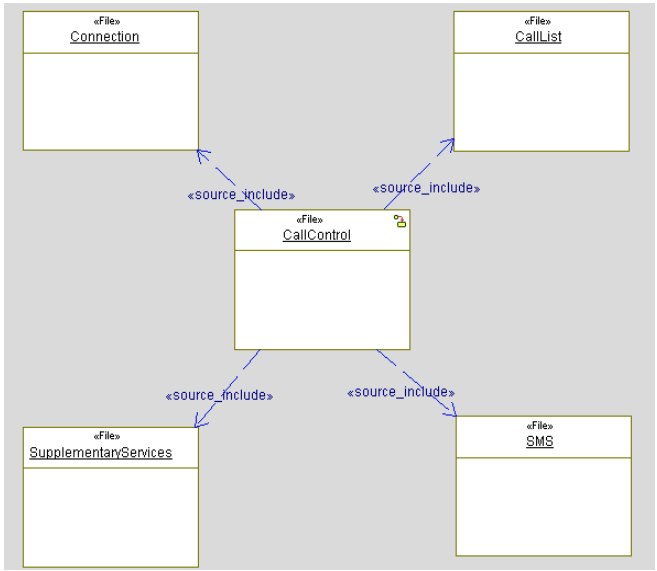


Domain-specific modeling

- Extend Rhapsody to create models for your domain
  - Use profiling to create your domain artifacts instead of UML artifacts
  - White boarding allows free formed design
  - Include your own graphic design elements
- SysML, DoDAF, AUTOSAR and Graphical C profiles available

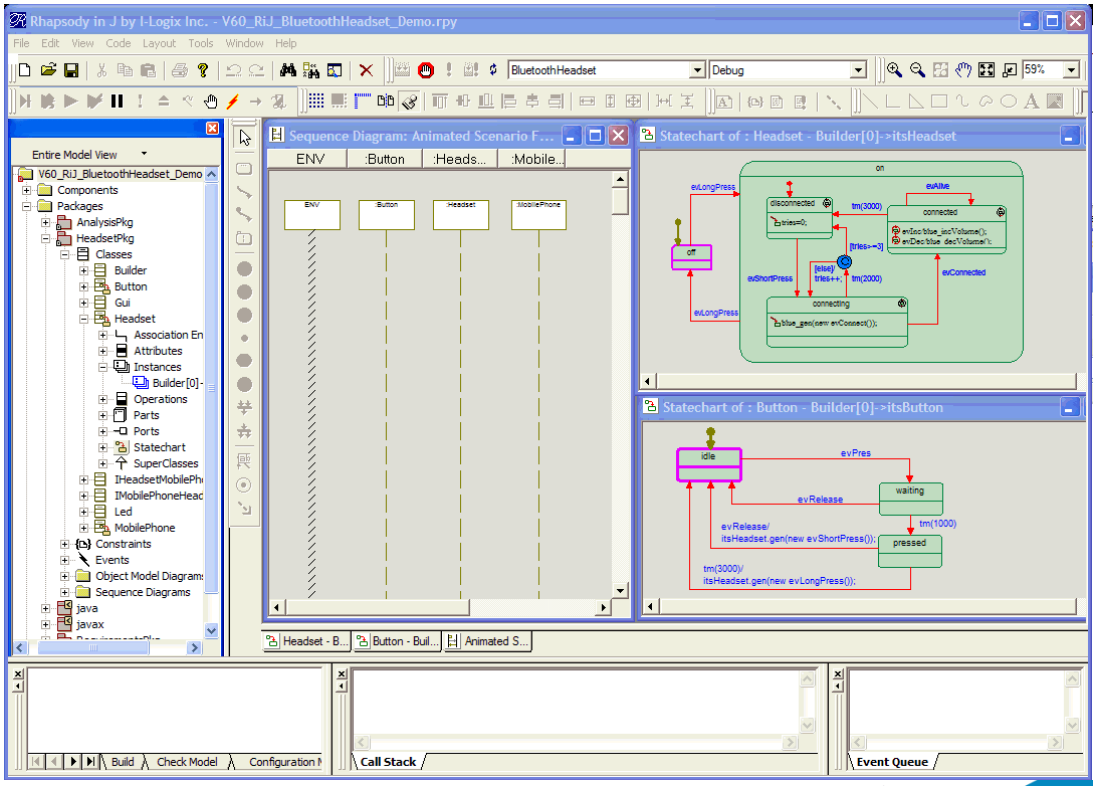
Graphical C





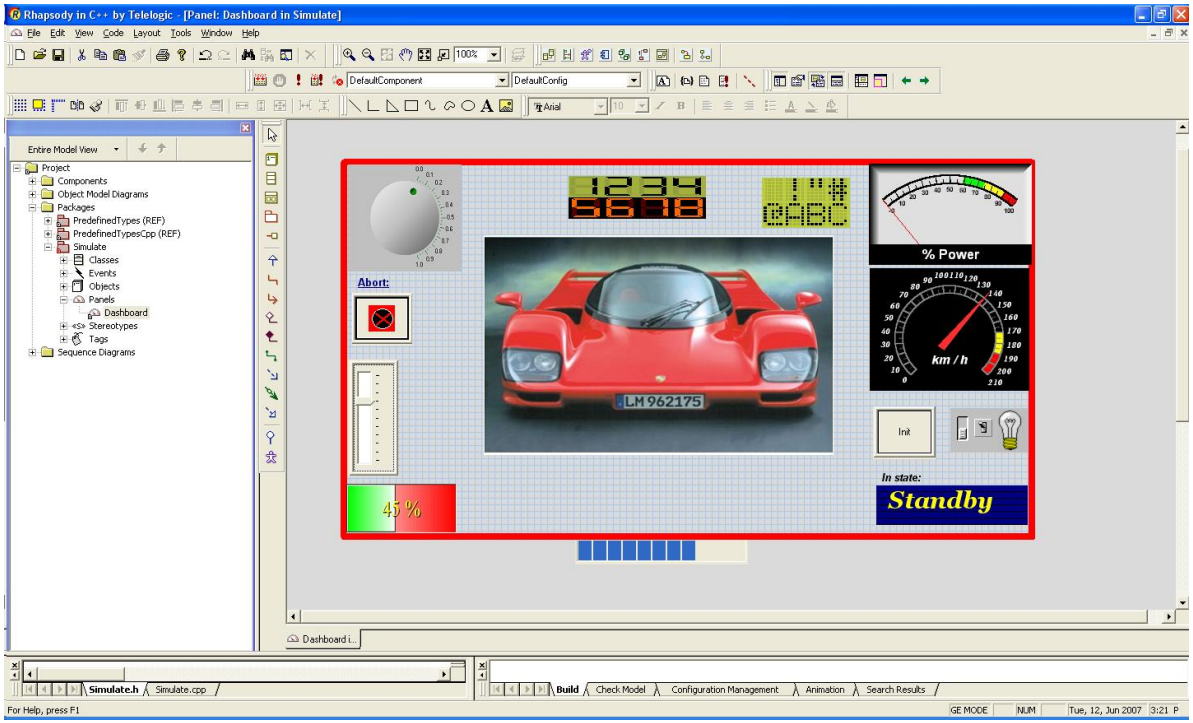
DFT - System Simulation, Execution & Animation

- Simulate to verify that model is correct
  - Reduces errors & therefore reduces development cost
- Virtual prototype / Panel graphics support
  - Ideal communications aid for design reviews and to share information.



Graphical panels

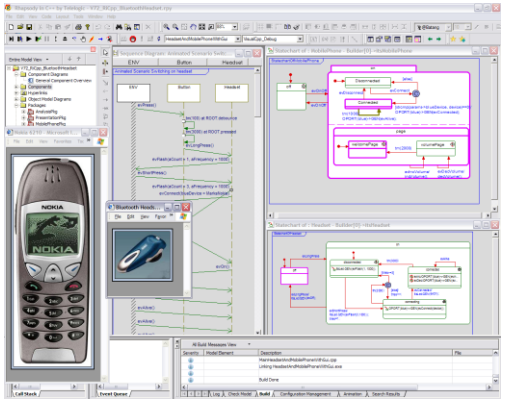
- Create mock-ups of interface to effectively communicate intended design behavior to customers
- Modify, monitor and analyze data values during simulation to help ensure that the design is correct early in the process



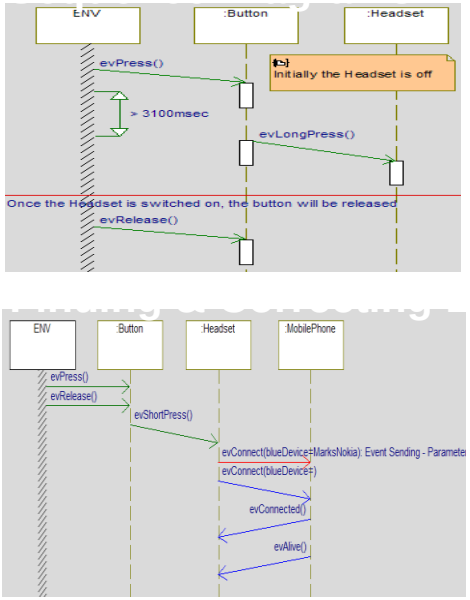
Model-driven testing

- Bring the benefits of abstraction and automation to testing
- Reduce defects early in the process when they are less costly to fix
- Deliver products meeting customer expectations

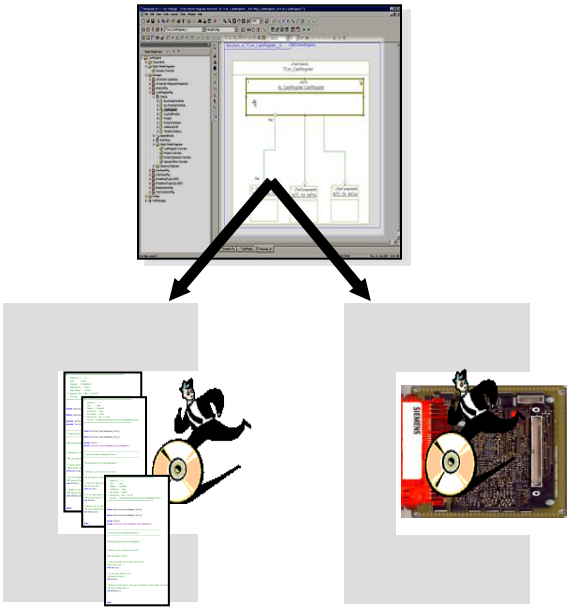
Simulation



Requirements-based testing



Automated unit testing



Host based

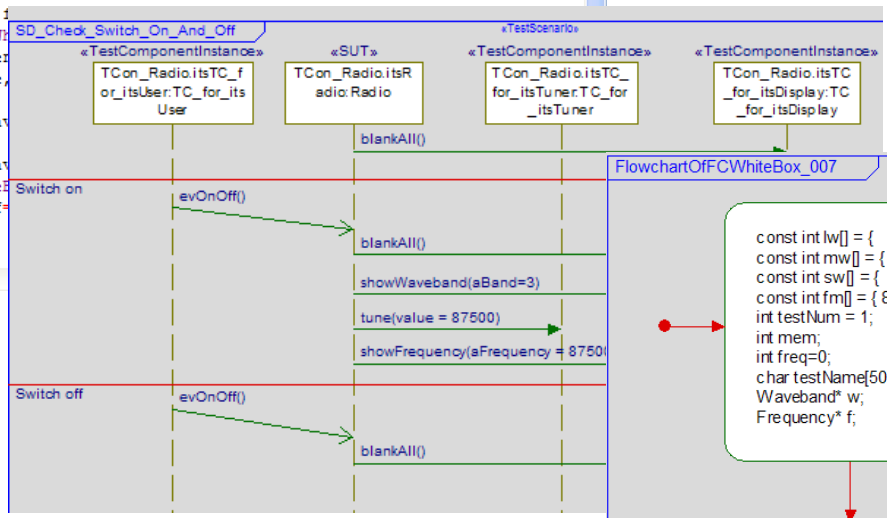
Target based

Test with Model Artifacts

- Test Your Model the Same Way You Design It

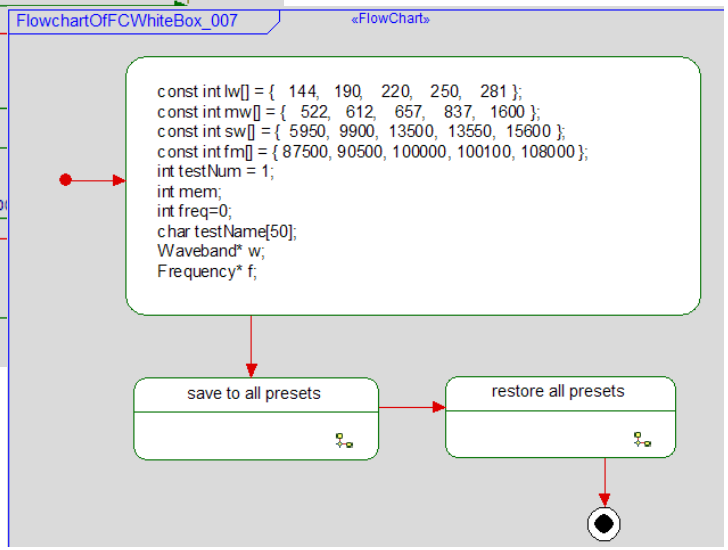
Code

```
int f;
int freq;
int testNum = 1;
char testName[50];

itsRadio.nextWaveband();

// Test MW  522KHz to 1620KHz step 9KHz
itsRadio.nextWaveband();
for ( freq=522; freq<=1620; i
    sprintf ( testName, "CDWh
    f = itsRadio.getItsCurrer
    RTC_ASSERT_NAME(testName,
    testNum++;
    itsRadio.getItsCurrentWav
}
f = itsRadio.getItsCurrentWav
sprintf ( testName, "CDWhiteB
RTC_ASSERT_NAME(testName, (f=
```



Sequence Diagrams

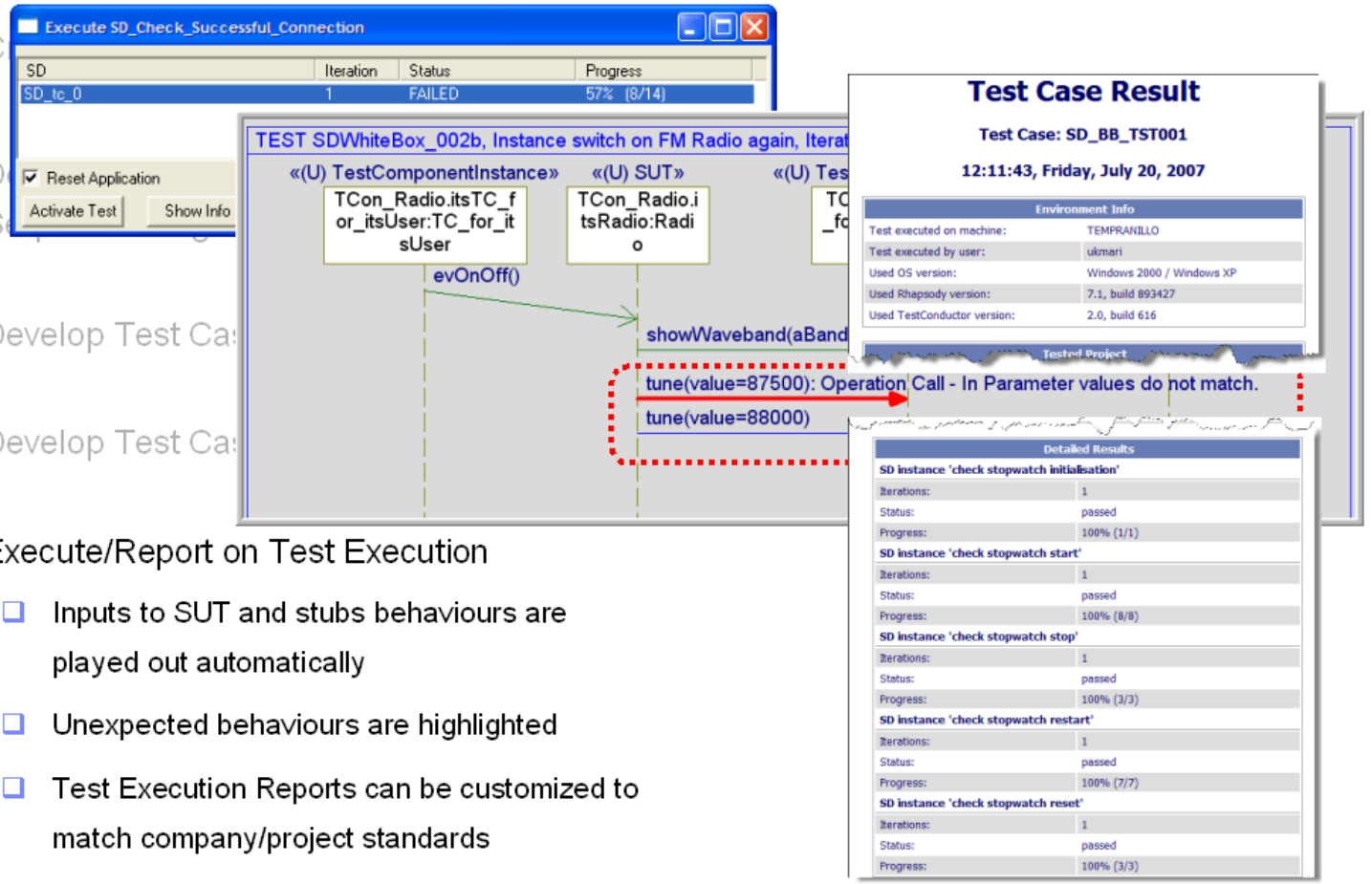
Flow Charts



- Tests Execute on Your Desktop and on Your Target

Model Driven Testing with IBM Rational Rhapsody Test Conductor

Test Execution & Test Reporting & Model Coverage

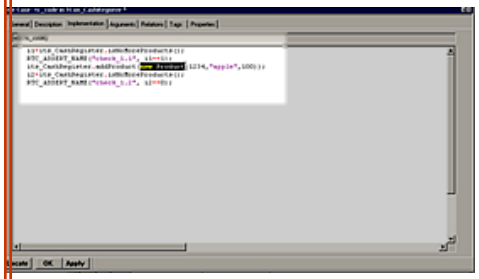


- Develop Test Ca
- Develop Test Ca
- Execute/Report on Test Execution
  - Inputs to SUT and stubs behaviours are played out automatically
  - Unexpected behaviours are highlighted
  - Test Execution Reports can be customized to match company/project standards

Multiple types of test cases

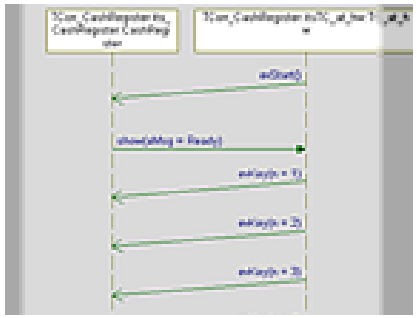
- "Use the right tool for the right job"
- Can describe complex testing scenarios
- Allow for testing of complex designs

Code



- *Minimal learning curve*
- *Only snippets of code*
- *Integrate with existing test cases*

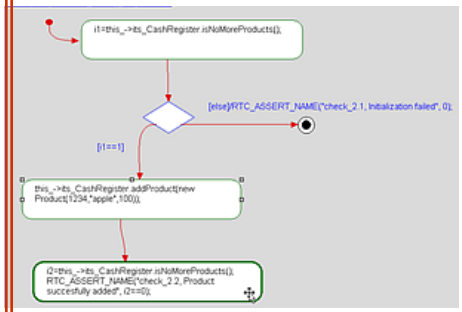
Sequence diagrams



Rhapsody TestConductor™

- *Capture required behaviors*
- *Capture approved behaviors for regression testing*

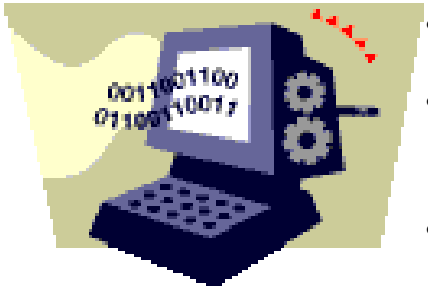
Flow charts



- *Capture complex testing scenarios*
- *Graphical*
- *Easy to communicate*

ATG

Auto test generation

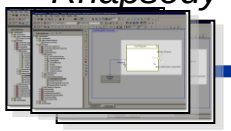


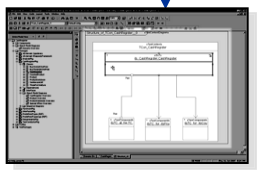
- *Automatic*
- *Very high coverage*
- *Regression testing*

Model-driven testing process

Coding Test Benches Manually... → *Coding Test Cases Manually...* → *Code on Host...* → *Code on Target...*

Rhapsody



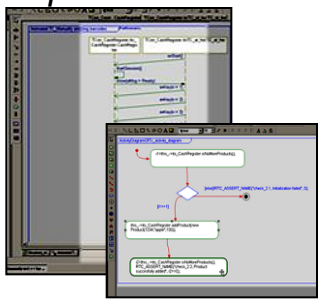


AutoGen Graphical TestBench

UML testing profile

Code TestCases

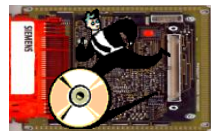
Graphical TestCases



AutoGen TestCases

Host based

Target based



DFT: Automatic Test Generation

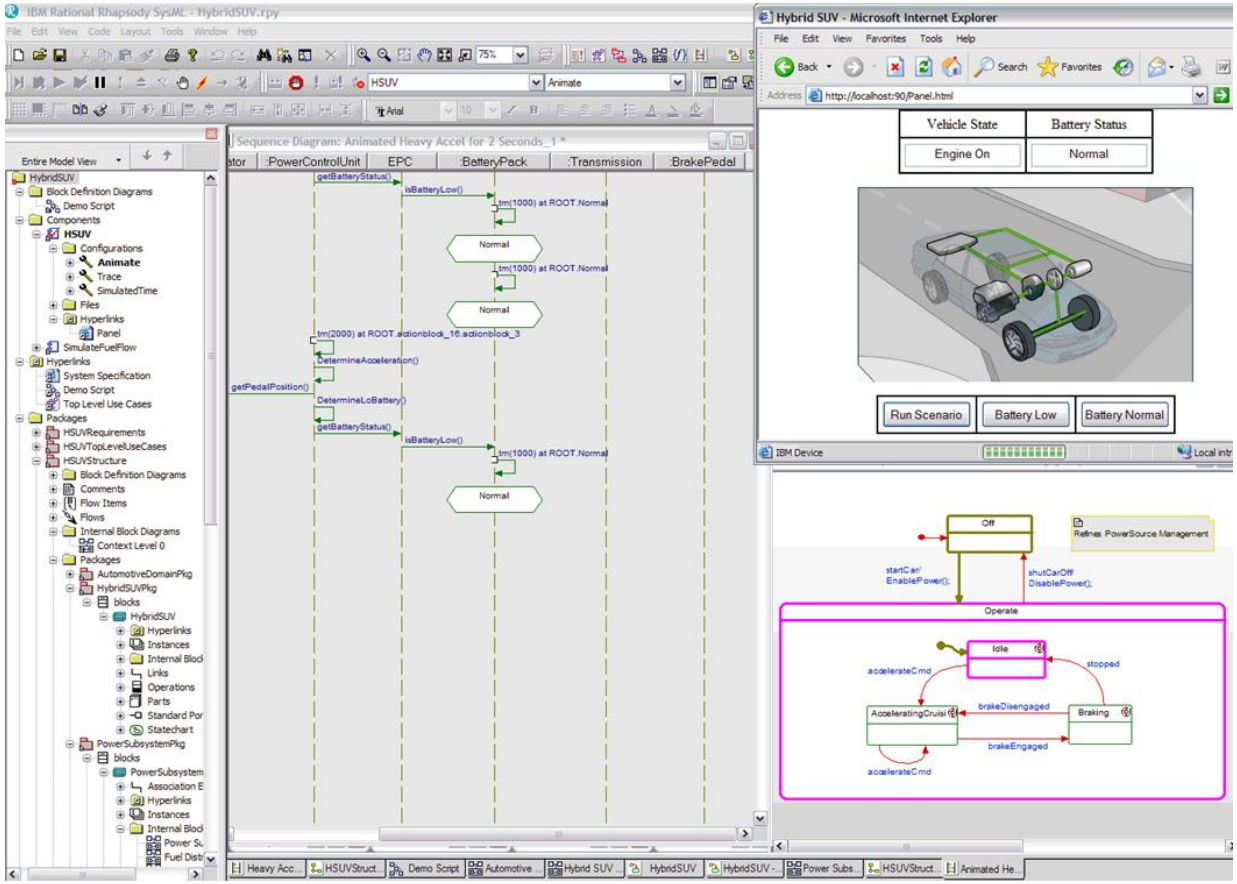
- ATG (Automatic Test Generation) offers Model Driven Test Generation (Consistent with the emerging UML Testing profile)
- Generates test cases with high coverage of the model
  - Covers states, transitions, operations, generation of events
  - Covers all relevant combinations of inputs for MC/DC analysis
  - Model and MC/DC coverage – Required for DO178B/ED12B
  - Identifies cases for potentially dead portions of the model
  - Test Cases can be exported and reused (as sequence diagrams and XMI to Test Conductor / 3rd party tools

HeadsetAndMobilePhone_atg__TestHeadset_Part1.cpp

```
int cantpp_main(char* argv[])
{
    CONFIGURE_ENHANCED_OUTPUT(  cppth_enhanced_op_stdout |
                                cppth_enhanced_op_fail |
                                cppth_enhanced_op_covg);
    char tmpSysCall[20];
    int nTestCase = atoi(argv[0]);

    if(!strcmp(argv[0],TEST_BY_TEST))
    {
        for(int i=1;i<=max_testfunc_calls+1;i++)
        {
            sprintf(tmpSysCall,"HeadsetAndMobilePhone.exe %d",i);
            system(tmpSysCall);
            Sleep(2000);
        }
        return 0;
    }
    // starts test execution without application stop
    else if (!strcmp(argv[0],TEST_IN_LOOP))
    {
        OPEN_LOG("HeadsetAndMobilePhone_atg__TestHeadset_Part1_noRes
        SET_LOG_LEVEL(cppth_ll_normal);
        START_SCRIPT("HeadsetAndMobilePhone_atg__TestHeadset_Part1",

        TEST_CLASS(HeadsetAndMobilePhone_atg__TestHeadset_Part1) tes

        test_object.run_tests(-1,false);
        return !END_SCRIPT(true);
    }
    else if(nTestCase!=NULL)
```

Executable Models on Host & Target

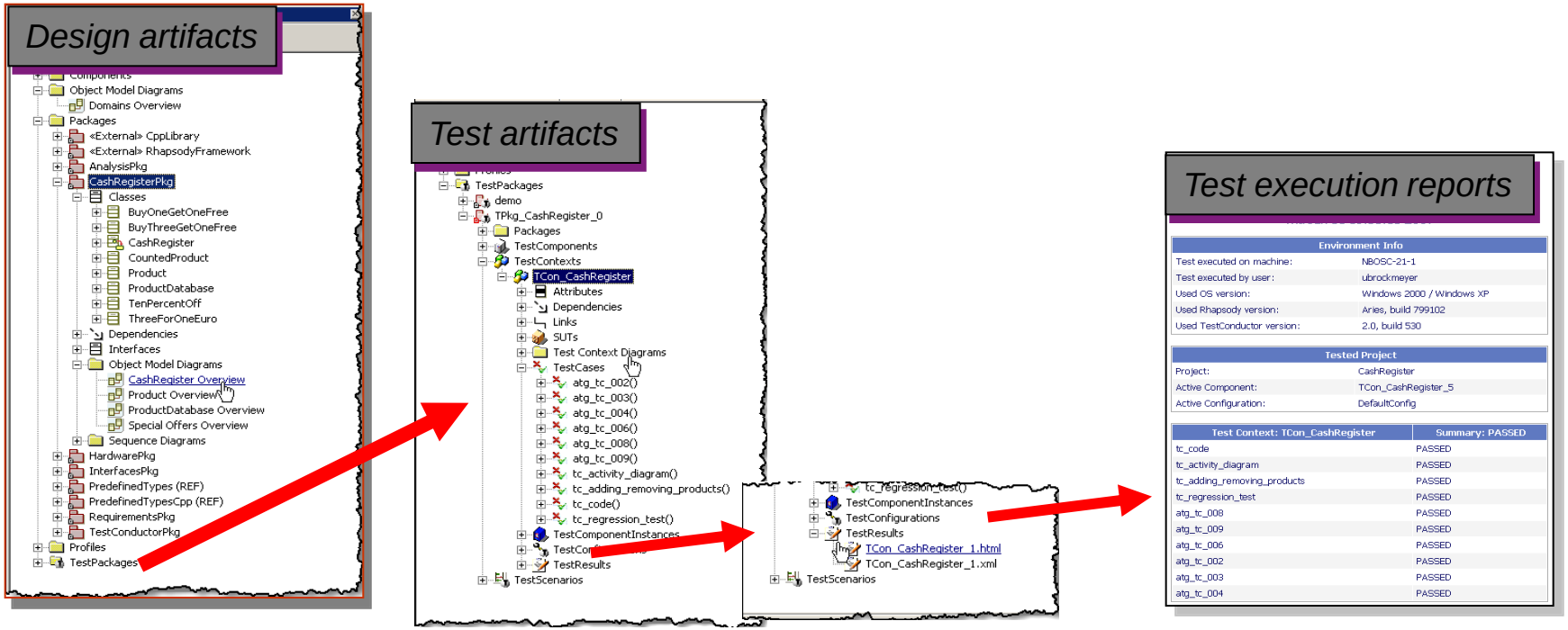
- **Execute** to verify that model is correct
  - **Remove errors** when they are introduced
  - **Rapid execution** at the design level
- **Virtual prototype** with graphics support
  - **Communications aid** for design reviews



You can't test what you can't execute!

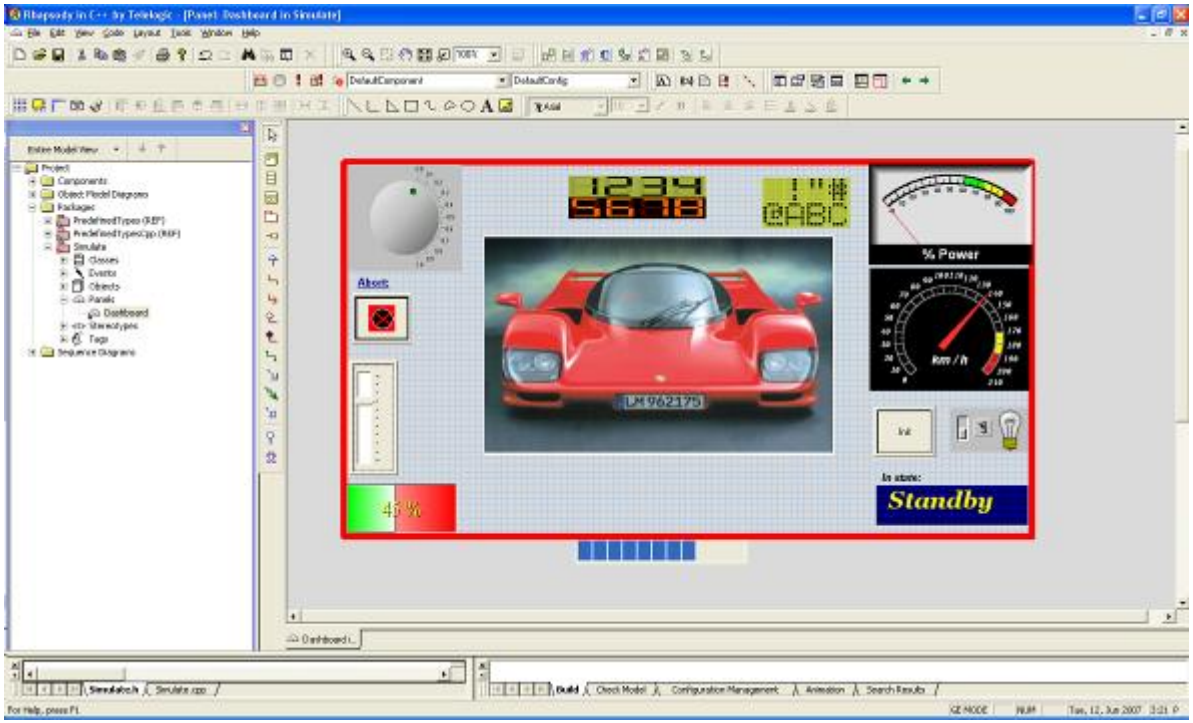
Design and test process integration

- Virtually seamless integrated process, based on UML 2.0 testing profile
  - Requirements linked to test cases
  - Straightforward navigation between design and test artifacts
  - Design and test—virtually always in sync
  - Automatically generated test execution reports



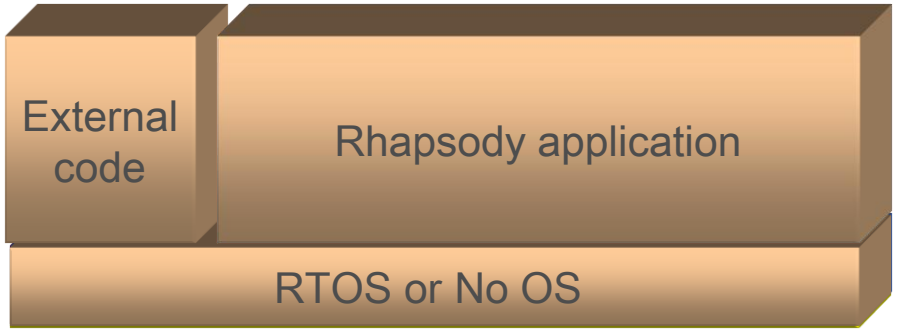
Graphical panels

- Create mock-ups of interface to effectively communicate intended design behavior to customers
- Modify, monitor and analyze data values during simulation to help ensure that the design is correct early in the process

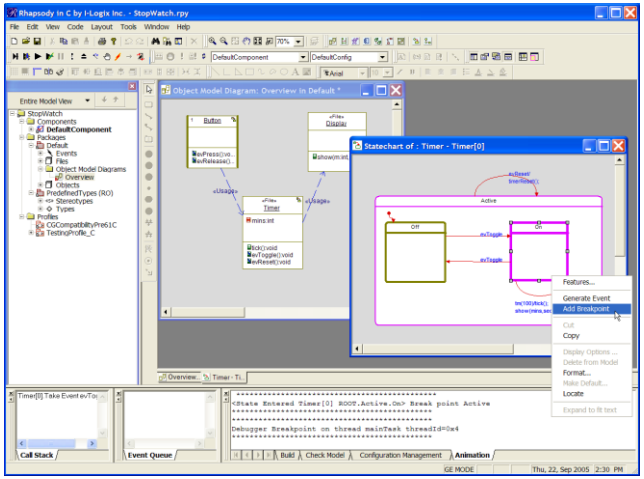


Full application generation

- Meet-time-to market pressures with complete applications, not frames
  - Generate C, C++, Java™ and Ada applications
  - Rhapsody generates very clean, readable code, readily debugged through any commercial IDE, including Eclipse
- Rapidly deploy your design onto any target platform
  - Provides platform-independent models (PIMs)
- Flexible development environment, work at code or model level

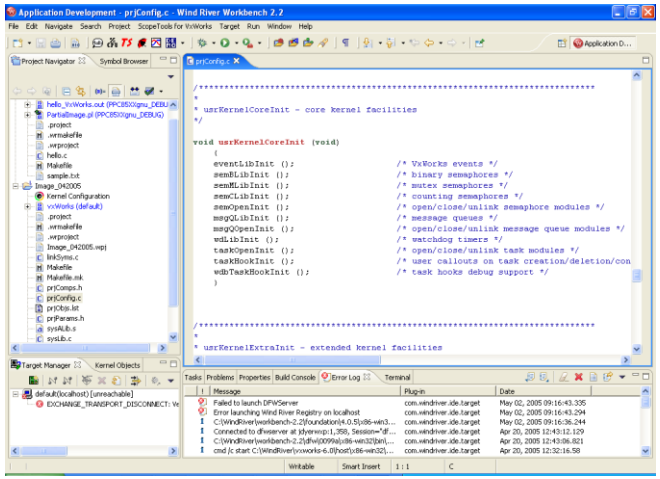


Rhapsody®



MDD
Code Generation C, C++ Ada
Combined source and Design-level debugging

Workbench

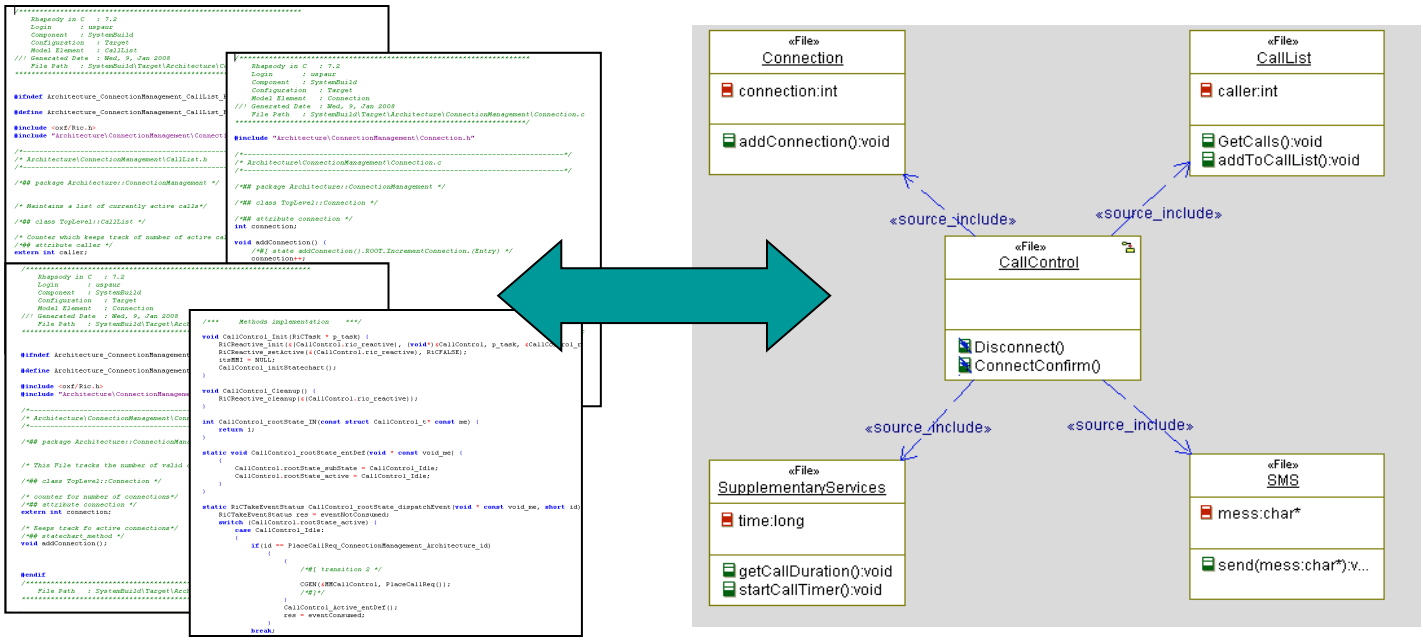


Automatic Download
Synchronized Breakpoints
Unit Testing

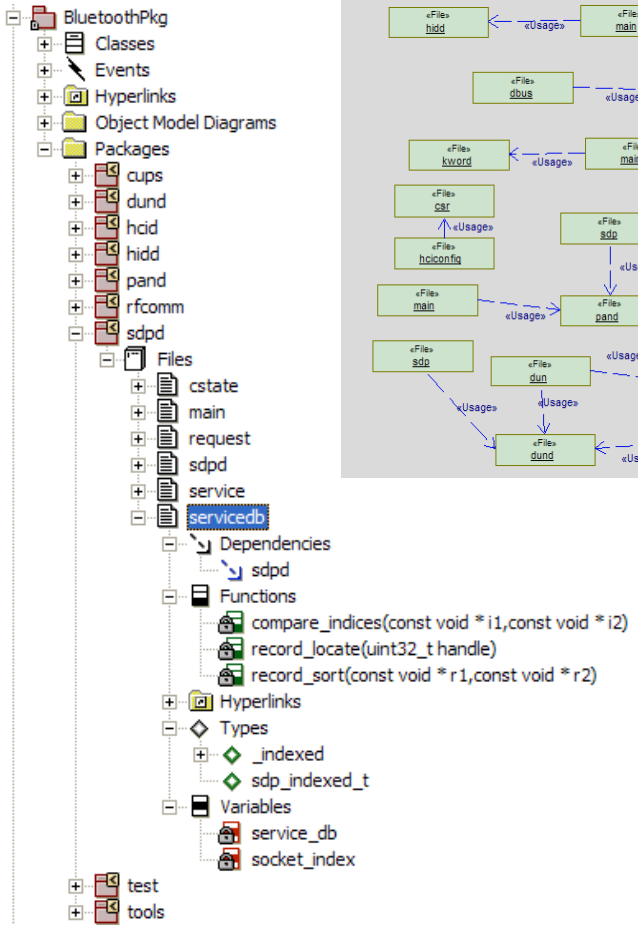
Targets

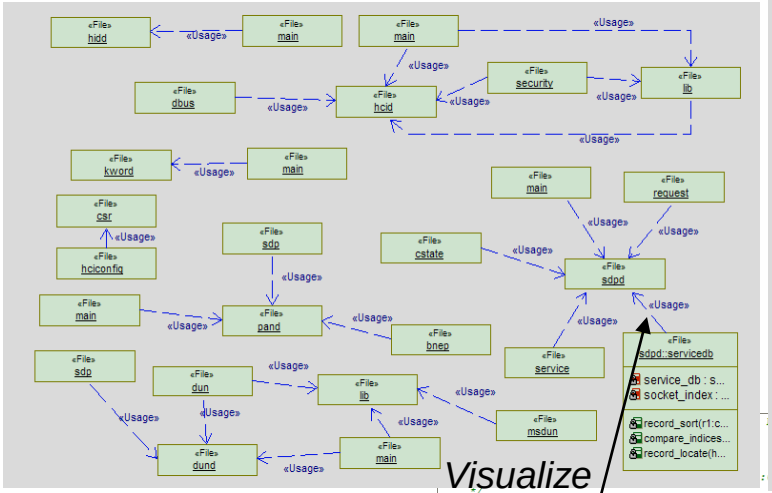
Reuse existing code (IP)

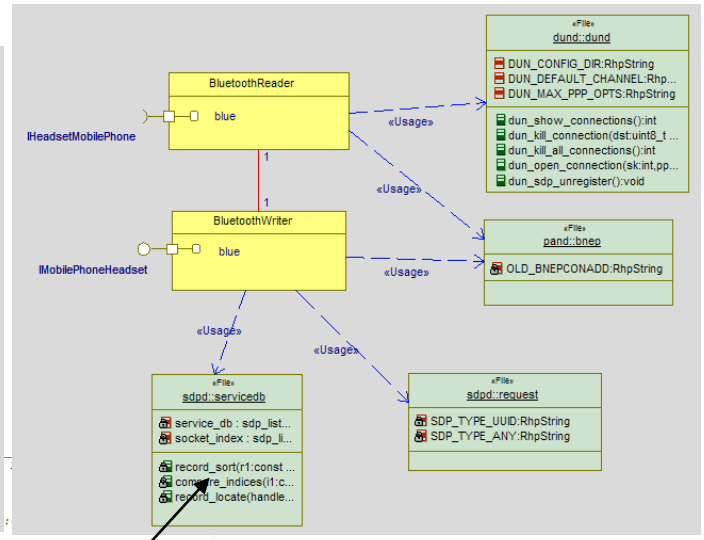
- Reuse code from other projects
- Integrate code developed by a third party
- Visualize in the model for better understanding
- Reference third-party code within Rhapsody



Code visualization example







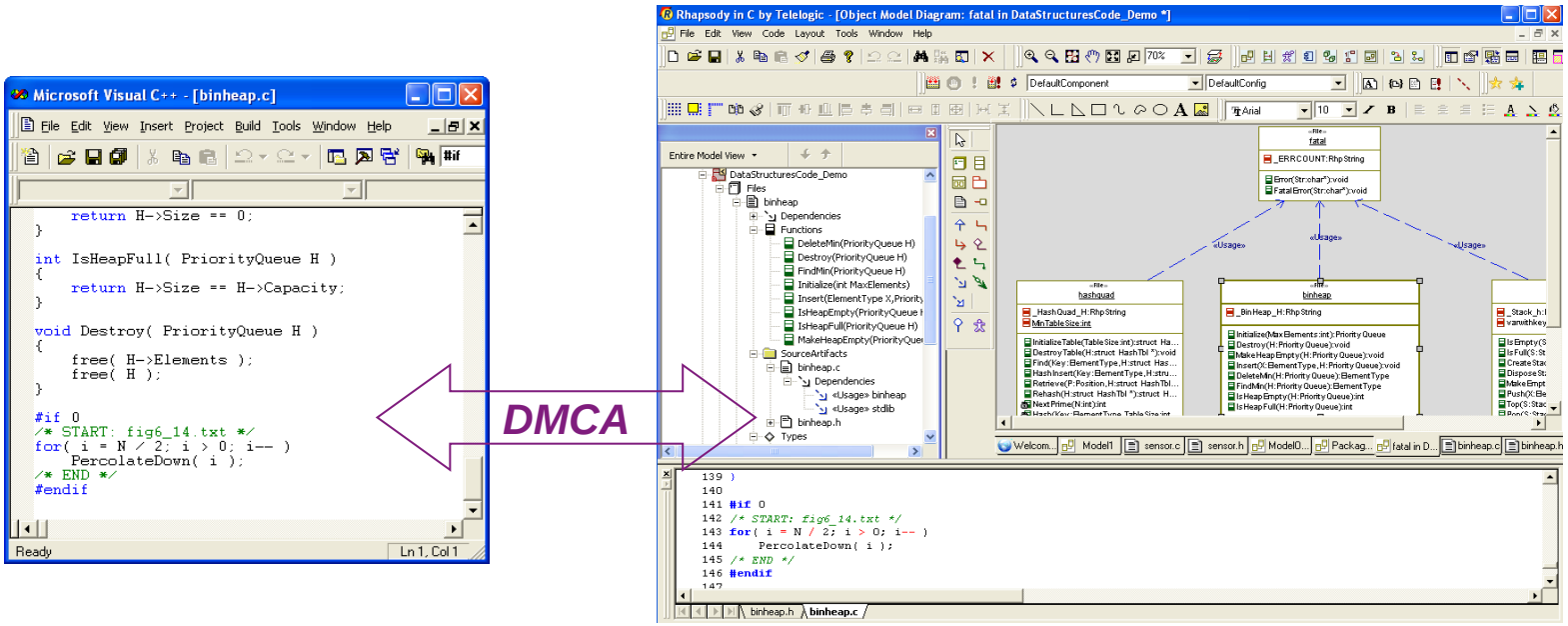
Visualize

Reference

```
#ifdef HAVE_CONFIG_H
#include <config.h>
#endif

#include <malloc.h>
#include <syslog.h>
#include <sys/socket.h>

#include <bluetooth/bluetooth.h>
#include <bluetooth/l2cap.h>
#include <bluetooth/sdp.h>
#include <bluetooth/sdp_lib.h>

#include "sdpd.h"

static sdp_list_t *service_db;

/*
 * Ordering function called when inserting a service record.
 * The service repository is a linked list in sorted order
 * and the service record handle is the sort key
 */
static int record_sort(const void *r1, const void *r2)
{
    const sdp_record_t *rec1 = (const sdp_record_t *)r1;
    const sdp_record_t *rec2 = (const sdp_record_t *)r2;

    if (!rec1 || !rec2) {
        SDPERR("NULL RECORD LIST FATAL\n");
        return -1;
    }
    return rec1->handle - rec2->handle;
```

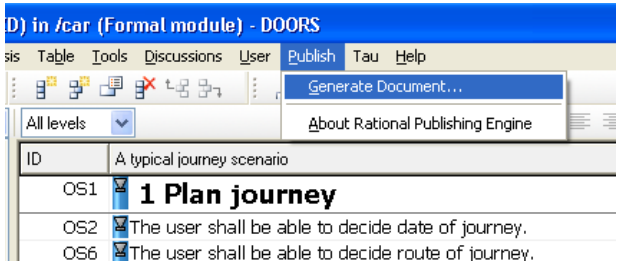
Rhapsody works the way you do

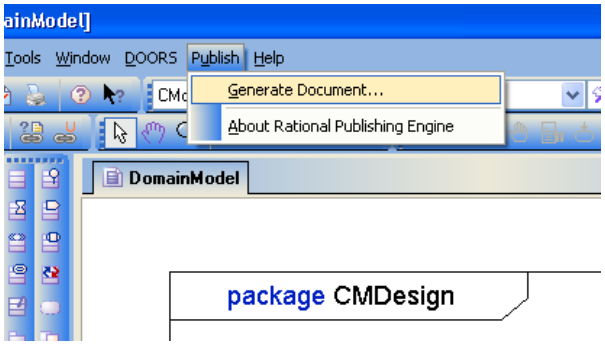
- Work at the code or model level
- Reduce learning curve and increase effectiveness
- Dynamic Model Code Associativity (DMCA) keeps design and code in sync
  - Change one view, the others change automatically
  - Critical for realtime embedded software development

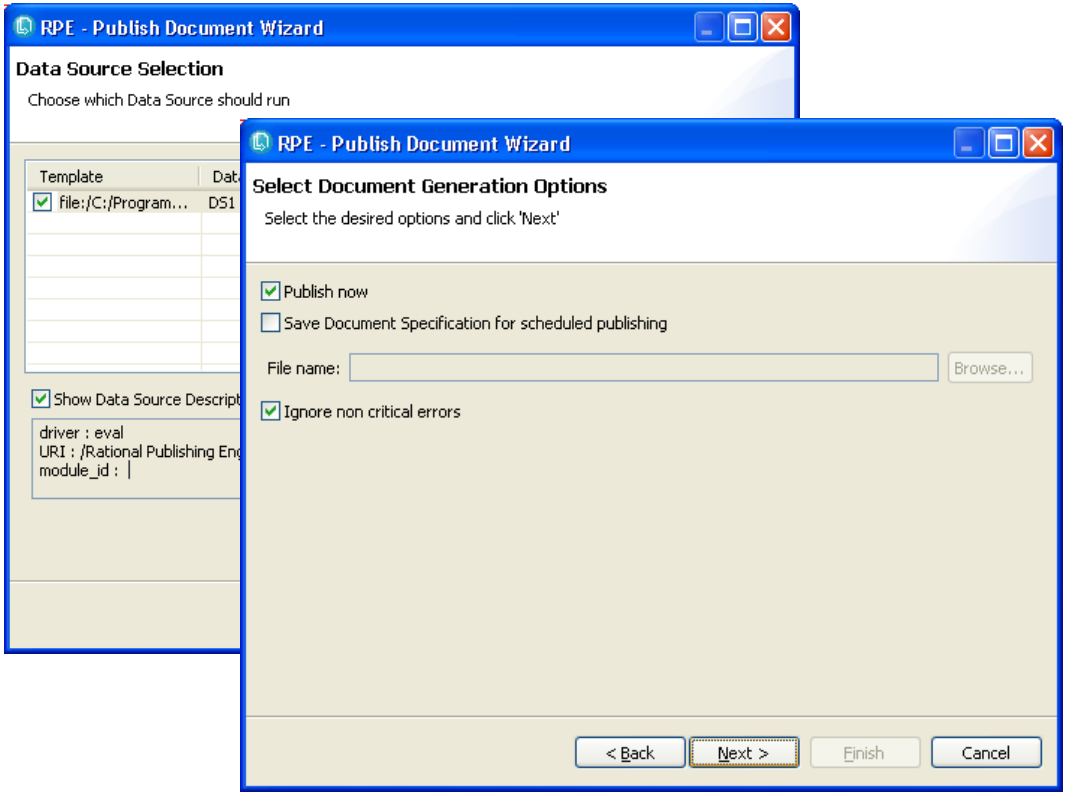


Rational Publishing Engine
Document Creation Made Simple

- Create documents from within and/or extract data from Rational DOORS and Rational Rhapsody, ClearCase, ClearQuest, Quality Manager, Test Manager, Requisite Pro, XML data sources…)

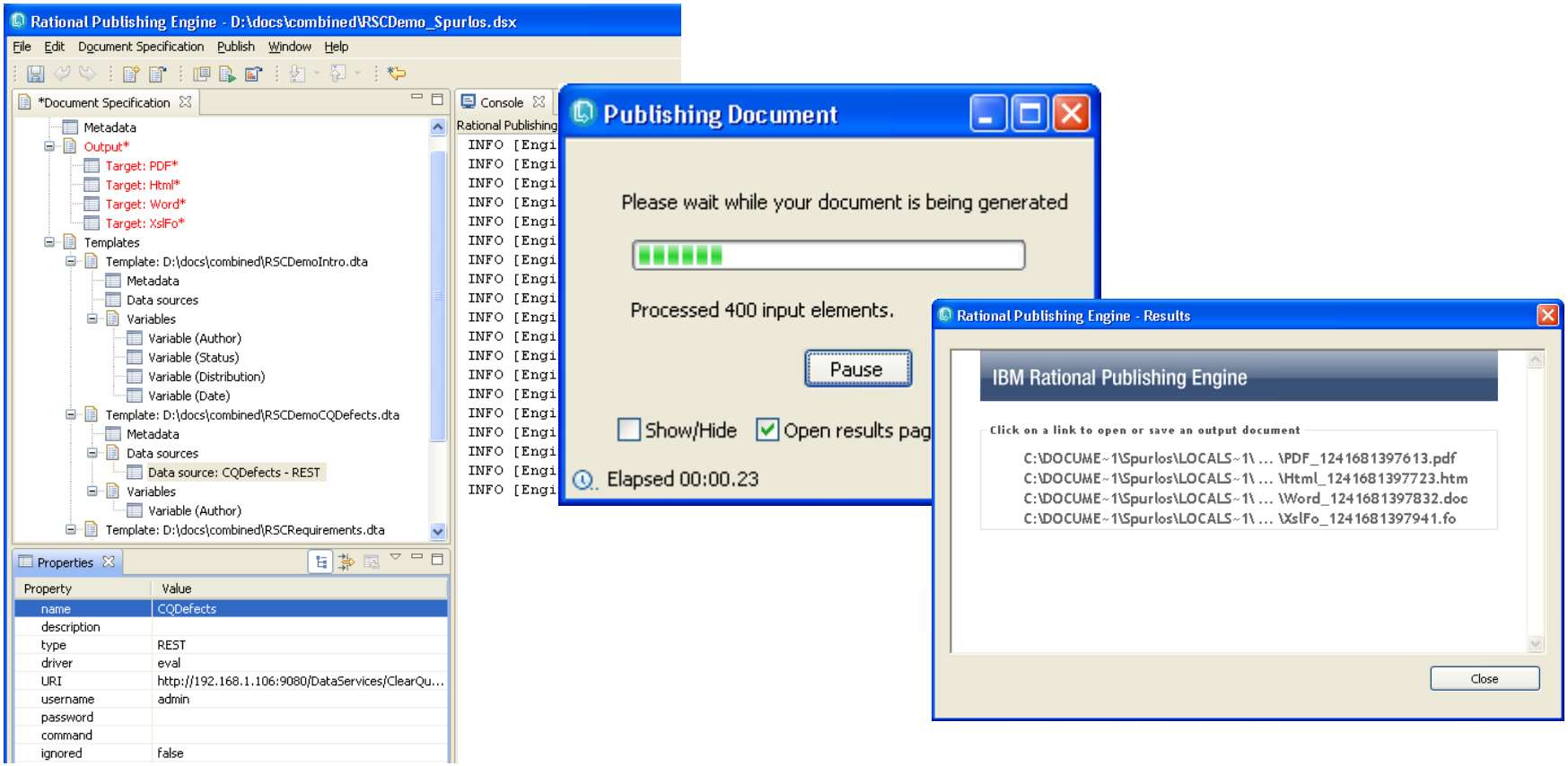






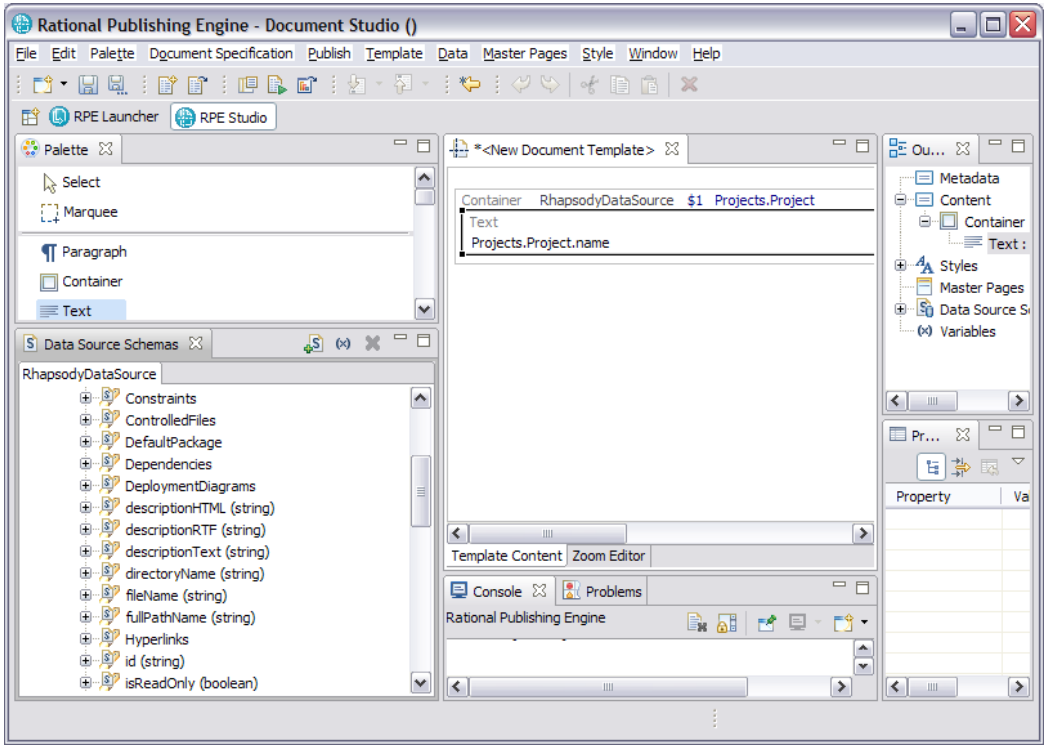
Rational Publishing Engine
Document Creation Made Simple

- Create documents using the easy and intuitive Rational Publishing Engine Launcher



Rational Publishing Engine
Document Creation Made Simple

- Build, share and reuse templates
- Use out-of-the-box templates and integrated preview capability for quick ROI
- Leverage templates for industry standards
- Create and modify templates using intuitive editing environment
  - Drag and drop capability
  - Powerful scripting language support (Javascript) with expression editor for ease of use



Advantages of MDD

- Precision
- Models constructed in formal (or semi-formal) languages are more precise than text
- Validation
- Models can be executed, simulated, or analyzed
- Improved Handoff from systems engineering to downstream engineering
- Precise models are less likely to be misinterpreted
- If systems and software engineers use the same modeling languages, then no translation is required
- Improved understanding of architecture
- Improved visualization of functional, structural, and behavioral aspects
- Decreased design learning time

Productivity improvements via Model driven development

- Early design validation via simulations
- Code generated in parallel with model development avoiding tedious and error prone manual coding
- Software is developed inside the design elements maintaining compatibility with design
- Requirements traceability between the requirements to the model to the code
- Support for industry specific solutions such as Autosar, DoDAF, Net Centric Operations, Telecom Handsets, Medical FDA certification support, etc
- linkage with embedded operating systems so it is possible to validate the design on the target
- Formal testing at a model level via Model driven testing (MDT)
- Documentation is automatically generated by the tools
- Product is much easier to visualise, understand, prove , maintain and reuse in Model form

IBM Rational Team Concert

Software innovation through collaboration

- **Real time, in-context team collaboration**
  - Make software development more automated, transparent and predictive
- **"Think and work in unison"**
  - Integrated planning, source control, work item, build management and project visibility
- **Deliver end-to-end governance**
  - Assess real-time project health
  - Capture data automatically and unobtrusively
  - Integrate document collaboration with enterprise infrastructure
- **Automate best practices**
  - Dynamic processes accelerate team workflow
  - Out-of-the-box choice of agile processes or customize
- **Unify software teams**
  - Broad array of clients: Web, Eclipse, Visual Studio
  - Extends the value of ClearQuest and ClearCase
  - Support for System i and System z

Open and extensible on Jazz

- ✓ Collaborate
- ✓ Automate
- ✓ Report

Rational Team Concert – a closer look

Iteration Planning

- Integrated iteration planning and execution
- Task estimation linked to key milestones
- Out of the box agile process templates

Project Transparency

- Customizable web based dashboards
- Real time metrics and reports
- Project milestone tracking and status

SCM

- Integrated stream management with flow relationships
- Component level baselines
- Server-based sandboxes
- Identifies component in streams and available baselines
- ClearCase connector

Work Items

- Defects, enhancements and conversations
- Query results view and share queries with team or member
- Support for approvals and discussions
- ClearQuest connector
- Query editor interface

Build

- Work item and change set traceability
- Local or remote build servers
- Supports ant and command line tools
- Integration with build forge
- Build definitions for team and private builds

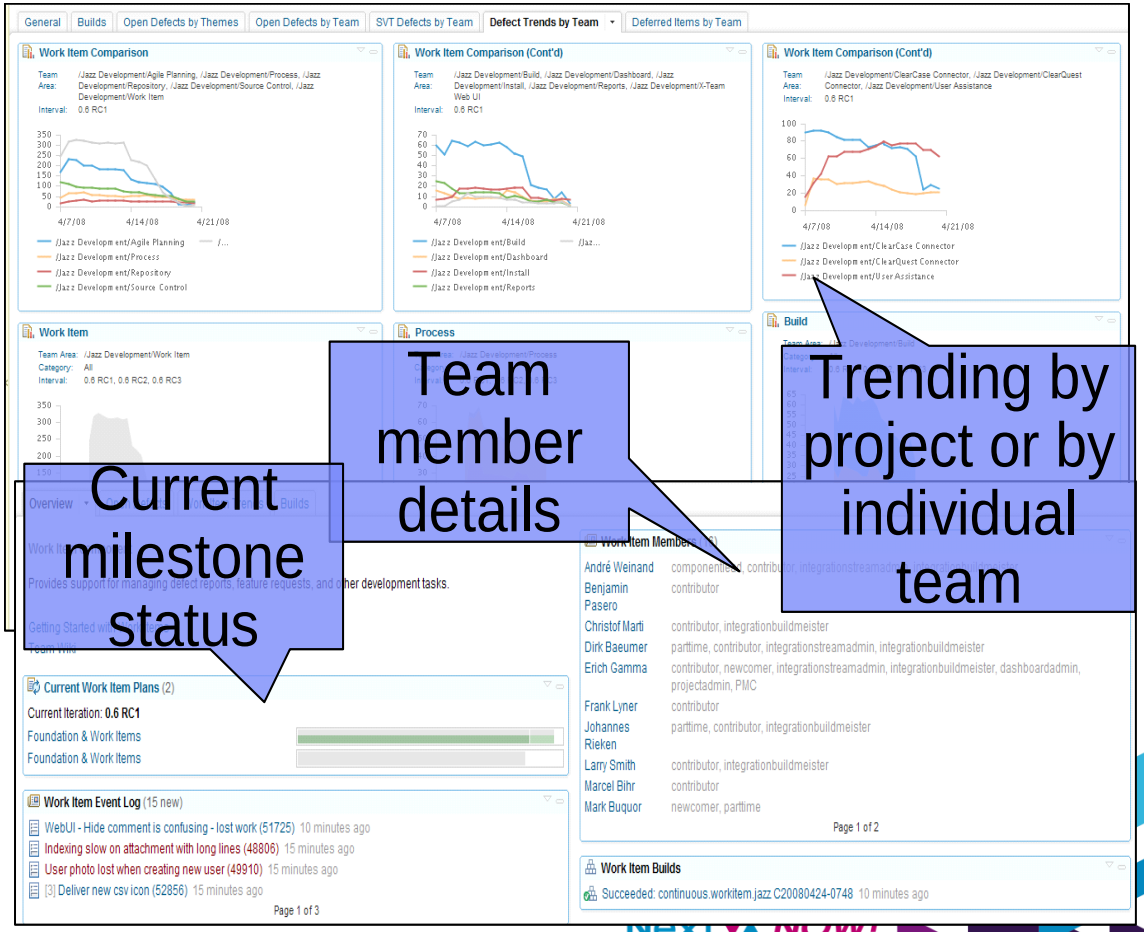
Jazz Team Server

- Single structure for project related artifacts
- World-class team on-boarding / off-boarding including team membership, sub-teams and project inheritance
- Role-based operational control for flexible definition of process and capabilities
- Team advisor for defining / refining “rules” and enabling continuous improvement
- Process enactment and enforcement
- In-context collaboration shows team members and status of their work

Collaborate, plan and manage change across diverse teams

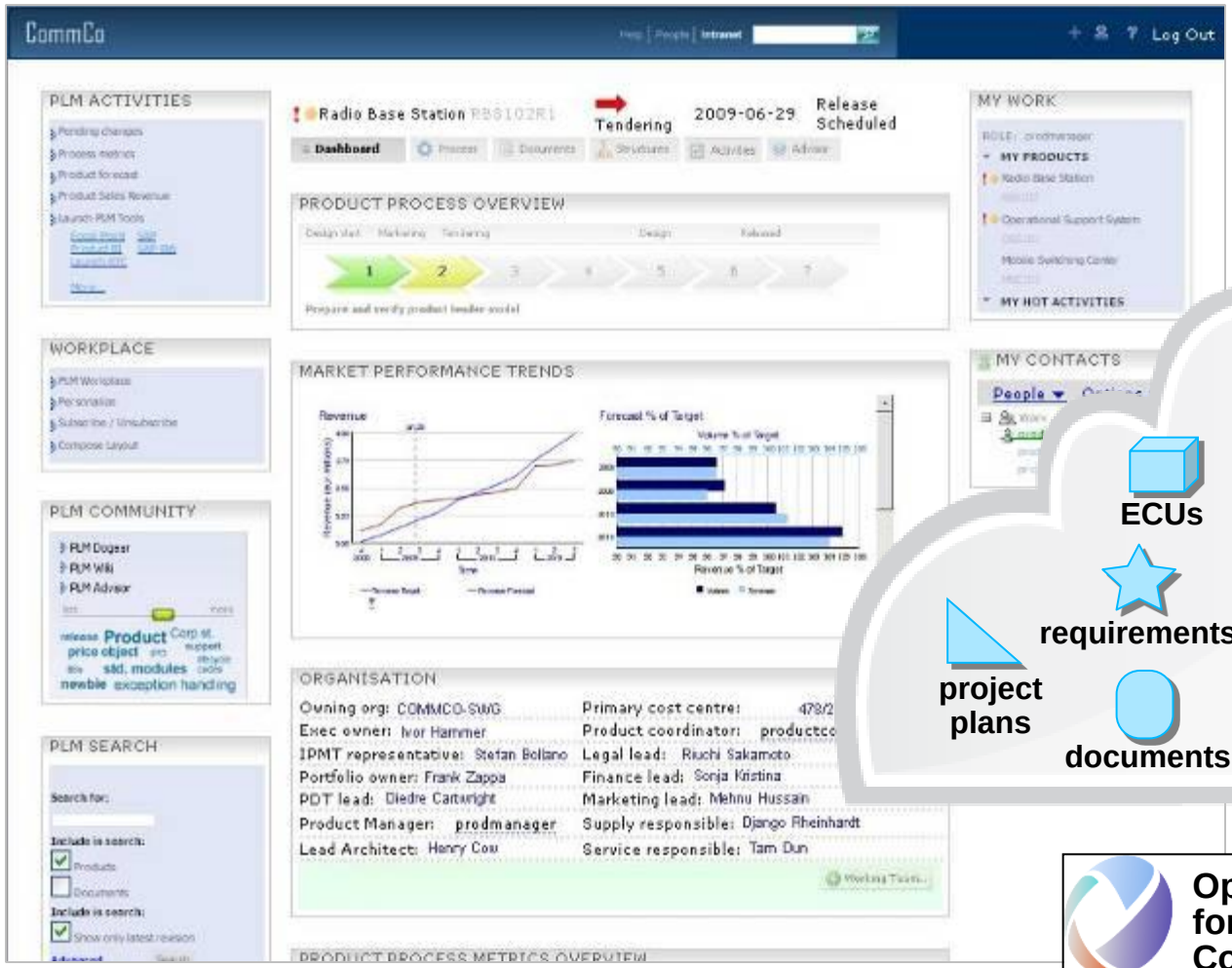
- Establish a Web-based collaboration hub
- Increase visibility with real time Dashboards
- Manage changes to requirements

- Respond faster with Integrated Planning
- Collaborate in context
- Link all artifacts to work items

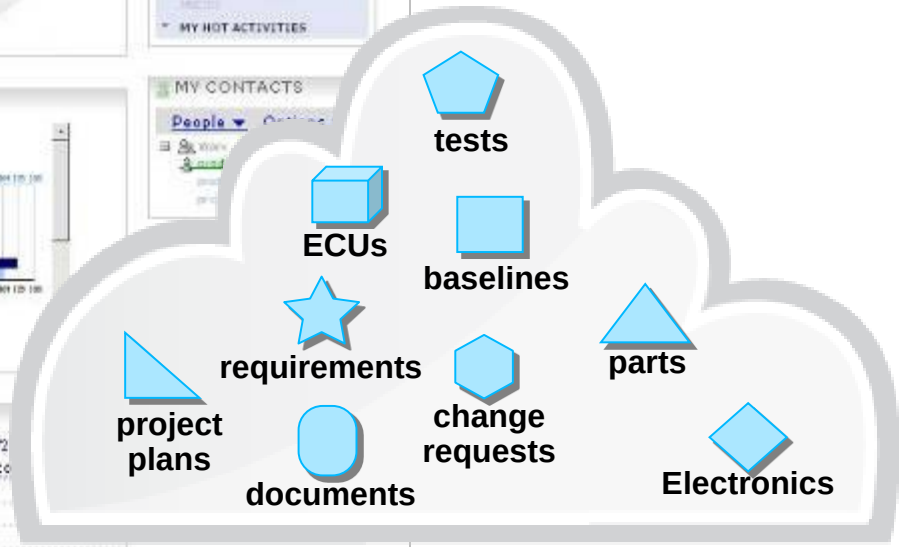


Establish a Platform for Integration

Enable a loosely coupled "web" of linked engineering data



Capture disparate data for project visibility



Jazz Dashboard

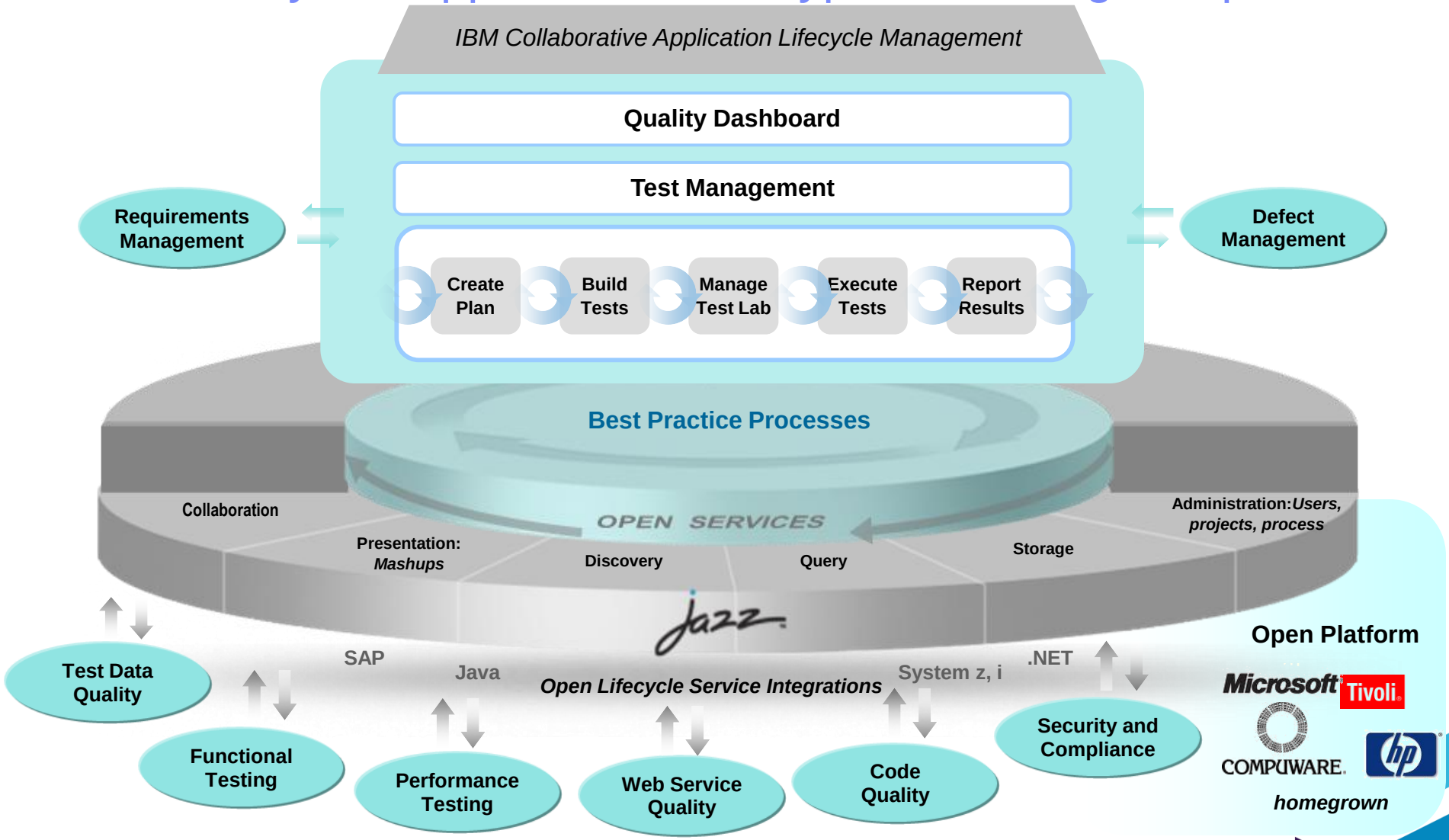
Integration based on the principals of the Internet

Open Services for Lifecycle Collaboration

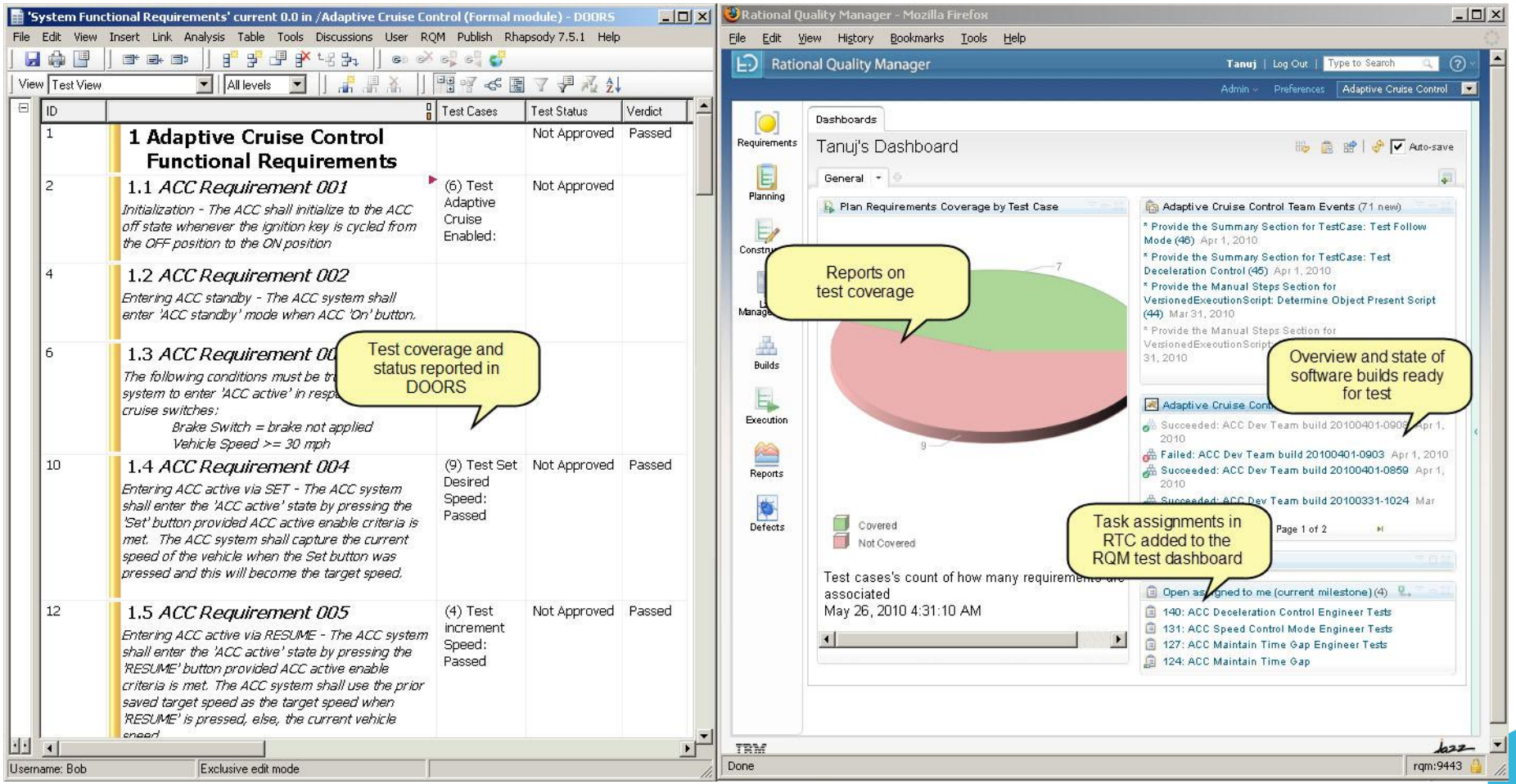
Open interfaces.
Open possibilities.

www.open-services.net

Quality Management offers a centralized test management hub and full lifecycle support across all types of testing and platforms



Test management integrates test planning and execution with requirements



IBM Rational Quality Manager

A central hub for business-driven software quality

Mitigate business risk with collaboration

- ✔ Stakeholder and team coordination reduces mistakes
- ✔ Risk identification and management leads to educated prioritization decisions
- ✔ Test traceability linked to business requirements improves customer satisfaction

Improve operational efficiency with automation

- ✔ Running tests earlier leads to reduced repair costs
- ✔ Running more tests in less time improves coverage
- ✔ Reducing manual labor leads to fewer testing errors
- ✔ Lab configuration automation improves efficiency and asset utilization

Make confident decisions with effortless reporting

- ✔ Real-time dashboards enable proactive risk management
- ✔ Customizable reports facilitate ongoing process improvement

105

Cut risk and cost

Collaborate seamlessly to reduce rework and the cost of bugs with integrated processes aligned to business goals

Customer Speak!

Unify the team through real-time, in-context collaboration

A single, dynamic quality contract provides clear and accountable direction

*"Some large projects have found that **41%** of all defects have their origin in bad requirements."**

Avoid disruption and achieve better business stability and project delivery predictability

Achieve quality objectives by understanding and controlling sources of risk

I just got a budget cut, what testing should I eliminate? What impact will it have on application production quality?

Lower the cost of delivering quality solutions

Orchestrate across teams with ALM integration for maximum transparency and traceability of assets

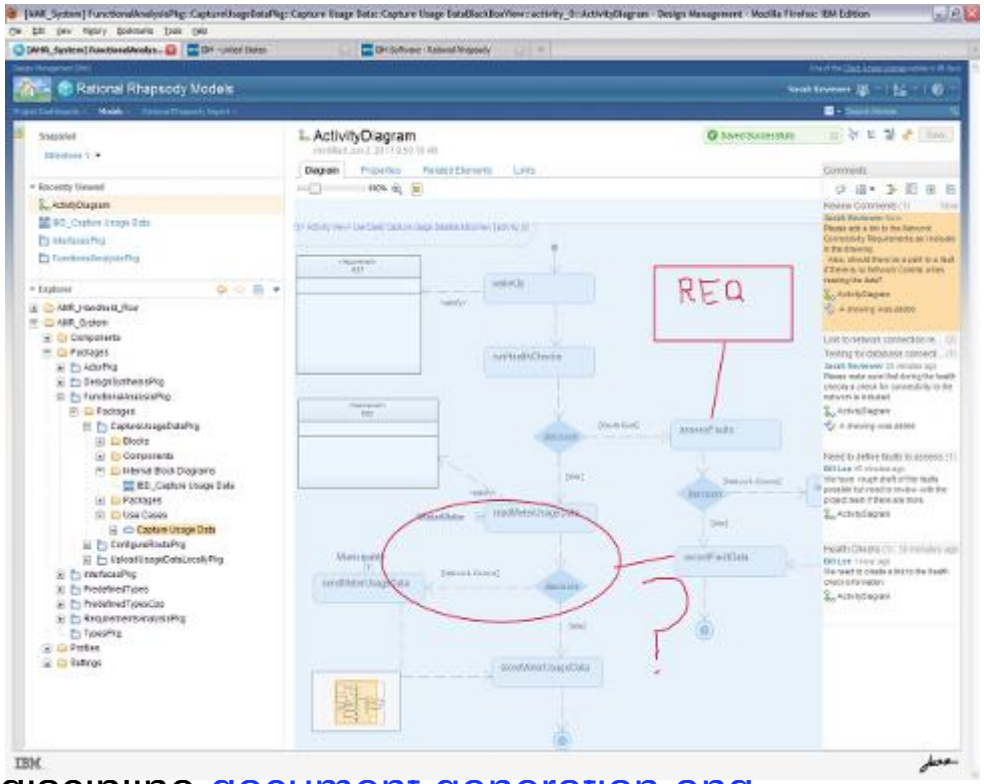
"Testing consumes **20% to 40+%** of the average software application life cycle effort"*

** Source: IBM*

Enable Cross-Domain Collaboration

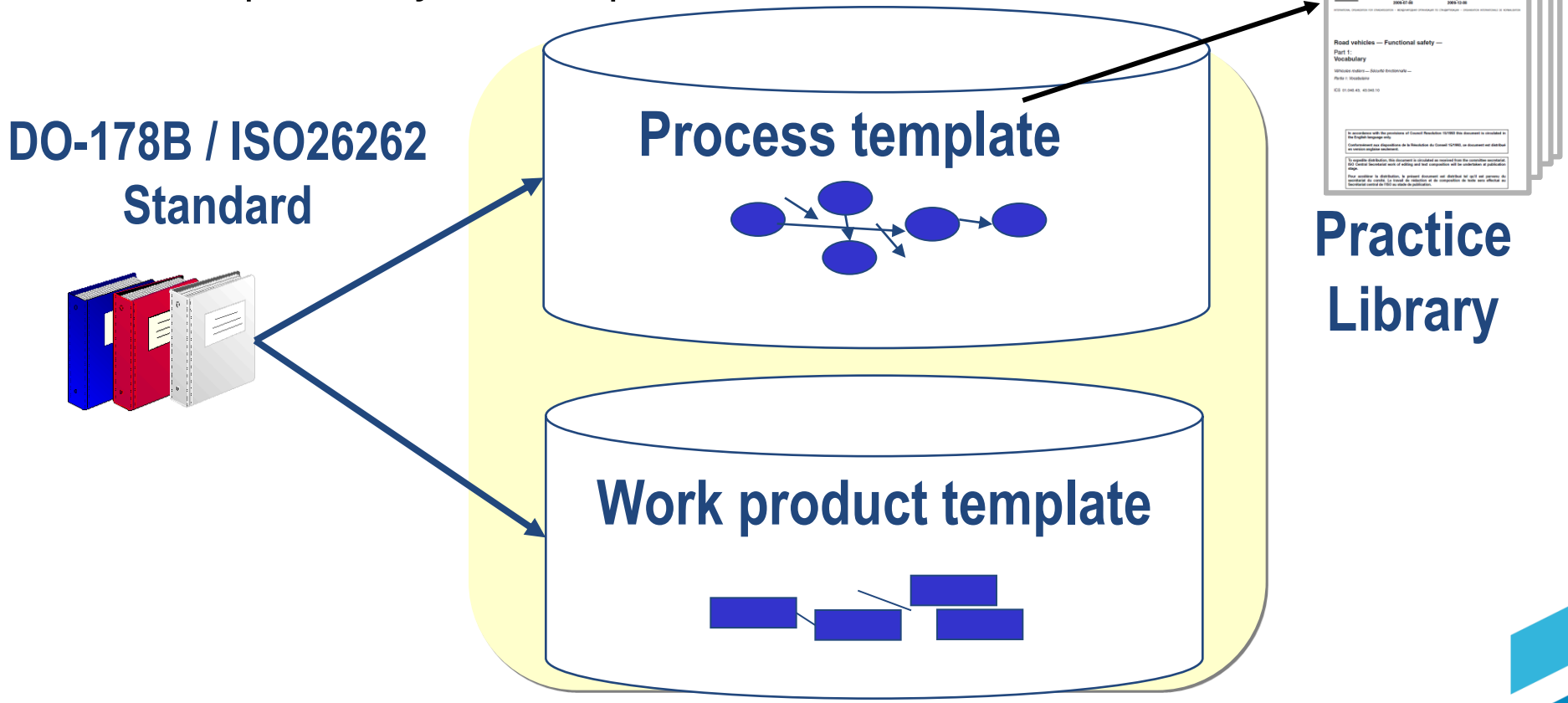
Enhance cross-team collaboration in systems & software design

- Maximize productivity and lower costs with a central location to store and access designs
- Collaborate among stakeholders on software architectures, deployment plans and system designs
- Shorten time-to-market with faster design reviews
- Satisfy regulatory demands with multi-discipline document generation and reporting



IBM practices for DO-178B and ISO 26262

- Supports processes and work products defined in the standards
- Implemented in the Rational Solution for Systems and Software Engineering
- Customizable for your business processes
- Tools to implement your own processes



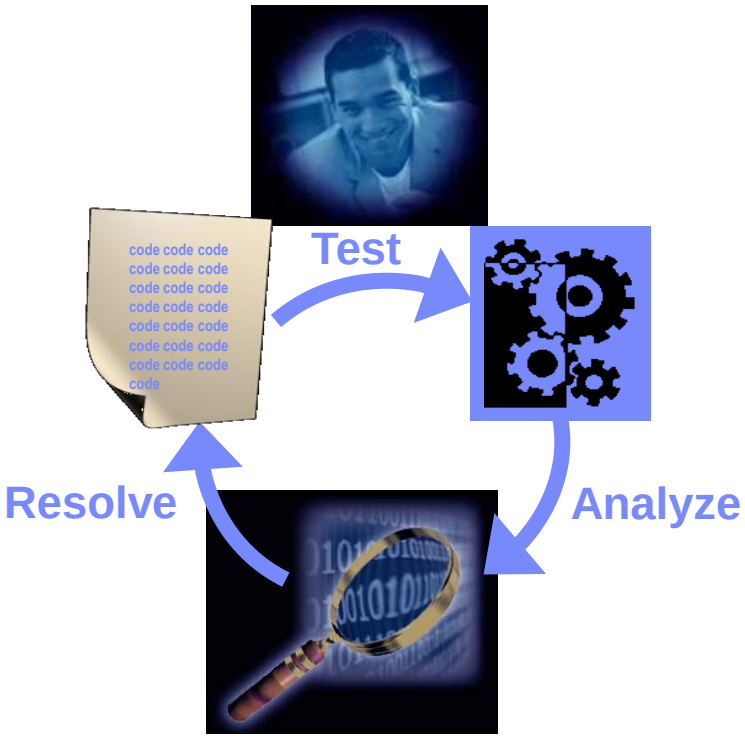
Embedded and real-time software testing challenges

Embedded and real-time systems are complex by nature

- Application Complexity
  - Strong timing constraints
  - Low memory footprints
  - Concurrent/Distributed/Networked
- Environment Complexity
  - Multiple RTOS/IDE/Chips vendors
  - Limited host-target connectivity
  - Low built-in debugging capabilities
- Process Complexity
  - Requirements and
  - Design translation errors
  - Difficult to maintain
  - Poor performance

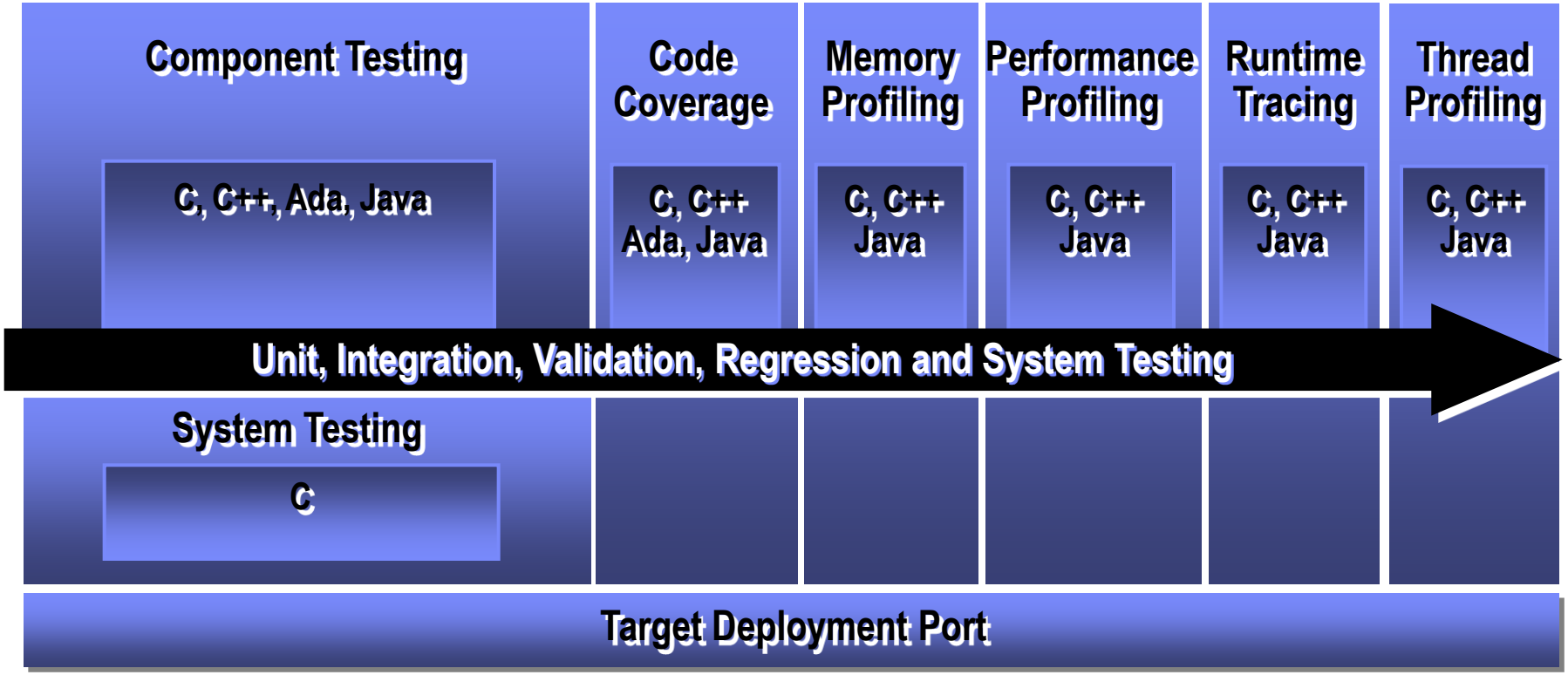
Test, analyze and resolve during development

- Test as you code
- Analyze while you test
  - Code coverage analysis
  - Memory profiling
  - Performance profiling
  - Runtime tracing
  - Thread profiling



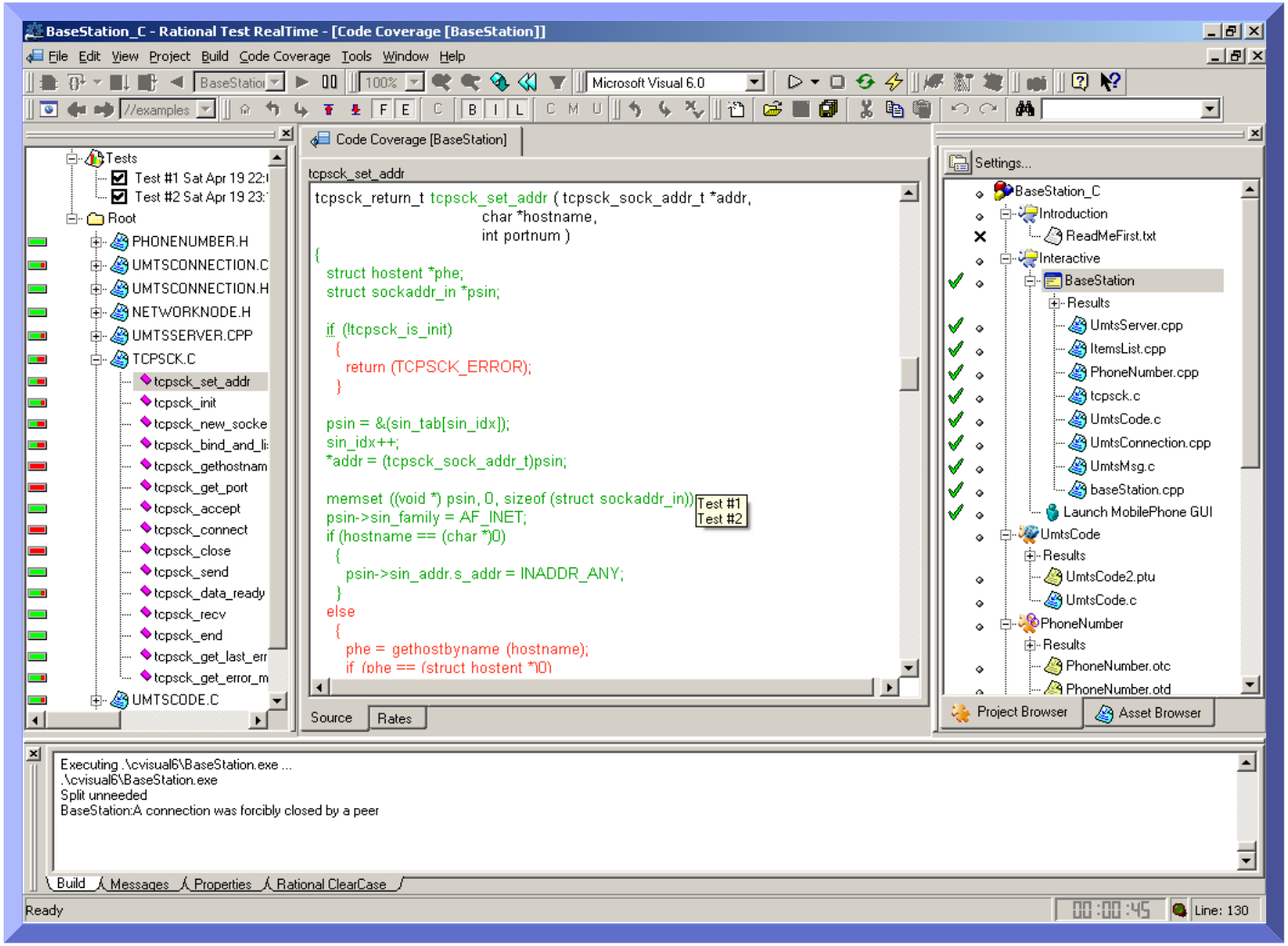
Test, analyze and resolve during development

Overview of IBM Rational Test RealTime features

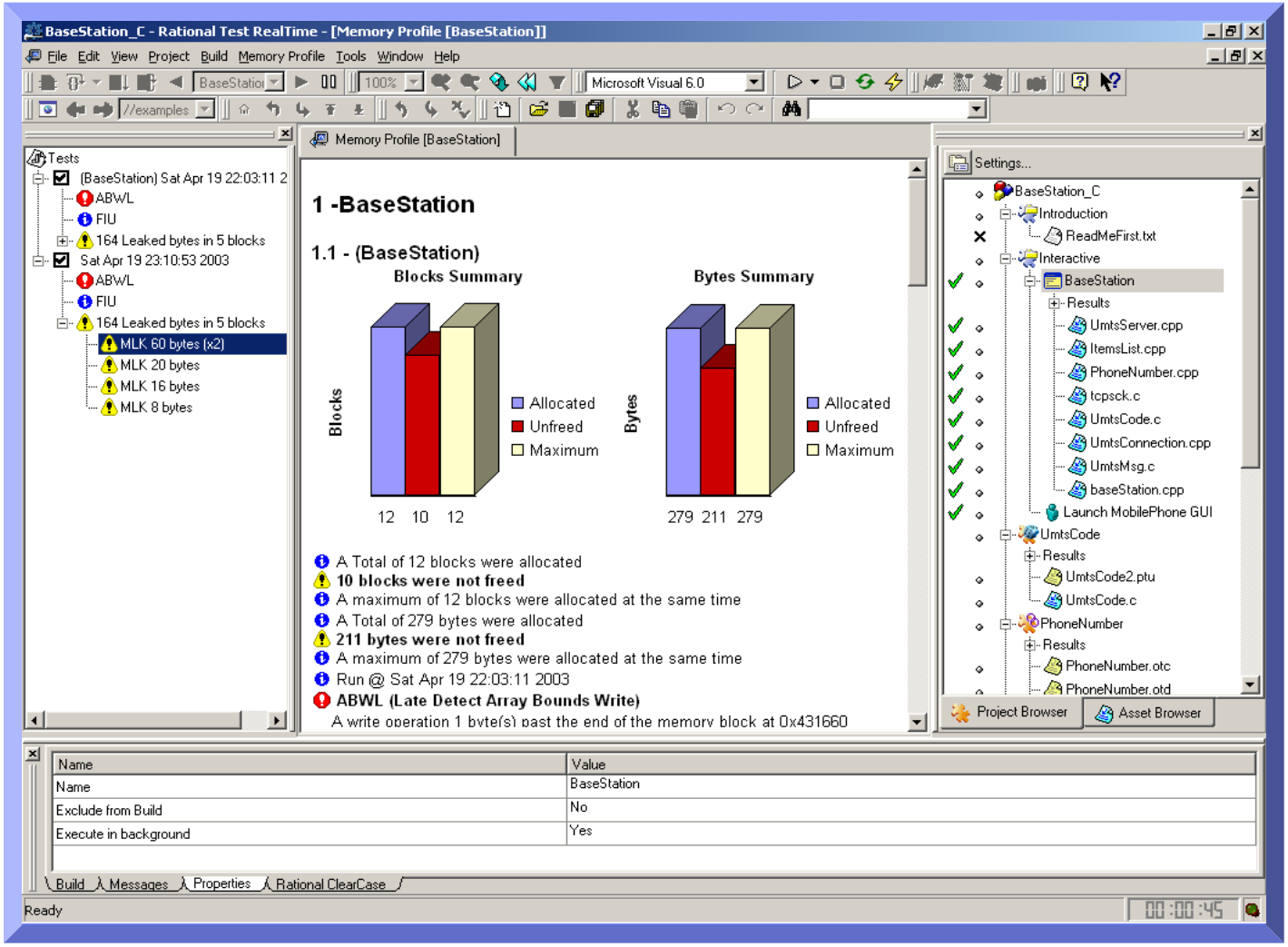


- **Built to achieve standards compliance**
  - DO-178B (Avionics), MISRA (Automotive), DEF STAN 00-55 (Defense)

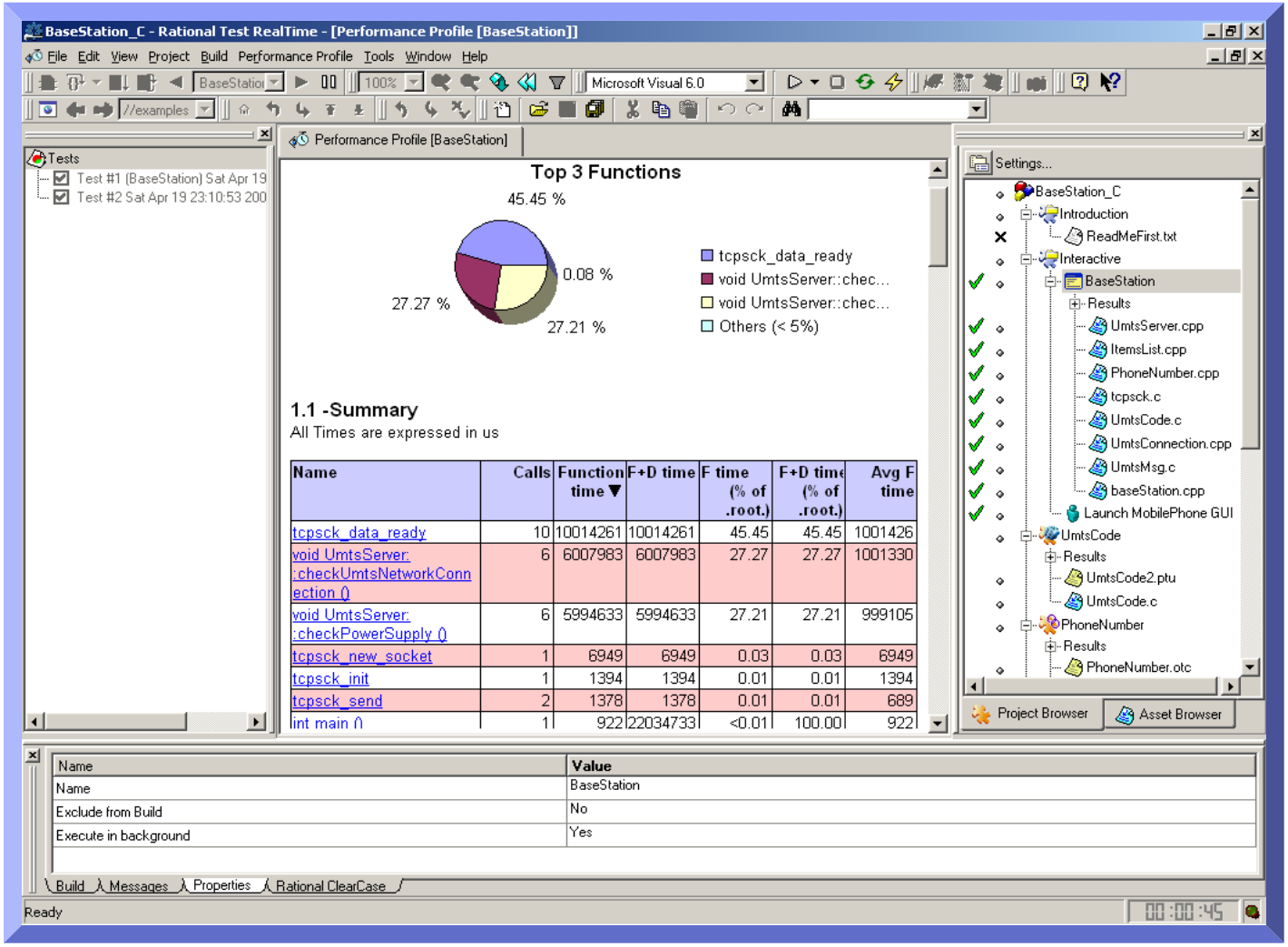
IBM Rational Test RealTime: *Code Coverage*



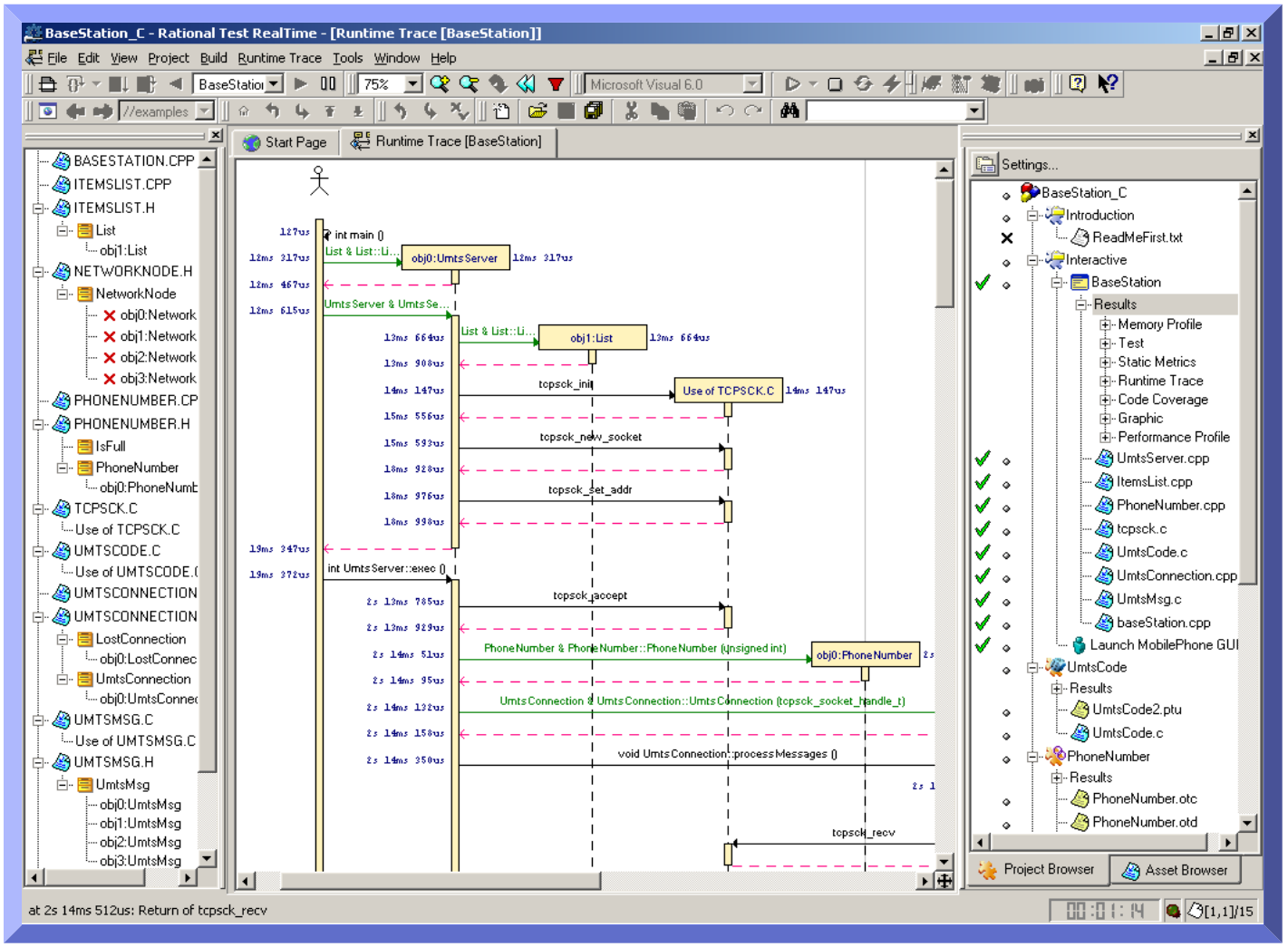
IBM Rational Test RealTime: *Memory Profiling*



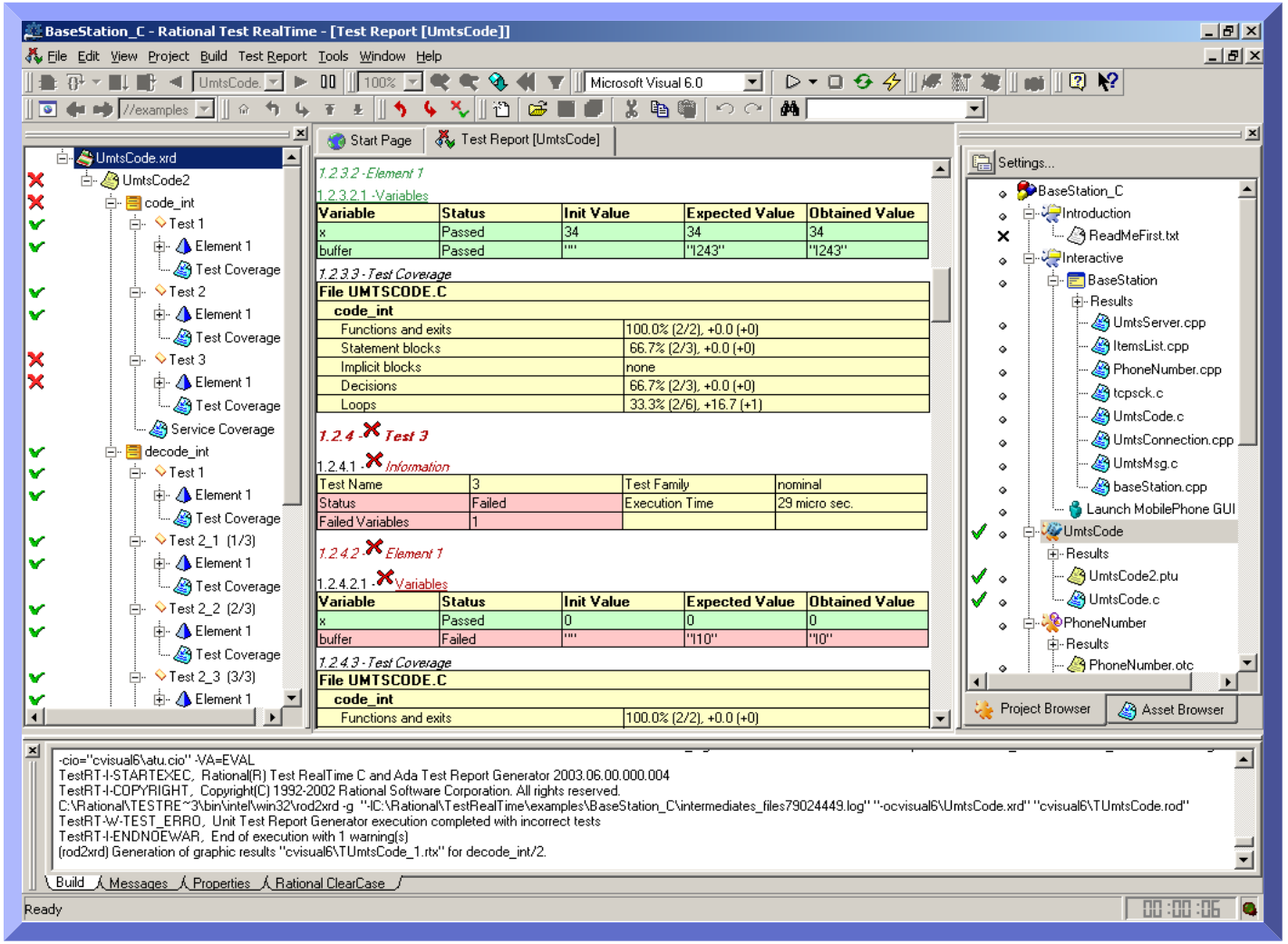
IBM Rational Test RealTime: *Performance Profiling*



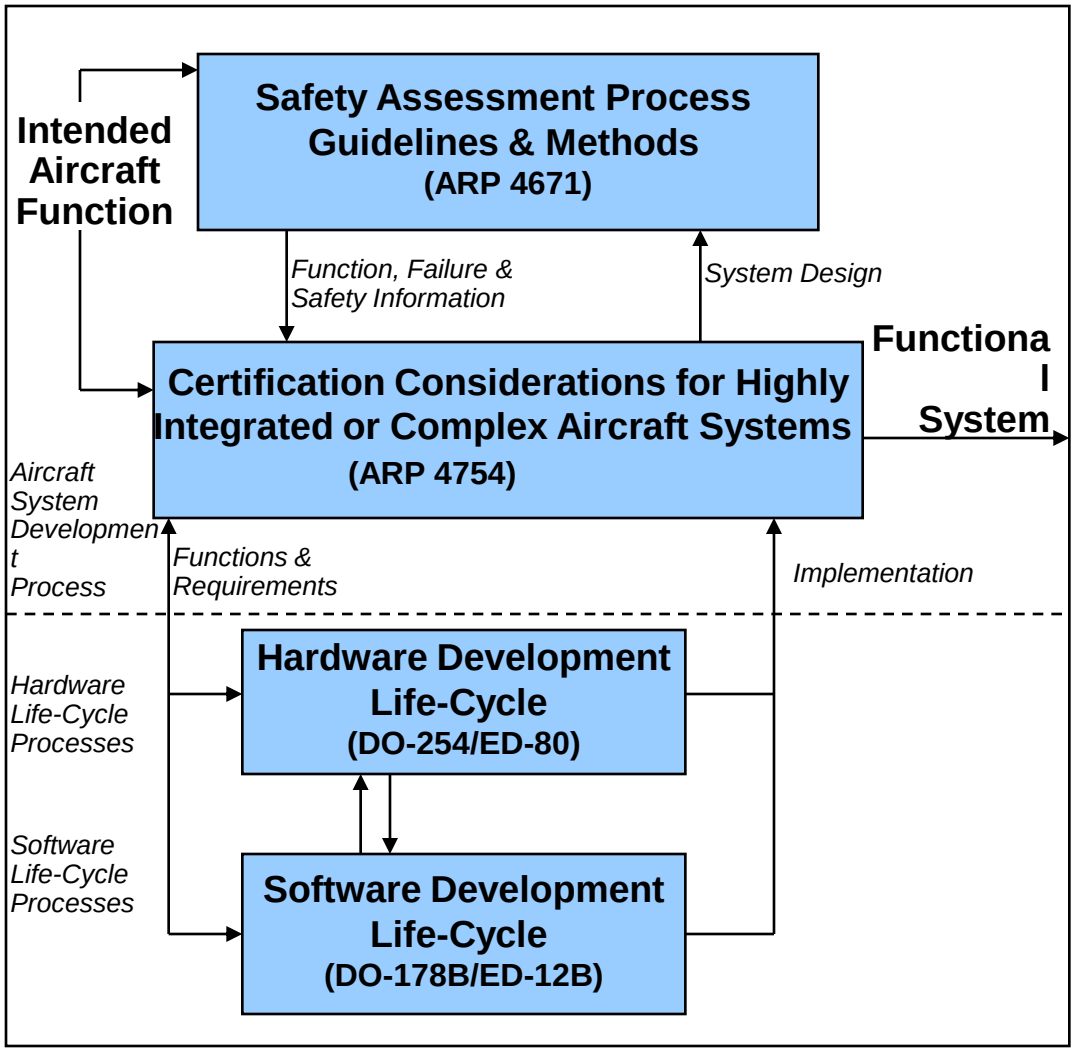
IBM Rational Test RealTime: *Runtime Tracing*



IBM Rational Test RealTime: *Test Report*



Overview of Certification Guidance for System, Safety, SW, HW Processes



- *As defined by SAE in ARP 4754*

Related Standards

Information Assurance

- NIST
- DIACAP (DoD)
- FISMA
- CC EAL levels

Other Standards

- IEEE/EIA 12207 (MIL-STD 498 ,J-STD-016)
- MIL-STD 882D
- ISO/IEC 15288:2008
- CMMI,etc (process improvement)
- AQAP-160 (NATO)
- ITAR
- ISO/IEC 15939 (Software Measurement Process)

…

Details on DO-178B/ED-12B

- DO-178B and ED-12B were developed by a broad committee of industry representatives from around the world. The official working groups were RTCA SC-167 and EUROCAE WG-12, and comprised representatives of the FAA, CAA, Boeing, Aerospatiale, Bendix/King, Veridatas, NASA, British Aerospace, Smiths Industries, Litton Aero, Rockwell Collins, Honeywell, Deutsche Airbus, ARINC, SNECMA, GE Aircraft Engines, Pratt & Whitney, Rolls-Royce, and many others.
- DO-178B/ED-12B provides guidance on designing, specifying, developing, testing, and deploying software in safety-critical avionics systems. It covers software life cycles, software planning processes, software development processes, software verification processes, software configuration management processes, software quality assurance processes, and other aspects of creating quality software for a safety-critical environment.
- In sum, DO-178B/ED-12B (developed by RTCA and EUROCAE) is a guideline for determining, in a consistent manner and with an acceptable level of confidence, that the software aspects of airborne systems and equipment comply with FAA and EASA airworthiness requirements.

DO-178B-Background

The number of objectives to be satisfied (with independence) is determined by the software assurance level.

Level	Failure condition	Objectives	With independence
A	Catastrophic	66	25
B	Hazardous	65	14
C	Major	57	2
D	Minor	28	2
E	No effect	0	0

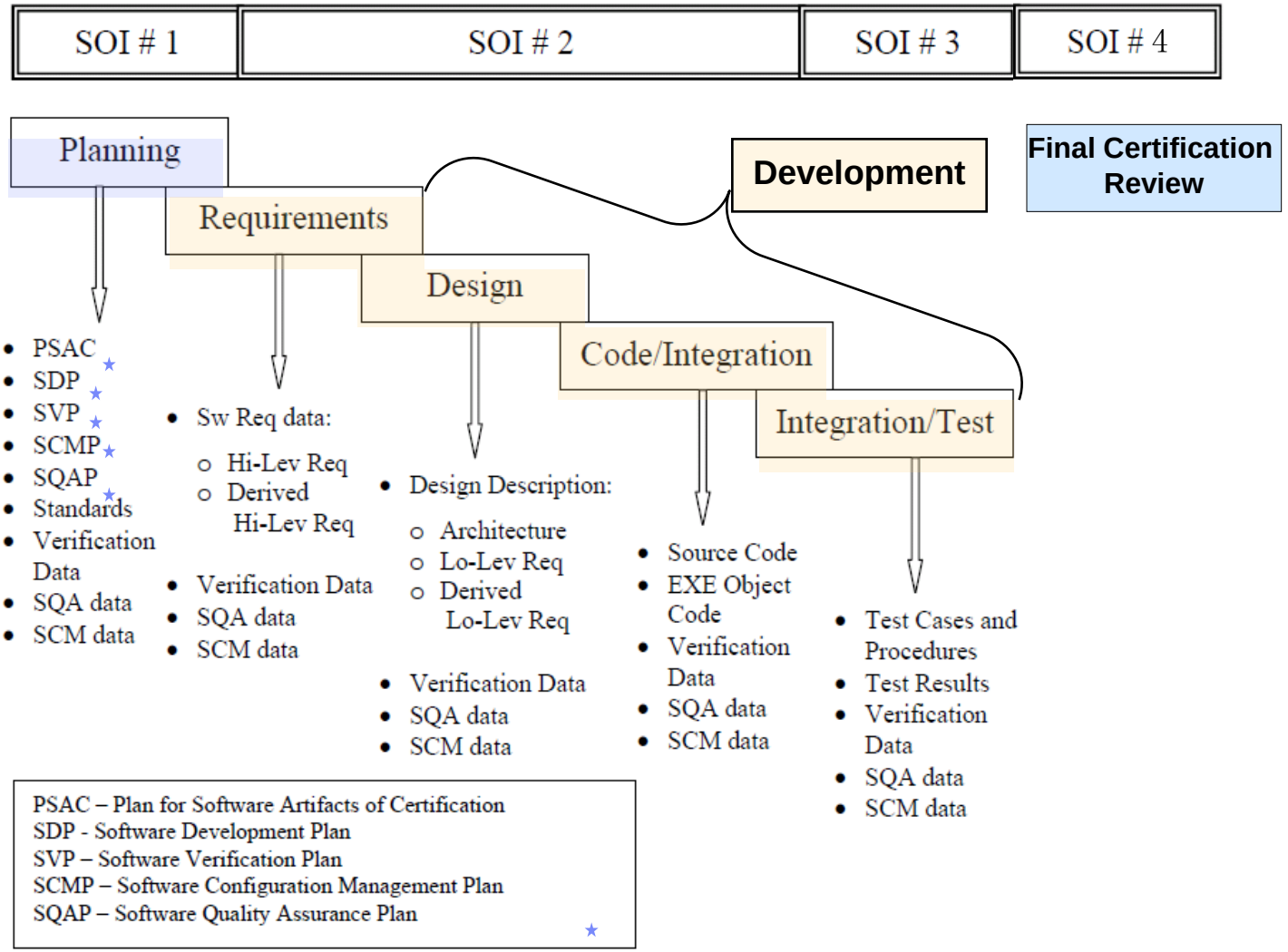
DO-178B-Background

DO-178B Processes

- Planning (section 4)
- Development (section 5)
- Verification (section 6)
- Configuration Management (section 7)
- Quality Assurance (section 8)

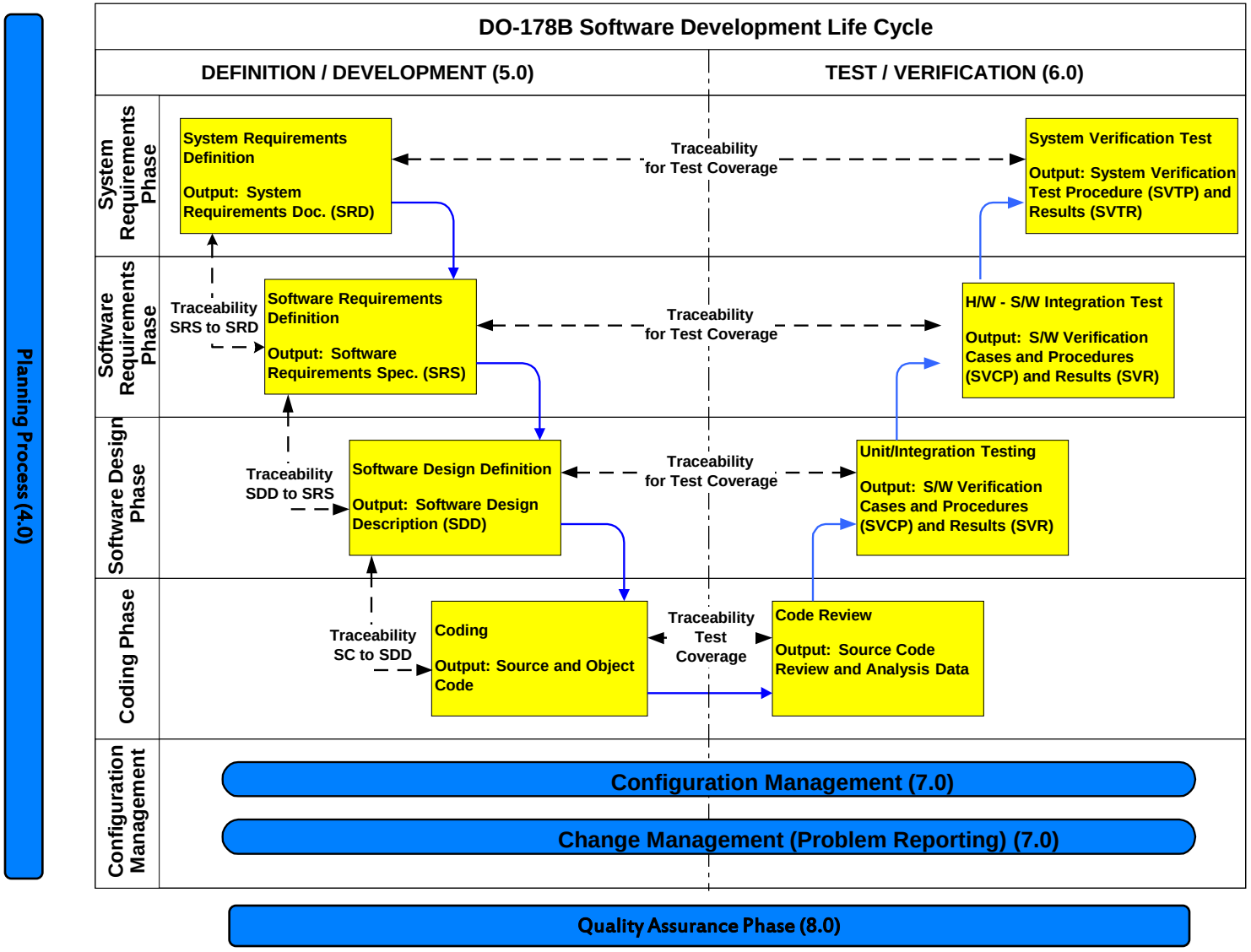
Processes have associated output documentation

DO-178B Processes

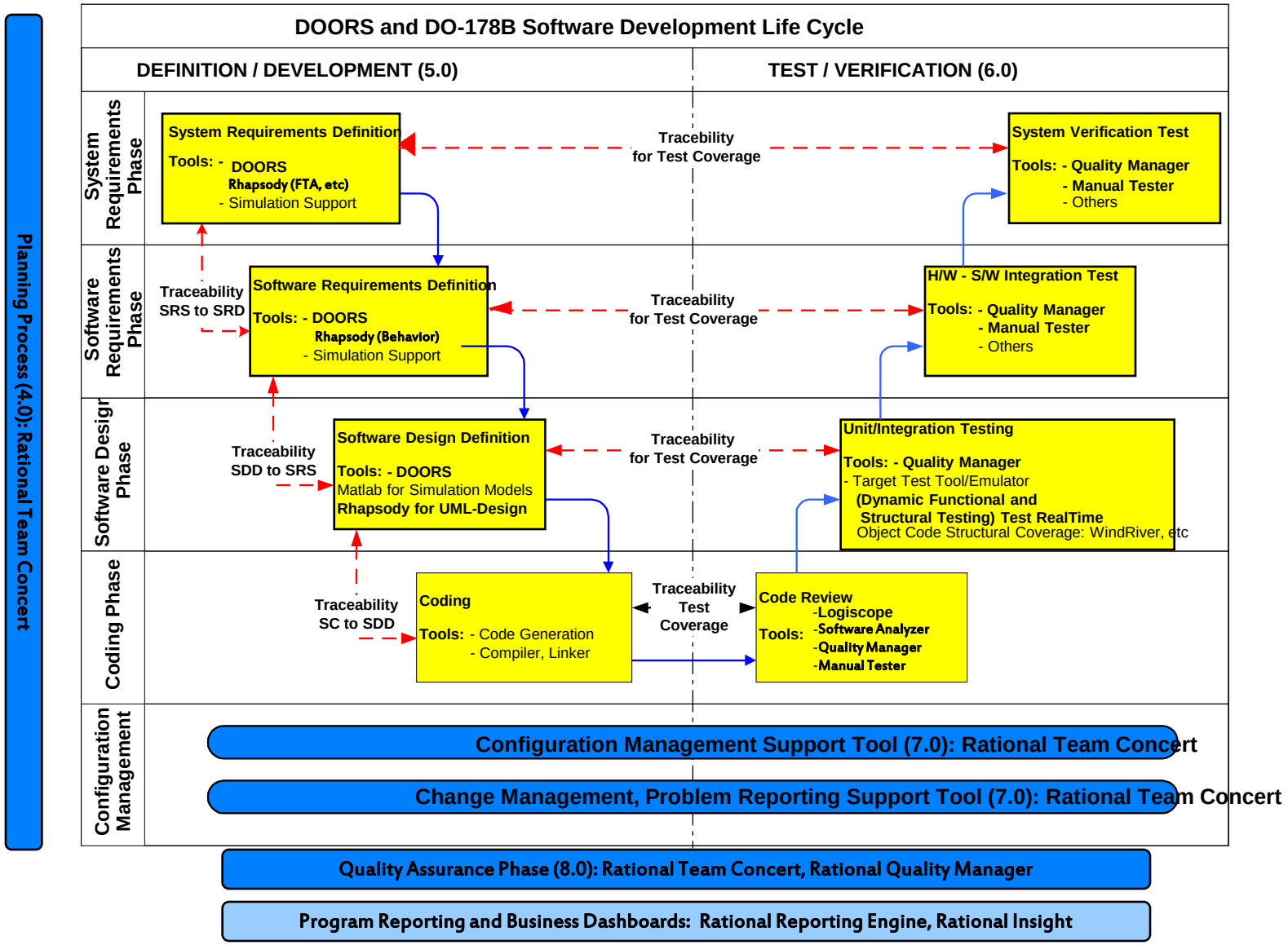


Verification, Configuration Mgmt, Quality Assurance

DO-178B Software Development Lifecycle



DO-178B Software Development Lifecycle

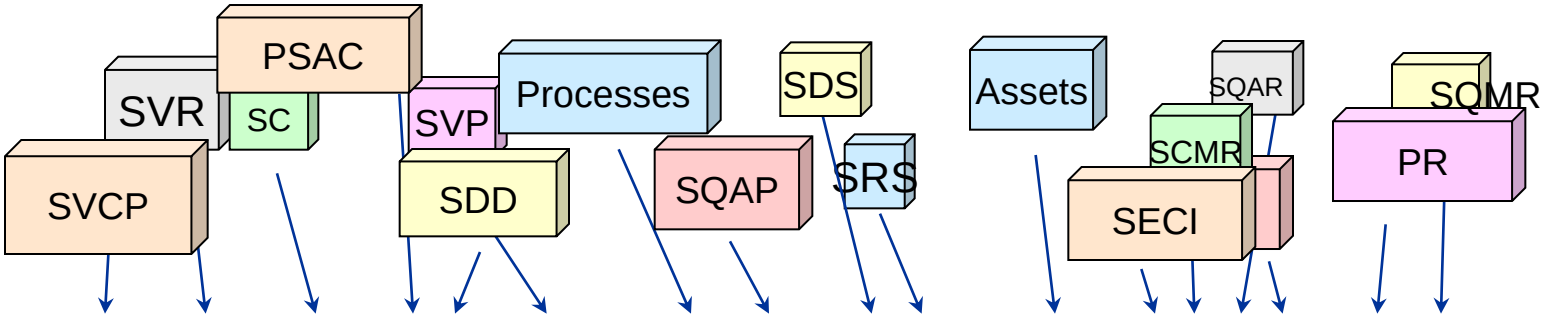


Planning process Process (4.0)

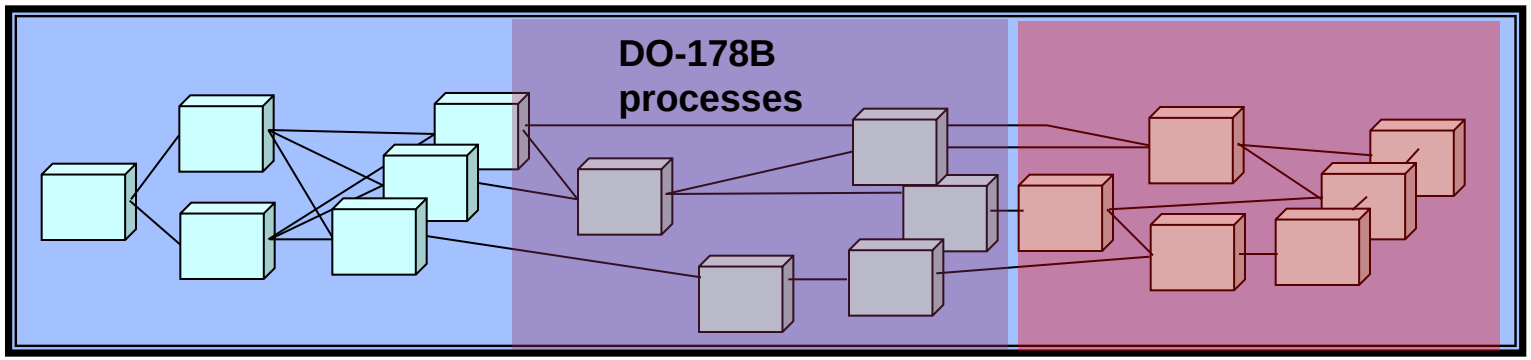
- Produces the software plans and standards that direct the software development processes and the integral processes (verification, SCM, SQA, certification liason).
- Identifies the transition criteria, interrelationship and sequencing among the processes.
- Software life cycle environment is defined
- Software development standards are defined.
- Ensure software plans conform to these documents.
- Ensure software plans are coordinated.

Visualize the DO-178B Development Process

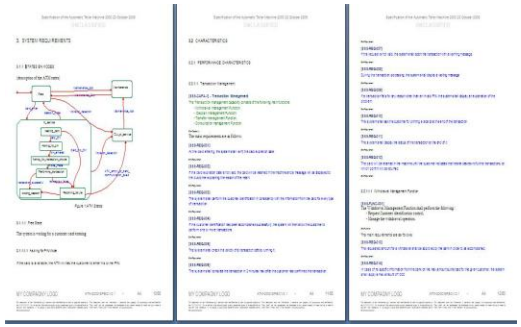
Disparate Data is Imported and Supplemented

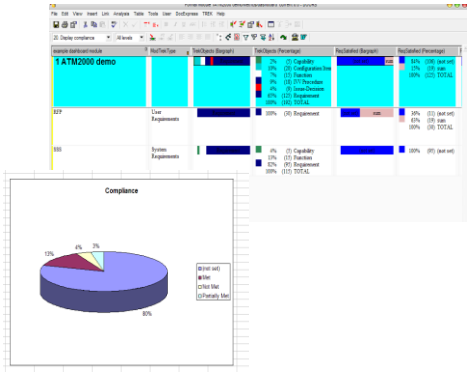


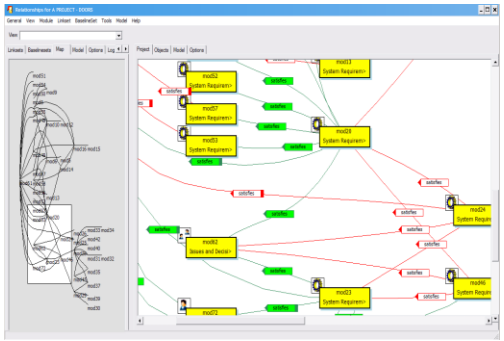
Visualized, Structured, Linked and Traced,



To Produce Reports of Managed Information For Certification



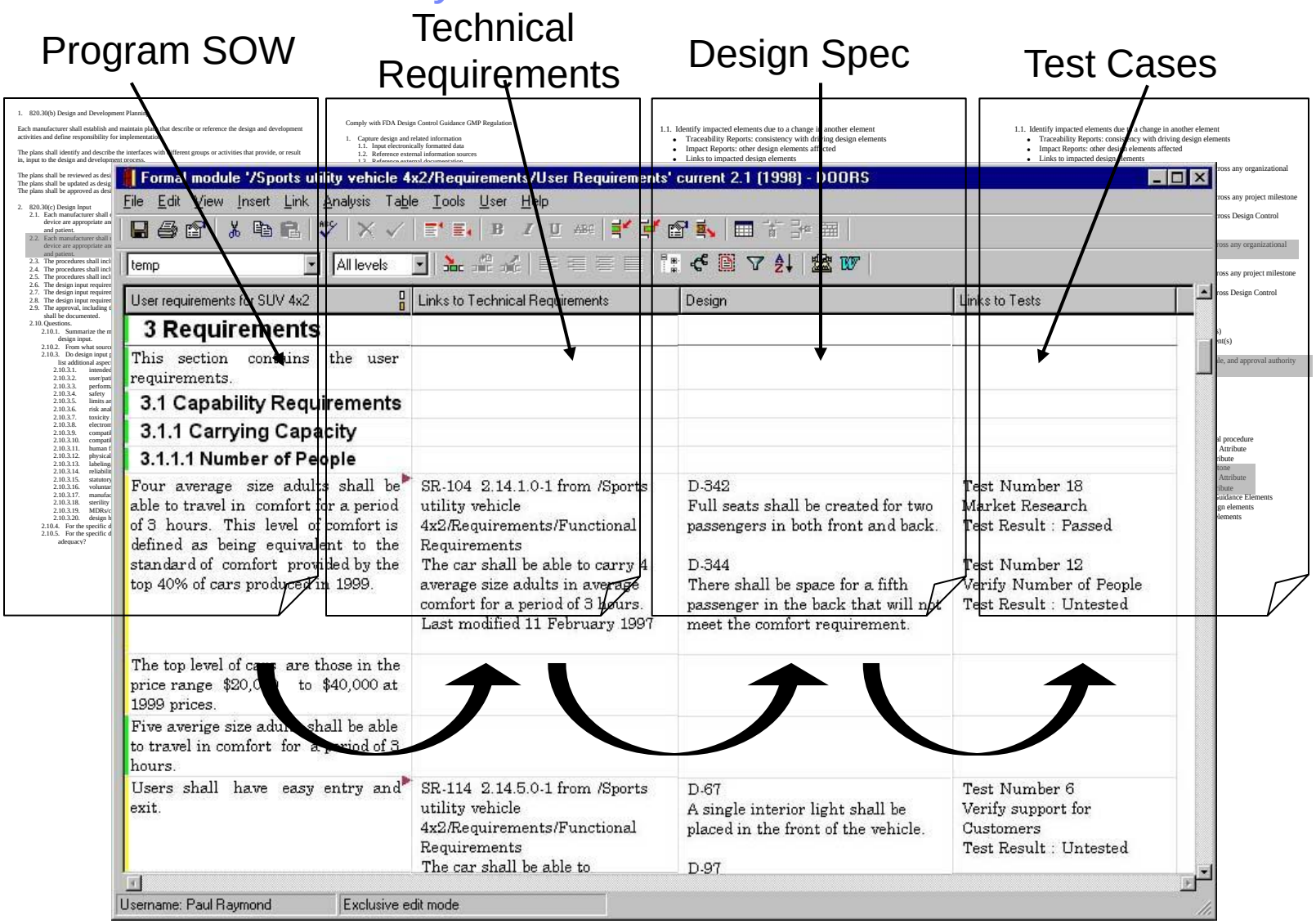




Requirements Process (5.1)

- Requirements must be verifiable, unambiguous, consistent, and well defined
  - If any requirement does not meet this criteria a Problem Report must be created to feed the issue back to the input source for clarification and correction
- System Requirements allocated to Software must be traceable to a High Level Software Requirement
- Each High Level Software Requirement must trace to one or more System Requirement (except for derived requirements)
- Each High Level Software Requirement must trace to one or more Low Level Software Requirement.
- Each Low Level Software Requirement must trace to one or more High Level Software Requirement (except for derived requirements).
- All derived requirements must be provided to the system safety assessment process
- All source code that is developed should be traceable, verifiable, consistent and correctly implement the Low Level Software Requirements

DOORS Traceability view

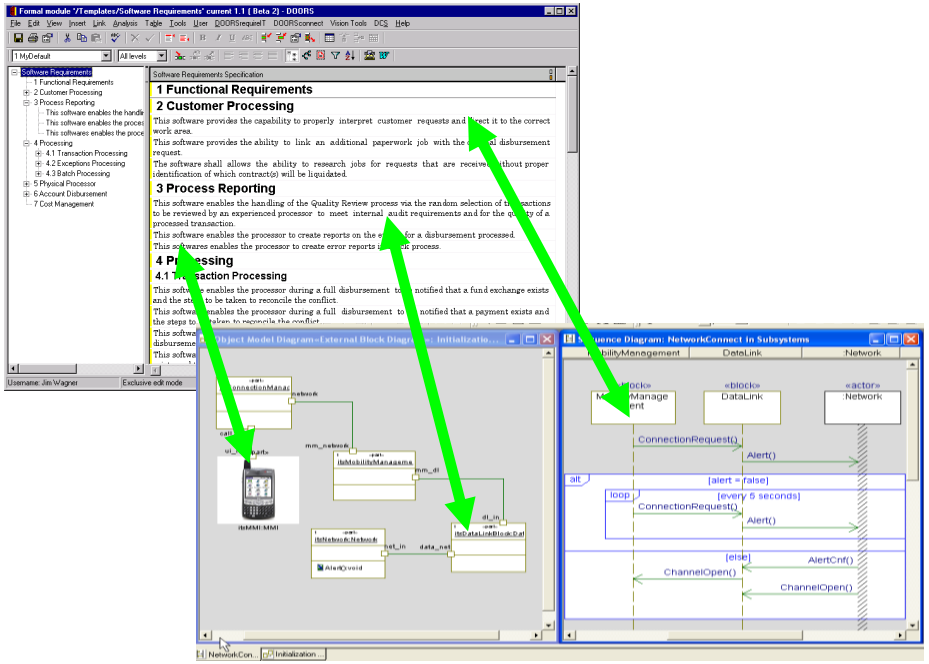


Design and Coding Processes (5.2 & 5.3)

- Low Level Software Requirements and software architecture should conform to the Software Design Standard and be traceable, verifiable, and consistent
- All source code that is developed should be traceable, verifiable, consistent and correctly implement the Low Level Software Requirements

Linking requirements to Rhapsody design models

- Traceability helps prove conformance and compliance
- Easily check for:
  - Requirements not satisfied by the design
  - Design elements with no linked requirements – 'gold plating'
- Fast and complete impact analysis
  - Assess full impact of changes BEFORE they are made
  - Ensure approved changes are fully implemented



Integration Process (5.4)

- Object code is loaded onto the target computer for hardware/software integration
- Inadequate or erroneous inputs detected require creating a Problem Report and feeding the information back to the appropriate process for clarification and correction.
- Evidence that deactivated code is disabled should be available

Traceability (5.5)

- Traceability between system requirements and software requirements should be provided to enable verification of the complete implementation of the system requirements and give visibility to the derived requirements

- Traceability between the low-level requirements and high-level requirements should be provided to give visibility to the derived requirements and the architectural design decisions made during the software design process, and allow verification of the complete implementation of the high-level requirements.

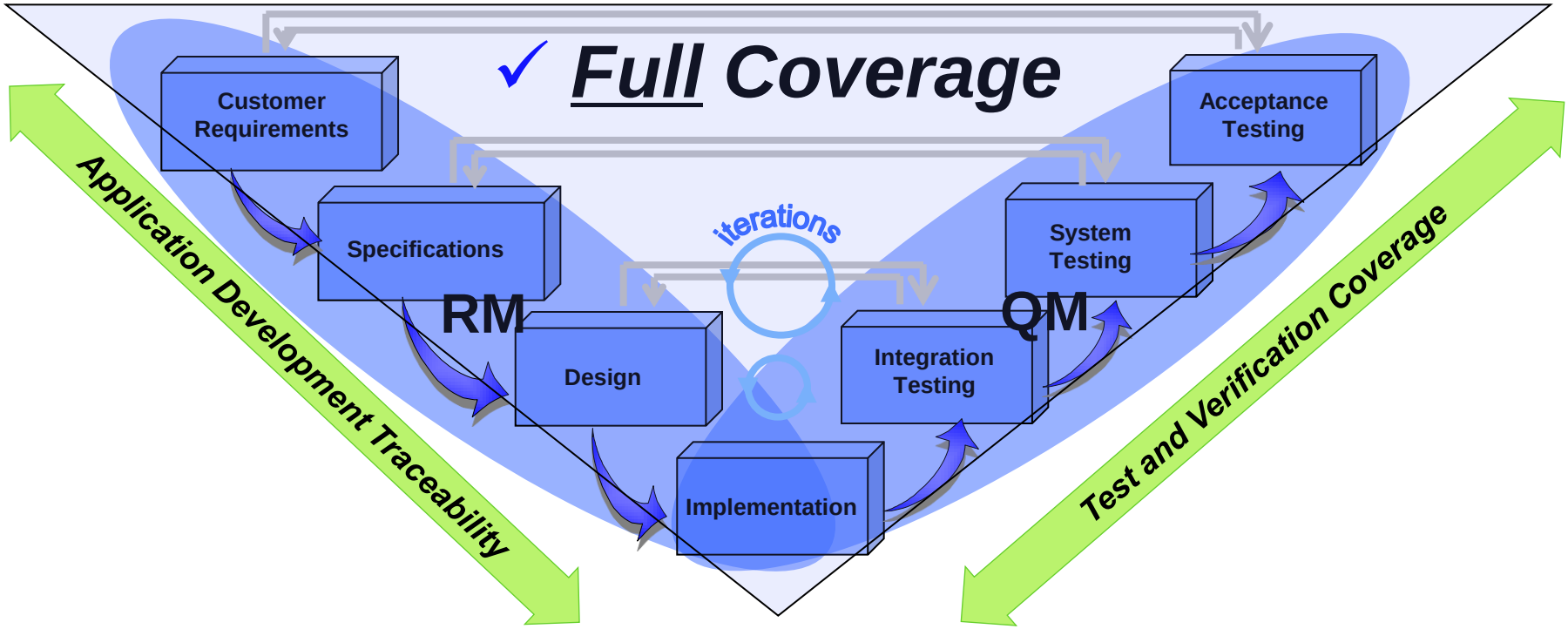
- Traceability between Source Code and low-level requirements should be provided to enable verification of the absence of undocumented Source Code and verification of the complete implementation of the low-level requirements.

Verification Process (6.0)

- Verification process ensures the software fulfills all the requirements and is not simply testing (detecting for errors), but showing the absence of errors.
- It Verifies that all lower level artifacts satisfy higher level artifacts
- Traceability between Requirements and Test Cases is accomplished through requirements based coverage analysis.
- Traceability between code structure and test cases is accomplished through structural coverage analysis
- Each Requirement is traceable to the code that implements it and the review, test, or analysis that verifies it
- Ensure that implemented functionality traces back to requirements and tests test for this. Dead code or code not traceable to requirements needs to be eliminated

IBM Offers A Unique Solution-DO-178B V&V coverage

That Ensures Entire Lifecycle Collaboration and Traceability



IBM's full life cycle coverage and traceability solution

- ✓ Common set of clear requirements shared by team
- ✓ Don't miss out critical requirements
- ✓ Assess requirements change impact
- ✓ Identify most critical requirements to test
- ✓ Prove compliance (audit-ability)

DO-178B Detailed Testing Requirements

You have to test every line, every branch, every condition using Reqs. Based Testing!

DO-178B defines specific verification objectives that must be satisfied;these include:

1. Functional Verification of software
   a. Requirements-based testing
   b. Robustness testing
2. Structural Coverage Analysis for each DO-178B level:

Level	Coverage	Coverage Requirements
Level A	MCDC	Level B + 100% Modified Condition Decision Coverage
Level B	DC	Level C + 100% Decision Coverage
Level C	SC	Level D + 100% Statement (or Line) Coverage
Level D		100% Requirements Coverage Requirements
Level E		No Coverage Requirements

- DO-178B Section 6.4.1 – Need to test on target
- DO-178B Section 12.2 – Tools that can introduce or miss errors in code need to be qualified

Coverage Criteria	Statement Coverage	Decision Coverage	Condition Coverage	Condition/ Decision Coverage	MC/DC	Multiple Condition Coverage
Every point of entry and exit in the program has been invoked at least once		•	•	•	•	•
Every statement in the program has been invoked at least once	•					
Every decision in the program has taken all possible outcomes at least once		•		•	•	•
Every condition in a decision in the program has taken all possible outcomes at least once			•	•	•	•
Every condition in a decision has been shown to independently affect that decision's outcome					•	•
Every combination of condition outcomes within a decision has been invoked at least once						•

DO-178B Detailed Testing Requirements

DO-178B Qualification Kits Available

IBM Rational Solutions:

- **IBM Rational Test RealTime (System Test, Dynamic Code Coverage for Level A MC/DC & Multiple Decision Coverage, Static Analysis, Memory, Performance & Thread profiling Analysis, Dynamic Trace Capture, Unit Test Automation, Software Metrics, Reporting)**

Configuration Management Process (7.0)

- DO-178B requires
  - Each configuration item to be uniquely identified
  - Baselines of configuration items that can be protected from change
  - A configuration item should be traced to the configuration item it was derived from (lineage and history)
  - Baselines should be traceable to the baselines from which they are derived
  - Builds should be reproducible (replicate executable object code)
  - Provide evidence of change approvals
    - Software configuration index (SCI)
    - Software life cycle environment configuration index (SECI)

TABLE 7-1
SCM PROCESS OBJECTIVES ASSOCIATED WITH CC1 and CC2 DATA

SCM Process Objective	Reference	CC1	CC2
Configuration Identification	7.2.1	•	•
Baselines	7.2.2a, b, c, d, e	•	
Traceability	7.2.2f, g	•	•
Problem Reporting	7.2.3	•	
Change Control - integrity and identification	7.2.4a, b	•	•
Change Control - tracking	7.2.4c, d, e	•	
Change Review	7.2.5	•	
Configuration Status Accounting	7.2.6	•	
Retrieval	7.2.7a	•	•
Protection against Unauthorized Changes	7.2.5b(1)	•	•
Media Selection, Refreshing, Duplication	7.2.7b(2), (3), (4), c	•	
Release	7.2.7d	•	
Data Retention	7.2.7e	•	•

Change Management (7.2.3)

- DO-178B requires a Problem Reporting system to document any modification to formal baseline
- This means at a certain stage of the project a Problem Report(PR) has to be generated to document the modification
- PR's are also used to cover change request from the customer.
- PR's need to identify/trace to the items to be modified (files, requirements, documents, test cases, etc.)

DO-178B Objectives

- Following tables describing the 10 categories of DO-178B objectives…

Table A-1

Software Planning Process

	Objective		Applicability by SW Level				Output		Control Category by SW level			
	Description	Ref.	A	B	C	D	Description	Ref.	A	B	C	D
1	Software development and integral processes activities are defined.	4.1a 4.3	○	○	○	○	Plan for Software Aspects of Certification	11.1	①	①	①	①
							Software Development Plan	11.2	①	①	②	②
							Software Verification Plan	11.3	①	①	②	②
							SCM Plan	11.4	①	①	②	②
							SQA Plan	11.5	①	①	②	②
2	Transition criteria, inter-relationships and sequencing among processes are defined.	4.1b 4.3	○	○	○							
3	Software life cycle environment is defined.	4.1c	○	○	○							
4	Additional considerations are addressed.	4.1d	○	○	○	○						
5	Software development standards are defined.	4.1e	○	○	○		SW Requirements Standards	11.6	①	①	②	
							SW Design Standards	11.7	①	①	②	
							SW Code Standards	11.8	①	①	②	
6	Software plans comply with this document.	4.1f 4.6	○	○	○		SQA Records	11.19	②	②	②	
							Software Verification Results	11.14	②	②	②	
7	Software plans are coordinated.	4.1g 4.6	○	○	○		SQA Records	11.19	②	②	②	
							Software Verification Results	11.14	②	②	②	

LEGEND:		
	●	The objective should be satisfied with independence.
	○	The objective should be satisfied.
	Blank	Satisfaction of objective is at applicant's discretion.
	①	Data satisfies the objectives of Control Category 1 (CC1).
	②	Data satisfies the objectives of Control Category 2 (CC2).

Table A-2
Software Development Processes

	Objective		Applicability by SW Level				Output		Control Category by SW level			
	Description	Ref.	A	B	C	D	Description	Ref.	A	B	C	D
1	High-level requirements are developed.	5.1.1a	○	○	○	○	Software Requirements Data	11.9	①	①	①	①
2	Derived high-level requirements are defined.	5.1.1b	○	○	○	○	Software Requirements Data	11.9	①	①	①	①
3	Software architecture is developed.	5.2.1a	○	○	○	○	Design Description	11.10	①	①	②	②
4	Low-level requirements are developed.	5.2.1a	○	○	○	○	Design Description	11.10	①	①	②	②
5	Derived low-level requirements are defined.	5.2.1b	○	○	○	○	Design Description	11.10	①	①	②	②
6	Source Code is developed.	5.3.1a	○	○	○	○	Source Code	11.11	①	①	①	①
7	Executable Object Code is produced and integrated in the target computer.	5.4.1a	○	○	○	○	Executable Object Code	11.12	①	①	①	①

LEGEND:

●	The objective should be satisfied with independence.
○	The objective should be satisfied.
Blank	Satisfaction of objective is at applicant's discretion.
①	Data satisfies the objectives of Control Category 1 (CC1).
②	Data satisfies the objectives of Control Category 2 (CC2).

Table A-3
Verification of Outputs of Software Requirements Process

	Objective		Applicability by SW Level				Output		Control Category by SW level			
	Description	Ref.	A	B	C	D	Description	Ref.	A	B	C	D
1	Software high-level requirements comply with system requirements.	6.3.1a	●	●	○	○	Software Verification Results	11.14	②	②	②	②
2	High-level requirements are accurate and consistent.	6.3.1b	●	●	○	○	Software Verification Results	11.14	②	②	②	②
3	High-level requirements are compatible with target computer.	6.3.1c	○	○			Software Verification Results	11.14	②	②		
4	High-level requirements are verifiable.	6.3.1d	○	○	○		Software Verification Results	11.14	②	②	②	
5	High-level requirements conform to standards.	6.3.1e	○	○	○		Software Verification Results	11.14	②	②	②	
6	High-level requirements are traceable to system requirements.	6.3.1f	○	○	○	○	Software Verification Results	11.14	②	②	②	②
7	Algorithms are accurate.	6.3.1g	●	●	○		Software Verification Results	11.14	②	②	②	

LEGEND:		
	●	The objective should be satisfied with independence.
	○	The objective should be satisfied.
	Blank	Satisfaction of objective is at applicant's discretion.
	①	Data satisfies the objectives of Control Category 1 (CC1).
	②	Data satisfies the objectives of Control Category 2 (CC2).

Table A-4
Verification of Outputs of Software Design Process

	Objective		Applicability by SW Level				Output		Control Category by SW level			
	Description	Ref.	A	B	C	D	Description	Ref.	A	B	C	D
1	Low-level requirements comply with high-level requirements.	6.3.2a	●	●	○		Software Verification Results	11.14	②	②	②	
2	Low-level requirements are accurate and consistent.	6.3.2b	●	●	○		Software Verification Results	11.14	②	②	②	
3	Low-level requirements are compatible with target computer.	6.3.2c	○	○			Software Verification Results	11.14	②	②		
4	Low-level requirements are verifiable.	6.3.2d	○	○			Software Verification Results	11.14	②	②		
5	Low-level requirements conform to standards.	6.3.2e	○	○	○		Software Verification Results	11.14	②	②	②	
6	Low-level requirements are traceable to high-level requirements.	6.3.2f	○	○	○		Software Verification Results	11.14	②	②	②	
7	Algorithms are accurate.	6.3.2g	●	●	○		Software Verification Results	11.14	②	②	②	
8	Software architecture is compatible with high-level requirements.	6.3.3a	●	○	○		Software Verification Results	11.14	②	②	②	
9	Software architecture is consistent.	6.3.2b	●	○	○		Software Verification Results	11.14	②	②	②	
10	Software architecture is compatible with target computer.	6.3.3c	○	○			Software Verification Results	11.14	②	②		
11	Software architecture is verifiable.	6.3.3d	○	○			Software Verification Results	11.14	②	②		
12	Software architecture conforms to standards.	6.3.3e	○	○	○		Software Verification Results	11.14	②	②	②	
13	Software partitioning integrity is confirmed.	6.3.3f	●	○	○	○	Software Verification Results	11.14	②	②	②	②

LEGEND:

●	The objective should be satisfied with independence.
○	The objective should be satisfied.
Blank	Satisfaction of objective is at applicant's discretion.
①	Data satisfies the objectives of Control Category 1 (CC1).
②	Data satisfies the objectives of Control Category 2 (CC2).

Table A-5
Verification of Outputs of Software Coding & Integration Processes

	Objective		Applicability by SW Level				Output		Control Category by SW level			
	Description	Ref.	A	B	C	D	Description	Ref.	A	B	C	D
1	Source Code complies with low-level requirements.	6.3.4a	●	●	○		Software Verification Results	11.14	②	②	②	
2	Source Code complies with software architecture.	6.3.4b	●	○	○		Software Verification Results	11.14	②	②	②	
3	Source Code is verifiable.	6.3.4c	○	○			Software Verification Results	11.14	②	②		
4	Source Code conforms to standards.	6.3.4d	○	○	○		Software Verification Results	11.14	②	②	②	
5	Source Code is traceable to low-level requirements.	6.3.4e	○	○	○		Software Verification Results	11.14	②	②	②	
6	Source Code is accurate and consistent.	6.3.4f	●	○	○		Software Verification Results	11.14	②	②	②	
7	Output of software integration process is complete and correct.	6.3.5	○	○	○		Software Verification Results	11.14	②	②	②	

LEGEND:		
	●	The objective should be satisfied with independence.
	○	The objective should be satisfied.
	Blank	Satisfaction of objective is at applicant's discretion.
	①	Data satisfies the objectives of Control Category 1 (CC1).
	②	Data satisfies the objectives of Control Category 2 (CC2).

Table A-6
Testing of Outputs of Integration Process

	Objective		Applicability by SW Level				Output		Control Category by SW level			
	Description	Ref.	A	B	C	D	Description	Ref.	A	B	C	D
1	Executable Object Code complies with high-level requirements.	6.4.2.1 6.4.3	○	○	○	○	Software Verification Cases and Procedures Software Verification Results	11.13 11.14	① ②	① ②	② ②	② ②
2	Executable Object Code is robust with high-level requirements.	6.4.2.2 6.4.3	○	○	○	○	Software Verification Cases and Procedures Software Verification Results	11.13 11.14	① ②	① ②	② ②	② ②
3	Executable Object Code complies with low-level requirements.	6.4.2.1 6.4.3	●	●	○		Software Verification Cases and Procedures Software Verification Results	11.13 11.14	① ②	① ②	② ②	
4	Executable Object Code is robust with low-level requirements.	6.4.2.2 6.4.3	●	○	○		Software Verification Cases and Procedures Software Verification Results	11.13 11.14	① ②	① ②	② ②	
5	Executable Object Code is compatible with target computer.	6.4.3a	○	○	○	○	Software Verification Cases and Procedures Software Verification Results	11.13 11.14	① ②	① ②	② ②	② ②

LEGEND:

●	The objective should be satisfied with independence.
○	The objective should be satisfied.
Blank	Satisfaction of objective is at applicant's discretion.
①	Data satisfies the objectives of Control Category 1 (CC1).
②	Data satisfies the objectives of Control Category 2 (CC2).

Table A-7
Verification of Verification Process Results

	Objective		Applicability by SW Level				Output		Control Category by SW level			
	Description	Ref.	A	B	C	D	Description	Ref.	A	B	C	D
1	Test procedures are correct.	6.3.6b	●	○	○		Software Verification Cases and Procedures	11.13	②	②	②	
2	Test results are correct and discrepancies explained.	6.3.6c	●	○	○		Software Verification Results	11.14	②	②	②	
3	Test coverage of high-level requirements is achieved.	6.4.4.1	●	○	○	○	Software Verification Results	11.14	②	②	②	②
4	Test coverage of low-level requirements is achieved.	6.4.4.1	●	○	○		Software Verification Results	11.14	②	②	②	
5	Test coverage of software structure (modified condition/decision) is achieved.	6.4.4.2	●				Software Verification Results	11.14	②			
6	Test coverage of software structure (decision coverage) is achieved.	6.4.4.2a 6.4.4.2b	●	●			Software Verification Results	11.14	②	②		
7	Test coverage of software structure (statement coverage) is achieved.	6.4.4.2a 6.4.4.2b	●	●	○		Software Verification Results	11.14	②	②	②	
8	Test coverage of software structure (data coupling and control coupling) is achieved.	6.4.4.2c	●	●	○		Software Verification Results	11.14	②	②	②	

LEGEND:		
	●	The objective should be satisfied with independence.
	○	The objective should be satisfied.
	Blank	Satisfaction of objective is at applicant's discretion.
	①	Data satisfies the objectives of Control Category 1 (CC1).
	②	Data satisfies the objectives of Control Category 2 (CC2).

Table A-8
Software Configuration Management Process

	Objective		Applicability by SW Level				Output		Control Category by SW level			
	Description	Ref.	A	B	C	D	Description	Ref.	A	B	C	D
1	Configuration items are identified.	7.2.1	○	○	○	○	SCM Records	11.18	②	②	②	②
2	Baselines and traceability are established.	7.2.2	○	○	○	○	Software Configuration Index SCM Records	11.16 11.18	① ②	① ②	① ②	① ②
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	○	○	○	○	Problem Reports SCM Records	11.17 11.18	② ②	② ②	② ②	② ②
4	Archive, retrieval, and release are established.	7.2.7	○	○	○	○	SCM Records	11.18	②	②	②	②
5	Software load control is established.	7.2.8	○	○	○	○	SCM Records	11.18	②	②	②	②
6	Software life cycle environment control is established.	7.2.9	○	○	○	○	Software Life Cycle Environment Configuration Index SCM Records	11.15 11.18	① ②	① ②	① ②	② ②

LEGEND:		
	●	The objective should be satisfied with independence.
	○	The objective should be satisfied.
	Blank	Satisfaction of objective is at applicant's discretion.
	①	Data satisfies the objectives of Control Category 1 (CC1).
	②	Data satisfies the objectives of Control Category 2 (CC2).

www.ibm.com/software/rational

www.ibm.com/software/rational