IBM Rational Software Conference 2009

Prasad Bhat
Product Consultant, Rational Services
prasad.bhat@in.ibm.com

Rational. software

Go to IBM

Industry Axiom?

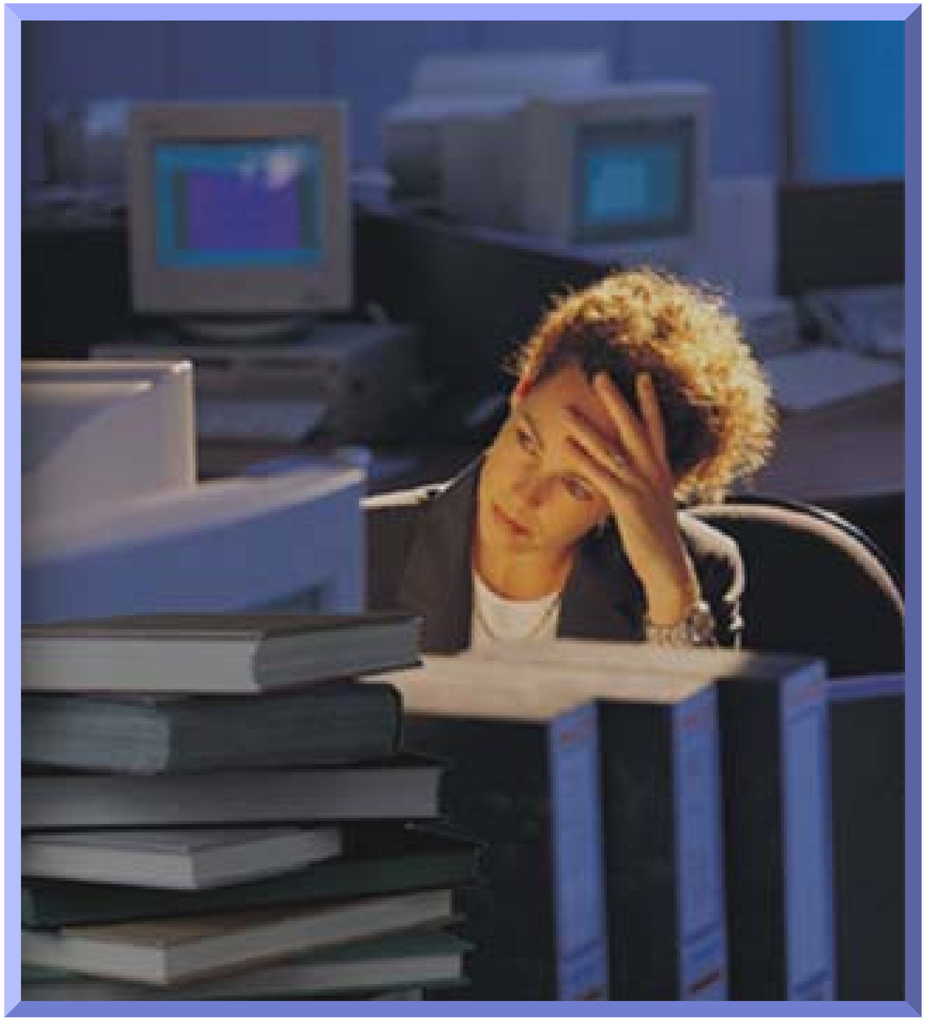
Quality
Time to Market

Embedded Software Development Challenges

- **Application Complexity**
  - Strong timing constraints
  - Low memory footprints
  - Concurrent/Distributed/ Networked
- **Environment Complexity**
  - Multiple RTOS vendors
  - Multiple chip vendors
  - Multiple IDEs
  - Limited host-target connectivity
  - Low built-in debugging capabilities
- **Process Complexity**
  - Requirements shift
  - Design translation errors
  - Lack of understanding
  - Difficult to maintain
  - Poor performance
  - Late discovery
  - Incomplete integration

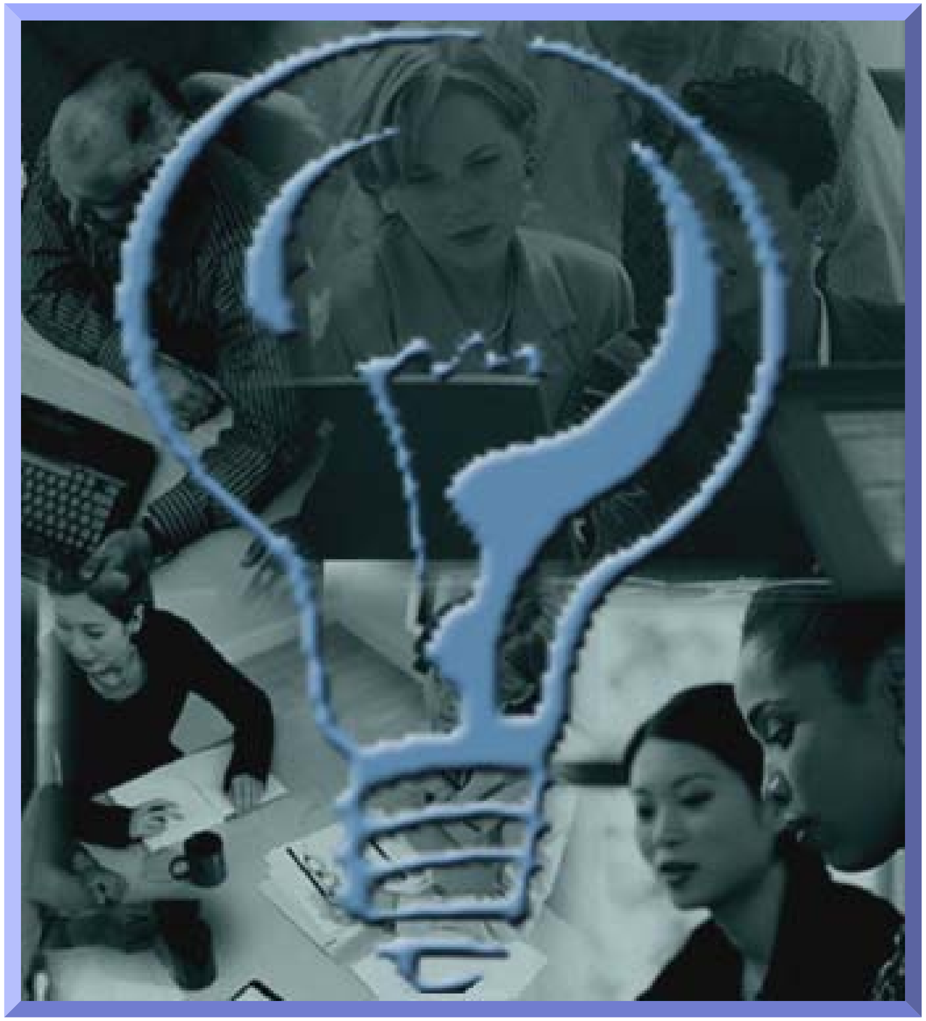
Experience Tells Us...

- **Debugging is not testing!**
  - Systematic testing tools and methods are necessary to ensure proactive and early problem discovery
- **A debugger details symptoms, but debugging requires a diagnosis**
  - Runtime analysis is a key practice for simplifying the diagnosis of issues impacting reliability, scalability and durability



Would This Accelerate Quality?

- **Automated component testing at all levels of complexity**
  - From the simplest function to distributed systems
- **Extensive runtime analysis capabilities**
  - Memory and performance profiling, code coverage, runtime tracing
- **Static metrics calculation**
  - Assist with test prioritization
- **Dynamic links between code, test results, and visual model**
- **Full regression testing abilities**



IBM Rational Test RealTime

- Automated component testing at all levels of complexity – from the simplest function to distributed systems
- Memory and performance profiling, code coverage measurement, runtime tracing and thread profiling
- Static metrics calculation to assist with test prioritization
- Dynamic links between code, test, and visual model
- Full regression testing capabilities

All in one tool
For any IDE - Hosted on any target!

Fix Your Code Before It Breaks

- **Rational Solutions and the Embedded Systems Market**
- **IBM Rational Test RealTime**
  - Serves and Empowers Any Test and Debug Process
  - Delivers a Unified Component Testing and Runtime Analysis Solution
  - Delivers Total Environment Adaptability
- **What Do You Need?**

Rational Platform: Supporting Embedded Industry

Medical

- ✓ Scanners
- ✓ Surgical Lasers
- ✓ Pace-makers

Telecom Devices

- ✓ Pagers
- ✓ Phones
- ✓ Switches
- ✓ Routers

Rational Solution: ***Addressing your embedded design, test, and management needs***

Aerospace / Military

- ✓ Aircraft
- ✓ Spacecraft
- ✓ Missiles

Automotive

- ✓ Body Electronics
- ✓ Power train
- ✓ Chassis

Fix Your Code Before It Breaks

- **Rational Solutions and the Embedded Systems Market**
- **IBM Rational Test RealTime**
  - Serves and Empowers Any Test and Debug Process
  - Delivers a Unified Component Testing and Runtime Analysis Solution
  - Delivers Total Environment Adaptability
- **What Do You Need?**

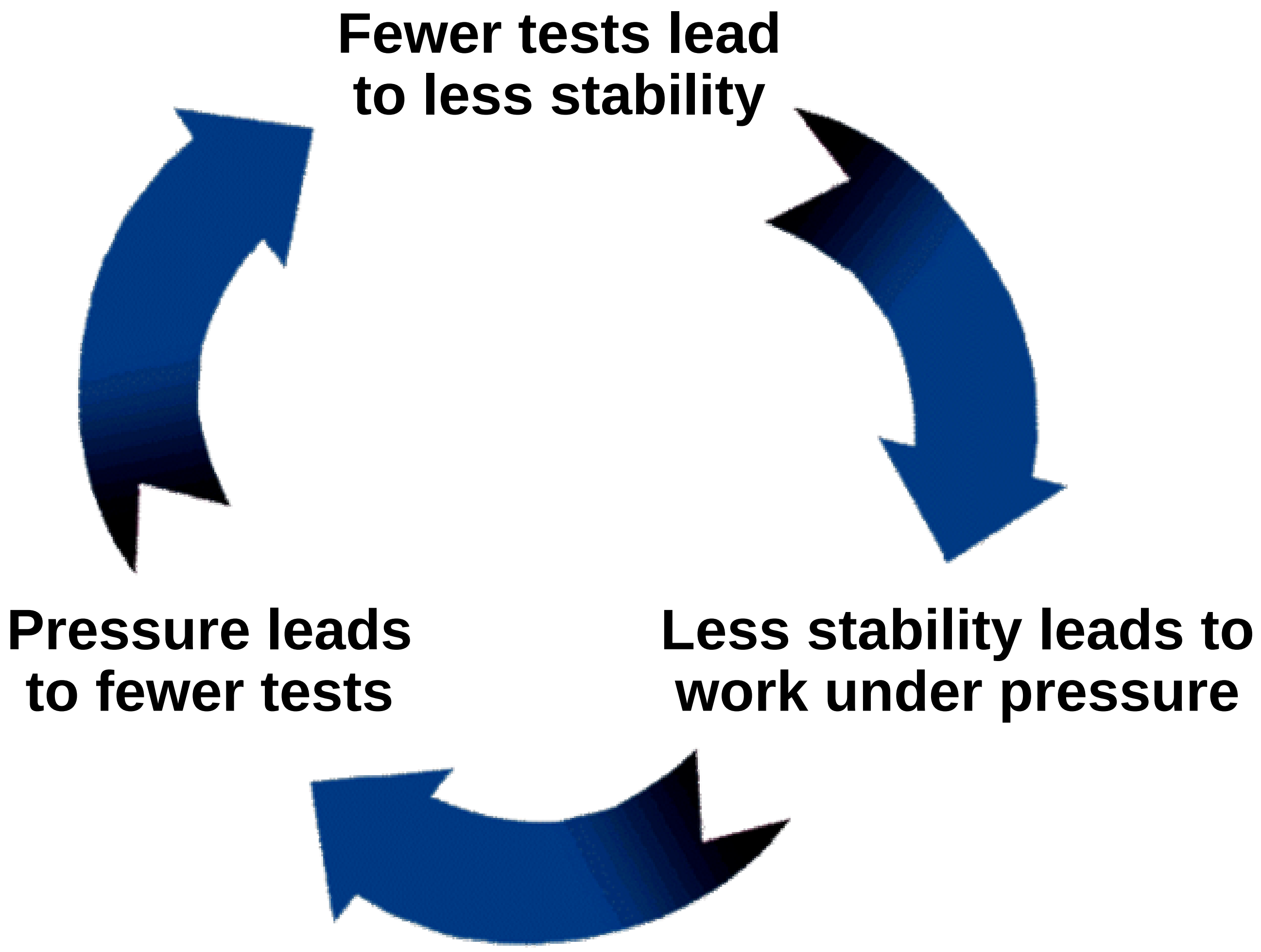
What We Have Seen

Every developer knows testing ensures quality code…but few do more than debug

"I'll test in the next project – I've got too much code to write and fix"

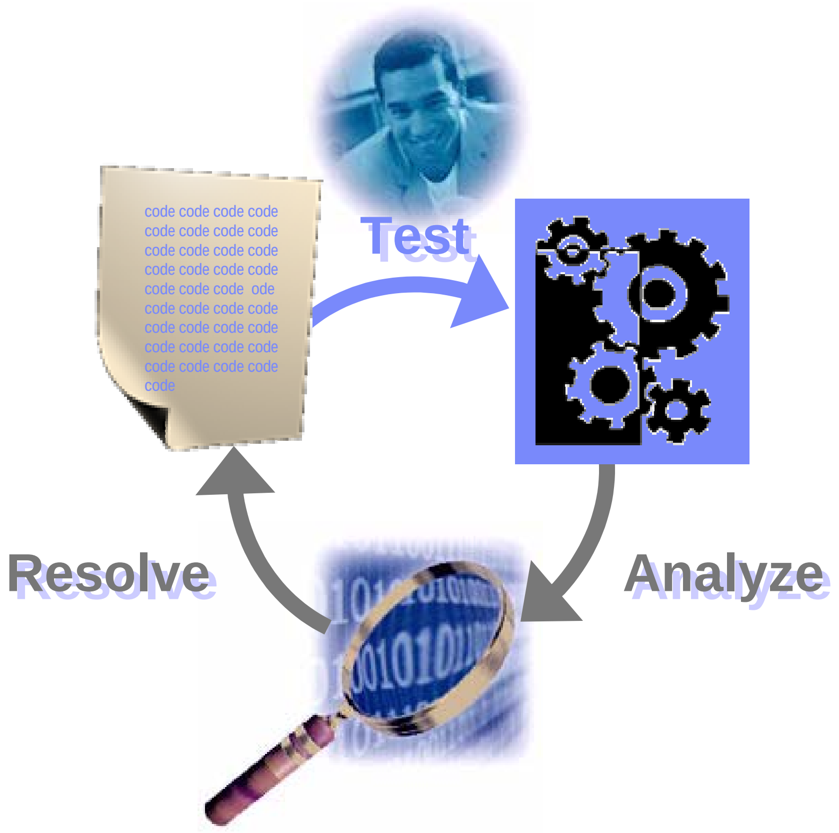
But is there room for risk in the embedded world?

A Day in the Life


Fewer tests lead to less stability
Less stability leads to work under pressure
Pressure leads to fewer tests

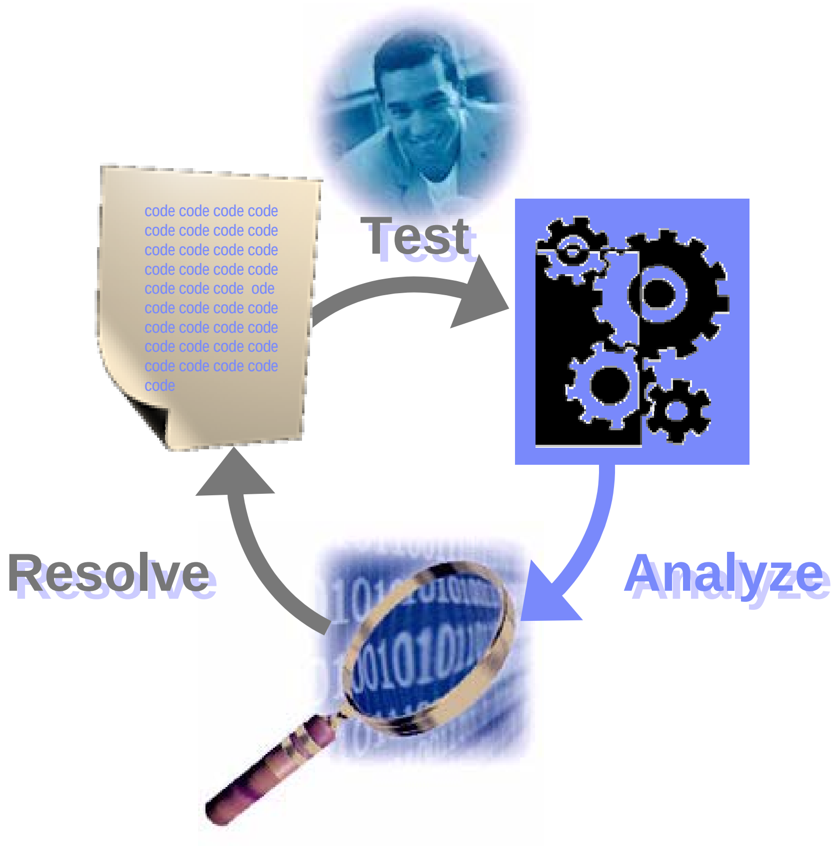
Serves and Empowers any Test and Debug Process

- **Test as you code**
  - Automatic component test template and data generation
    - Black- and white-box testing
    - All levels of complexity: From single functions to distributed systems
    - Static metric calculation for:
      - tests prioritization
      - complexity estimation
    - Full regression testing



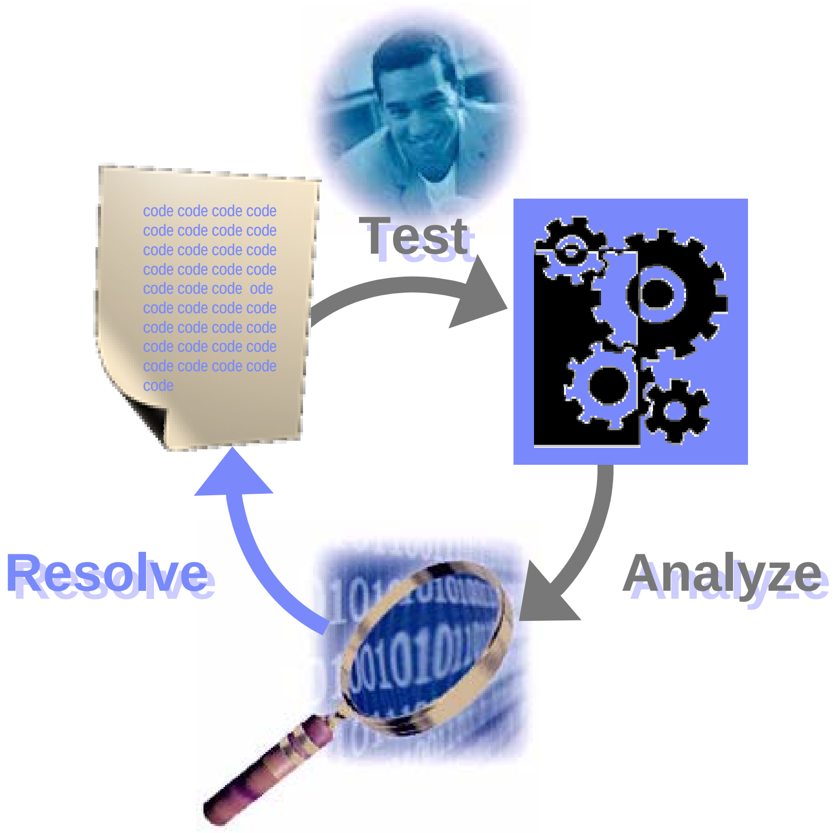
Serves and Empowers any Test and Debug Process

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


code code code code code code code code code code code code code code code code code code code ode code code code code code code code code code code code code code code code code code

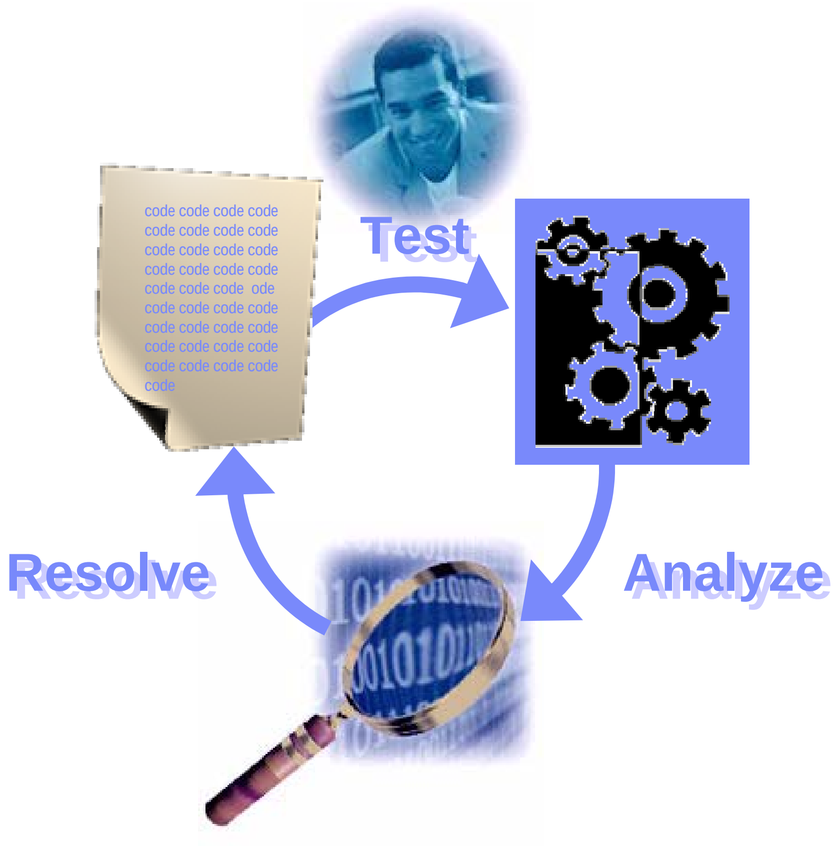
Serves and Empowers any Test and Debug Process

- **Test as you code**
- **Analyze while you test**
- **Resolve what you have uncovered**
  - Test execution integrated with your debugger
  - Consolidated, detailed to-the-point test reporting
  - Test data hyperlinked to runtime analysis results and code



Serves and Empowers any Test and Debug Process

- Test as you code
- Analyze while you test
- Resolve what you have uncovered



Now fix the defects, enhance your tests
And move on!

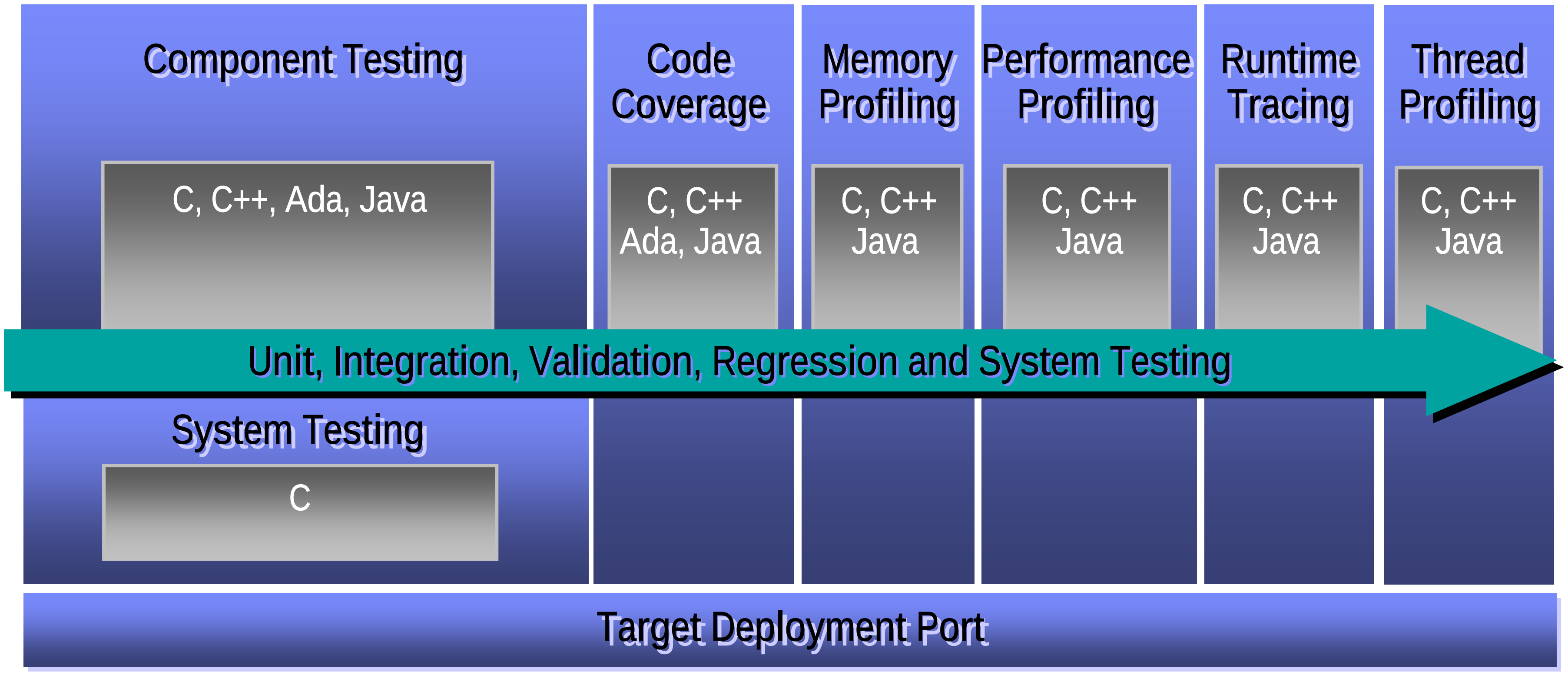
Serves and Empowers any Test and Debug Process

- ✓ **Eases transition from manual testing to automation**
  - Source-code aware and thus easily adoptable
  - Powerful test languages drive robust data-driven tests
  - Creates shared debug and test vocabulary for your team
- ✓ **Flexible to minimally impact preexisting processes**
  - Process agnostic, so you can stay in your comfort zone
  - Shortens time-to-problem-resolution, maximizes time-to-code
  - Mix and match functionality to accommodate your needs
- ✓ **Built explicitly for the rigors of embedded development**
  - Manages target environment so you can focus on good test creation
  - Handles all test enablement activities so you test early and often
  - Guarantees test reuse when the environment changes to ensure your testing effort will pay off in regression testing dividends

Fix Your Code Before It Breaks

- **Rational Solutions and the Embedded Systems Market**
- **IBM Rational Test RealTime**
  - Serves and Empowers Any Test and Debug Process
  - Delivers a Unified Component Testing and Runtime Analysis Solution
  - Delivers Total Environment Adaptability
- **What Do You Need?**

IBM Rational Test RealTime - Overview



- **Built to achieve standards compliance**
  - DO-178B
  - MISRA
  - Defense Standard 00-55

A Unified Component Testing and Runtime Analysis Solution

- **Combines entire features set into a single, unified tool**
  - Optimizes test effort through addition of runtime analysis functionality
  - Accelerates problem resolution through shared team assets
  - Simplifies refactoring verification
- **Enables quality verification for all test granularity levels and certification standards**
  - Focuses your efforts on a single toolset
  - Produces information required for code certification
  - Provides a means for achieving pervasive quality
- **Accommodates the larger IBM Rational lifecycle solution**
  - Alleviates friction generated by use of non-integrated tools
  - Improves team and asset stability through traceability
  - Single vendor relationship ensures future integration improvement

Fix Your Code Before It Breaks

- **Rational Solutions and the Embedded Systems Market**
- **IBM Rational Test RealTime**
  - Serves and Empowers Any Test and Debug Process
  - Delivers a Unified Component Testing and Runtime Analysis Solution
  - Delivers Total Environment Adaptability
- **What Do You Need?**

Delivers Total Environment Adaptability

Target Deployment Technology

A low-overhead, versatile target deployment technology

- Compiler-independent high level scripting API
- Debugger-independent test harness deployment
- Target-independent results upload & report creation

Full Target Independence!

Delivers Total Environment Adaptability

4-Bit to 64-Bit Cross-Development Environments Used By Our Customers		
WindRiver	*Montavista*	*Tasking*
GreenHills	*TI*	*CAD-UL*
ARM	*NEC*	*Cosmic*
Enea	*Hitachi*	*Hiware*
Windows CE	*Apex*	*Hitex*
LynuxWorks	*Sun*	*Symbian*
Lauterbach	*Microtec*	*……*

Languages

- C
- C++
- Ada
- J2ME/J2SE

Platforms

- *Windows*
- *Solaris*
- *Linux*
- *HP-UX*
- *AIX*

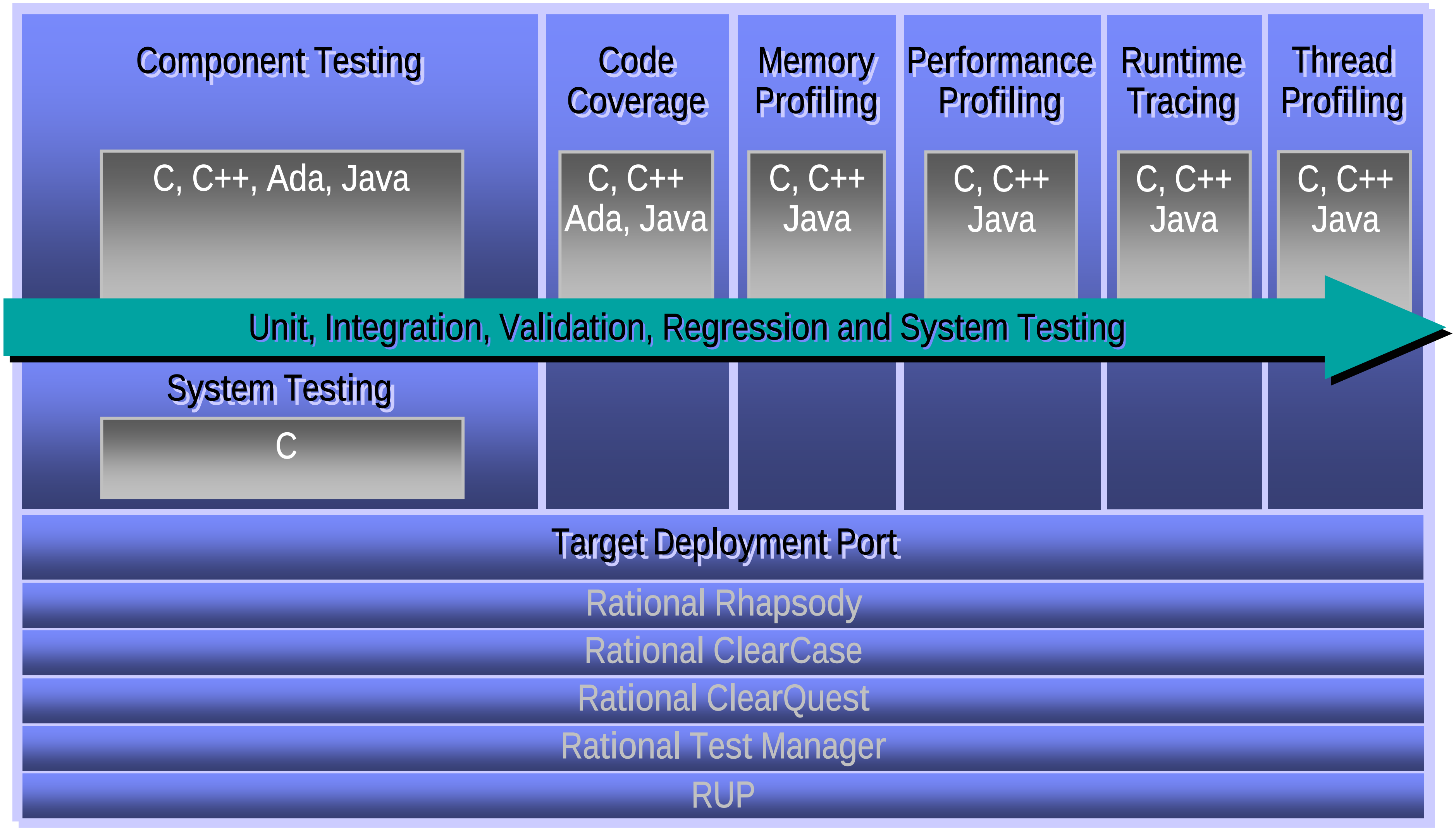
Delivers Total Environment Adaptability

- ✓ **Customizable to support a complete range of embedded targets**
  - Assures tool adoption
  - Reduces your ramp-up time when target configuration changes
  - Guarantees the reuse of test assets despite target constraints
- ✓ **Host, Build and Target Environment Agnostic**
  - Ensures portability of test and runtime analysis processes
  - Simplifies multiple team deployment
  - Optimizes ROI in comparison to home-grown test solutions
- ✓ **Size and Speed Optimized to Limit Target Impact**
  - Enables full control to minimize instrumentation overhead
  - Frees your tests from having to compensate for target restraints
  - Avoids overtasking your system

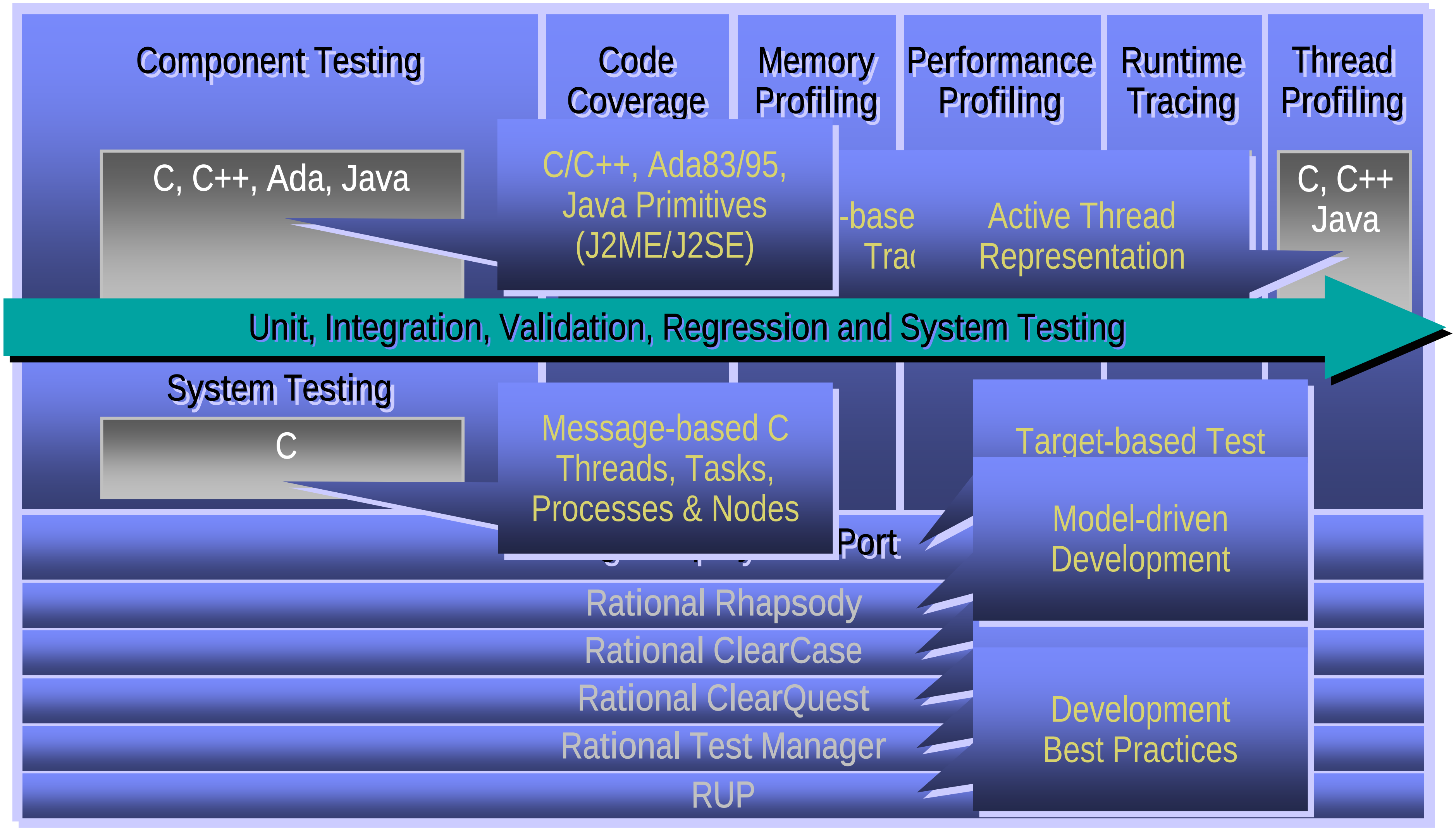
Fix Your Code Before It Breaks

- **Rational Solutions and the Embedded Systems Market**
- **IBM Rational Test RealTime**
  - Serves and Empowers Any Test and Debug Process
  - Delivers a Unified Component Testing and Runtime Analysis Solution
  - Delivers Total Environment Adaptability
- **What Do You Need?**

IBM Rational Test RealTime

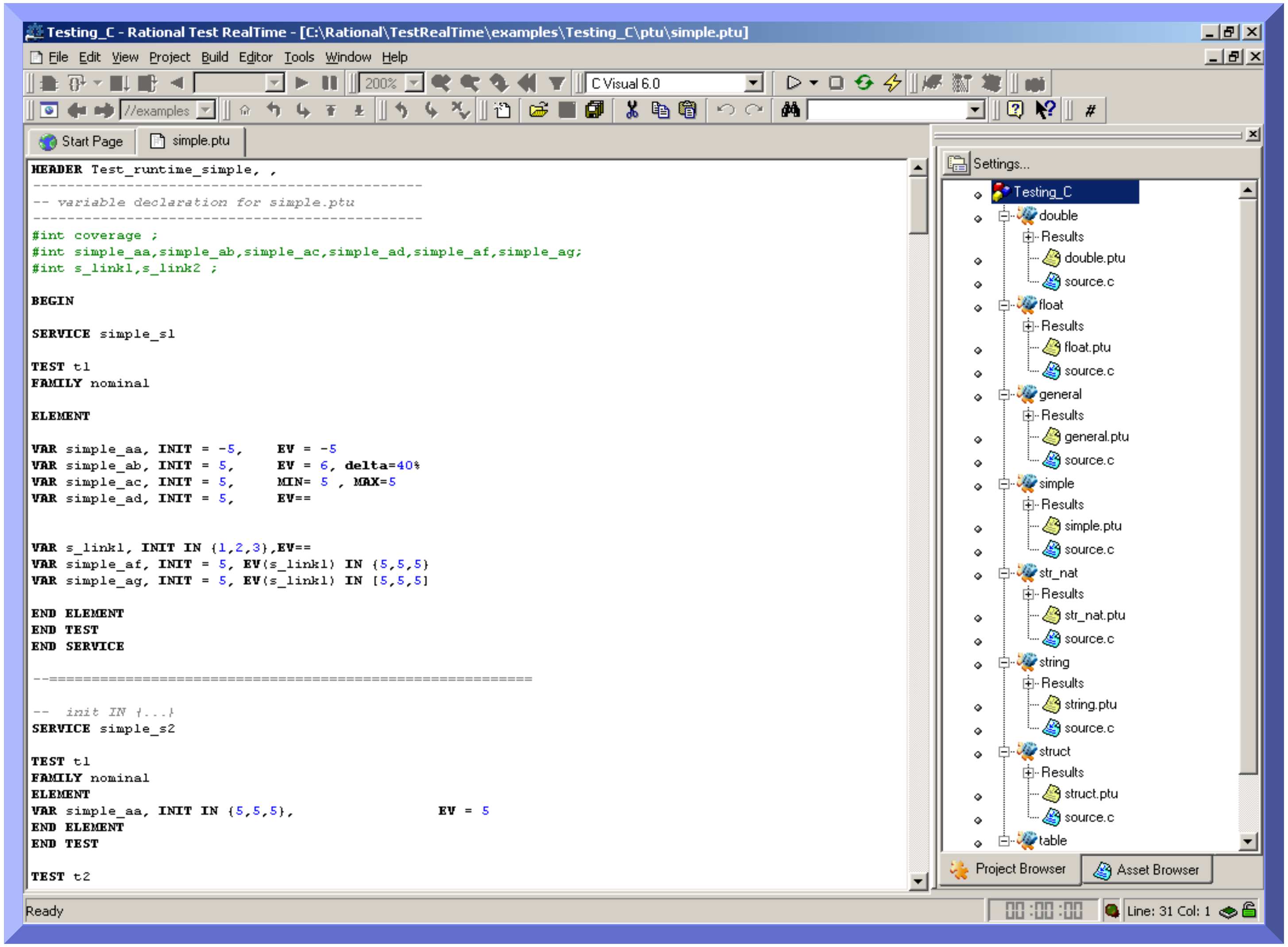


IBM Rational Test RealTime



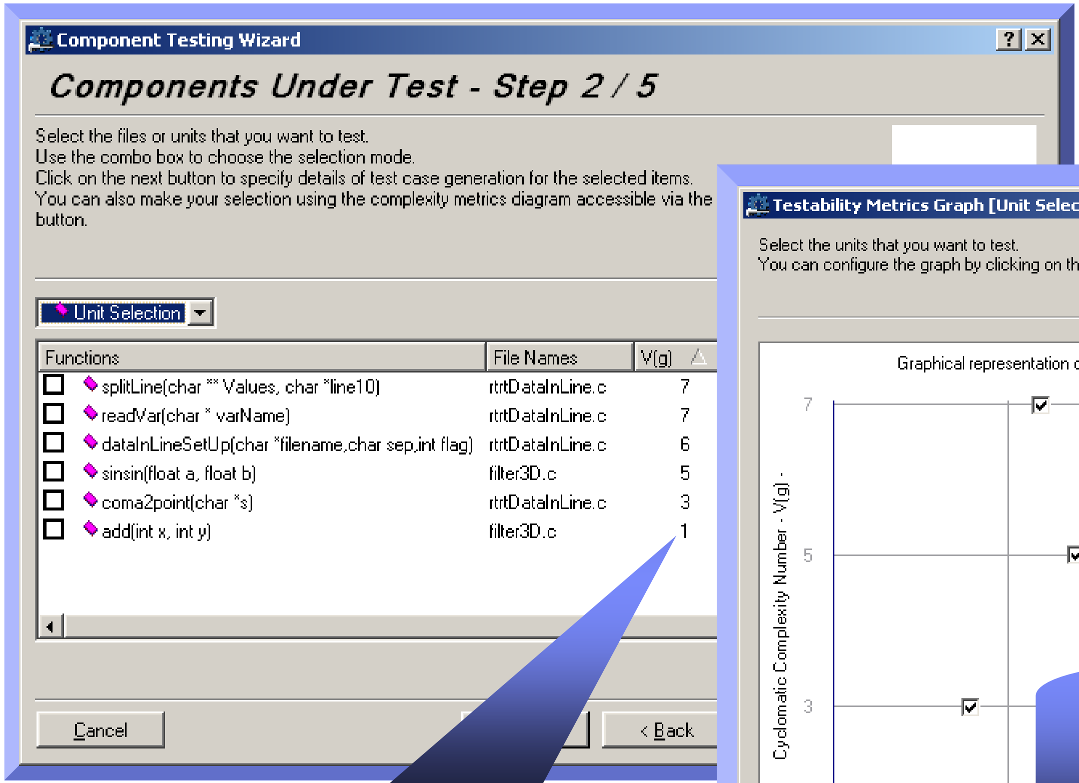
DEMO

IBM Rational Test RealTime: *Test Script*



```
HEADER Test_runtime_simple, ,
----------------------------------------------
-- variable declaration for simple.ptu
----------------------------------------------
#int coverage ;
#int simple_aa,simple_ab,simple_ac,simple_ad,simple_af,simple_ag;
#int s_link1,s_link2 ;

BEGIN

SERVICE simple_s1

TEST t1
FAMILY nominal

ELEMENT

VAR simple_aa, INIT = -5,     EV = -5
VAR simple_ab, INIT = 5,      EV = 6, delta=40%
VAR simple_ac, INIT = 5,      MIN= 5 , MAX=5
VAR simple_ad, INIT = 5,      EV==


VAR s_link1, INIT IN {1,2,3},EV==
VAR simple_af, INIT = 5, EV(s_link1) IN {5,5,5}
VAR simple_ag, INIT = 5, EV(s_link1) IN [5,5,5]

END ELEMENT
END TEST
END SERVICE

--============================================================

--  init IN {...}
SERVICE simple_s2

TEST t1
FAMILY nominal
ELEMENT
VAR simple_aa, INIT IN {5,5,5},                   EV = 5
END ELEMENT
END TEST

TEST t2
```

IBM Rational Test RealTime: *Metrics Calculation - 1*



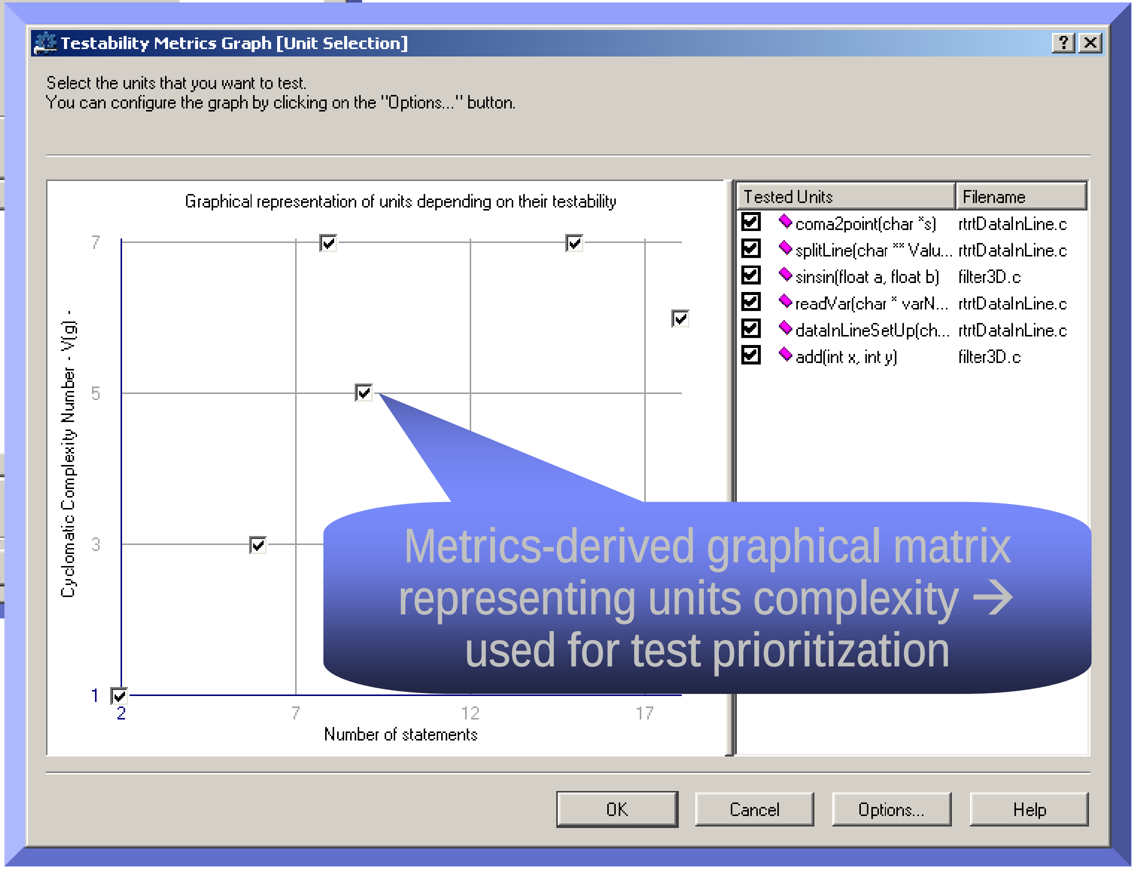
Component Testing Wizard

Components Under Test - Step 2 / 5

Select the files or units that you want to test.
Use the combo box to choose the selection mode.
Click on the next button to specify details of test case generation for the selected items.
You can also make your selection using the complexity metrics diagram accessible via the
button.

Unit Selection

Functions	File Names	V(g)
☐ splitLine(char ** Values, char *line10)	rtrtDataInLine.c	7
☐ readVar(char * varName)	rtrtDataInLine.c	7
☐ dataInLineSetUp(char *filename,char sep,int flag)	rtrtDataInLine.c	6
☐ sinsin(float a, float b)	filter3D.c	5
☐ coma2point(char *s)	rtrtDataInLine.c	3
☐ add(int x, int y)	filter3D.c	1



Testability Metrics Graph [Unit Selection]

Select the units that you want to test.
You can configure the graph by clicking on the "Options..." button.

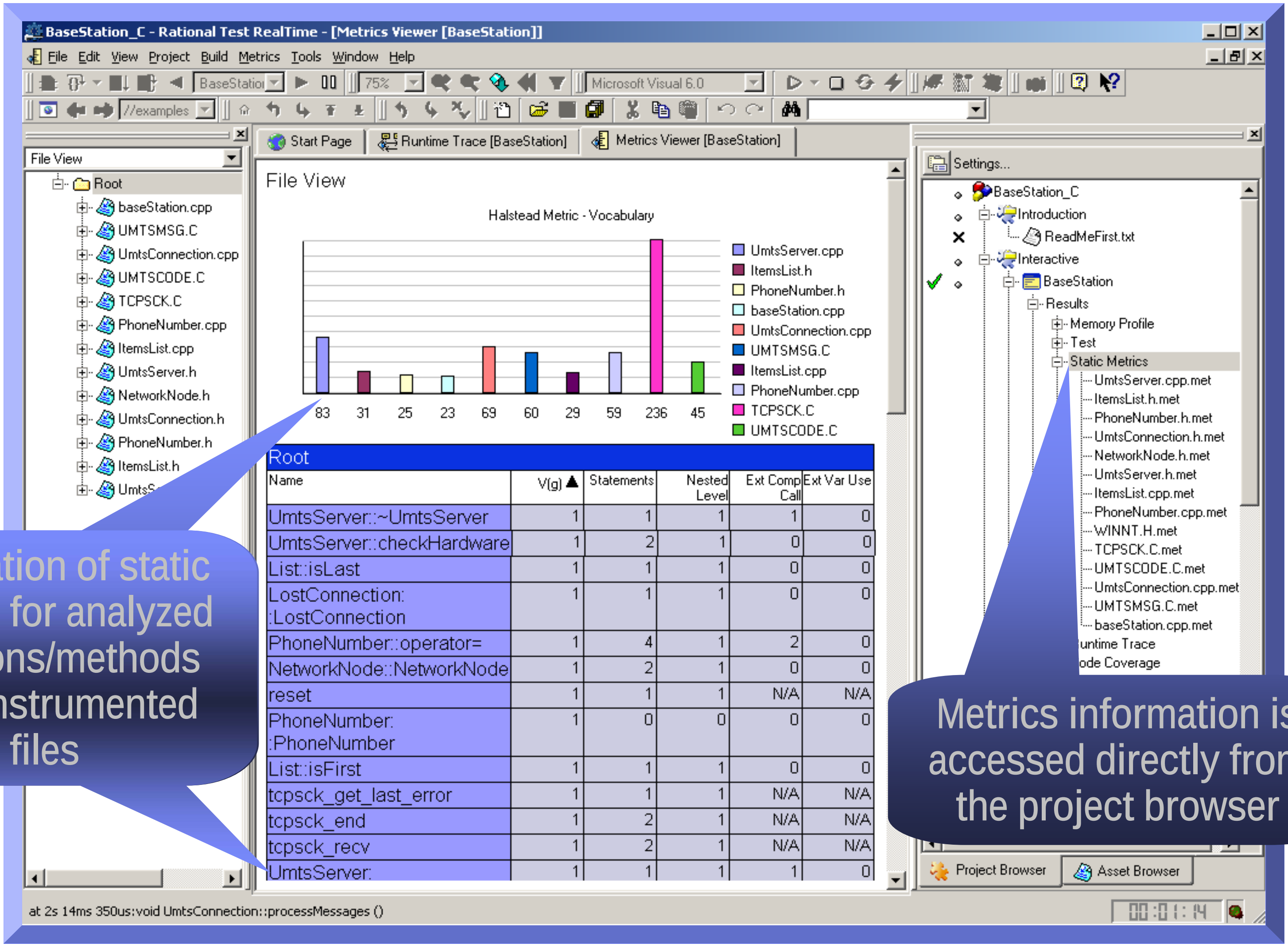
Graphical representation of units depending on their testability
Cyclomatic Complexity Number - V(g) -
Number of statements
7
5
3
1
2
7
12
17

Tested Units	Filename
☑ coma2point(char *s)	rtrtDataInLine.c
☑ splitLine(char ** Valu...	rtrtDataInLine.c
☑ sinsin(float a, float b)	filter3D.c
☑ readVar(char * varN...	rtrtDataInLine.c
☑ dataInLineSetUp(ch...	rtrtDataInLine.c
☑ add(int x, int y)	filter3D.c

Metrics-derived graphical matrix representing units complexity → used for test prioritization

Calculation of static metrics for tested classes/functions from selected components

IBM Rational Test RealTime: *Metrics Calculation - 2*


BaseStation_C - Rational Test RealTime - [Metrics Viewer [BaseStation]]
File Edit View Project Build Metrics Tools Window Help
File View
Start Page
Runtime Trace [BaseStation]
Metrics Viewer [BaseStation]
Settings...
Project Browser
Asset Browser

File View

Halstead Metric - Vocabulary

🟦 UmtsServer.cpp	83
🟥 ItemsList.h	31
🟨 PhoneNumber.h	25
🟦 baseStation.cpp	23
🟥 UmtsConnection.cpp	69
🟦 UMTSMSG.C	60
🟪 ItemsList.cpp	29
🟪 PhoneNumber.cpp	59
🟪 TCPSCK.C	236
🟩 UMTSCODE.C	45

Root

Name	V(g) ▲	Statements	Nested Level	Ext Comp Call	Ext Var Use
UmtsServer::~UmtsServer	1	1	1	1	0
UmtsServer::checkHardware	1	2	1	0	0
List::isLast	1	1	1	0	0
LostConnection::LostConnection	1	1	1	0	0
PhoneNumber::operator=	1	4	1	2	0
NetworkNode::NetworkNode	1	2	1	0	0
reset	1	1	1	N/A	N/A
PhoneNumber::PhoneNumber	1	0	0	0	0
List::isFirst	1	1	1	0	0
tcpsck_get_last_error	1	1	1	N/A	N/A
tcpsck_end	1	2	1	N/A	N/A
tcpsck_recv	1	2	1	N/A	N/A
UmtsServer:	1	1	1	1	0

File View tree: Root; baseStation.cpp; UMTSMSG.C; UmtsConnection.cpp; UMTSCODE.C; TCPSCK.C; PhoneNumber.cpp; ItemsList.cpp; UmtsServer.h; NetworkNode.h; UmtsConnection.h; PhoneNumber.h; ItemsList.h; UmtsS

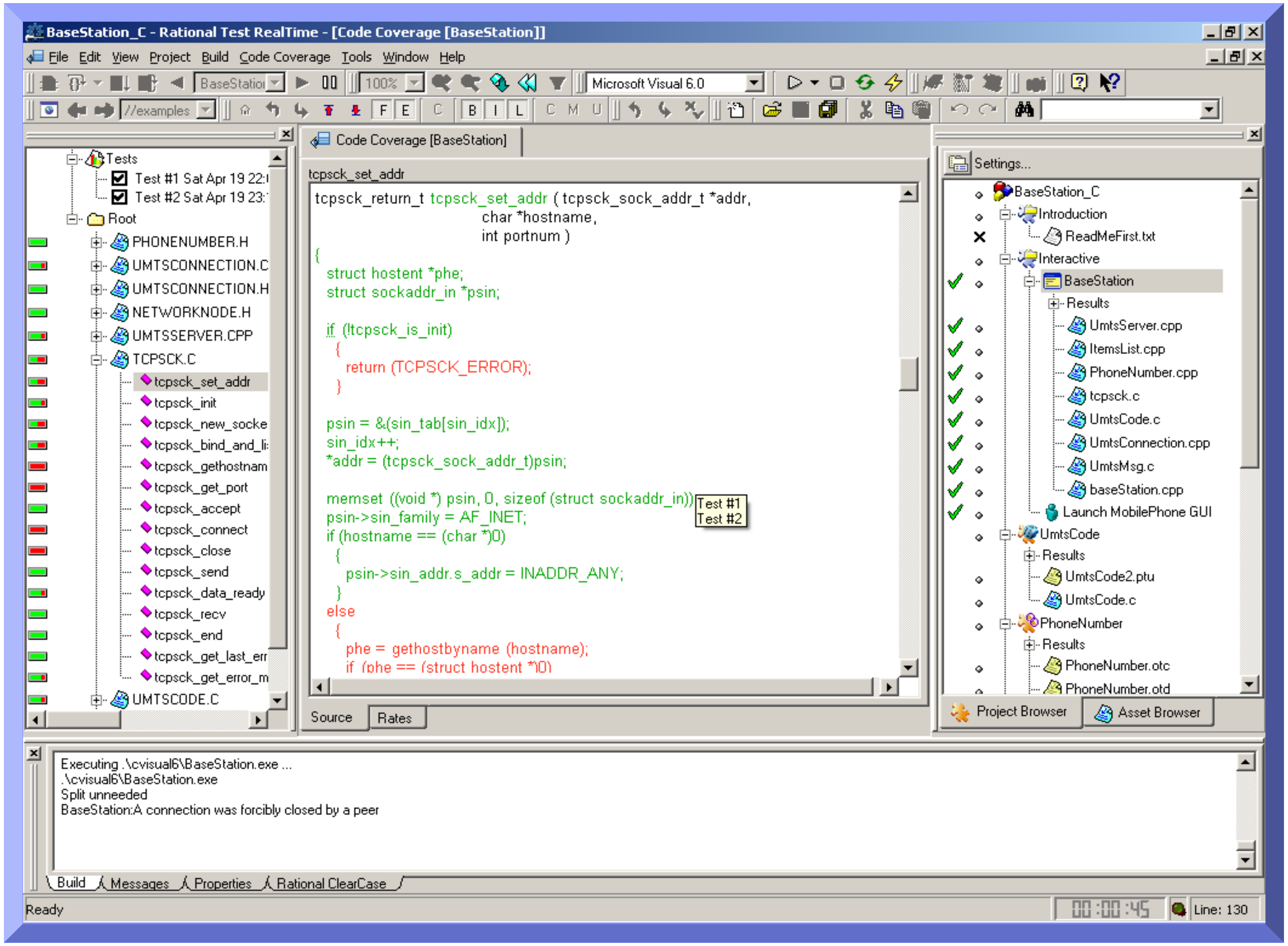
Project Browser tree: BaseStation_C; Introduction; ReadMeFirst.txt; Interactive; BaseStation; Results; Memory Profile; Test; Static Metrics; UmtsServer.cpp.met; ItemsList.h.met; PhoneNumber.h.met; UmtsConnection.h.met; NetworkNode.h.met; UmtsServer.h.met; ItemsList.cpp.met; PhoneNumber.cpp.met; WINNT.H.met; TCPSCK.C.met; UMTSCODE.C.met; UmtsConnection.cpp.met; UMTSMSG.C.met; baseStation.cpp.met; untime Trace; ode Coverage

at 2s 14ms 350us:void UmtsConnection::processMessages ()

Calculation of static metrics for analyzed functions/methods from instrumented files

Metrics information is accessed directly from the project browser

IBM Rational Test RealTime: *Code Coverage*

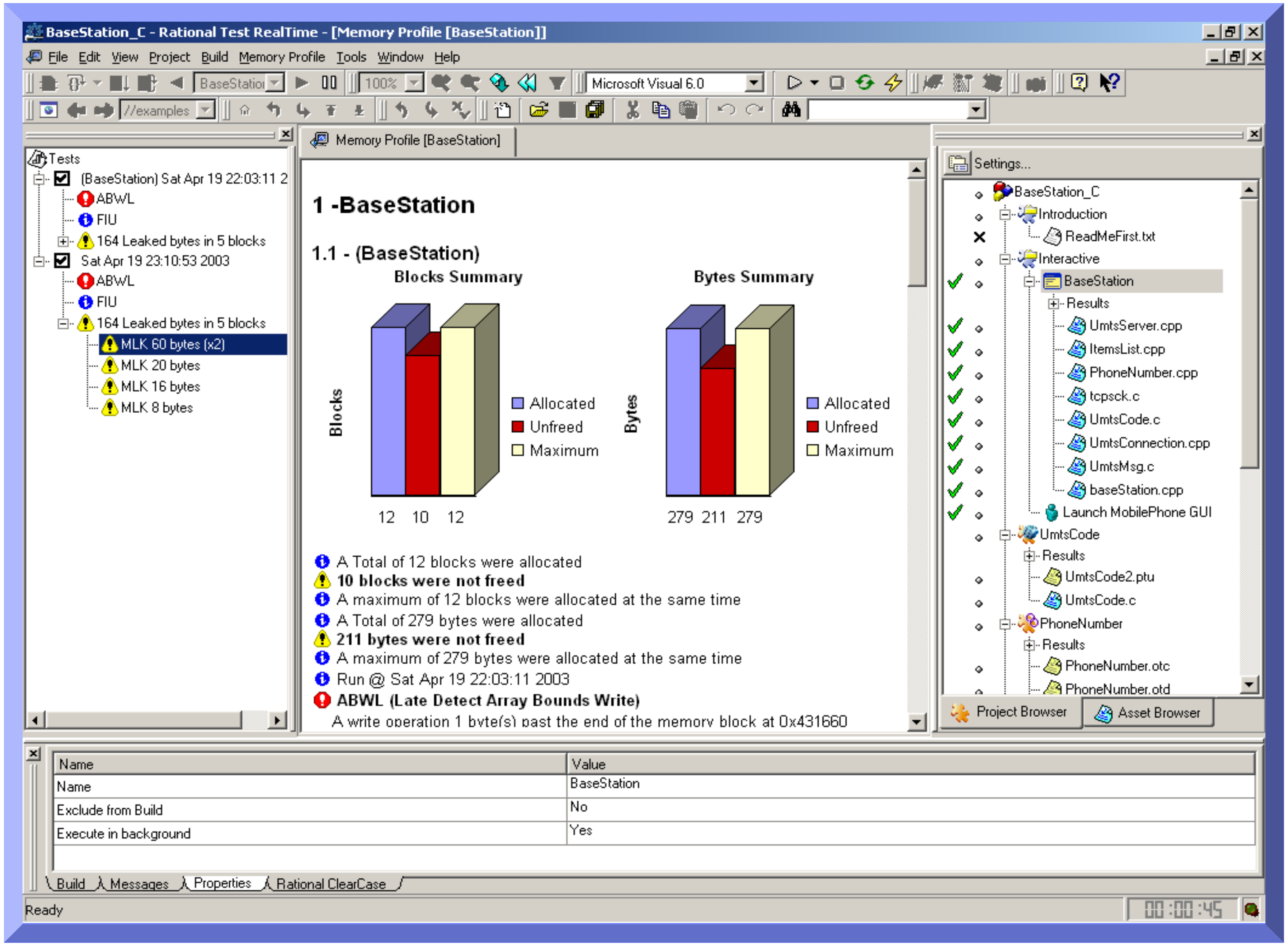


tcpsck_set_addr

```
tcpsck_return_t tcpsck_set_addr ( tcpsck_sock_addr_t *addr,
                                  char *hostname,
                                  int portnum )
{
  struct hostent *phe;
  struct sockaddr_in *psin;

  if (!tcpsck_is_init)
    {
      return (TCPSCK_ERROR);
    }

  psin = &(sin_tab[sin_idx]);
  sin_idx++;
  *addr = (tcpsck_sock_addr_t)psin;

  memset ((void *) psin, 0, sizeof (struct sockaddr_in));Test #1
  psin->sin_family = AF_INET;                              Test #2
  if (hostname == (char *)0)
    {
      psin->sin_addr.s_addr = INADDR_ANY;
    }
  else
    {
      phe = gethostbyname (hostname);
      if (phe == (struct hostent *)0)
```

```
Executing .\cvisual6\BaseStation.exe ...
.\cvisual6\BaseStation.exe
Split unneeded
BaseStation:A connection was forcibly closed by a peer
```

IBM Rational Test RealTime: *Memory Profiling*



1 -BaseStation

1.1 - (BaseStation)

Blocks Summary

Blocks	Allocated	Unfreed	Maximum
	12	10	12

Bytes Summary

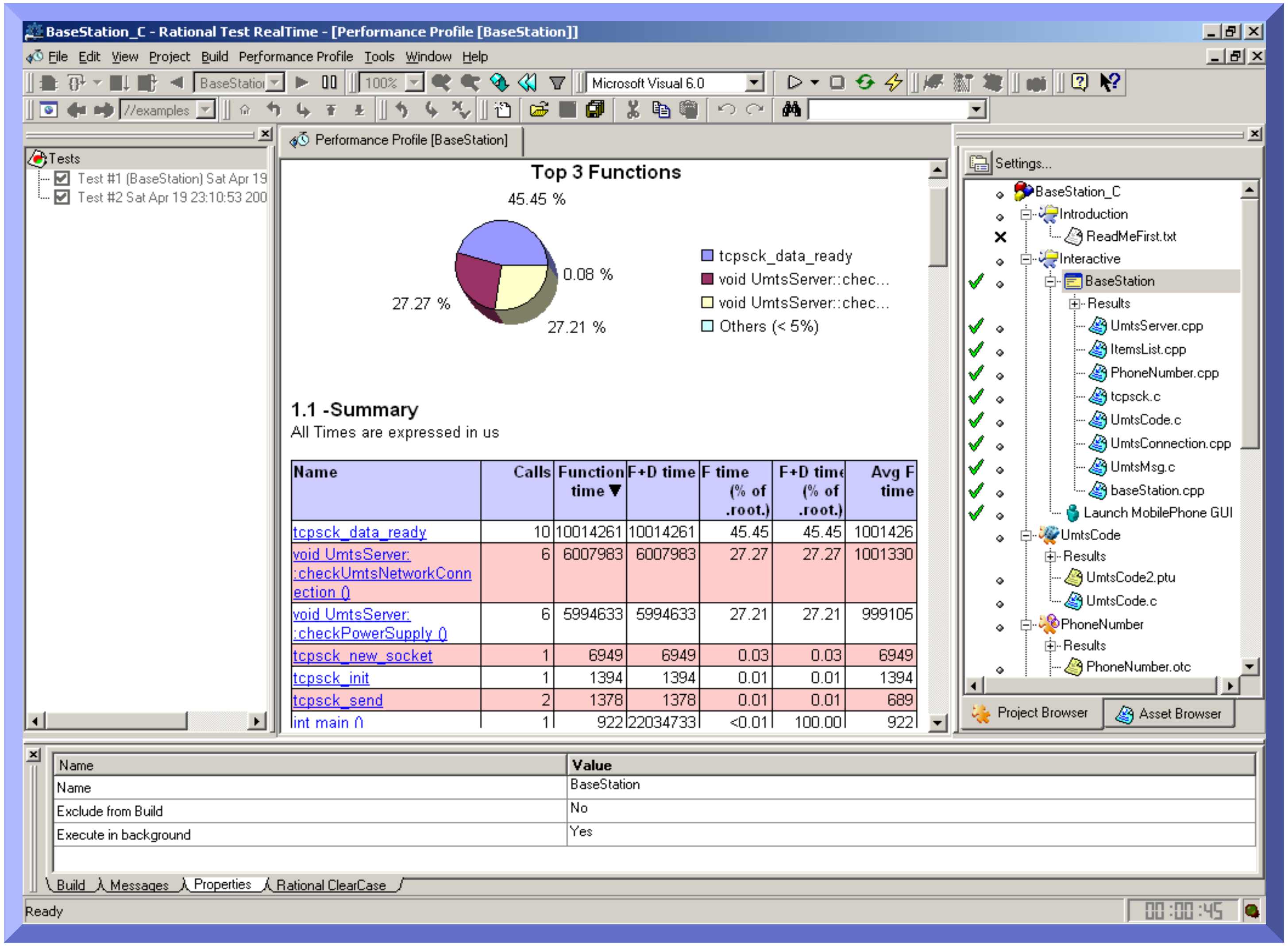
Bytes	Allocated	Unfreed	Maximum
	279	211	279

- A Total of 12 blocks were allocated
- **10 blocks were not freed**
- A maximum of 12 blocks were allocated at the same time
- A Total of 279 bytes were allocated
- **211 bytes were not freed**
- A maximum of 279 bytes were allocated at the same time
- Run @ Sat Apr 19 22:03:11 2003
- **ABWL (Late Detect Array Bounds Write)**

A write operation 1 byte(s) past the end of the memory block at 0x431660

Name	Value
Name	BaseStation
Exclude from Build	No
Execute in background	Yes

IBM Rational Test RealTime: *Performance Profiling*



Top 3 Functions

45.45 %
0.08 %
27.27 %
27.21 %

- ▢ tcpsck_data_ready
- ■ void UmtsServer::chec...
- ▢ void UmtsServer::chec...
- ▢ Others (< 5%)

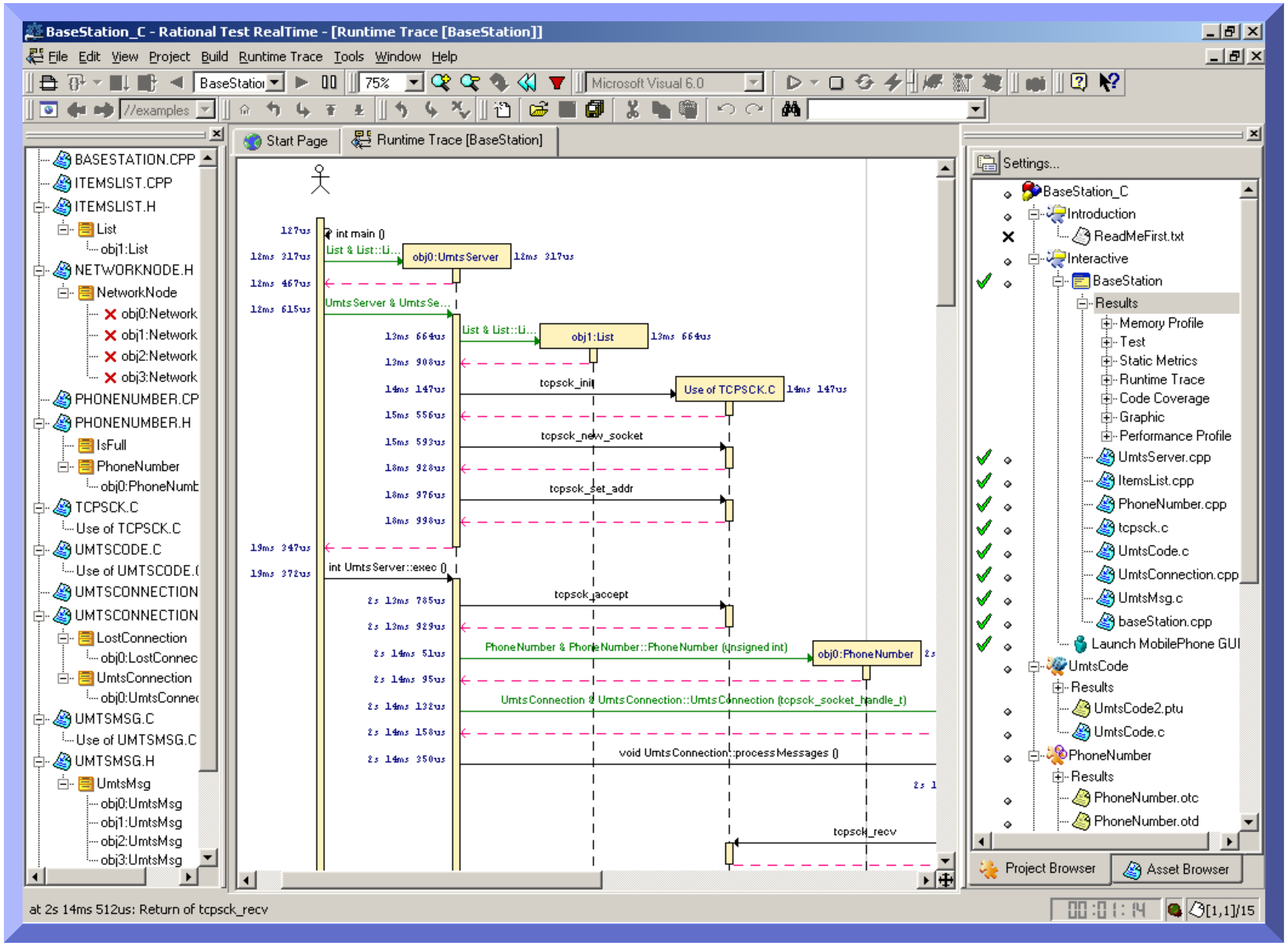
1.1 -Summary

All Times are expressed in us

Name	Calls	Function time ▼	F+D time	F time (% of .root.)	F+D time (% of .root.)	Avg F time
tcpsck_data_ready	10	10014261	10014261	45.45	45.45	1001426
void UmtsServer::checkUmtsNetworkConnection ()	6	6007983	6007983	27.27	27.27	1001330
void UmtsServer::checkPowerSupply ()	6	5994633	5994633	27.21	27.21	999105
tcpsck_new_socket	1	6949	6949	0.03	0.03	6949
tcpsck_init	1	1394	1394	0.01	0.01	1394
tcpsck_send	2	1378	1378	0.01	0.01	689
int main ()	1	922	22034733	<0.01	100.00	922

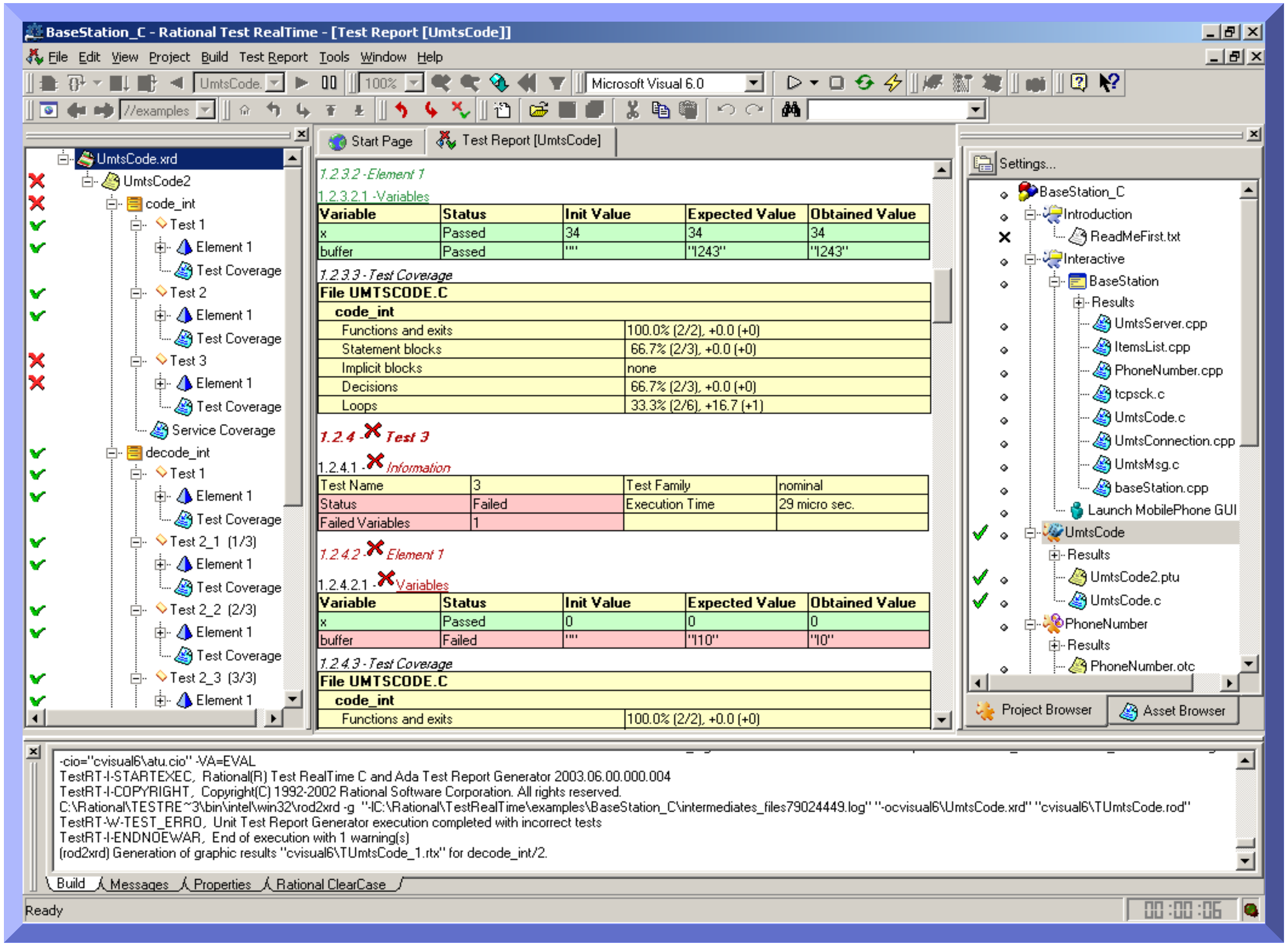
Name	Value
Name	BaseStation
Exclude from Build	No
Execute in background	Yes

IBM Rational Test RealTime: *Runtime Tracing*



at 2s 14ms 512us: Return of tcpsck_recv

IBM Rational Test RealTime: *Test Report*



1.2.3.2 - Element 1

1.2.3.2.1 - Variables

Variable	Status	Init Value	Expected Value	Obtained Value
x	Passed	34	34	34
buffer	Passed	""	"I243"	"I243"

1.2.3.3 - Test Coverage

File UMTSCODE.C	
code_int	
Functions and exits	100.0% (2/2), +0.0 (+0)
Statement blocks	66.7% (2/3), +0.0 (+0)
Implicit blocks	none
Decisions	66.7% (2/3), +0.0 (+0)
Loops	33.3% (2/6), +16.7 (+1)

1.2.4 - Test 3

1.2.4.1 - *Information*

Test Name	3	Test Family	nominal
Status	Failed	Execution Time	29 micro sec.
Failed Variables	1		

1.2.4.2 - Element 1

1.2.4.2.1 - Variables

Variable	Status	Init Value	Expected Value	Obtained Value
x	Passed	0	0	0
buffer	Failed	""	"I10"	"I0"

1.2.4.3 - Test Coverage

File UMTSCODE.C	
code_int	
Functions and exits	100.0% (2/2), +0.0 (+0)

```
-cio="cvisual6\atu.cio" -VA=EVAL
TestRT-I-STARTEXEC,  Rational(R) Test RealTime C and Ada Test Report Generator 2003.06.00.000.004
TestRT-I-COPYRIGHT,  Copyright(C) 1992-2002 Rational Software Corporation. All rights reserved.
C:\Rational\TESTRE~3\bin\intel\win32\rod2xrd -g  "-IC:\Rational\TestRealTime\examples\BaseStation_C\intermediates_files79024449.log" "-ocvisual6\UmtsCode.xrd" "cvisual6\TUmtsCode.rod"
TestRT-W-TEST_ERRO,  Unit Test Report Generator execution completed with incorrect tests
TestRT-I-ENDNOEWAR,  End of execution with 1 warning(s)
(rod2xrd) Generation of graphic results "cvisual6\TUmtsCode_1.rtx" for decode_int/2.
```

IBM Rational Test RealTime

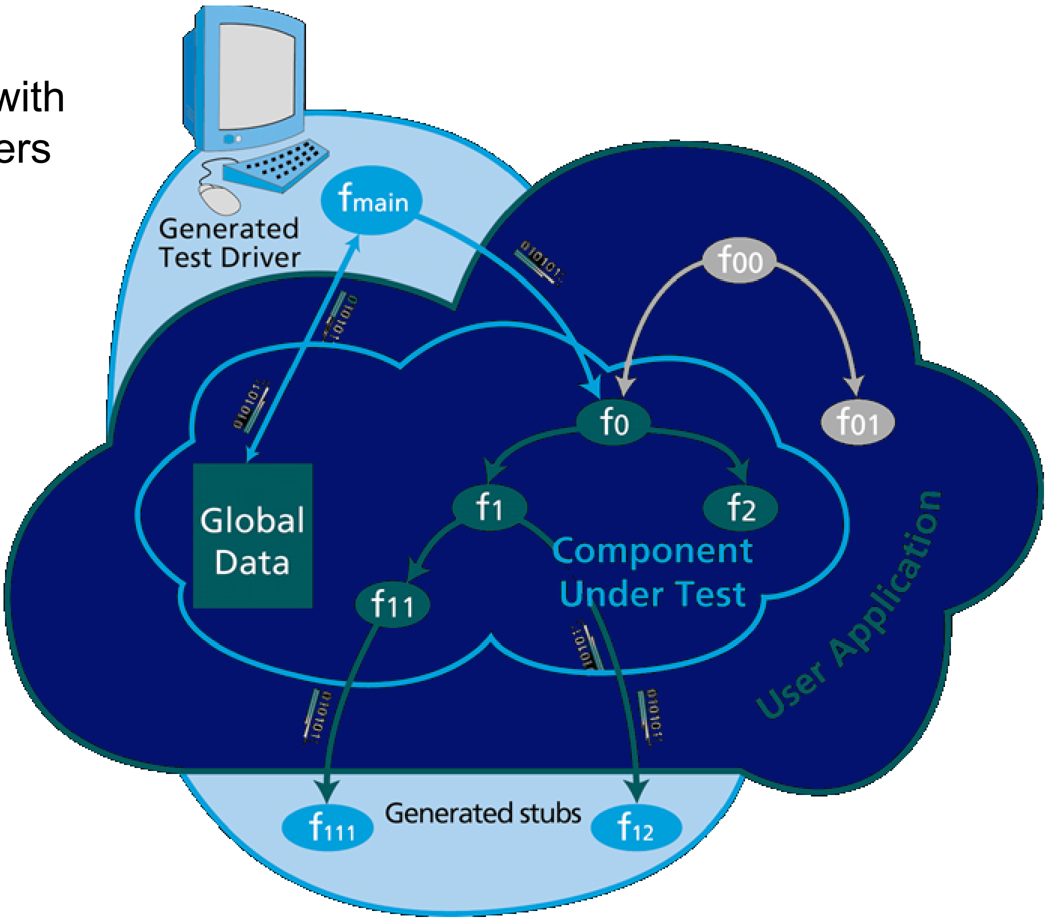
Component Testing for C and Ada Presentation

Test RealTime: *Component Testing for C & Ada*

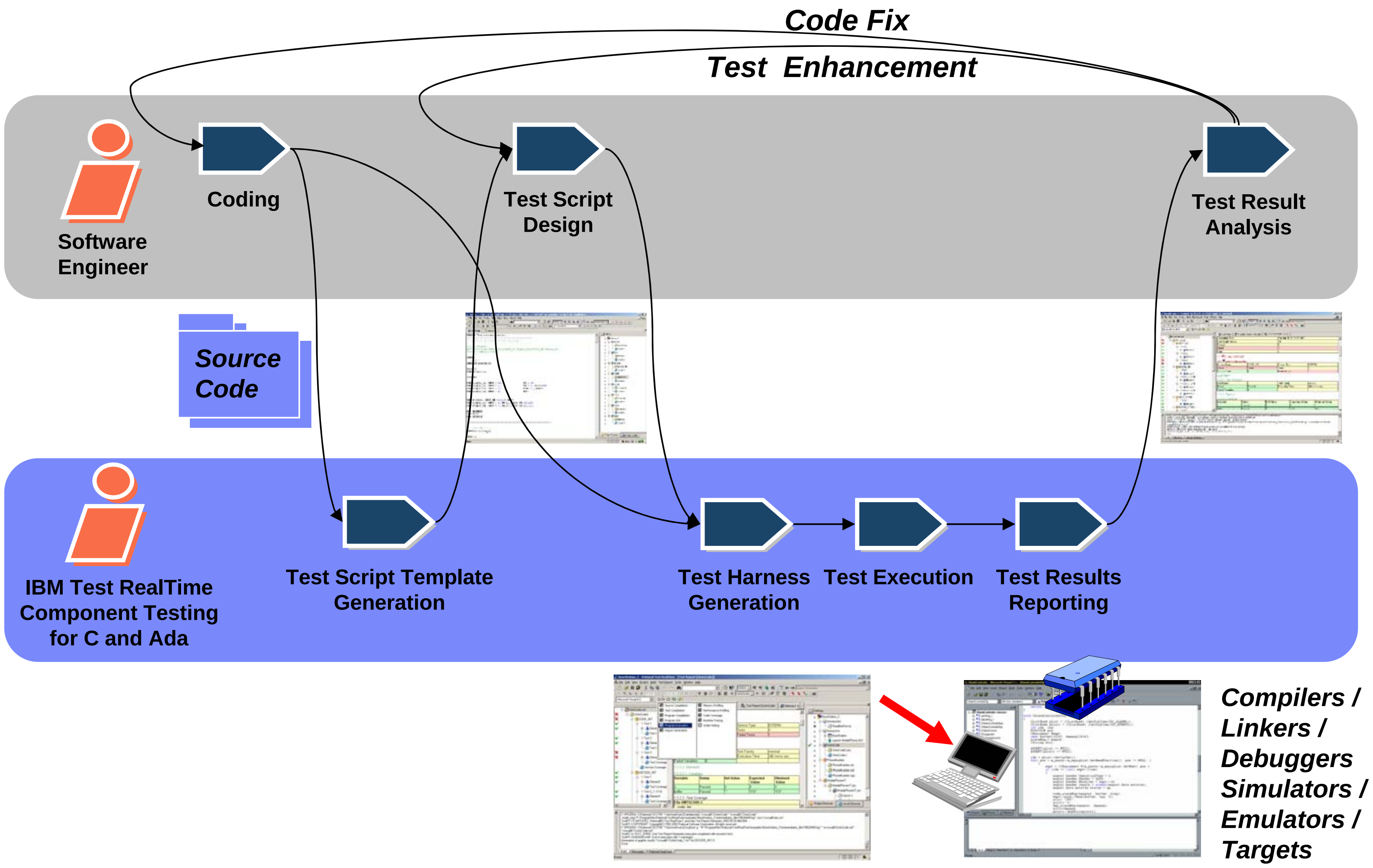
- **Source code-driven, data-intensive functional and structural testing via function calls**
- **Automatic**
  - Test template generation from source code
  - Test data generation from pattern language
  - Stub creation
  - Regression Testing
- **Detailed reporting**
- **Use of static metrics for test prioritization**
  - Software Complexity Level
- **Works with Test RealTime Runtime Analysis features**
  - Memory and Performance Profiling
  - Code Coverage and Runtime Tracing

Test Harness Architecture and Responsibilities

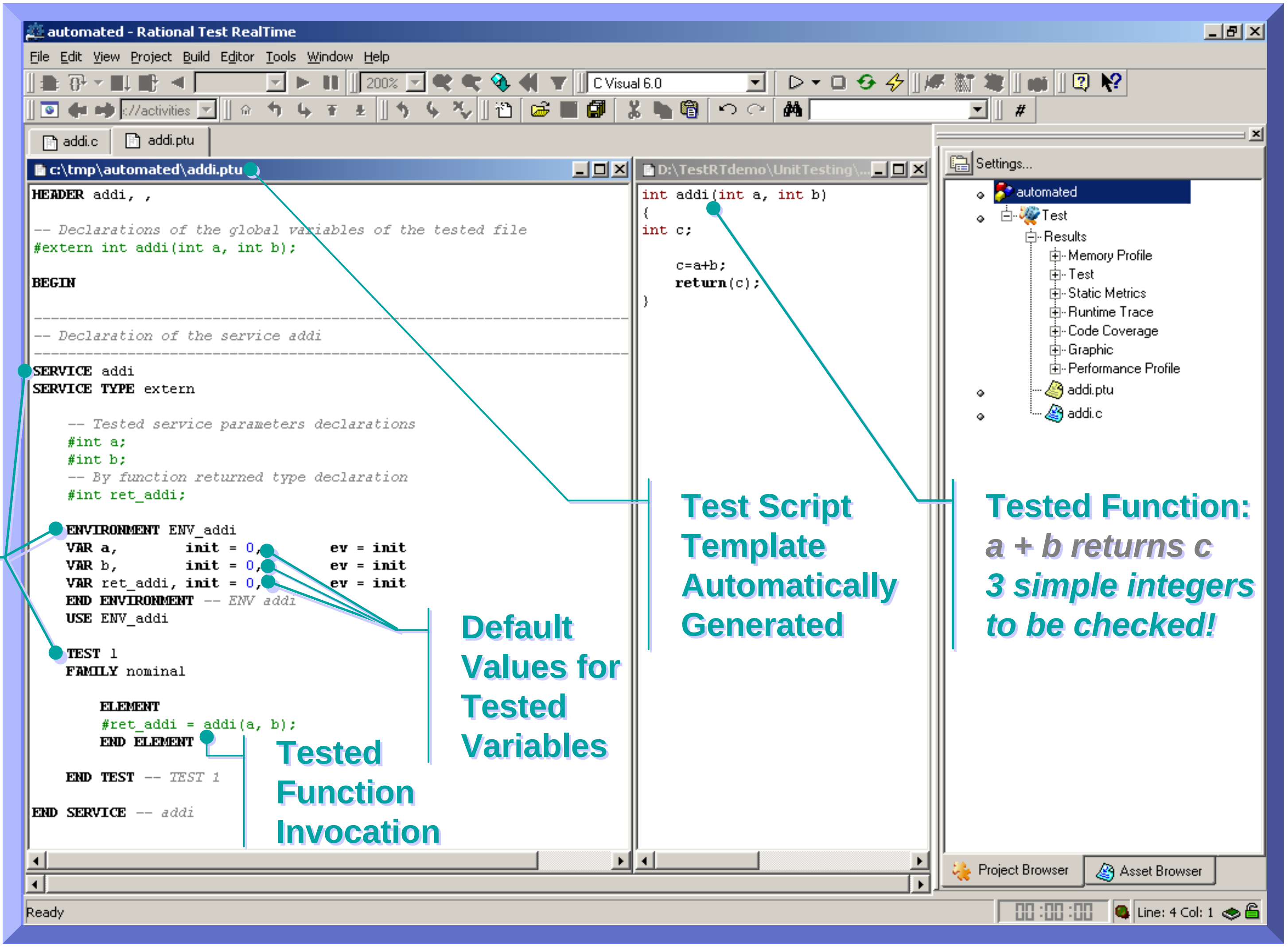
1. **Test driver**
   - Calls the function-under-test with the desired range of parameters
   - Checks returned parameters
   - Accesses global variables
2. **Stubs**
   - Check input parameters
   - Return desired parameters



Component Testing for C & Ada: *Process*



Component Testing for C & Ada: *Test Script Pattern*



automated - Rational Test RealTime

c:\tmp\automated\addi.ptu

```
HEADER addi, ,

-- Declarations of the global variables of the tested file
#extern int addi(int a, int b);

BEGIN

----------------------------------------------------------------------
-- Declaration of the service addi
----------------------------------------------------------------------
SERVICE addi
SERVICE TYPE extern

    -- Tested service parameters declarations
    #int a;
    #int b;
    -- By function returned type declaration
    #int ret_addi;

    ENVIRONMENT ENV_addi
    VAR a,        init = 0,        ev = init
    VAR b,        init = 0,        ev = init
    VAR ret_addi, init = 0,        ev = init
    END ENVIRONMENT -- ENV_addi
    USE ENV_addi

    TEST 1
    FAMILY nominal

        ELEMENT
        #ret_addi = addi(a, b);
        END ELEMENT

    END TEST -- TEST 1

END SERVICE -- addi
```

D:\TestRTdemo\UnitTesting\...

```
int addi(int a, int b)
{
int c;

    c=a+b;
    return(c);
}
```

Default Test Structure

Tested Function Invocation

Default Values for Tested Variables

Test Script Template Automatically Generated

Tested Function: *a + b returns c*
3 simple integers to be checked!

Component Testing for C & Ada: *Test Language*

Input values initialization:
- Multiple,
- Ranges, etc.

Expected values definition:
- According to requirements
- Using Range, delta, etc.

The function under test

A single instruction to define all test data

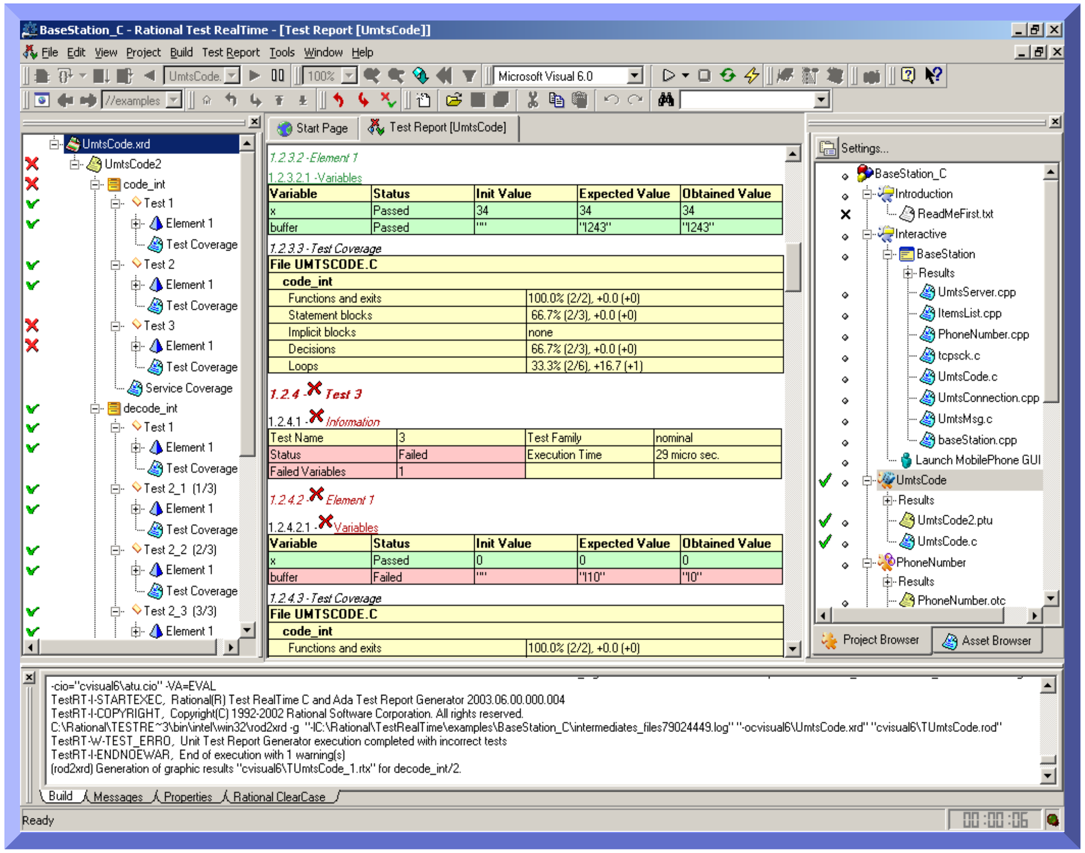
```
TEST 1

  ELEMENT

   #ret_val=myfunction(y,a,z,b,c);

   VAR glob,     init=0                    ,   ev==
   VAR y,        init in {-1,glob,0}       ,   ev in {-1,0,0}
   VAR a[1..10],init from 1 to 1000 step 1 ,   ev(y) in {0,2,3}
   VAR z.field1,init=a[2]                  ,   ev=ret_val
   VAR b,        init==                    ,   min=y,  max=y*10
   VAR c,        init=b                    ,   ev=10,  delta=10%
   VAR ret_val,  init=MY_DEFINE            ,   ev=init

   STUB alloc_block, 0=>(100)&a, OTHERS=>()NIL

  END ELEMENT

END TEST
```

STUBs:
- Check parameters,
- Return values

In 2 lines, 3x1000=3000 test cases are generated!

Component Testing for C & Ada: *Reporting*

- **Easy to understand report**
  - Passed and failed test cases at a glance
  - Initial, expected, obtained values for all managed variables and stubs
  - Source code coverage information from Rational Test RealTime Code Coverage feature
- **Exports to HTML great for**
  - Distributed development
  - Test subcontracting


BaseStation_C - Rational Test RealTime - [Test Report [UmtsCode]]

1.2.3.2 -Element 1

1.2.3.2.1 -Variables

Variable	Status	Init Value	Expected Value	Obtained Value
x	Passed	34	34	34
buffer	Passed	""	"I243"	"I243"

1.2.3.3 - Test Coverage

File UMTSCODE.C	
code_int	
Functions and exits	100.0% (2/2), +0.0 (+0)
Statement blocks	66.7% (2/3), +0.0 (+0)
Implicit blocks	none
Decisions	66.7% (2/3), +0.0 (+0)
Loops	33.3% (2/6), +16.7 (+1)

1.2.4 - Test 3

1.2.4.1 - *Information*

Test Name	3	Test Family	nominal
Status	Failed	Execution Time	29 micro sec.
Failed Variables	1		

1.2.4.2 - Element 1

1.2.4.2.1 - Variables

Variable	Status	Init Value	Expected Value	Obtained Value
x	Passed	0	0	0
buffer	Failed	""	"I10"	"I0"

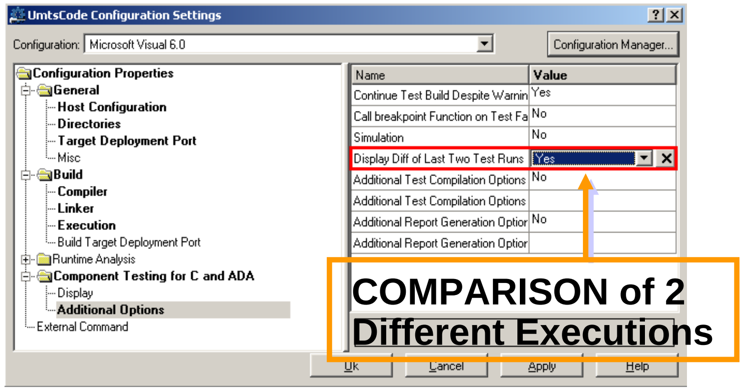
1.2.4.3 - Test Coverage

File UMTSCODE.C	
code_int	
Functions and exits	100.0% (2/2), +0.0 (+0)

```
-cio="cvisual6\atu.cio" -VA=EVAL
TestRT-I-STARTEXEC, Rational(R) Test RealTime C and Ada Test Report Generator 2003.06.00.000.004
TestRT-I-COPYRIGHT, Copyright(C) 1992-2002 Rational Software Corporation. All rights reserved.
C:\Rational\TESTRE~3\bin\intel\win32\rod2xrd -g "-IC:\Rational\TestRealTime\examples\BaseStation_C\intermediates_files79024449.log" "-ocvisual6\UmtsCode.xrd" "cvisual6\TUmtsCode.rod"
TestRT-W-TEST_ERRO, Unit Test Report Generator execution completed with incorrect tests
TestRT-I-ENDNOEWAR, End of execution with 1 warning(s)
(rod2xrd) Generation of graphic results "cvisual6\TUmtsCode_1.rtx" for decode_int/2.
```

Component Testing for C & Ada: *Tests Comparison*

- **Between:**
  - 2 iterations of same software component
  - 2 different development environments
  - Instrumented vs. non-instrumented code
  - Generated code vs. manual code



UmtsCode Configuration Settings

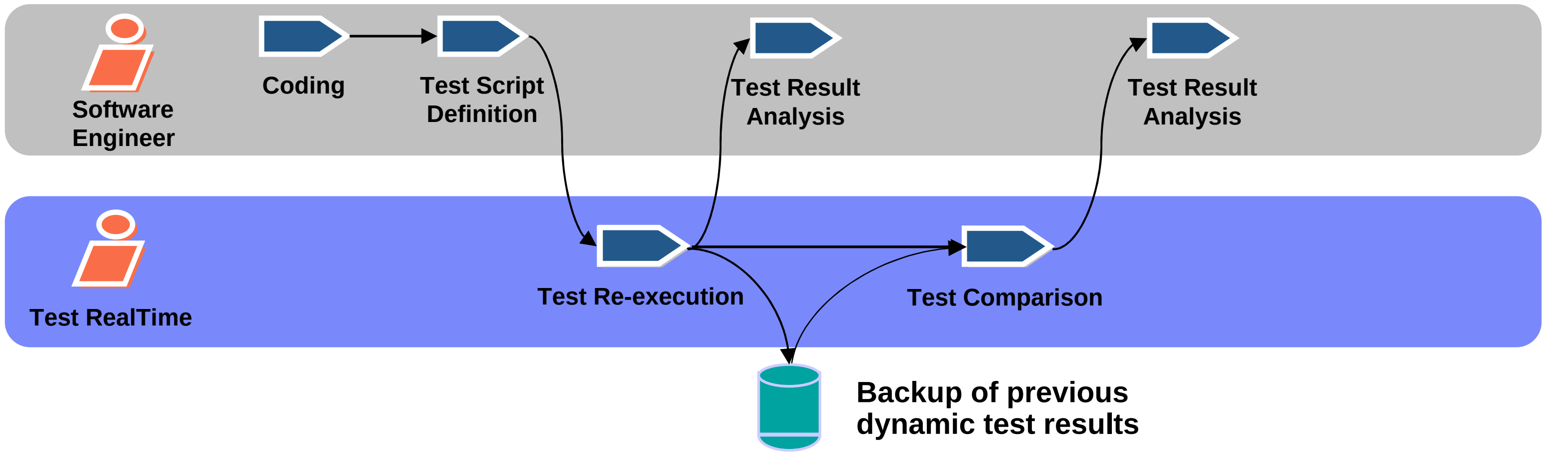
Configuration: Microsoft Visual 6.0 | Configuration Manager...

Configuration Properties
- General
  - Host Configuration
  - Directories
  - Target Deployment Port
  - Misc
- Build
  - Compiler
  - Linker
  - Execution
  - Build Target Deployment Port
- Runtime Analysis
- Component Testing for C and ADA
  - Display
  - Additional Options
- External Command

Name	Value
Continue Test Build Despite Warnin	Yes
Call breakpoint Function on Test Fa	No
Simulation	No
Display Diff of Last Two Test Runs	Yes
Additional Test Compilation Options	No
Additional Test Compilation Options	
Additional Report Generation Optior	No
Additional Report Generation Optior	

Ok | Cancel | Apply | Help

COMPARISON of 2 Different Executions



Variable	Status	Init Value	Expected Value	Obtained Value	Obtained value Comparison
x1	Failed	9	9	10	9
y1	Failed	9	9	10	9

IBM Rational Test RealTime

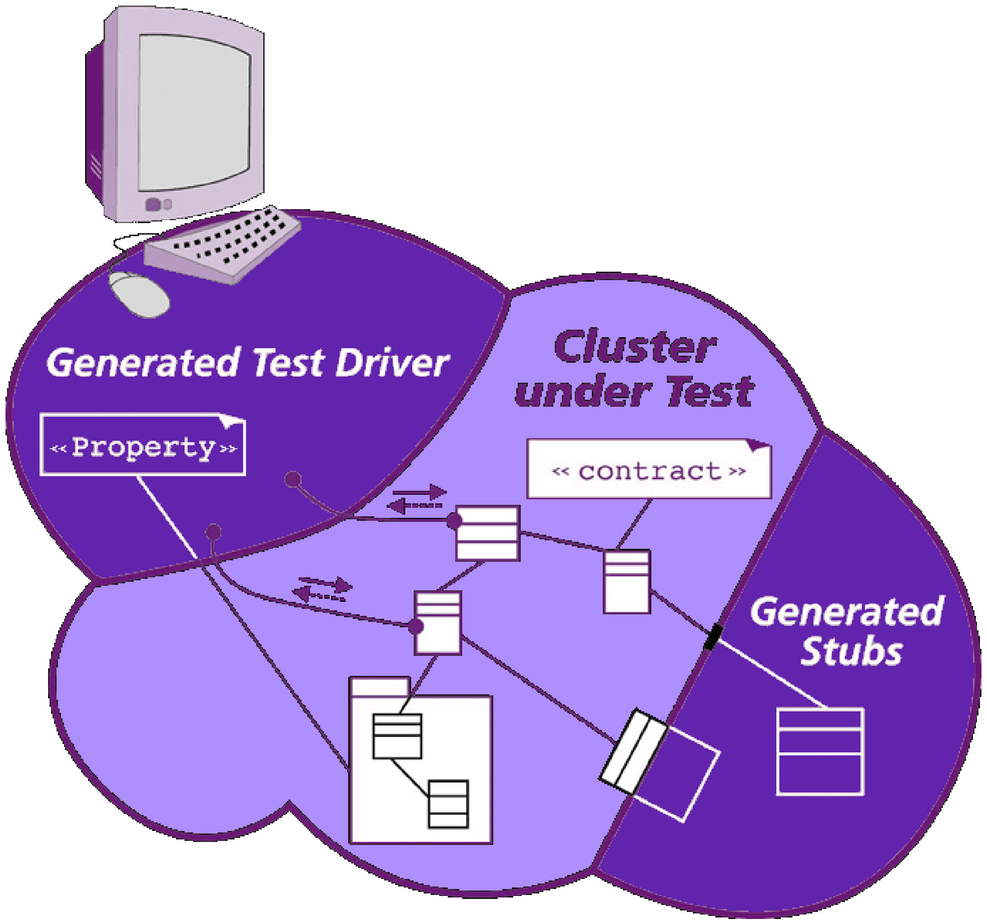
Component Testing for C++ Presentation

Test RealTime: *Component Testing for C++*

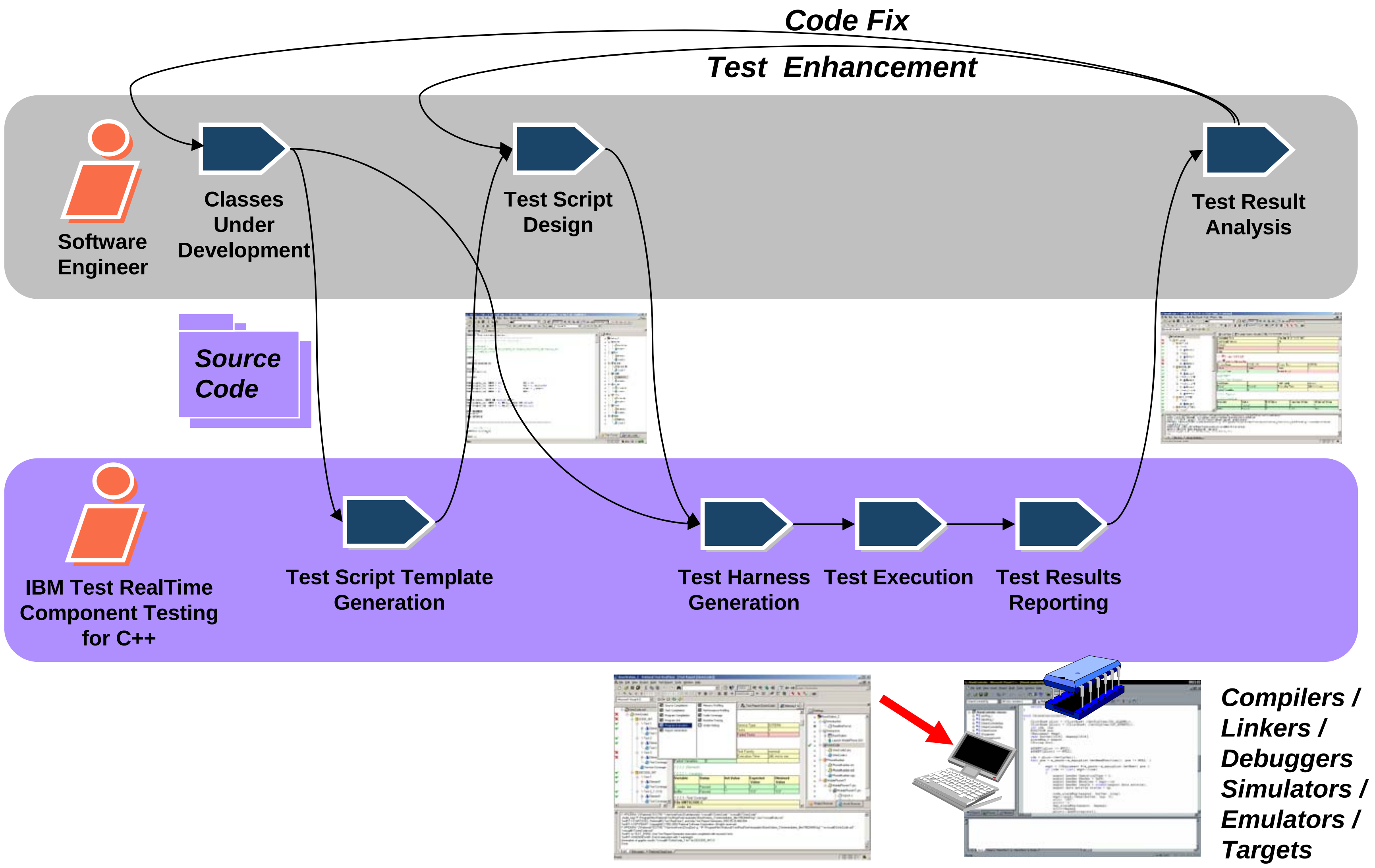
- **Source code–driven, functional and structural testing via method invocation scenarios**
- **Automatic**
  - Test template generation from source code
  - Assertion checking
  - Stub creation
  - Regression Testing
- **Detailed reporting**
- **Use of static metrics for test prioritization**
  - Software Complexity Level
- **Works with Test RealTime Runtime Analysis features**
  - Memory and Performance Profiling
  - Code Coverage and Runtime Tracing

Test Harness Architecture and Responsibilities

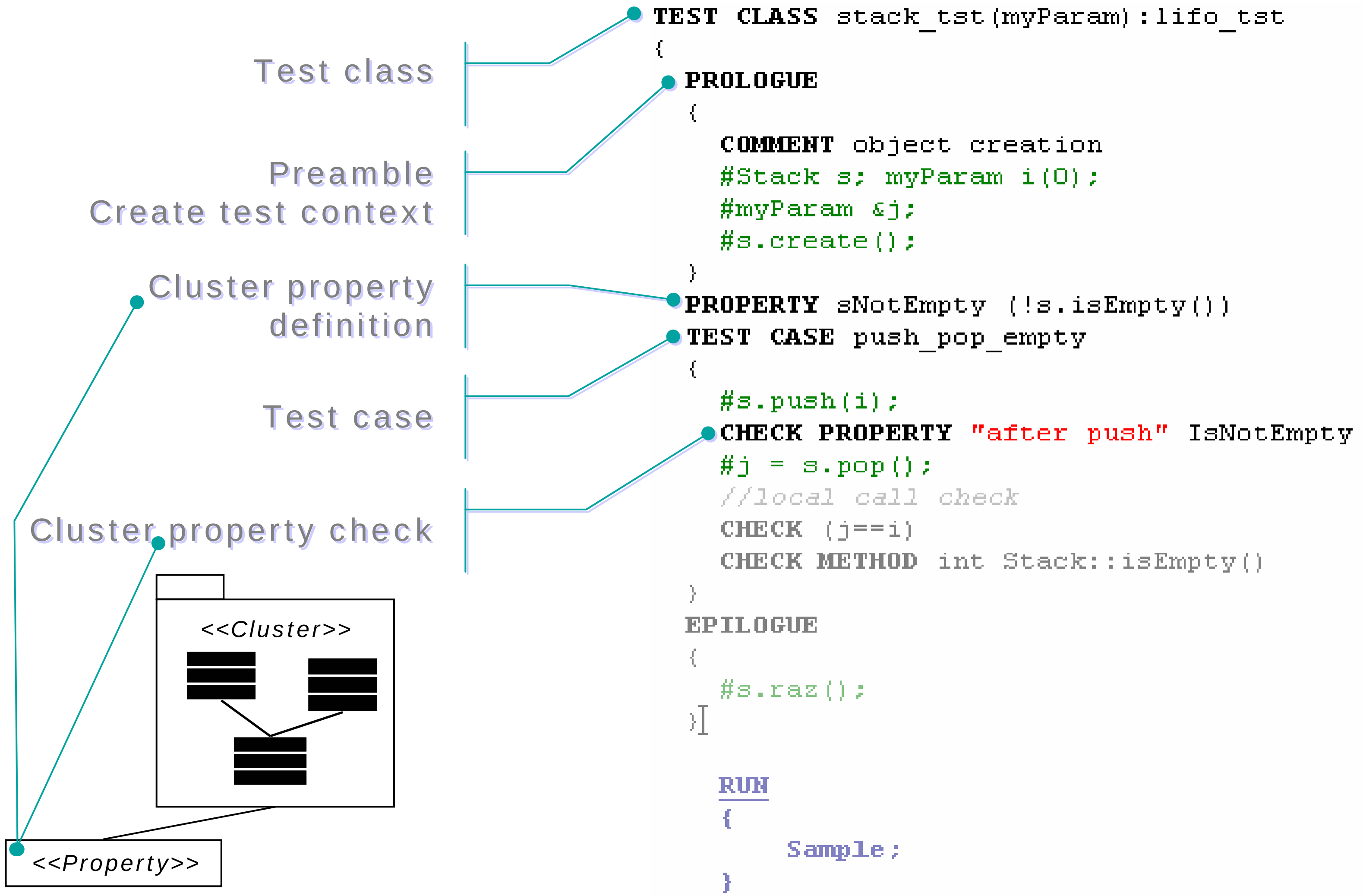
1. **Test driver**
   - Invokes a sequence of methods in the set of classes under test
   - Assesses returned values
   - Assesses cluster properties
2. **Stubs**
   - Check input parameters
   - Return desired parameters
3. **Class assertion checks**
   - Based on user-defined contracts



Component Testing for C++: *Process*

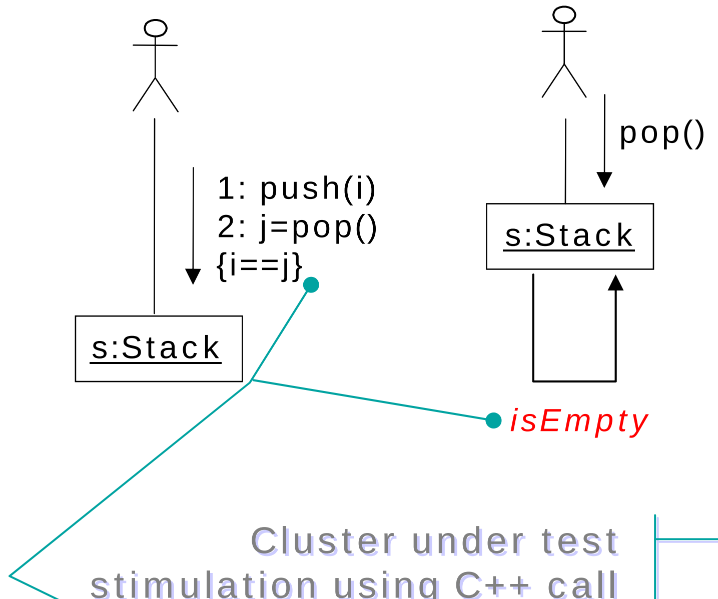

Code Fix
Test Enhancement
Software Engineer
Classes Under Development
Test Script Design
Test Result Analysis
Source Code
IBM Test RealTime Component Testing for C++
Test Script Template Generation
Test Harness Generation
Test Execution
Test Results Reporting
Compilers / Linkers / Debuggers Simulators / Emulators / Targets

Component Testing for C++: *Cluster Properties*



```
TEST CLASS stack_tst(myParam):lifo_tst
{
 PROLOGUE
 {
   COMMENT object creation
   #Stack s; myParam i(0);
   #myParam &j;
   #s.create();
 }
 PROPERTY sNotEmpty (!s.isEmpty())
 TEST CASE push_pop_empty
 {
   #s.push(i);
   CHECK PROPERTY "after push" IsNotEmpty
   #j = s.pop();
   //local call check
   CHECK (j==i)
   CHECK METHOD int Stack::isEmpty()
 }
 EPILOGUE
 {
   #s.raz();
 }

   RUN
   {
       Sample;
   }
```

Component Testing for C++: *Cluster Properties*



Cluster under test stimulation using C++ call

Checks
Called methods
Boolean expressions

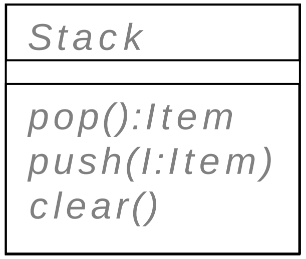
Postamble:
Clean test context

Test class instantiation

```
TEST CLASS stack_tst(myParam):lifo_tst
{
  PROLOGUE
  {
    COMMENT object creation
    #Stack s; myParam i(0);
    #myParam &j;
    #s.create();
  }
  PROPERTY sNotEmpty (!s.isEmpty())
  TEST CASE push_pop_empty
  {
    #s.push(i);
    CHECK PROPERTY "after push" IsNotEmpty
    #j = s.pop();
    //local call check
    CHECK (j==i)
    CHECK METHOD int Stack::isEmpty()
  }
  EPILOGUE
  {
    #s.raz();
  }

    RUN
    {
        Sample;
    }
```

Component Testing for C++: *Class Assertion*

A set of conditions expected to *always* be true

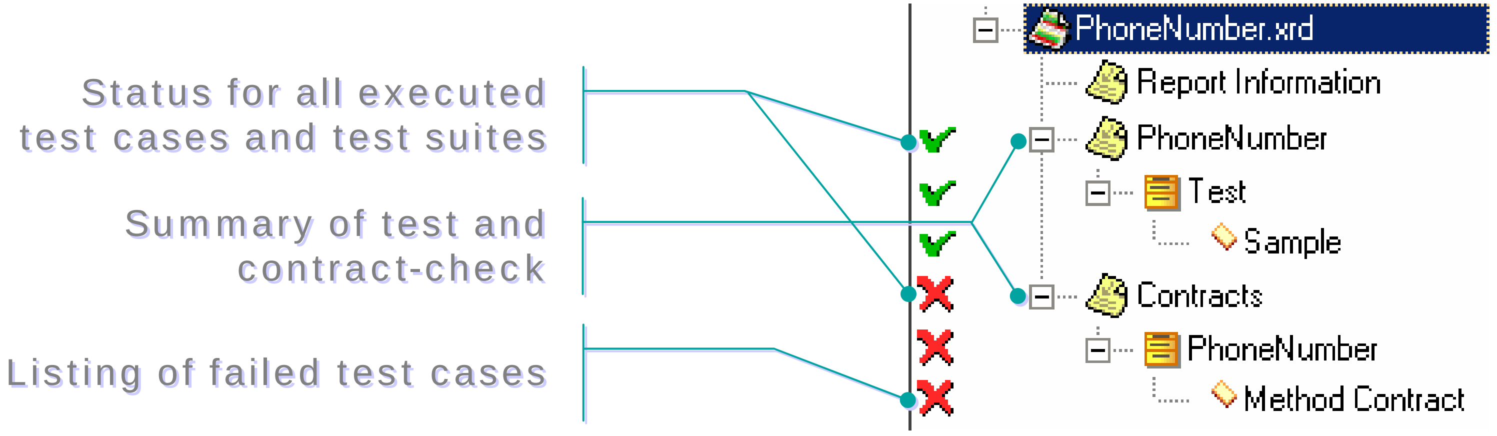


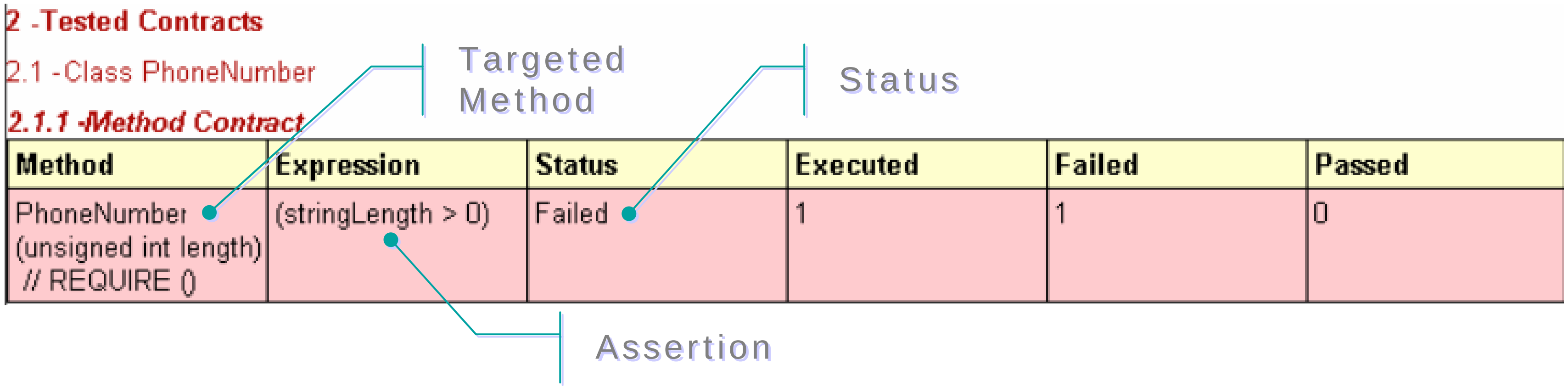
- Class context
- Class invariant
- Method pre-condition
- Method post-condition
- State definitions
- Transitions between states

```
CLASS Stack
{
    // Invariants
    INVARIANT ((first == 0 && last == 0) || (first != 0 && last != 0));

    // Pre- and postconditions
    WRAP pop
    REQUIRE (first != 0)

    WRAP clear
    ENSURE (count () == 0)

    // States
    STATE Empty { (count () == 0) }
    STATE NotEmpty { (count () > 0) }

    // Transitions
    TRANSITION Empty TO NotEmpty;
    TRANSITION NotEmpty TO Empty;
}
```

Component Testing for C++: *Test Report*

Easy to understand report:
Passed and failed test cases and assertion at a glance


Status for all executed test cases and test suites
Summary of test and contract-check
Listing of failed test cases
PhoneNumber.xrd
Report Information
PhoneNumber
Test
Sample
Contracts
PhoneNumber
Method Contract



2 -Tested Contracts

2.1 -Class PhoneNumber

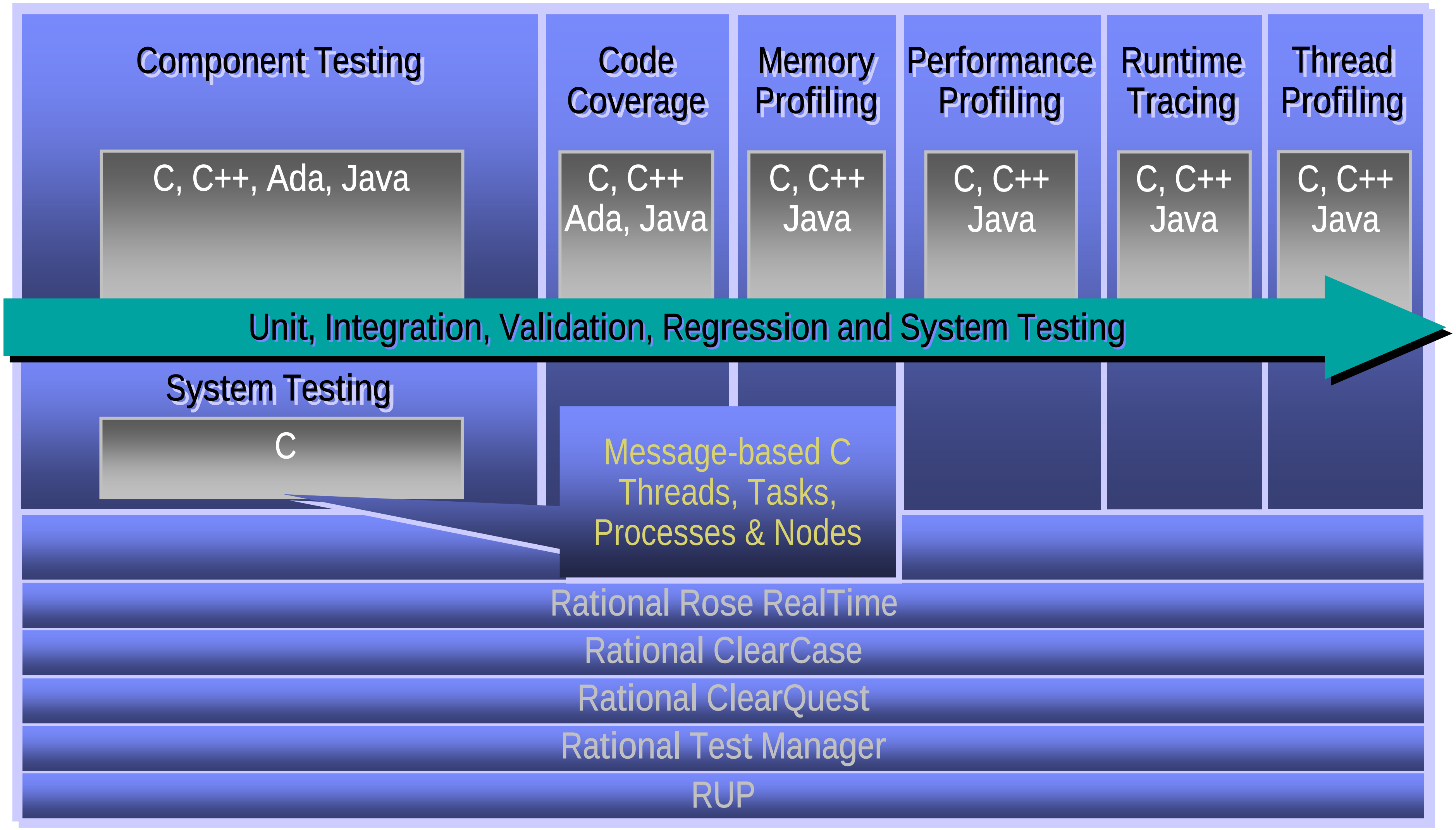
2.1.1 -Method Contract

Method	Expression	Status	Executed	Failed	Passed
PhoneNumber (unsigned int length) // REQUIRE ()	(stringLength > 0)	Failed	1	1	0

IBM Rational Test RealTime

System Testing for C Presentation

Test RealTime: *System Testing for C*

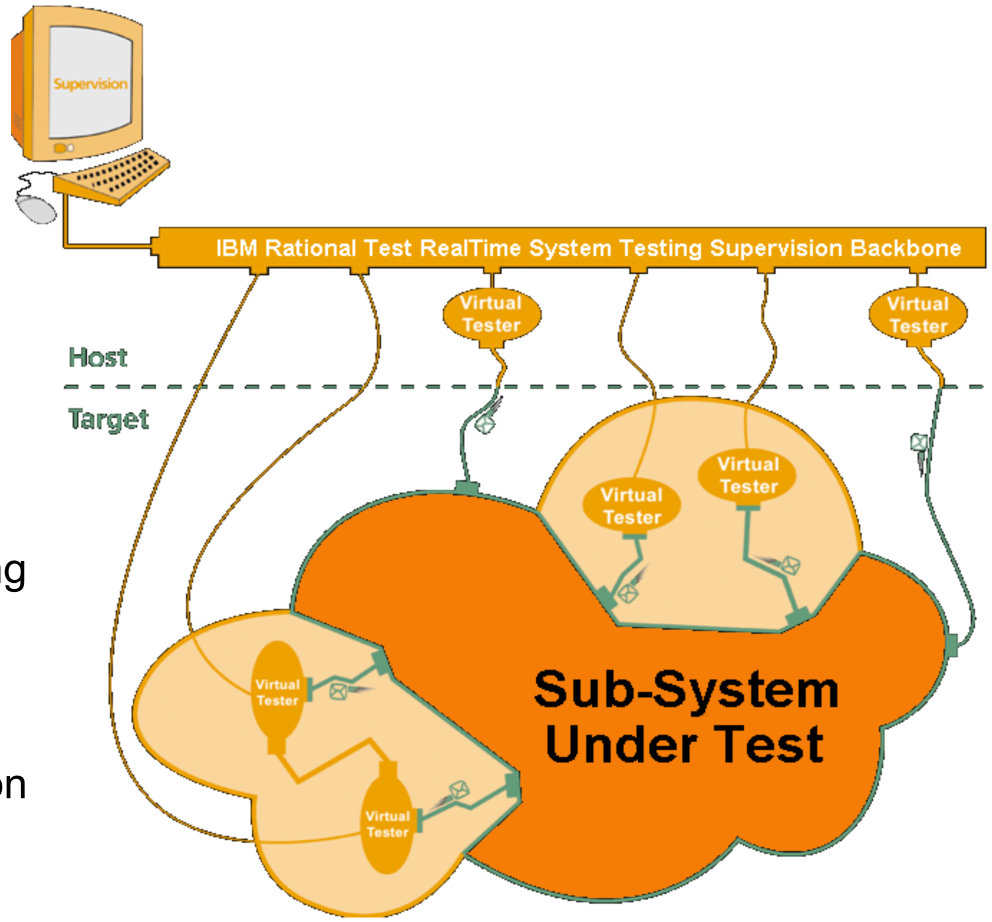


System Testing for C: *Message-Passing Testing*

- **Integration and validation testing from:**
  - Single thread … *up to*
  - Task(s) … *up to*
  - Node(s) … *up to*
  - Large networked system
- **Functional, load, and performance testing via message-passing API**
- **Powerful scripting language**
- **Detailed reporting**
- **Regression testing**
- **Works with Test RealTime runtime analysis features**
  - Memory and Performance Profiling
  - Code Coverage and Runtime Tracing

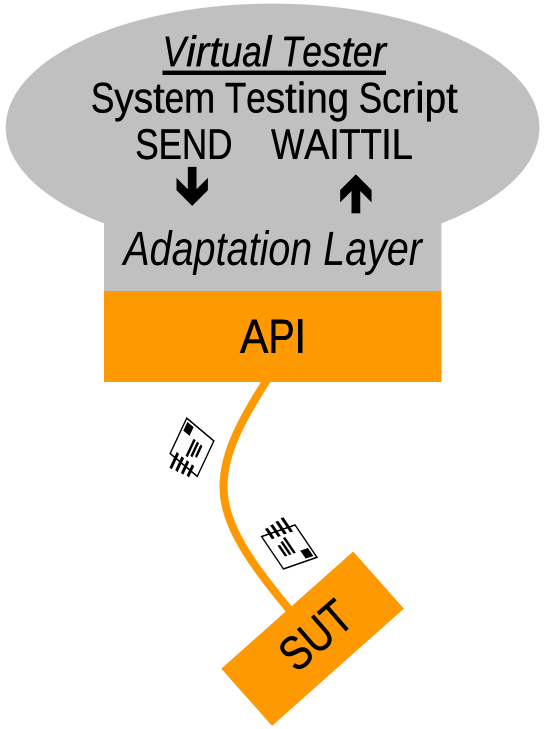
Test Harness Architecture and Responsibilities

1. **Virtual Testers**
   - Simulates external systems
   - Stubs internal actors
2. **Each Virtual Tester**
   - Sends events to the SUT
   - Controls the event flow
   - Checks event data and timing
   - Can be duplicated for load testing
3. **System Testing supervisor**
   - Monitoring services for virtual tester distribution, communication and synchronization



System Testing for C: *Message Adaptation Layer*

- **Full support of any communication interface**
  1. Adaptation layer
     - Built with C procedures to send and receive events (C-structs)
     - Symbolic management of events
     - Enables test script independence of the messaging API
  2. Message API
     - Part of the system-under-test
     - Provided by a communication card
     - Defines how to send and receive events



System Testing for C: *Test Script Structuring*

```
INITIALIZATION init_proc()
TERMINATION end_proc()
EXCEPTION recover_proc()

SCENARIO main

  SCENARIO test_case1

  END SCENARIO -- test_case1

  SCENARIO test_case2

     INSTANCE Virtual_Tester1

     END INSTANCE – Virtual_Tester1

     INSTANCE Virtual_Tester2

     END INSTANCE – Virtual_Tester2

  END SCENARIO -- test_case2

END SCENARIO -- main
```

Preamble, Postamble and Error recovery blocks

Test script composed of SCENARIOs and sub-SCENARIOs …

A scenario can be split into INSTANCE blocks to define asynchronous behaviors (Virtual Testers)

System Testing for C: *Test Script Behavior*

Loops

Synchronizations between Virtual Testers

IF statements

Init of outgoing Events with high level instructions

Send an Event via a defined communication channel

Set up a timer

Definition of constraints on incoming Events

Wait for complex conditions

Call to external C code

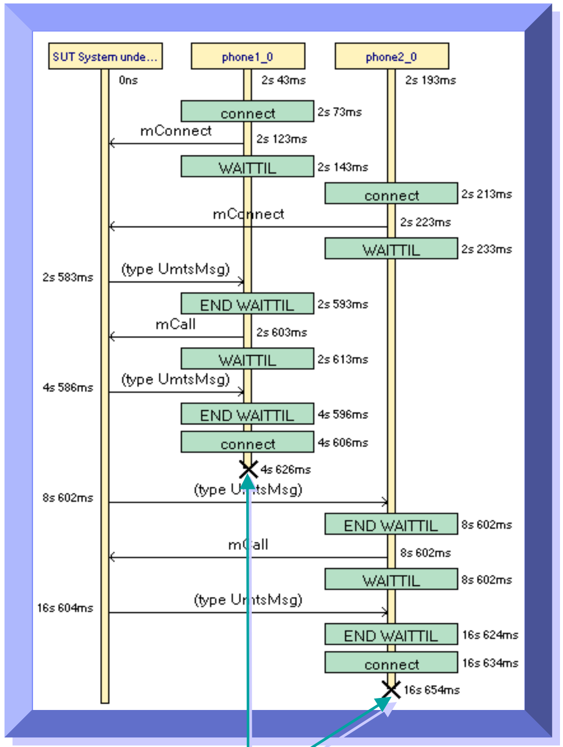
```
CHANNEL MY_COMMTYPE:mylink

SCENARIO example

  WHILE (TIME(mytimer) < 100)

     RENDEZVOUS start_example

     IF (sync == 0) THEN

      VAR creq, INIT={send=>…,neg=>{opt=> …, }}

      SEND (mylink, creq)

     END IF

     TIMER mytimer

     DEF_MESSAGE cresp, EV= { … }

     WAITTIL (MATCHED(cresp)||MATCHING(cack),WTIME>15)

     CALL myexternal_func()

  END WHILE

END SCENARIO -- example
```

System Testing for C: *Reporting*

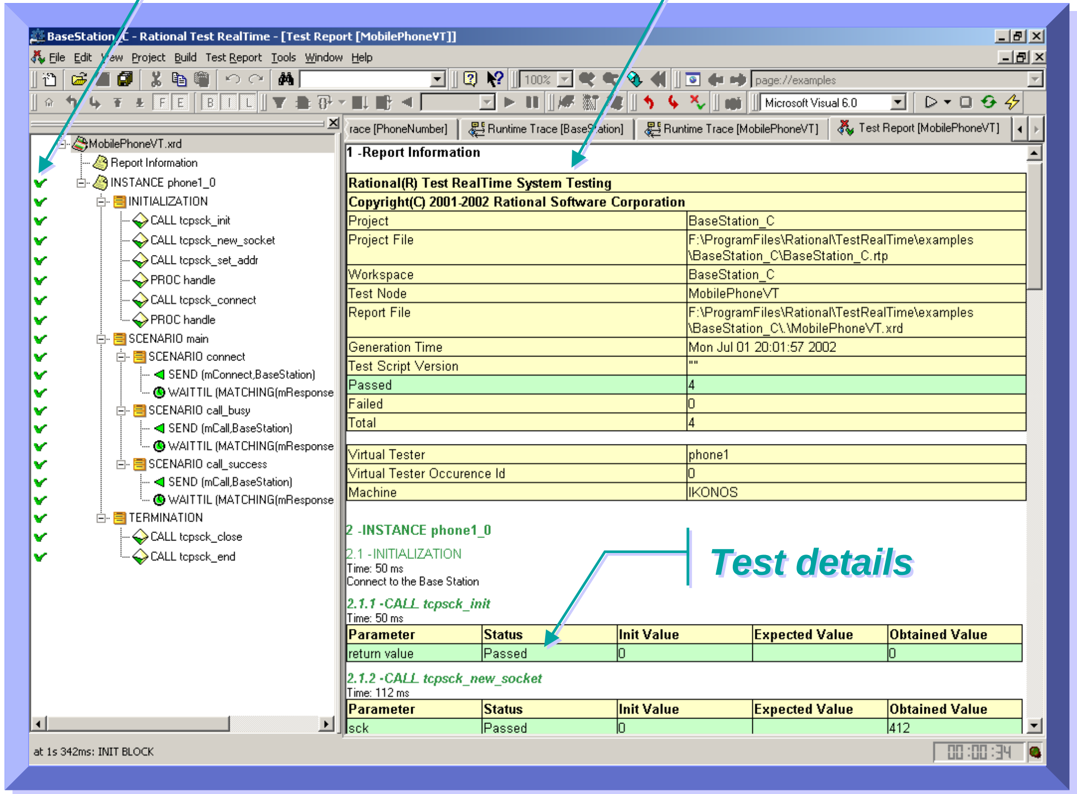
- **Dynamically jump between sequence diagram, test report, and source code**
- **Export to HTML**

Results Summary

Test Summary



Virtual Testers



1 -Report Information

Rational(R) Test RealTime System Testing	
Copyright(C) 2001-2002 Rational Software Corporation	
Project	BaseStation_C
Project File	F:\ProgramFiles\Rational\TestRealTime\examples \BaseStation_C\BaseStation_C.rtp
Workspace	BaseStation_C
Test Node	MobilePhoneVT
Report File	F:\ProgramFiles\Rational\TestRealTime\examples \BaseStation_C\.\MobilePhoneVT.xrd
Generation Time	Mon Jul 01 20:01:57 2002
Test Script Version	""
Passed	4
Failed	0
Total	4

Virtual Tester	phone1
Virtual Tester Occurence Id	0
Machine	IKONOS

2 -INSTANCE phone1_0

2.1 -INITIALIZATION
Time: 50 ms
Connect to the Base Station

2.1.1 -CALL tcpsck_init
Time: 50 ms

Parameter	Status	Init Value	Expected Value	Obtained Value
return value	Passed	0		0

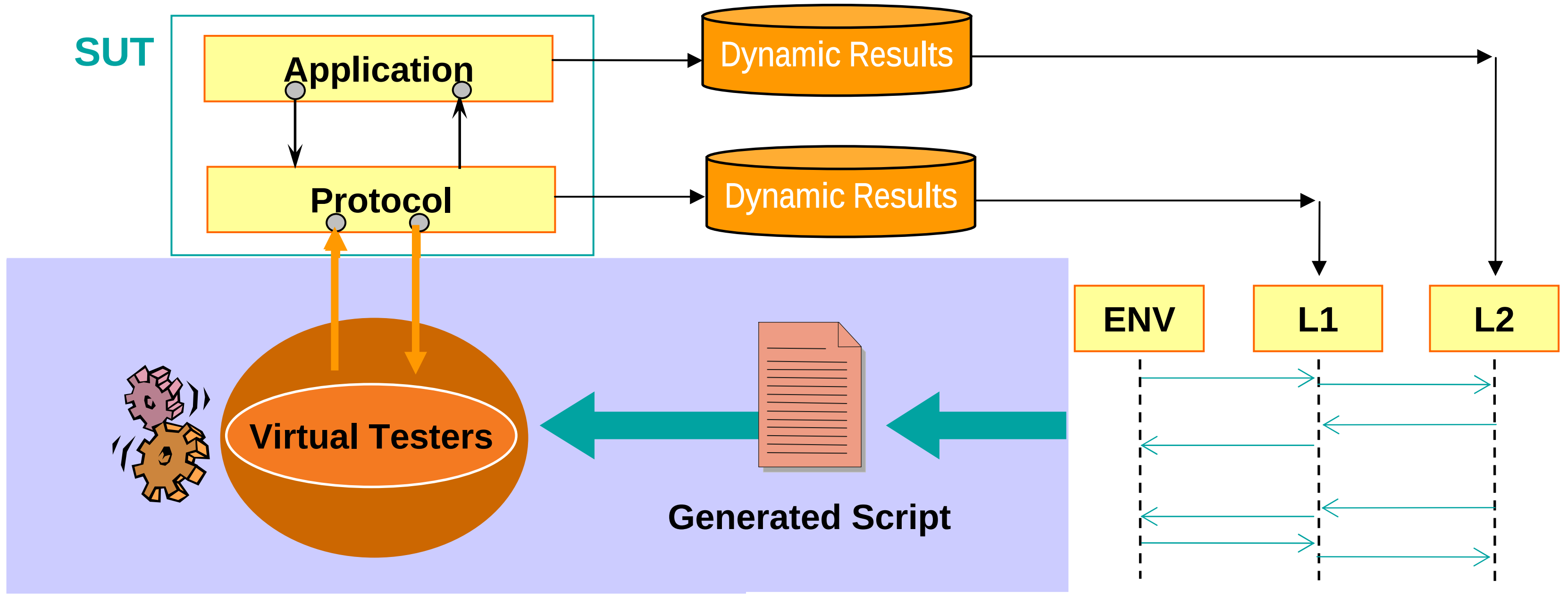
2.1.2 -CALL tcpsck_new_socket
Time: 112 ms

Parameter	Status	Init Value	Expected Value	Obtained Value
sck	Passed	0		412

Test details

Probe Feature: *Show Exchanges and Replay Them*

- **Deployment Process**
  - Define probes location in applications/simulators for sent and received messages
  - Execute normally using simulators/Hardware/GUIs to create/populate the test script
  - After execution, a UML/SD report shows all exchanged messages or events and generates the script to replay



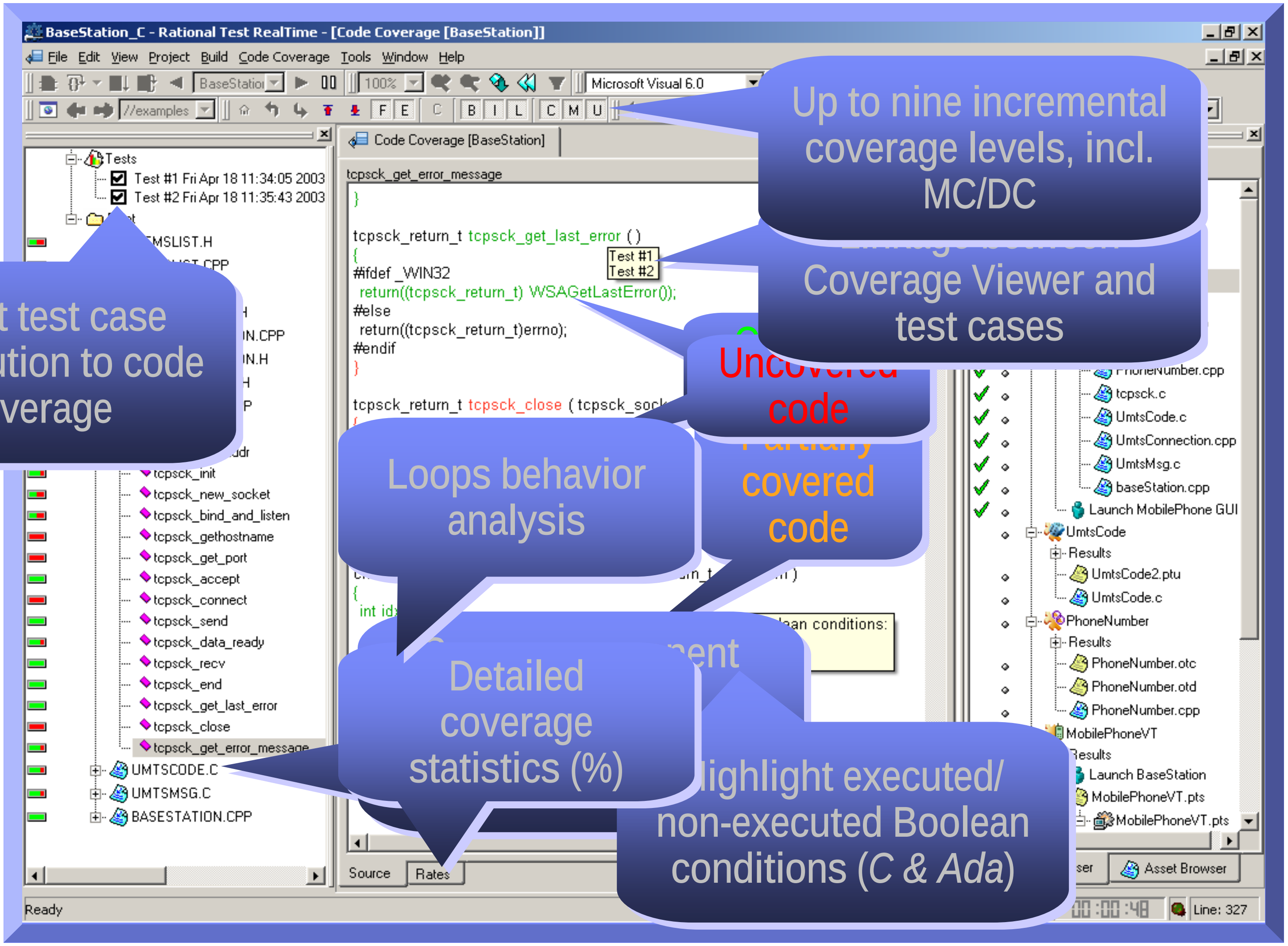
IBM Rational Test RealTime

Runtime Analysis Features Presentation

Benefits of Runtime Analysis Features

- **Get insights into program execution**
  - See how various components of a running application may affect each other during execution
- **Get an overall picture of an application's execution behavior over a period of time**
- **Detect hard-to-find problems**
  - Memory leaks
  - Performance bottlenecks
  - Unused and/or untested code
- **Directly correlate all analysis results to**
  - Test cases and code

Runtime Analysis Features: *Code Coverage*



BaseStation_C - Rational Test RealTime - [Code Coverage [BaseStation]]

Up to nine incremental coverage levels, incl. MC/DC

Linkage between Coverage Viewer and test cases

Select test case contribution to code coverage

Uncovered code

Partially covered code

Loops behavior analysis

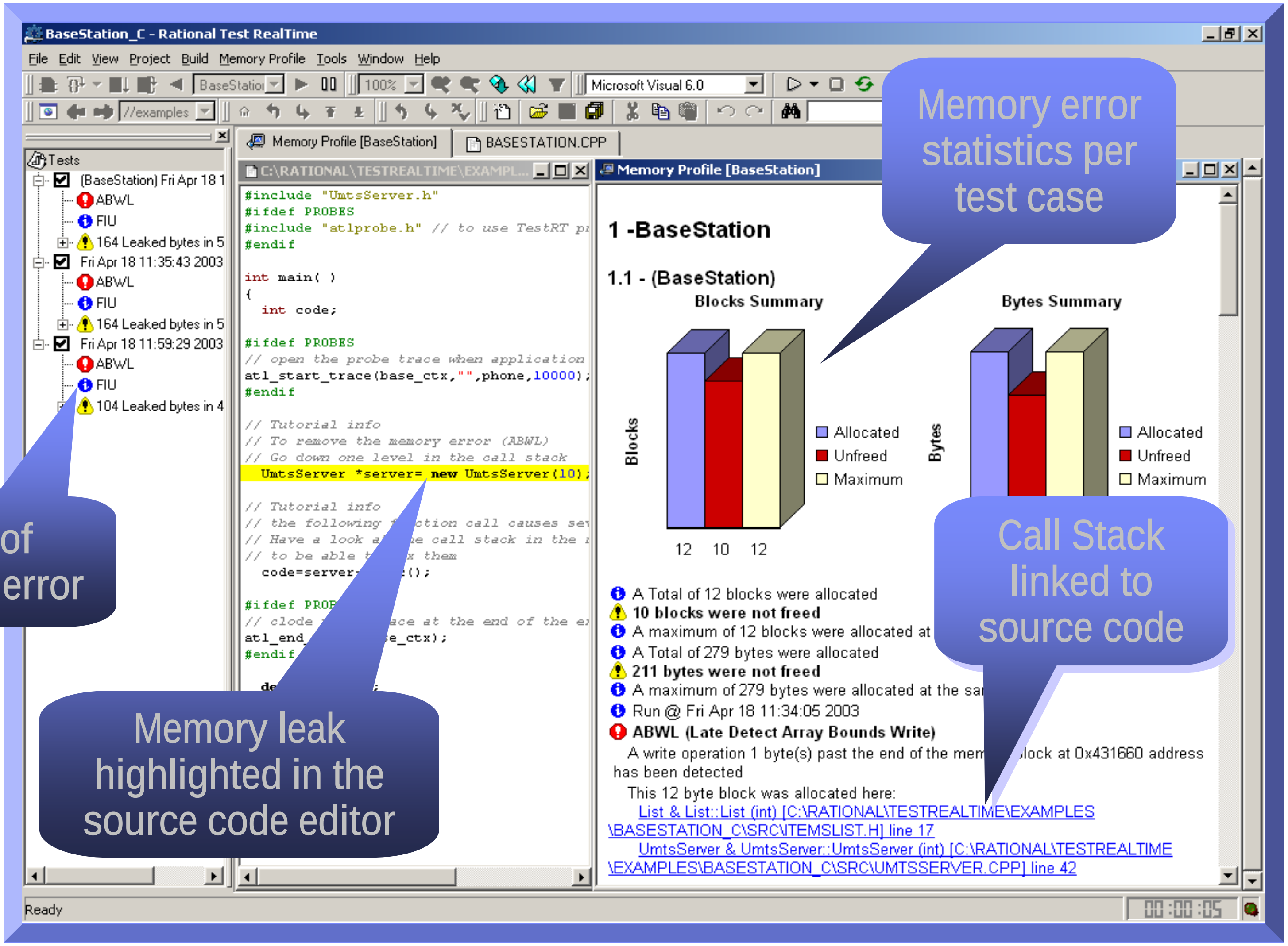
Comment

Detailed coverage statistics (%)

Highlight executed/ non-executed Boolean conditions (C & *Ada*)

```
tcpsck_get_error_message
}

tcpsck_return_t tcpsck_get_last_error ( )
{
#ifdef _WIN32
 return((tcpsck_return_t) WSAGetLastError());
#else
 return((tcpsck_return_t)errno);
#endif
}

tcpsck_return_t tcpsck_close ( tcpsck_sock
```

Runtime Analysis Features: *Memory Profiling*



C:\RATIONAL\TESTREALTIME\EXAMPL...

```
#include "UmtsServer.h"
#ifdef PROBES
#include "atlprobe.h" // to use TestRT pr
#endif

int main( )
{
  int code;

#ifdef PROBES
// open the probe trace when application
atl_start_trace(base_ctx,"",phone,10000);
#endif

// Tutorial info
// To remove the memory error (ABWL)
// Go down one level in the call stack
  UmtsServer *server= new UmtsServer(10);

// Tutorial info
// the following f   ction call causes sev
// Have a look a   he call stack in the r
// to be able t   x them
  code=server-   ();

#ifdef PROB
// clode       ace at the end of the ex
atl_end        e_ctx);
#endif

  de         ;
```

Memory Profile [BaseStation]

1 -BaseStation

1.1 - (BaseStation)

A Total of 12 blocks were allocated
10 blocks were not freed
A maximum of 12 blocks were allocated at
A Total of 279 bytes were allocated
211 bytes were not freed
A maximum of 279 bytes were allocated at the sa
Run @ Fri Apr 18 11:34:05 2003
ABWL (Late Detect Array Bounds Write)
A write operation 1 byte(s) past the end of the mem   block at 0x431660 address has been detected
This 12 byte block was allocated here:
List & List::List (int) [C:\RATIONAL\TESTREALTIME\EXAMPLES\BASESTATION_C\SRC\ITEMSLIST.H] line 17
UmtsServer & UmtsServer::UmtsServer (int) [C:\RATIONAL\TESTREALTIME\EXAMPLES\BASESTATION_C\SRC\UMTSSERVER.CPP] line 42

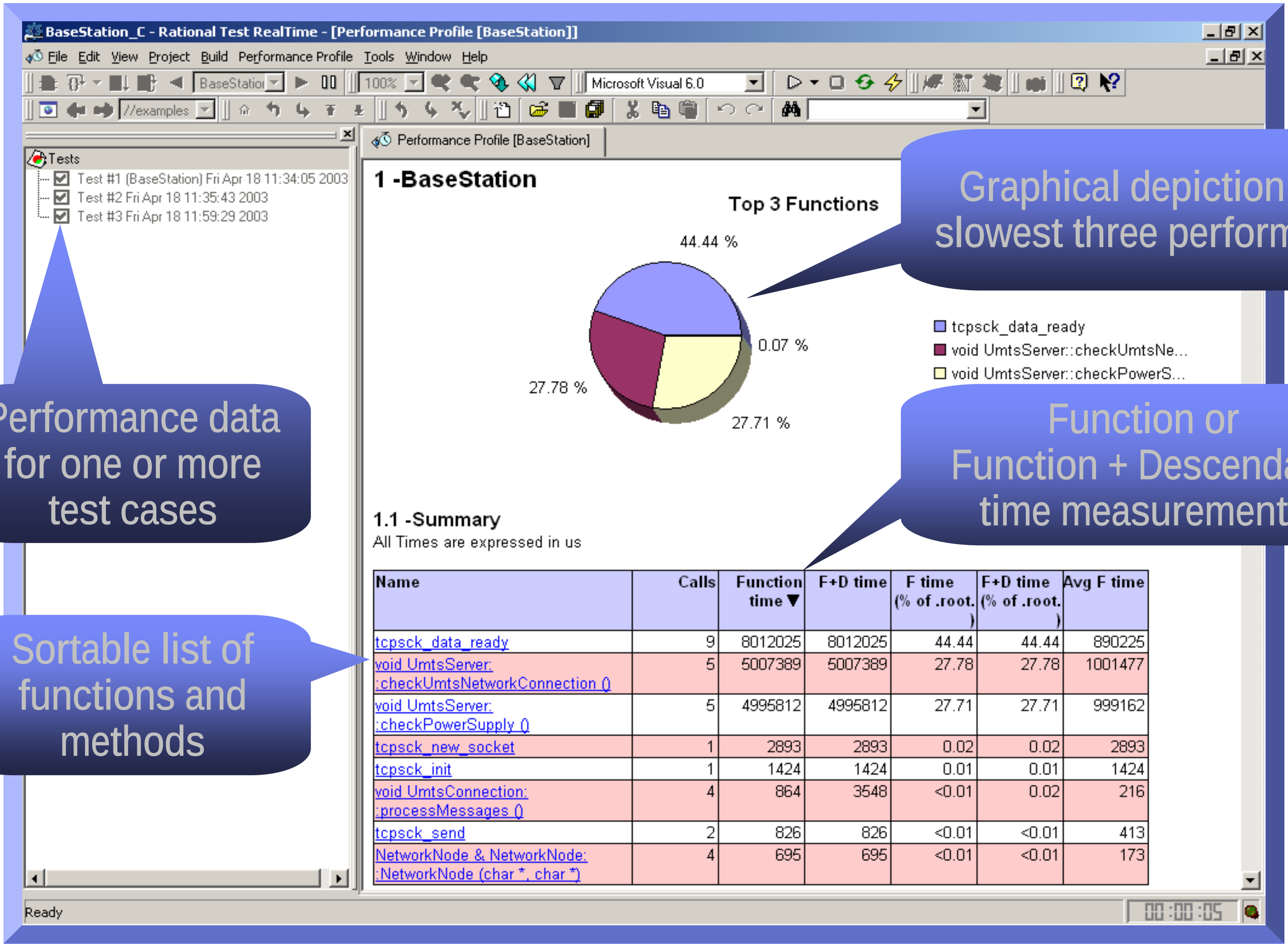
Memory error statistics per test case

Type of memory error

Memory leak highlighted in the source code editor

Call Stack linked to source code

Runtime Analysis Features: *Performance Profiling*


BaseStation_C - Rational Test RealTime - [Performance Profile [BaseStation]]
File Edit View Project Build Performance Profile Tools Window Help
BaseStation
100%
Microsoft Visual 6.0
//examples
Performance Profile [BaseStation]
Top 3 Functions
44.44 %
0.07 %
27.78 %
27.71 %
Ready
00:00:05

Tests
- ☑ Test #1 (BaseStation) Fri Apr 18 11:34:05 2003
- ☑ Test #2 Fri Apr 18 11:35:43 2003
- ☑ Test #3 Fri Apr 18 11:59:29 2003

1 -BaseStation

- 🟦 tcpsck_data_ready
- 🟥 void UmtsServer::checkUmtsNe...
- ☐ void UmtsServer::checkPowerS...

1.1 -Summary

All Times are expressed in us

Name	Calls	Function time ▼	F+D time	F time (% of .root.)	F+D time (% of .root.)	Avg F time
tcpsck_data_ready	9	8012025	8012025	44.44	44.44	890225
void UmtsServer::checkUmtsNetworkConnection ()	5	5007389	5007389	27.78	27.78	1001477
void UmtsServer::checkPowerSupply ()	5	4995812	4995812	27.71	27.71	999162
tcpsck_new_socket	1	2893	2893	0.02	0.02	2893
tcpsck_init	1	1424	1424	0.01	0.01	1424
void UmtsConnection::processMessages ()	4	864	3548	<0.01	0.02	216
tcpsck_send	2	826	826	<0.01	<0.01	413
NetworkNode & NetworkNode::NetworkNode (char *, char *)	4	695	695	<0.01	<0.01	173

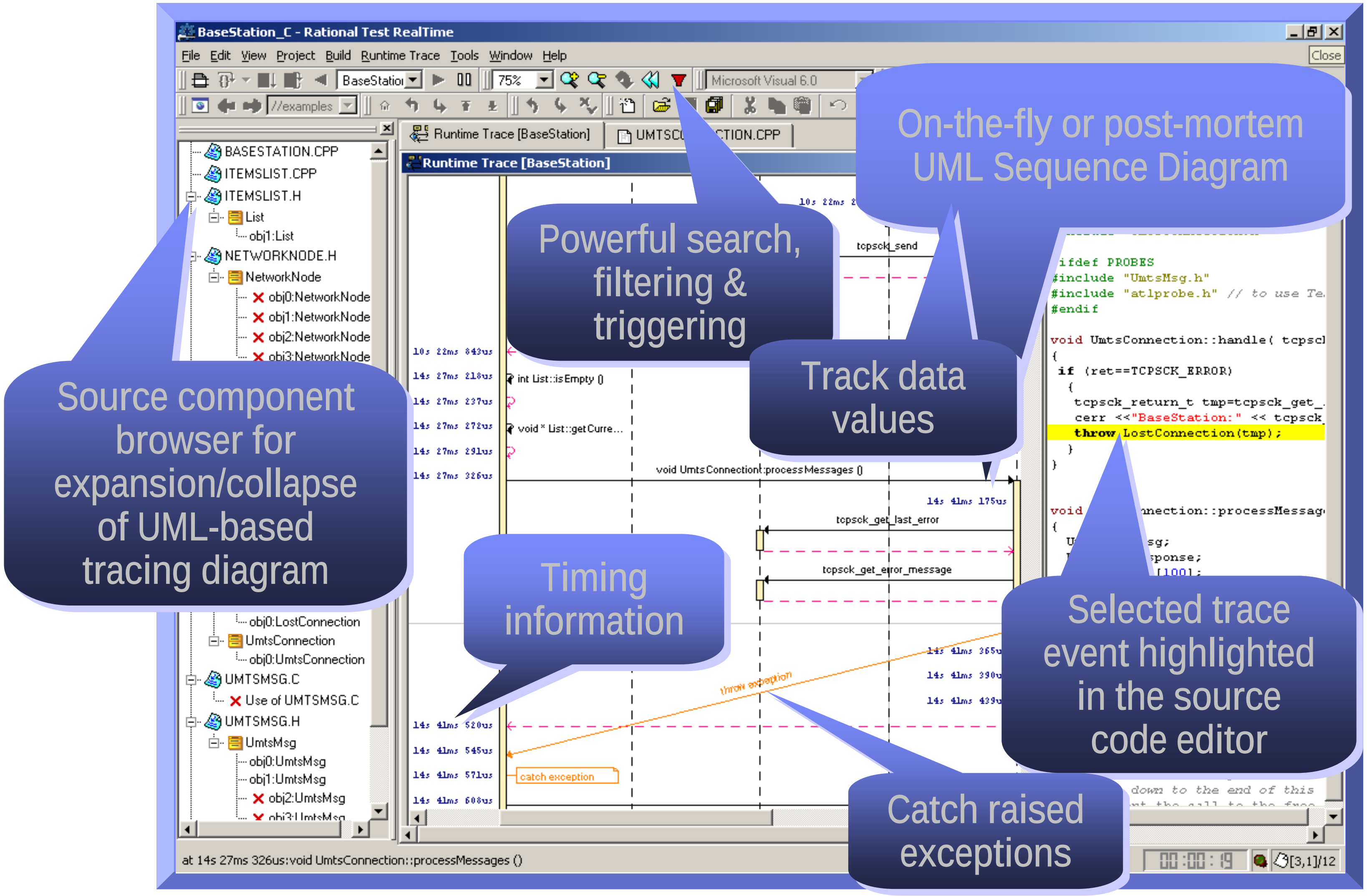
Graphical depiction of slowest three performers

Performance data for one or more test cases

Function or Function + Descendant time measurements

Sortable list of functions and methods

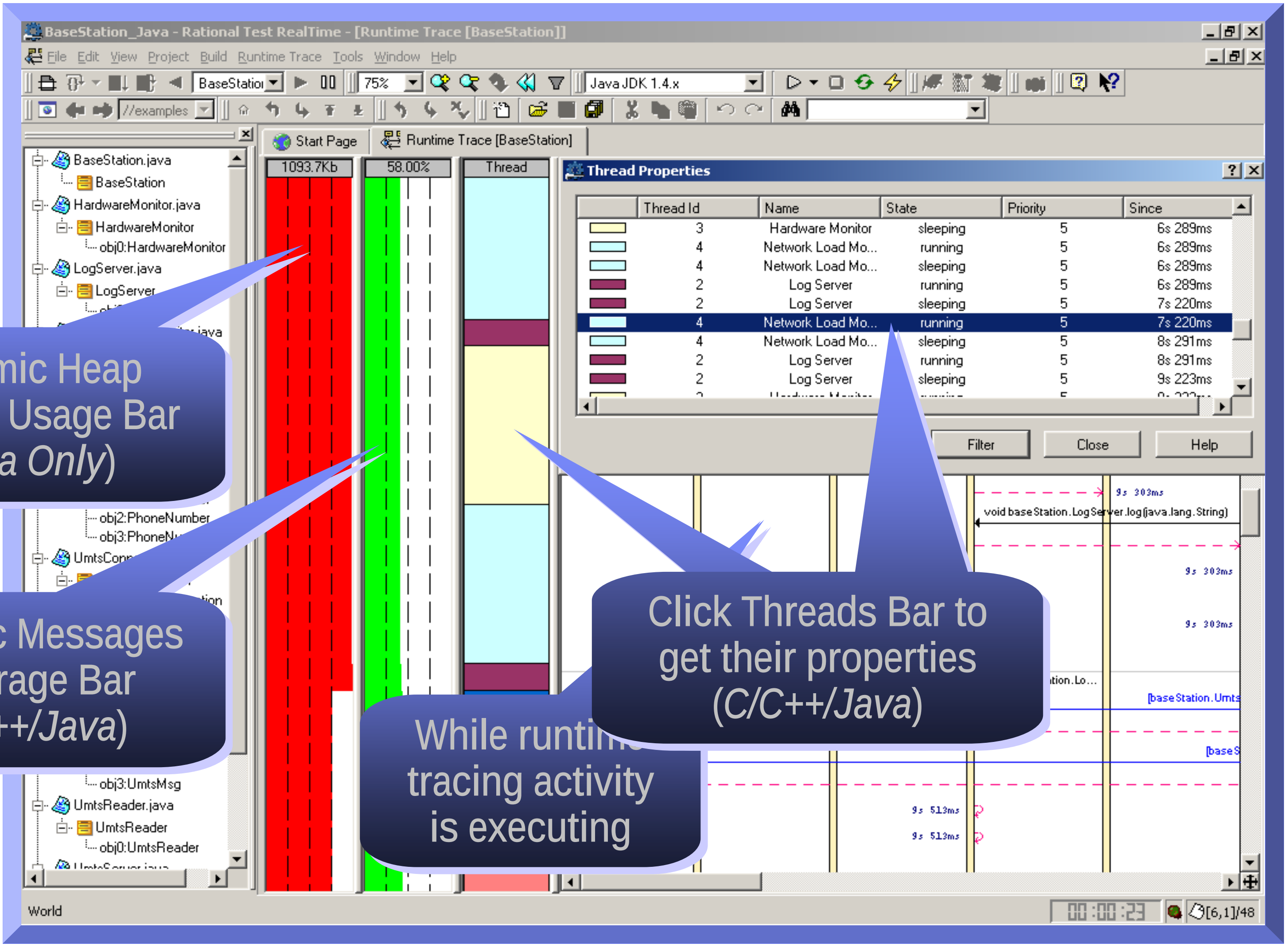
Runtime Analysis Features: *Runtime Tracing*



```
#ifdef PROBES
#include "UmtsMsg.h"
#include "atlprobe.h"  // to use Te
#endif

void UmtsConnection::handle( tcpsc
{
  if (ret==TCPSCK_ERROR)
  {
    tcpsck_return_t tmp=tcpsck_get_
    cerr <<"BaseStation:" << tcpsck
    throw LostConnection(tmp);
  }
}

void      nection::processMessag
{
```

at 14s 27ms 326us:void UmtsConnection::processMessages ()

Runtime Analysis Features: *Runtime Tracing Bars*


BaseStation_Java - Rational Test RealTime - [Runtime Trace [BaseStation]]
File Edit View Project Build Runtime Trace Tools Window Help
Start Page
Runtime Trace [BaseStation]
1093.7Kb
58.00%
Thread
Thread Properties
Filter
Close
Help
World
00:00:23
[6,1]/48

Dynamic Heap Memory Usage Bar (*Java Only*)

Dynamic Messages Coverage Bar (*C/C++/Java*)

While runtime tracing activity is executing

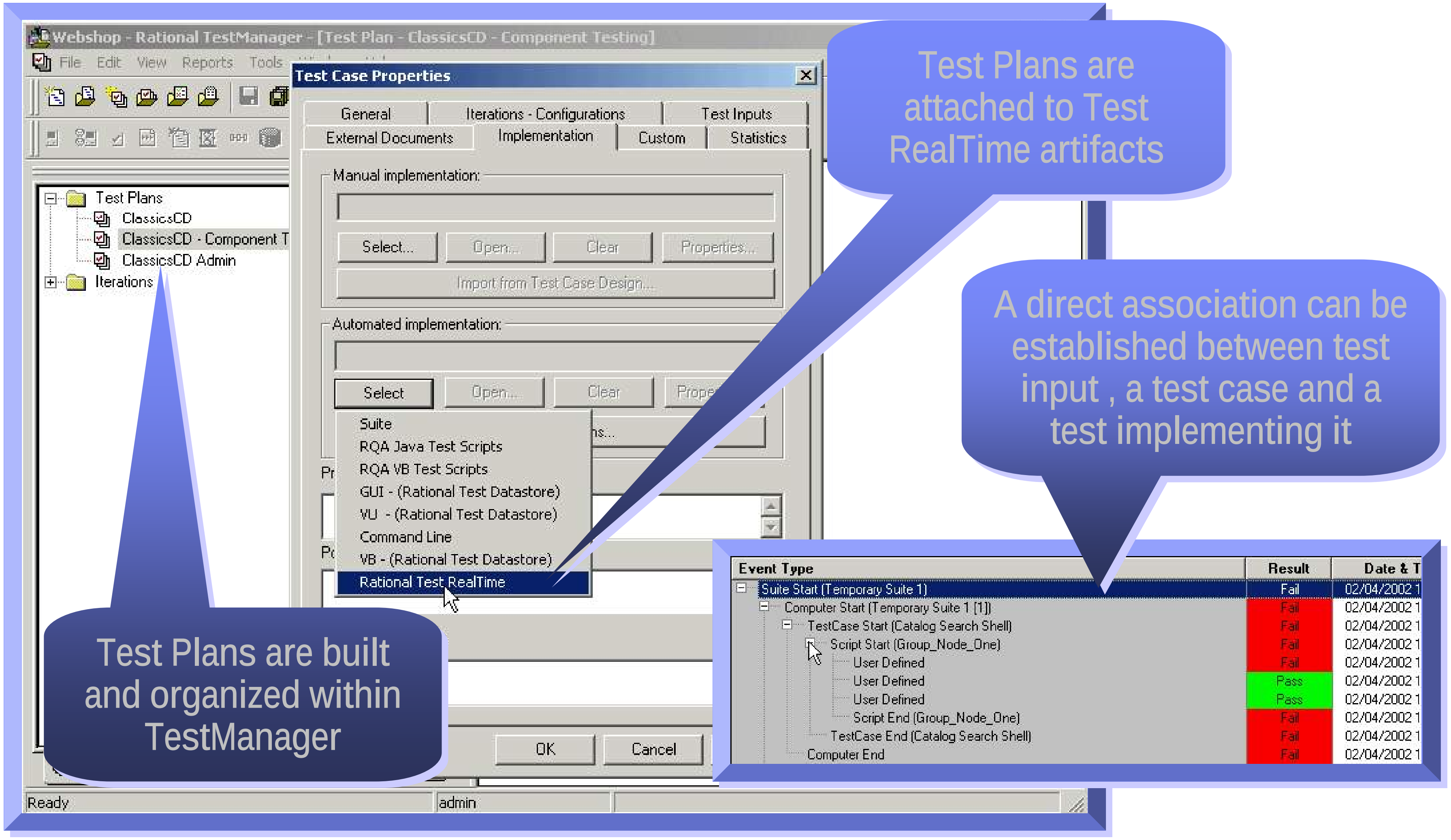
Click Threads Bar to get their properties (*C/C++/Java*)

Thread Properties

	Thread Id	Name	State	Priority	Since
🟨	3	Hardware Monitor	sleeping	5	6s 289ms
🟦	4	Network Load Mo...	running	5	6s 289ms
🟦	4	Network Load Mo...	sleeping	5	6s 289ms
🟥	2	Log Server	running	5	6s 289ms
🟥	2	Log Server	sleeping	5	7s 220ms
🟦	4	Network Load Mo...	running	5	7s 220ms
🟦	4	Network Load Mo...	sleeping	5	8s 291ms
🟥	2	Log Server	running	5	8s 291ms
🟥	2	Log Server	sleeping	5	9s 223ms

void baseStation.LogServer.log(java.lang.String)

Test RealTime: *Integration with Test Manager*



Event Type	Result	Date & T
Suite Start (Temporary Suite 1)	Fail	02/04/2002 1
Computer Start (Temporary Suite 1 [1])	Fail	02/04/2002 1
TestCase Start (Catalog Search Shell)	Fail	02/04/2002 1
Script Start (Group_Node_One)	Fail	02/04/2002 1
User Defined	Fail	02/04/2002 1
User Defined	Pass	02/04/2002 1
User Defined	Pass	02/04/2002 1
Script End (Group_Node_One)	Fail	02/04/2002 1
TestCase End (Catalog Search Shell)	Fail	02/04/2002 1
Computer End	Fail	02/04/2002 1

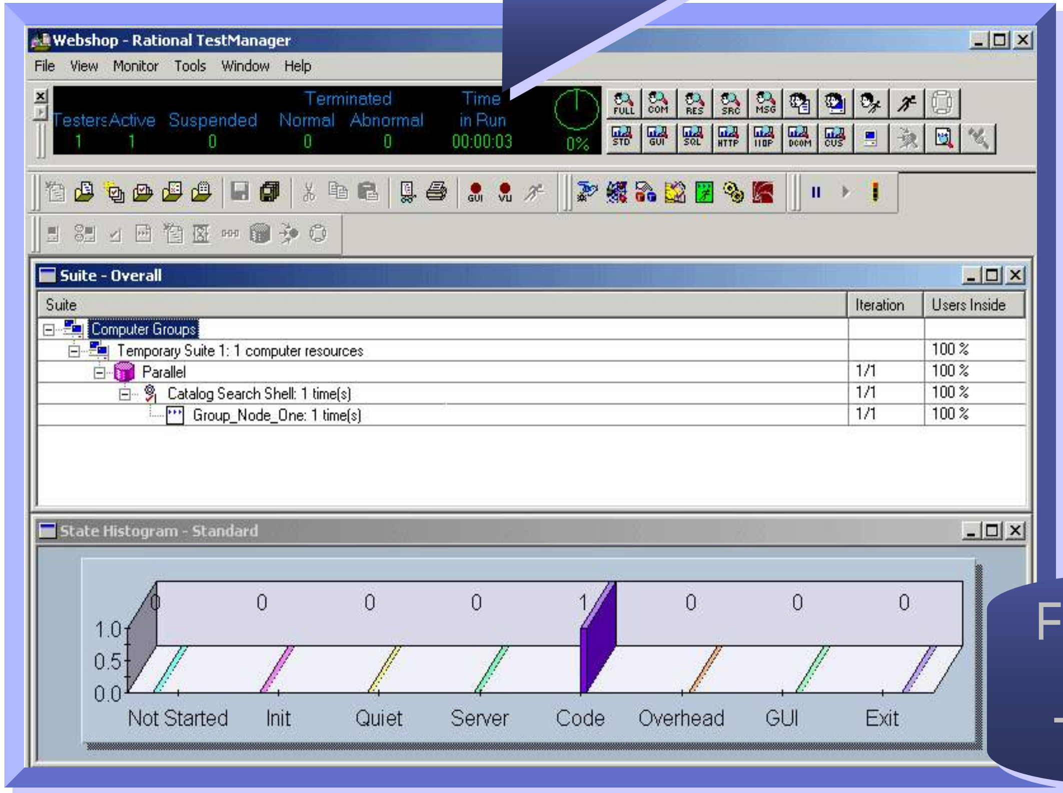
Test Plans are attached to Test RealTime artifacts

A direct association can be established between test input , a test case and a test implementing it

Test Plans are built and organized within TestManager

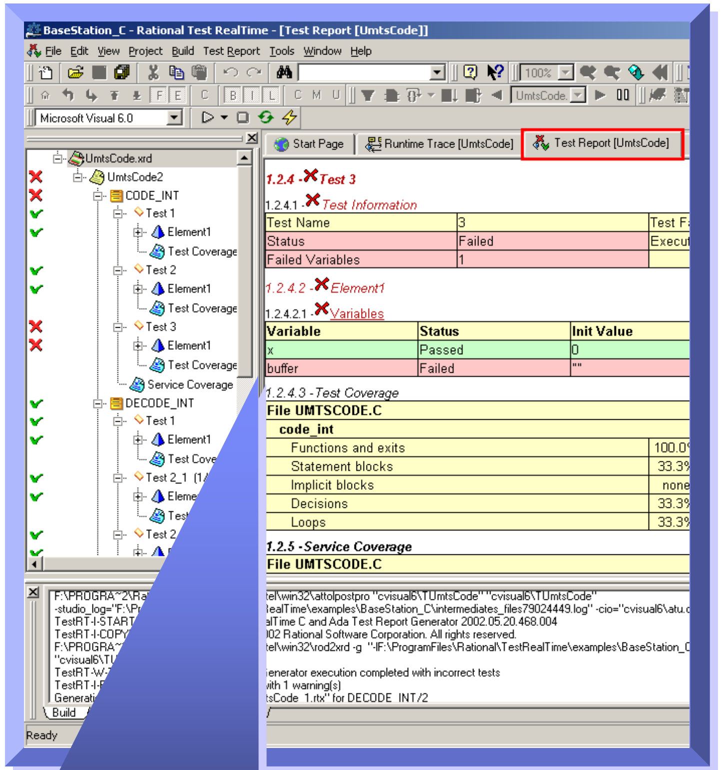
Test RealTime: *Integration with Test Manager*

Run and monitor the tests built with Test RealTime from TestManager



Suite	Iteration	Users Inside
Computer Groups		
Temporary Suite 1: 1 computer resources		100 %
Parallel	1/1	100 %
Catalog Search Shell: 1 time(s)	1/1	100 %
Group_Node_One: 1 time(s)	1/1	100 %

	Not Started	Init	Quiet	Server	Code	Overhead	GUI	Exit
	0	0	0	0	1	0	0	0



1.2.4 - Test 3

1.2.4.1 - *Test Information*

Test Name	3	Test F
Status	Failed	Execut
Failed Variables	1	

1.2.4.2 - Element1

1.2.4.2.1 - Variables

Variable	Status	Init Value
x	Passed	0
buffer	Failed	""

1.2.4.3 - Test Coverage

File UMTSCODE.C	
code_int	
Functions and exits	100.0
Statement blocks	33.3
Implicit blocks	none
Decisions	33.3
Loops	33.3

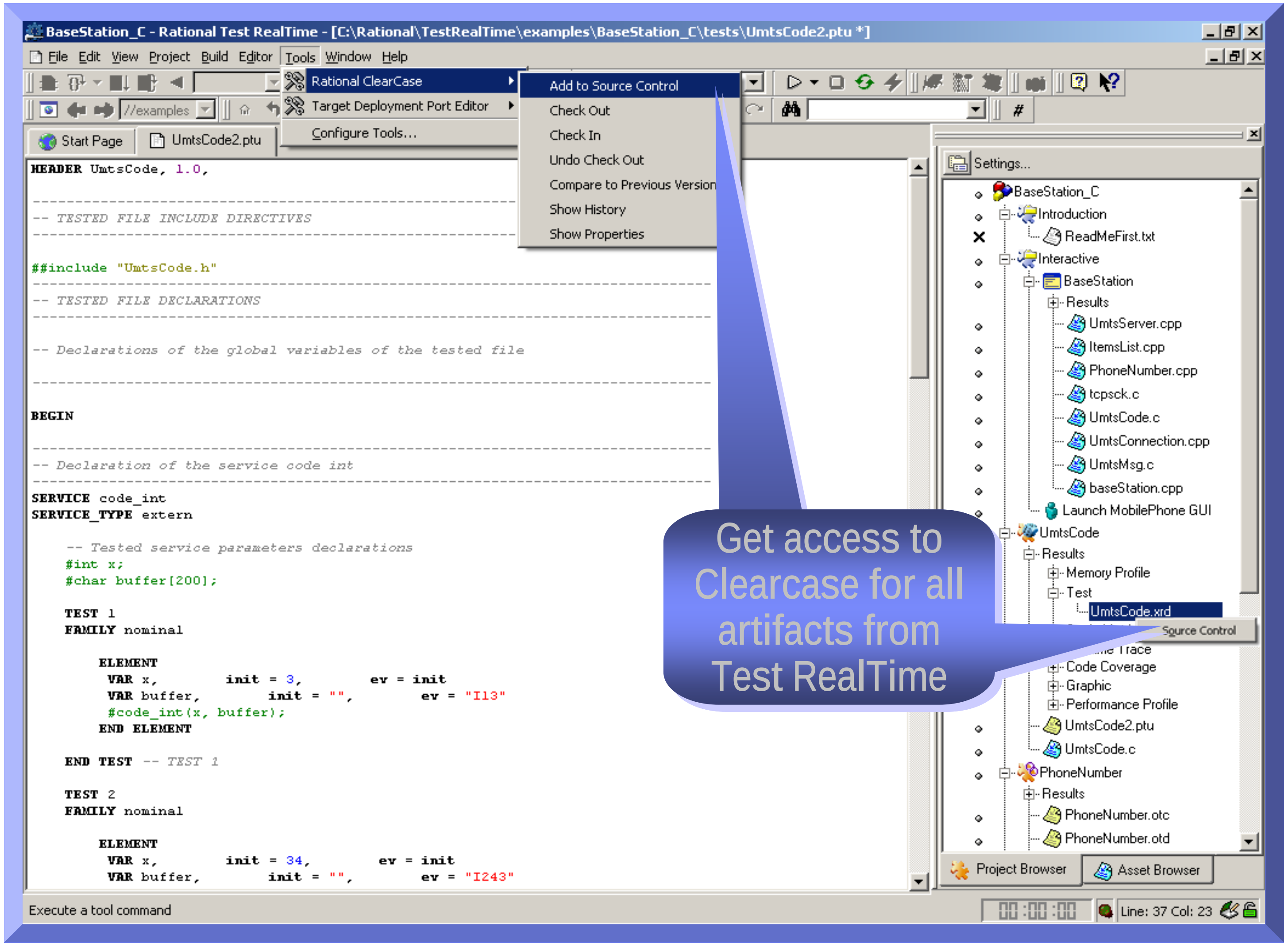
1.2.5 - Service Coverage

File UMTSCODE.C

```
F:\PROGRA~2\Ra   tel\win32\attolpostpro "cvisual6\TUmtsCode" "cvisual6\TUmtsCode"
-studio_log="F:\Pr   RealTime\examples\BaseStation_C\intermediates_files79024449.log" -cio="cvisual6\atu.c
TestRT-I-START   alTime C and Ada Test Report Generator 2002.05.20.468.004
TestRT-I-COPY   002 Rational Software Corporation. All rights reserved.
F:\PROGRA~   tel\win32\rod2xrd -g "-IF:\ProgramFiles\Rational\TestRealTime\examples\BaseStation_C
"cvisual6\TU
TestRT-W-   ienerator execution completed with incorrect tests
TestRT-I-F   with 1 warning(s)
Generati   tsCode 1.rtx" for DECODE_INT/2
```

Finally, inspect from Test RealTime GUI the detailed results when TestManager gives a failed status

Test RealTime: *Integration with ClearCase*



```
HEADER UmtsCode, 1.0,

------------------------------------------------------------
-- TESTED FILE INCLUDE DIRECTIVES
------------------------------------------------------------

##include "UmtsCode.h"
------------------------------------------------------------
-- TESTED FILE DECLARATIONS
------------------------------------------------------------

-- Declarations of the global variables of the tested file

------------------------------------------------------------

BEGIN

------------------------------------------------------------
-- Declaration of the service code int
------------------------------------------------------------
SERVICE code_int
SERVICE_TYPE extern

    -- Tested service parameters declarations
    #int x;
    #char buffer[200];

    TEST 1
    FAMILY nominal

        ELEMENT
         VAR x,          init = 3,        ev = init
         VAR buffer,         init = "",         ev = "I13"
         #code_int(x, buffer);
        END ELEMENT

    END TEST -- TEST 1

    TEST 2
    FAMILY nominal

        ELEMENT
         VAR x,          init = 34,        ev = init
         VAR buffer,         init = "",         ev = "I243"
```

Get access to Clearcase for all artifacts from Test RealTime

Test RealTime 7.5 features

- Integrated with new Rational Quality Manager Express Edition and Rational Quality Manager Standard Edition
- Integrated with new Rational Software Architect Standard Edition v7.5
- Updated support for Eclipse v3.4 integration
- New integration with Telelogic Rhapsody v7.4 and TestConductor (7.5)
- Updated integration with Wind River Workbench 3.0
- New out-of-the-box support for Symbian OS
- New support of IPv6 network infrastructure

Thank
You