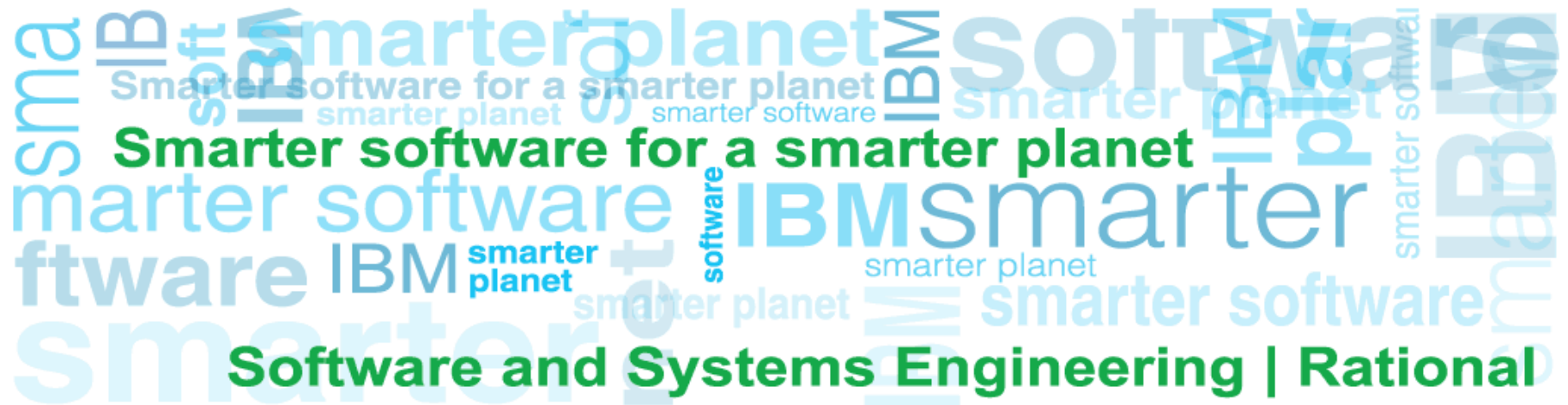


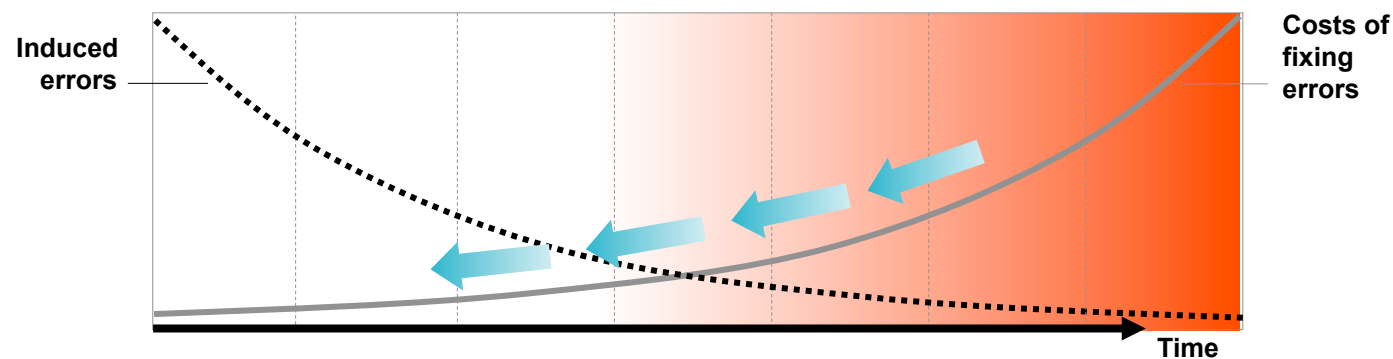
# IBM Rational Test RealTime v8.0

## What's New?



## IBM Rational Test RealTime

- **Reduce costs of fixing errors** by finding them earlier in the development cycle

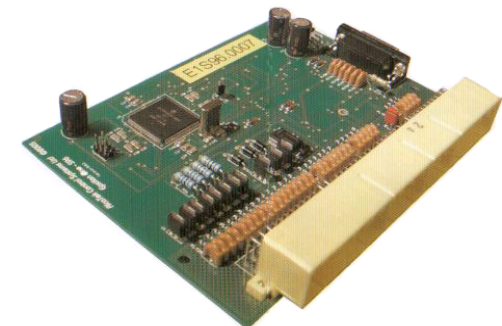


- Automate creation and execution of Unit Tests on **embedded target environment**
- Automatically **pinpoints hard to find errors**
  - such as memory corruption error
- **Highlight untested software**, assess code coverage
- Help teams **understand runtime software behavior** on target

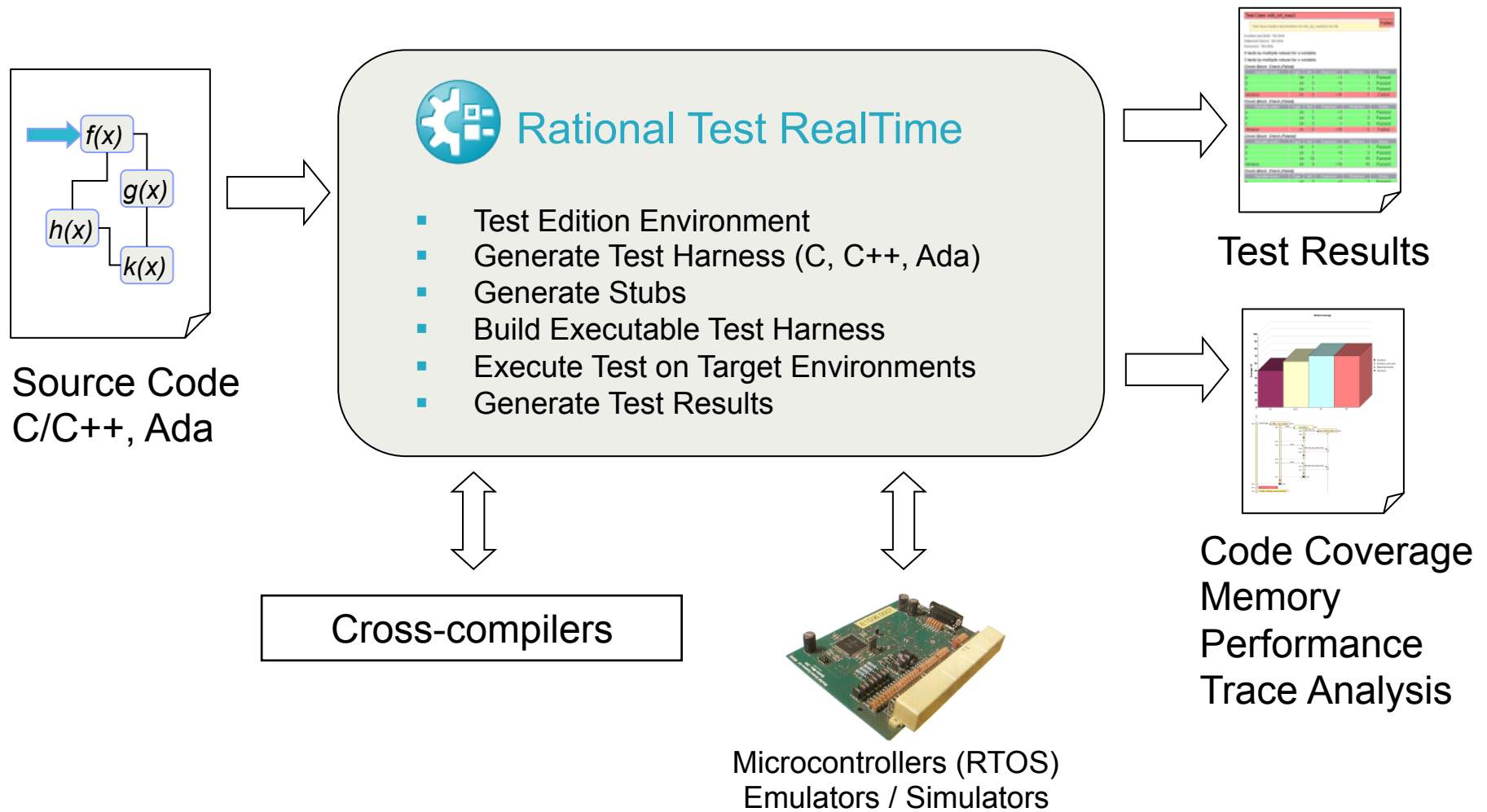


## IBM Rational Test RealTime - Overview

- A comprehensive **embedded software testing** solution
  - Software Unit & Integration Testing
  - Electronic Control Unit (ECU) / Hardware in the Loop (HIL) Validation
  - Runtime Analysis (Code coverage, Memory profiling, ...)
  - Static Code Analysis (MISRA-C)
- Targeting **embedded software developers** and systems QA professionals
  - Avionics, Rail, Automotive, HealthCare, Satellite, Telecom, ...
- Works on hosts (Windows, Linux, Unix) and **embedded target environments**
- Integrated with
  - Rational development & quality Tools
  - Microsoft Visual studio
- First product version shipped more than **20 years ago**
- Used by **over 200 customers** in all industry sectors, all Geos



# Testing Embedded Software with TestRT



## TestRT v8.0 – Nov-2011

- Script-less Unit Test Creation and Execution Environment for C language
  - **Assisted test creation** based on reverse engineering
  - **Script-less Visual Test Editor**
  - Brand new user interface build on top of Eclipse CDT
- TestRT 8.0 allows C/C++ embedded software developer to work in Eclipse
  - Enables to use any cross-compiler tool chain even if not integrated with Eclipse

The screenshot displays the TestRT v8.0 interface within an Eclipse IDE. On the left, a 'Call Graph' window shows a hierarchical view of system components, including 'main', 'SystemBuild\_Cy...', and various modules like 'ConnectionMa', 'DataLink\_initRe', and 'MobilityManag'. The 'main' component is highlighted.

In the center, the 'mlb\_int\_max\_tab' test case activity diagram is shown. It consists of three main activity nodes: 'Init. & Subs', 'Code', and 'Check'. The 'Check' node is currently selected.

On the right, the 'Details' panel for the 'Check' activity is visible. It shows the 'Checked Variables' section with a table of test data:

Name	Type	Initial Expression	Expected Expression	Obtained Value
size	int	0	== Same as Init	
tab	int[10]	4, 1..4=>5, 5..9=>Serie	[ 0=>0, 1..4=> Same as Init, 5....	
0	int	Multiple[4]	== 0	
1..4	int	5	== Same as Init	
5..9	int	Serie	== Same as Init	
retvalue	int	No Change	== 0	

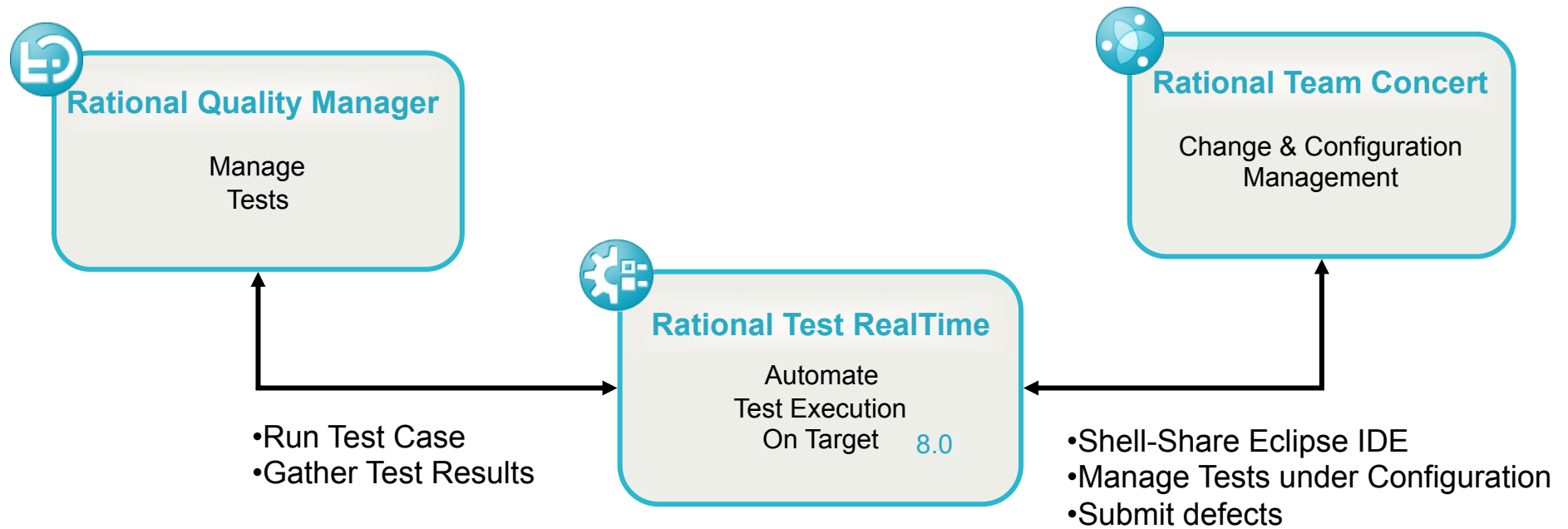
## Rational Test RealTime v8.0 – Capability Matrix

	C	C++	Ada
<b>Unit Testing</b>	✓	✓	✓
<b>Static Metrics</b>	✓	✓	✓
<b>Code Coverage</b>	✓	✓	✓
<b>Performance Profiling</b>	✓	✓	
<b>Runtime Tracing</b>	✓	✓	
<b>Memory Profiling</b>	✓	✓	
<b>Code Review</b>	✓	✓	
<b>System Testing</b>	✓		

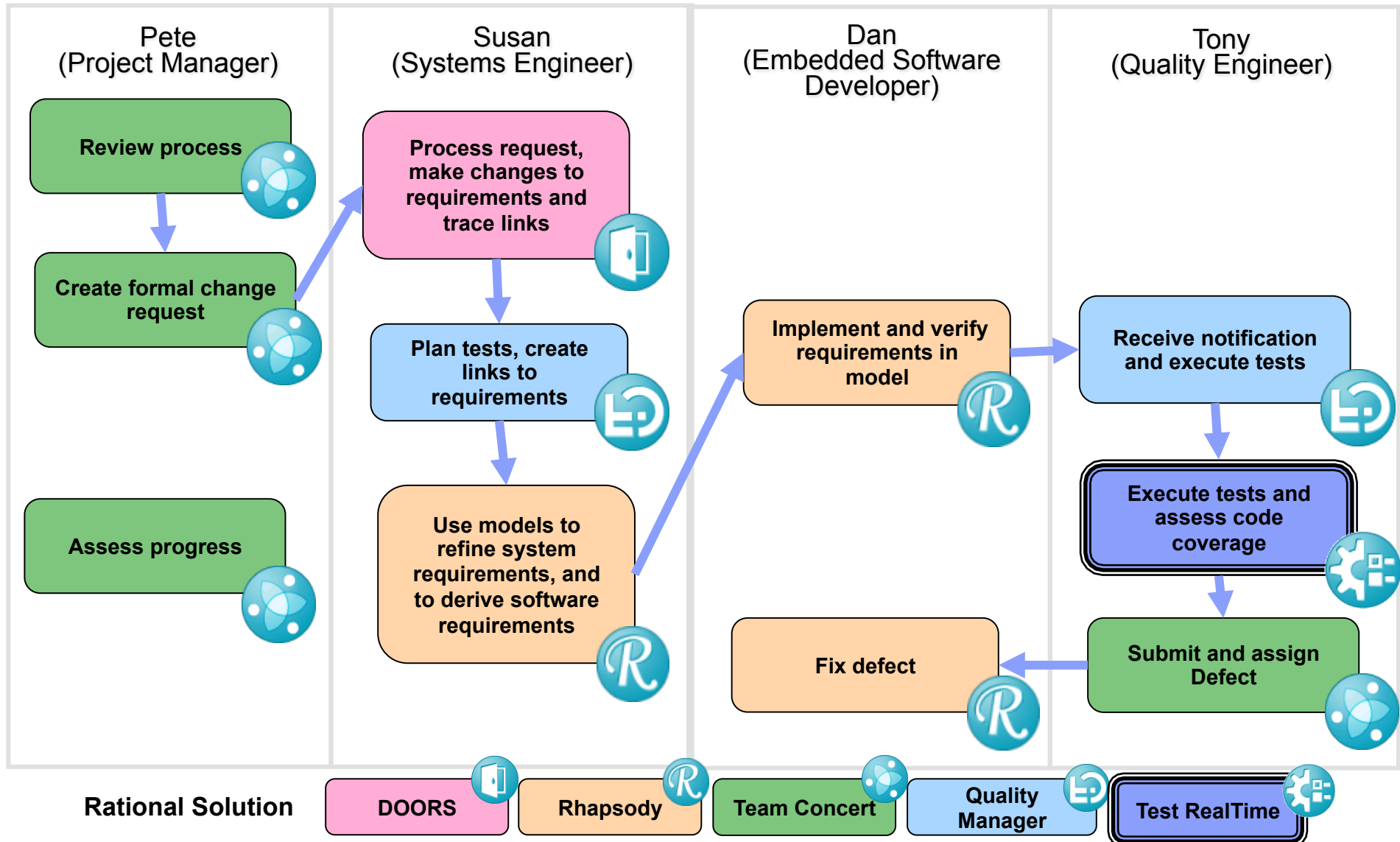
**New v8.0  
Eclipse IDE**

- Integrated with
  - Eclipse IDE 3.5, 3.6, 3.7, 4.2(comming)
  - Rational Quality Manager
  - Rational Team Concert
  - Microsoft Visual Studio
  - Rational Software Architect
  - Rational Rhapsody
- Tests run on Virtually any target environment
- Qualification Kit Available
  - DO178, FDA, EN50128, ISO 26262,...
- Systems Requirements - Eclipse IDE
  - Windows XP Professional Edition SP3 (x32)
  - Windows 7 Pro SP1 (x32, x64)
  - Red Hat Enterprise Linux 5.0 update 4 (x32, x64)
- Systems Requirements - Studio classic IDE
  - Windows 7 Pro SP1, XP, Server 2008 R2
  - Red Hat Enterprise Linux 6 & 5 (x32, x64)
  - SuSE Linux Enterprise 10.0 & 11.0 (x32, x64)
  - Ubuntu (x32, x64)
  - IBM AIX 5L 5.3 and 6.1
  - Sun SPARC Solaris 10

## Rational Test RealTime v8.0 Integrations

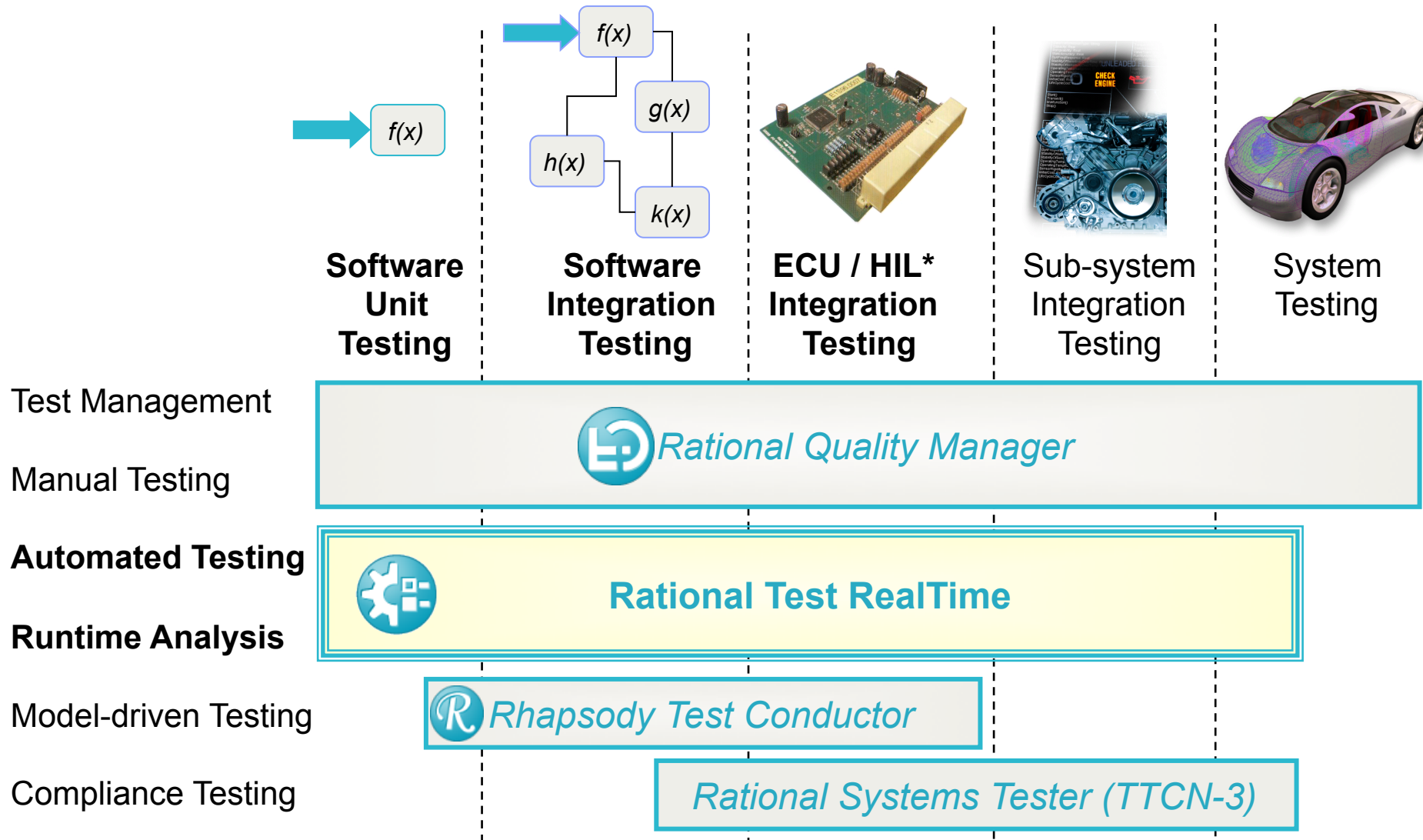


## Add-on to Software and Systems Engineering (SSE) Accelerator





# Rational Testing Solutions for Systems



(\*) ECU: Electronic Control Unit  
 HIL: Hardware-In-the-loop

## TestRT for eclipse IDE Offering

### Focus on development's performances

Give complete CLM solution accessible to C/C++ developers & Testers

- TestRT Eclipse allows developer/testers to works under Eclipse: no more need to use external tool chain to develop & build the C/C++ embedded applications
- RTC can becomes the Team developpment environment
- RQM integrations allows to manage tests against requirements

## TestRT for eclipse IDE Offering

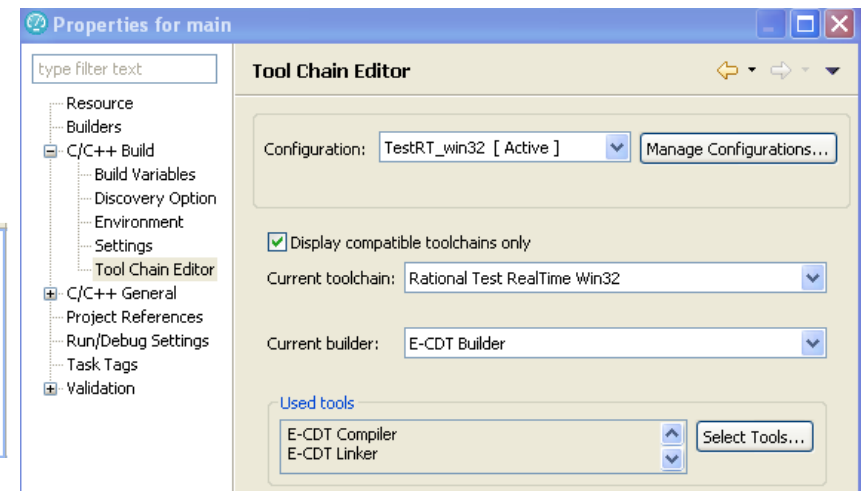
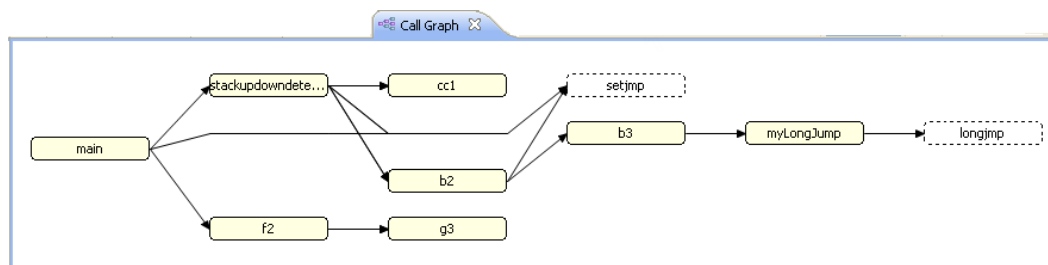
- Build with any compiler tool chain (even if not integrated into Eclipse)
- Build with Static and Dynamic analysis
- Make Testing using development environment
- Establish links to Requirements (RQM)
- Graphical Test Case Creation wizard
- Smart Test Case editor to enhance the tests
- Raise defects from report
- Easily share the work with colleagues (through RTC)
- Customisable reports (xml+xslt) that can be seen from internet browser
- Project Management (through RTC)
- Improve installation using IBM Installation Manager

# TestRT for eclipse IDE Offering

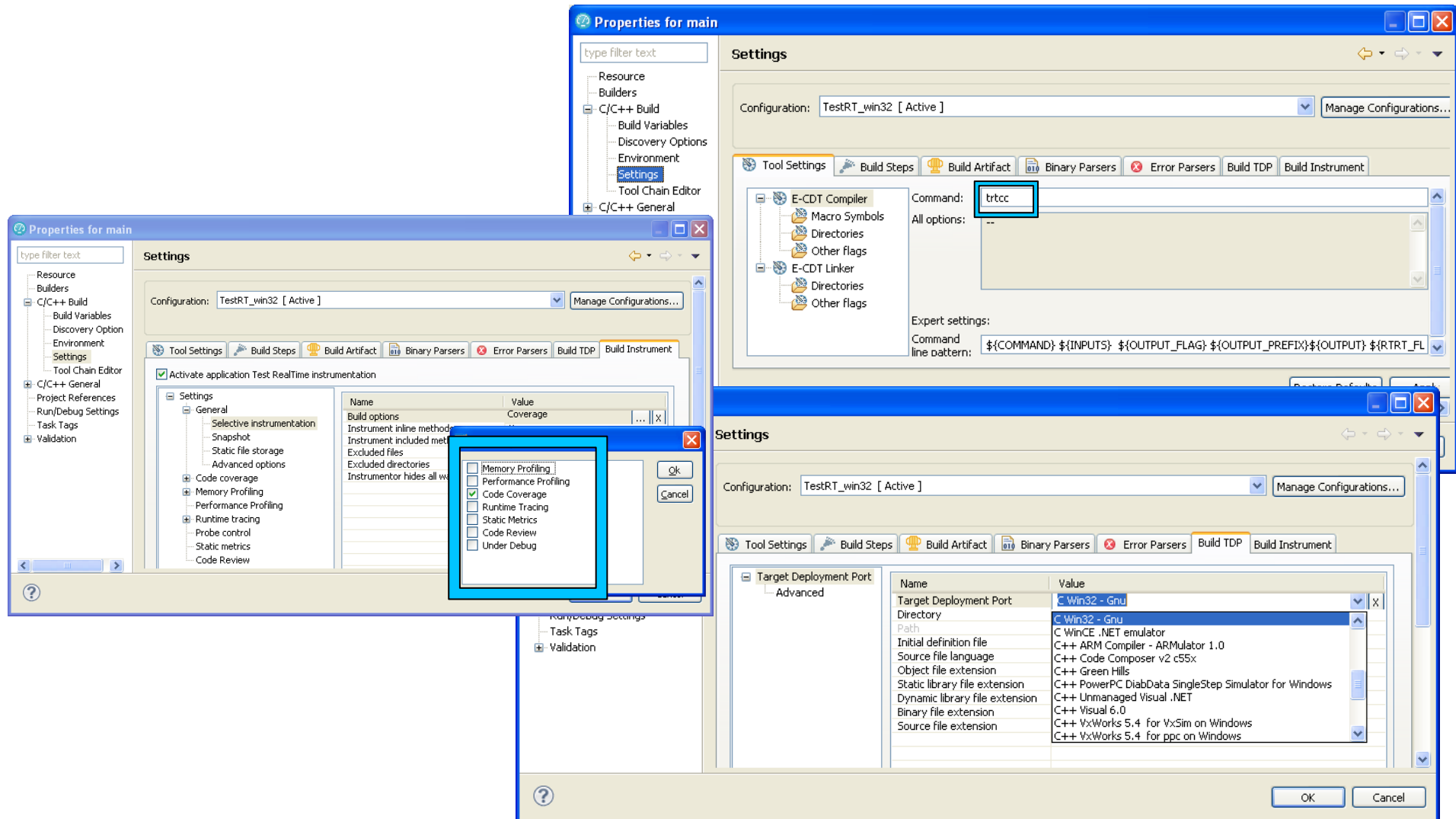
## Focus on adaptability and Flexibility

### TestRT eclipse Tool chain Embedded-CDT Feature

- TestRT tool chain (trtcc, trtld, trtex, trtclean) is the tool chain that use the TDP (Target Deployment Port) technology to compile, link, execute, ...
- The TDP technology allows to use native/cross compiler, debugger, and simulator that are not integrated into eclipse.
- The TDP selection from the E-CDT settings, allows to change the compiler and the way to execute on target/simulator.
- The E-CDT involves instrumentation & static metrics settings allowing static metrics computation during build and get application with runtime Analysis.
- Can launch debugger outside eclipse
- Offers an application CallGraph view
- View reports under eclipse
- Export Static & runtime Analysis reports in XML format for any internet browsers




# TestRT CDT Tool Chain for Embedded (E-CDT)

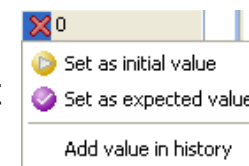
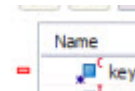
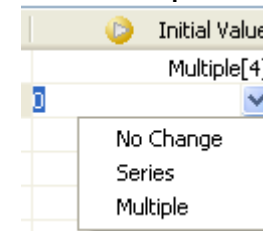


# TestRT for eclipse IDE Test Offering

## Focus on Accelerated Return to value

### Accelerated Use

- Introducing a new test paradigm based on a Visual interface:  
 source code → call graph → Test case definition → execution → Result report+coverage
- Provide a visual interface:  
 Tests can be created and edited without programming
- No language to learn:  
 All testing possibilities are offered via contextual menus.
- No Need to know variables & function prototypes:  
 The tool has parsed the application, So they are all listed in dialog boxes.
- The visual editor checks inputs permanently:  
 markers appears to warn the user ASAP on errors
- The visual editor generates code on save. 
- The report viewer is the test editor:  
 Faster way to change test values from execution report
- Test hierarchie enhance the re-usability:  
 Test cases and Test Stubs are re-usable.
- Test Stub allows multiple re-usable and enhanced behaviours:  
 each behaviour defines: parameter check/nocheck, and  
 return value or customized code if necessary
- Test Code generation based on source code parsing allows compiler changes with minimal  
 impact on test asset



# TestRT for eclipse IDE Test Offering - Details

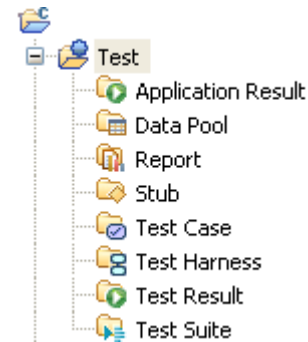
## Component Testing with Runtime Analysis & static metrics

### Test Assets


- Test Case
- Test Harness
- Test Suite
- Stub behavior

### Test RealTime Editors Viewers

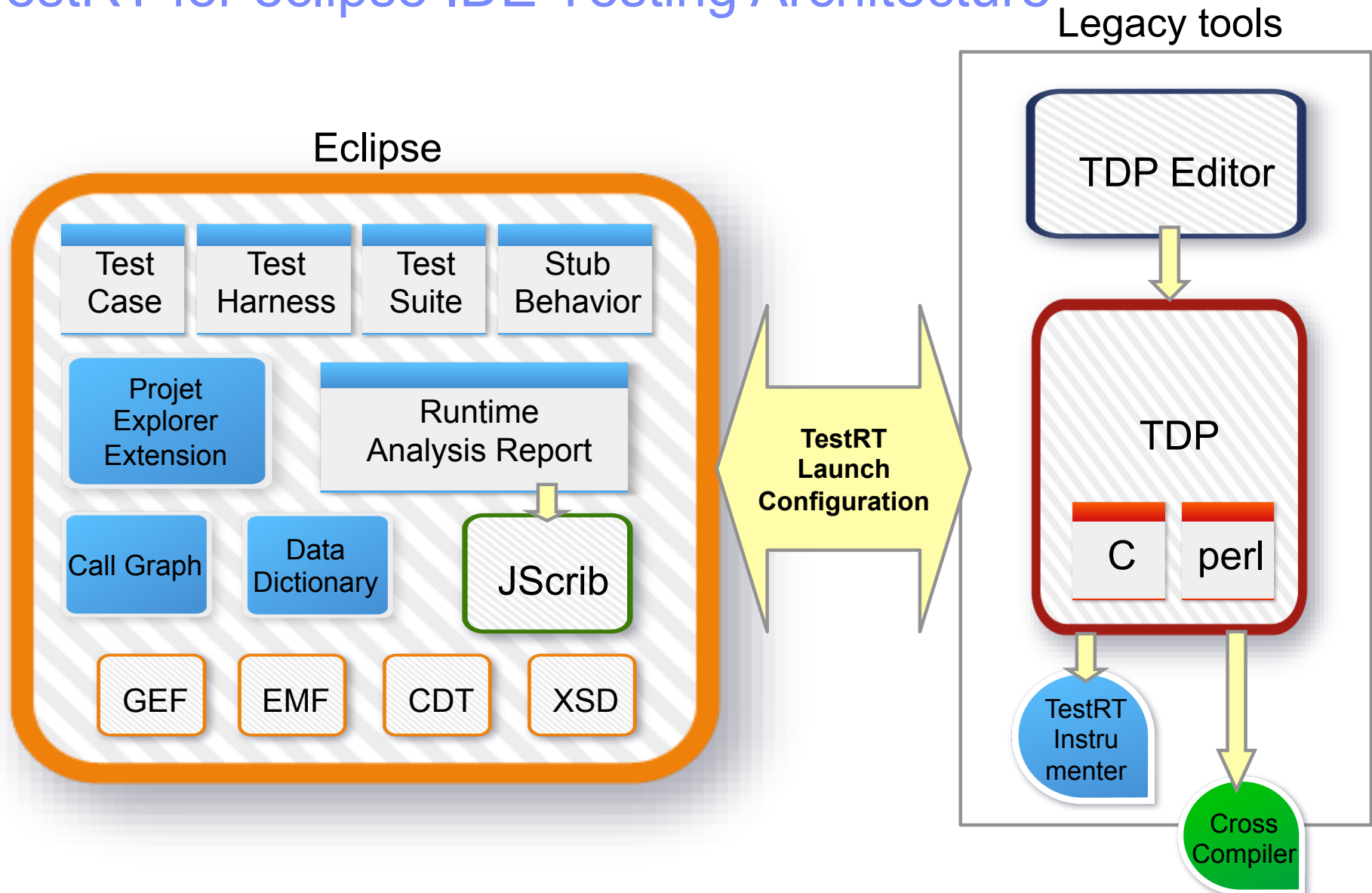
- Test Case Editor
- Test Harness Editor
- Test Suite Editor
- Stub Behavior Editor
- Data pool Editor
- Call Graph viewer
- Dictionary Viewer
- Project Explorer extension for TestRT



### Test Report

- Report generation in XML , include coverage, mode compare available
- Customization easy based on XSLT
- Runtime Analysis viewers (Coverage, memory & performance profiling, trace)
- Static Analysis viewers (metrics & code review)
- Chart & curve Viewer 

# TestRT for eclipse IDE Testing Architecture





# Visual Test TestCase Editor

The screenshot displays the Visual Test TestCase Editor interface. Key components include:

- Project Explorer:** Shows the project structure with files like `decode1`, `double2`, and `myHC08project`.
- Activity Diagram:** A flowchart showing the test case structure: `Init. & Stubs` (grey), `Code` (yellow), and `Check` (purple, marked as Failed).
- Details Panel:** Shows the selected `Check` activity. It includes a table of checked variables with their initial, expected, and obtained values.
- Results Table:** A table with columns: Name, Type, Initial Value, Initial Result, Expected Value, Expected Result, and Obtained Value.
- Integer Tool Tip:** A small window showing iteration values for the `integer` variable.
- Console:** Displays build output from the HC08 compiler, including error messages.
- Annotations:** Several callout boxes highlight specific features: 'run result selection', 'iteration result selection', 'coverage result', 'compilation error markers', 'Tool tip iteration values', and 'Cross Build console'.

Name	Type	Initial Value	Initial Result	Expected Value	Expected Result	Obtained Value
key	signed char*	encryptionKey1	4280456	No Check		4280456
retvalue	int	1	1	5	5	0
theVal	val_t	{ Multiple[3], 0, "AA..."	{ 567, 0, "SDFGSD" }	{ Same as Init, Sync...	{ 3, "01234" }	{ 567, 3, "732" }
integer	long	Multiple[3]	567	Same as Init		567
length	int	0	0	Synchronized with th...	3	3
hexa	char[20]	"AAAA"	"SDFGSD"	Synchronized with th...	"01234"	"732"

Index/Name	Initial Value
1	567
2	1024
3	1234567890

```

Test RealTime Visual Tester
Object file: C:\Program Files\Rational\TestRealTime\bin\win32\wave\perl_bug_0; format: COFF; architecture: i386
Code Size: 3193
Global objects: 58, Data Size (RAM): 290, Const Data Size (ROM): 16
HC08 Compiler: *** 0 error(s), 0 warning(s), 0 information message(s) ***
HC08 Compiler: *** Processing ok ***
...
Result buffer size reduced to 256 chars
    
```

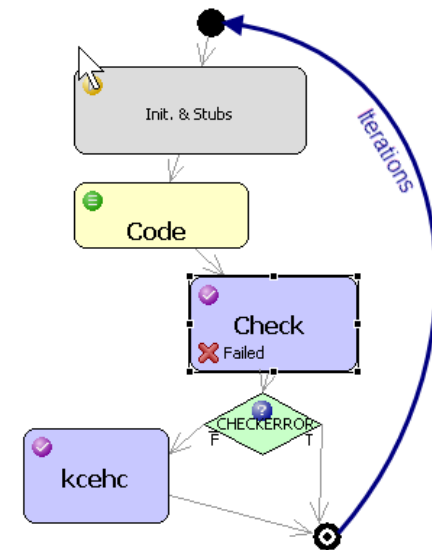
## TestRT - Visual Test Environment - Details

### Simple to complex test case

- Allows to execute check block conditionally
- Allows to make unit and integration testing
- Used conditions can be variable, block check status, or test status

### Test case enables multiple iterations

- Init with Multiple allows to use a set of
  - defined values
  - random values (between min and Max)
  - data Pool values (from spread sheet columns)
  - logical serie (from x to y step t)
- One execution iteration per
  - value in multiple,
  - combination of them,
  - Data Pool lines
- Iteration number can be used into init/expected value expression



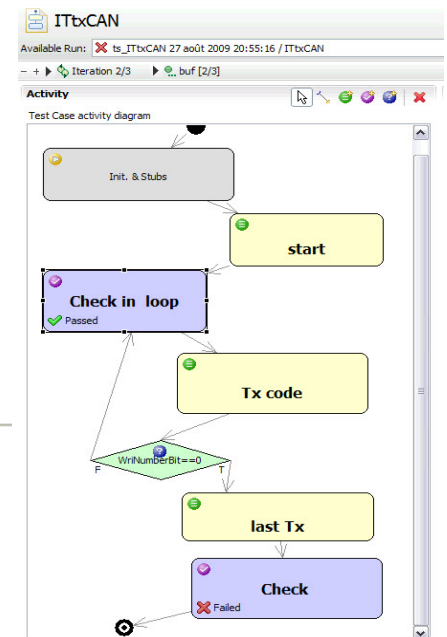
### Variables

- Variable types are known, so structures are expended
- Memory can be allocated and affected to pointers easily

Test Application Variable  
 Create Tested Variable  
 Allocate range ...

### Expected values

- Can be a range including/excluding each bound
- Can use all comparison types (==, >=, <=, !=, >, <)



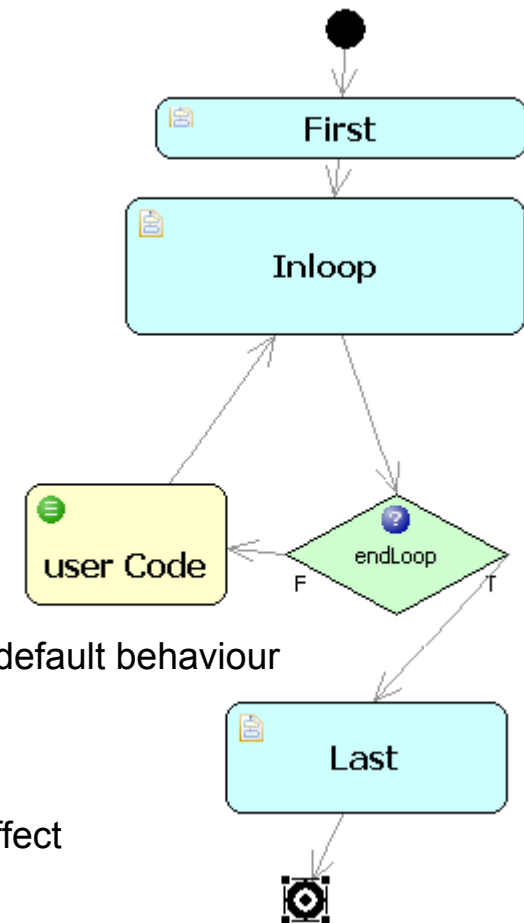
## TestRT - Visual Test Environment - Details

### Simple to complex test Harness

- Allows to execute Test Case block conditionally
- Allows to make unit and integration testing
- Allows to insert user code between Test Cases
- Used conditions can be variable, Test Case status

### Enhanced Stub

- The Test Harness defines the functions to be Stubbed
  - If there is no selected behaviour, then NoCall expected is the default behaviour
- Stub may have multiple stub behaviours
- The Test Case defines the stub behaviour for each stubbed function
  - Each behaviour can be re-used across multiple Test Cases
  - If the function is not stubbed the behaviour selection has no effect
- Each Stub behaviour may have multiple check definitions
- Each check definition determines for a range of call
  - the parameter to check and its comparison types (==, >=, <=, !=, >, <, or Range)
  - the return value OR the user code to execute
- The call index (ix\_<funcName>) can be used anywhere.



The screenshot displays the Rational Team Concert interface for a test harness. The main window shows a test harness activity diagram for 'array' with the following components:

- array**: Failed (marked with a red X)
- array1**: Passed (marked with a green checkmark)
- array2**: Failed (marked with a red X)
- \_TESTERROR**: A decision diamond with 'F' and 'T' paths.
- Code 1**: A yellow box representing code.
- array3**: Failed (marked with a red X)

The 'array' activity is expanded in the Project Explorer on the left, showing a tree structure of test harness components: array [2], iteration [1], array1 [2], array1, iteration [1], array2 [2], array2, iteration [1], array3 [2], and array3.

On the right, a sequence diagram shows the execution flow between lifelines 'array7', 'thsimple1', and 'array3'. The diagram includes several 'test' messages and a 'Test Case' message. A red box at the bottom of the sequence diagram indicates 'Errors detected'.

The bottom console window shows the following execution log:

```

<terminated> array [Rational Test RealTime Launch Test Harness] November 13, 2012 1:18 PM
c:\exe /ZI /EHsc /RTC1 /nologo -c "C:\PROGRA~1\IBM\TESTRE~1\targets\cvisual9\lib\TP.c" -Fo"C:\temp\workspace\Valid_runtime\cvisual\TP.obj"
TP.c
Linking C:\temp\workspace\Valid_runtime\cvisual\array.exe...
----
link.exe /NOLOGO /DEBUG /MACHINE:IX86 "C:\temp\workspace\Valid_runtime\cvisual\tst\array.test_harness.obj" "C:\temp\workspace\Valid_runtime\c
Executing C:\temp\workspace\Valid_runtime\cvisual\array.exe ...
C:\temp\workspace\Valid_runtime\cvisual\array.exe
    
```

# TestRT – RQM Integration

Use the Command Line Interface of Visual Test

- Extension of the existing RQM integration
- Able to run Test Harness and Test Suite as well as studio projects
- Multi-project executions

The screenshot shows the Rational Test RealTime Adapter Console interface. A dialog box titled 'Import Test Script: Use local test resources' is open, showing 'Step 2: Select Project path and Test Scripts'. The 'Project Path' is set to 'C:\Program Files\Rational\TestRealTime\examples'. Below the dialog, a table lists test scripts with their locations and types. The 'ChainedList/ChainedList.rtp' entry is selected and highlighted with a red box.

Name	Location	Type
<input type="checkbox"/> BaseStation_C/BaseStation_C.rtp	C:\Program Files\Rational\TestRealTime\examples	Rational Test RealTime
<input type="checkbox"/> BaseStation_Java/BaseStation_Java.rtp	C:\Program Files\Rational\TestRealTime\examples	Rational Test RealTime
<input type="checkbox"/> BroadcastServer/BroadcastServer.rtp	C:\Program Files\Rational\TestRealTime\examples	Rational Test RealTime
<input checked="" type="checkbox"/> ChainedList/ChainedList.rtp	C:\Program Files\Rational\TestRealTime\examples	Rational Test RealTime
<input type="checkbox"/> CheckFrequency/ABWLcheckFrequency.rtp	C:\Program Files\Rational\TestRealTime\examples	Rational Test RealTime

Below the dialog, the main console shows a table of adapters:

ID	Adapter Name	Machine Name	Adapter type	Status	Health	Host Name	IP	Polling Interval
<input type="checkbox"/> 294	TestRT on F2T000D60ACE4E8	F2T000D60ACE4E8	Rational Test RealTime	Available	●	F2T000D60ACE4E8	9.143.105.69	5 sec

A yellow message box at the bottom right states: 'Adapter deleted successfully.'