

IBM Software Group

# P17 System Testing

## Monday, September 24, 2007

### Module 8 : IBM Rational Testing Solutions

**Rational.** software

A decorative horizontal bar with a cyan background, featuring a series of colored squares (cyan, green, yellow, red, purple) followed by several icons: a crane, a circular arrow, a person's head, a grid, a globe, a starburst, and a square with four arrows pointing outwards.

Marty Swafford

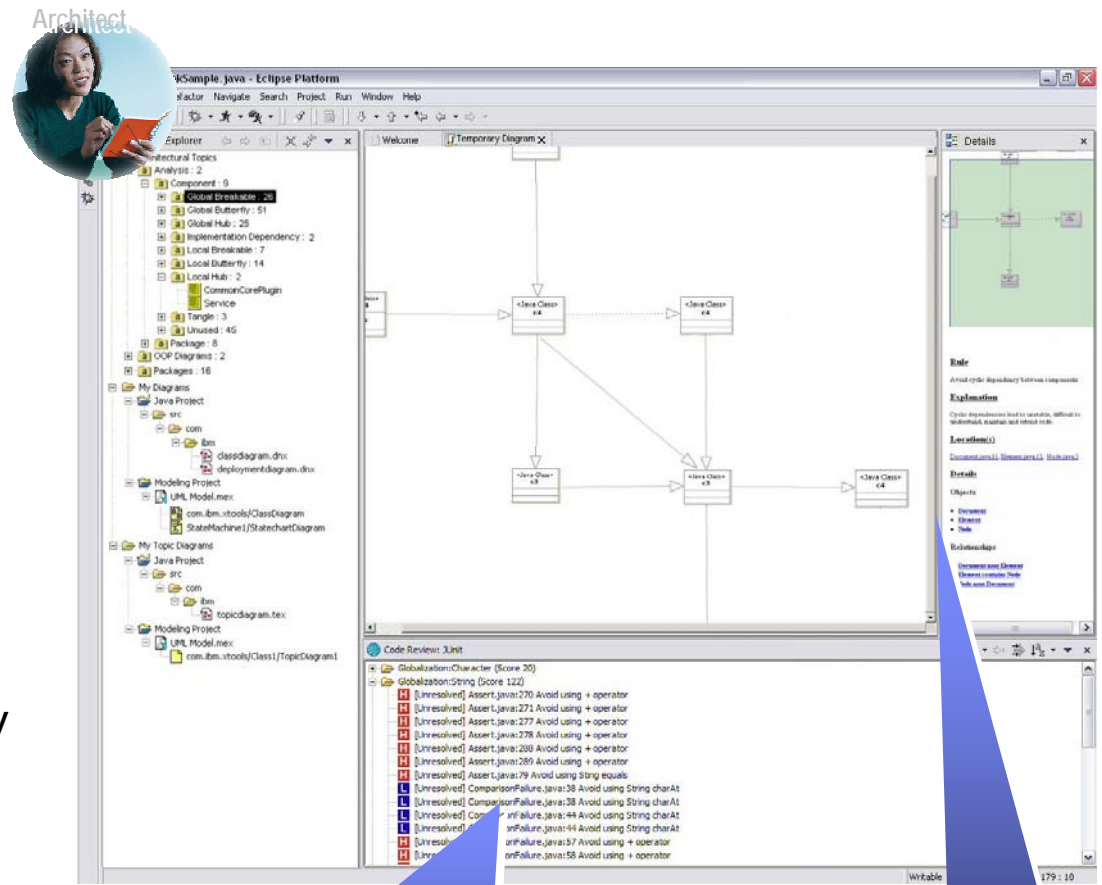
IBM Rational Software

IBM Certified Solution Designer - Rational Manual Tester, Rational Performance Tester, Rational Functional Tester for Java

[mswaffor@us.ibm.com](mailto:mswaffor@us.ibm.com)

## Design: Architecture Discovery, Analysis and Control

- Architect discovers the architecture as well as control the changes made during implementation**
- Automatic diagramming of significant structures (e.g. inheritance trees, class internals, package structure)
- Insight into the architecture through Diagram Browsing capability
- Validation of structural quality through structural antipattern rules
- Definition of custom architectural rules ensure design integrity



**Anti-pattern detection by category/files/severity**

**Anti-pattern visualization**

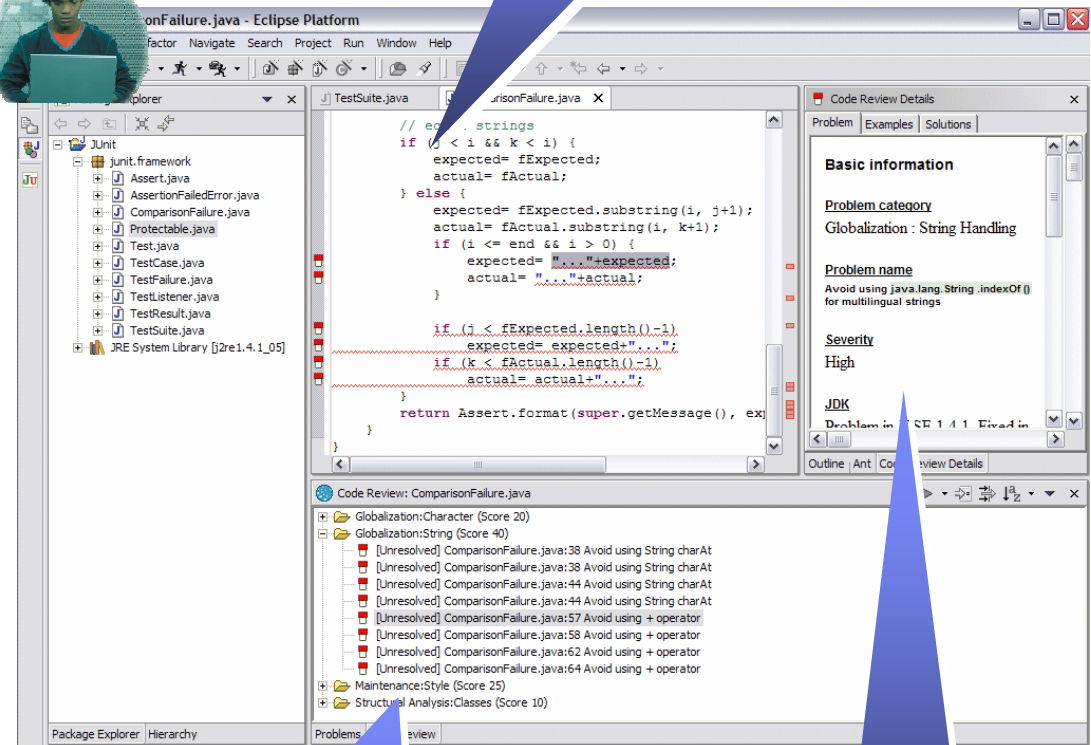


## Construction: Code Review

- **Developer implements the code and receives guidance for J2SE and J2EE best practices**
- Validation of code through sets of coding rules (200+) in various categories:
  - ▶ J2EE Best Practices
  - ▶ J2SE Best Practices
  - ▶ Performance
  - ▶ Private API
  - ▶ Coding style
  - ▶ Globalization
  - ▶ Naming conventions
  - ▶ Testability & Testing completeness
  - ▶ Design Principles
  - ▶ Structural Anti-patterns
- Users can define their own rules
- Eclipse's Quick Fix available
- Includes IBM Research IP



**Rule violation underlined in the code editor**



**Rule violations by category/files/severity**

**Why / Example / How to fix**



## Component Test: Verify component functionality

- **Developer automatically generates fully functional unit and API tests for Java classes, EJBs and Web Services (incl. .Net)**
- Provides testing guidance
  - ▶ What to test first based on metrics?
  - ▶ What test patterns apply to code?
- Automated test driver and stub generation
  - ▶ Includes both control flow and data
- Based on the JUnit framework
- Data-driven testing
- No need to write code!
- Integrated with debugging and line level code coverage

The screenshot displays the IBM WebSphere Studio Application Developer interface. The main editor shows the following Java code for a test method:

```

testMethods() throws Throwable {
    BidHelperLocal oneBidHelperLocal_1 = null;
    {
        oneBidHelperLocal_1 = oneBidHelperLocalHome.create();
    }
    {
        Integer itemTypeId = null;
        Long bidAmount = null;
        Integer bidIncrement = null;
        Long maxBid = null;
        Integer userID = null;
        int currency = 0;
        oneBidHelperLocal_1.bidItem(
            itemTypeId,
            bidAmount,
            bidIncrement,
            maxBid,
            userID,
            currency);
    }
}
  
```

Below the code editor, the Test Data Table is visible, showing the data used for the test:

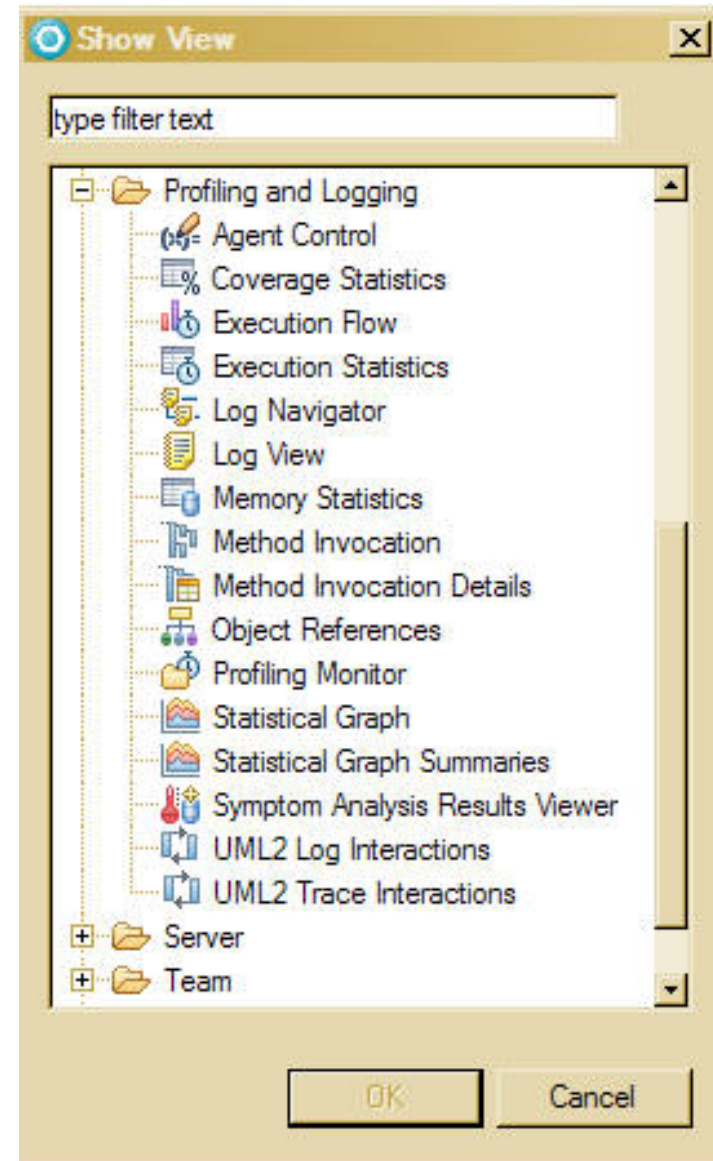
Action	Type	IN	default
oneBidHelperLocal_1 = oneBidH...			
oneBidHelperLocal_1.bidItem( it...			
itemTypeId	Integer	100546	
bidAmount	Long	100	
bidIncrement	Integer	[ 1 ... 10	
maxBid	Long	1000	
userID	Integer	83201	
currency	int	3	
ExpectedException	Throwable		

A blue callout box with a white border points to the Test Data Table and contains the text: **Test driver and stub datapool**



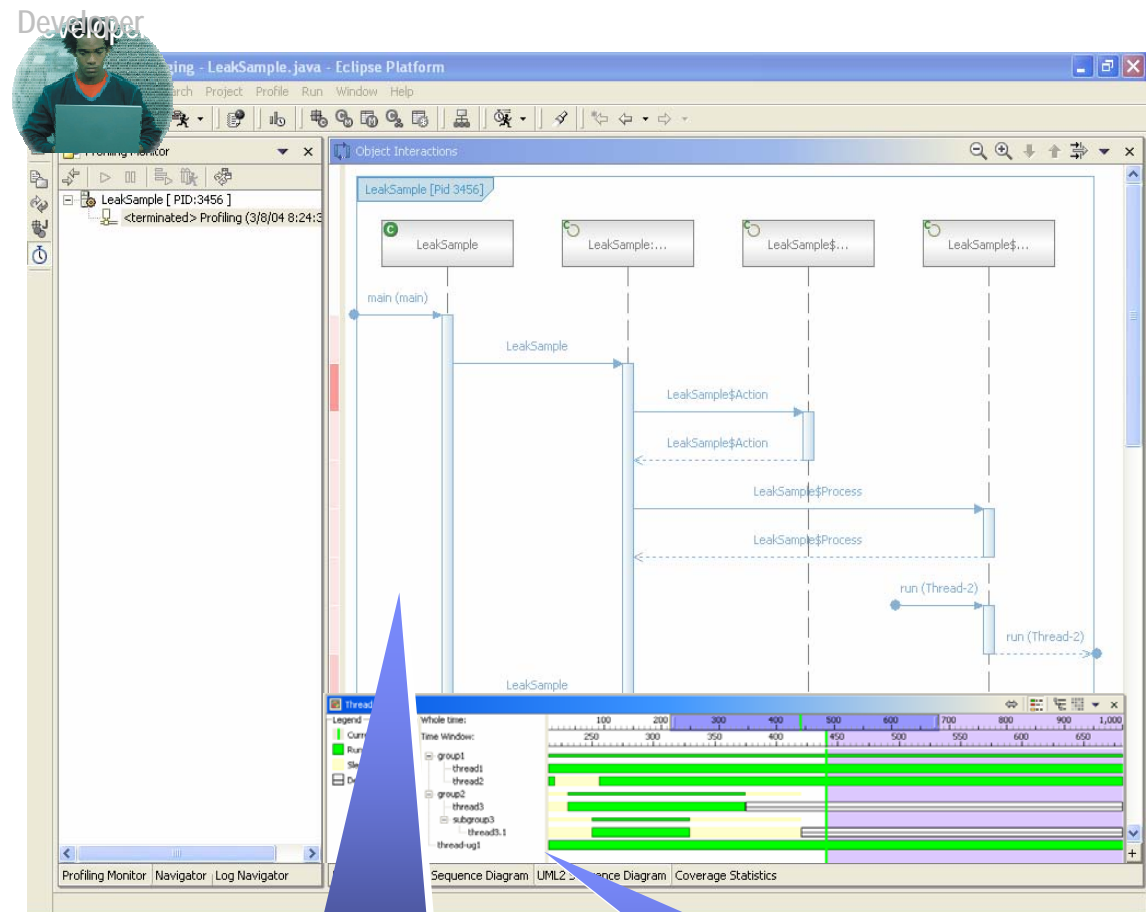
# Runtime Analysis

- Built-in tools helps developer isolate and fix performance problems
- Advanced features including:
  - ✓ Execution Stats
  - ✓ Log Navigator and View
  - ✓ Memory Stats
  - ✓ Method Invocation
  - ✓ Object References
  - ✓ Profiling Monitor
  - ✓ Statistical Graph and Graph Summaries
  - ✓ Statistical Data
  - ✓ UML Sequence Diagram
  - ✓ UML Log Interactions
  - ✓ UML Trace Interactions
  - ✓ Static and Dynamic Probe Kit
  - ✓ Probe Kit Editor
  - ✓ J2EE Request Profiler
  - ✓ PI-Agent
  - ✓ RAC Security Extension
- Profiling tools can seamlessly trace across multiple servers



## Debug: Understand behavior, performance & memory

- **Developer analyzes application for memory, performance, functional problems**
- Application tracing
  - ▶ User defined probes for logging
  - ▶ Thread analysis
  - ▶ Multi-tier UML 2.0 sequence diagrams
  - ▶ Automated log analysis with symptom databases
- Performance analysis
  - ▶ Call graph
  - ▶ Thread analysis
- Memory analysis
  - ▶ Local and remote memory leak analysis down to the line level
  - ▶ Includes IBM Research IP
- Code coverage



**Live UML2 sequence  
diagram generation**

**Thread Analysis**



# Examples of Run-Time Problem Determination

Run-time Problem Determination								
Memory analysis	Code Coverage	Performance Analysis	Execution Trace	Thread Analysis	System Monitoring	Probe based analysis		
						Logging	J2EE Trace	User Defined Probes

- ▶ Includes:
  - Memory profiling and leak detection
  - Performance profiling
  - Thread analysis and deadlock detection
  - Code coverage analysis
  - Runtime log and trace analysis



# Memory Analysis

Object Reference Graph [Heap : 3] - com.queues.TestThreeTierQueue at tvohra-laptop [ PID:4884 ]

Visible: 12/12      Highlighted: 5/5

Leak Candidates - com.queues.TestThreeTierQueue at tvohra-laptop [ PID:4884 ]

Heap dump: id: 1, Name: C:\br-252\workspace\ProfileProject\leakanalysisheapdir\optHeap.20040617.233215.0000004884.00.01.trchoh  
 Heap dump: id: 3, Name: C:\br-252\workspace\ProfileProject\leakanalysisheapdir\optHeap.20040617.233215.0000004884.00.05.trchoh

<Likelihood	Root of leak	Container type	What's leaking	Number of leaks	Bytes leaked	Objects leaked
000.84	TestThreeTierQueue.271...	Vector	String	9,000	0	9,000

**Highlighted leak candidate and container**

**Bytes and objects leaked**



# Threads view and UML2 thread interactions

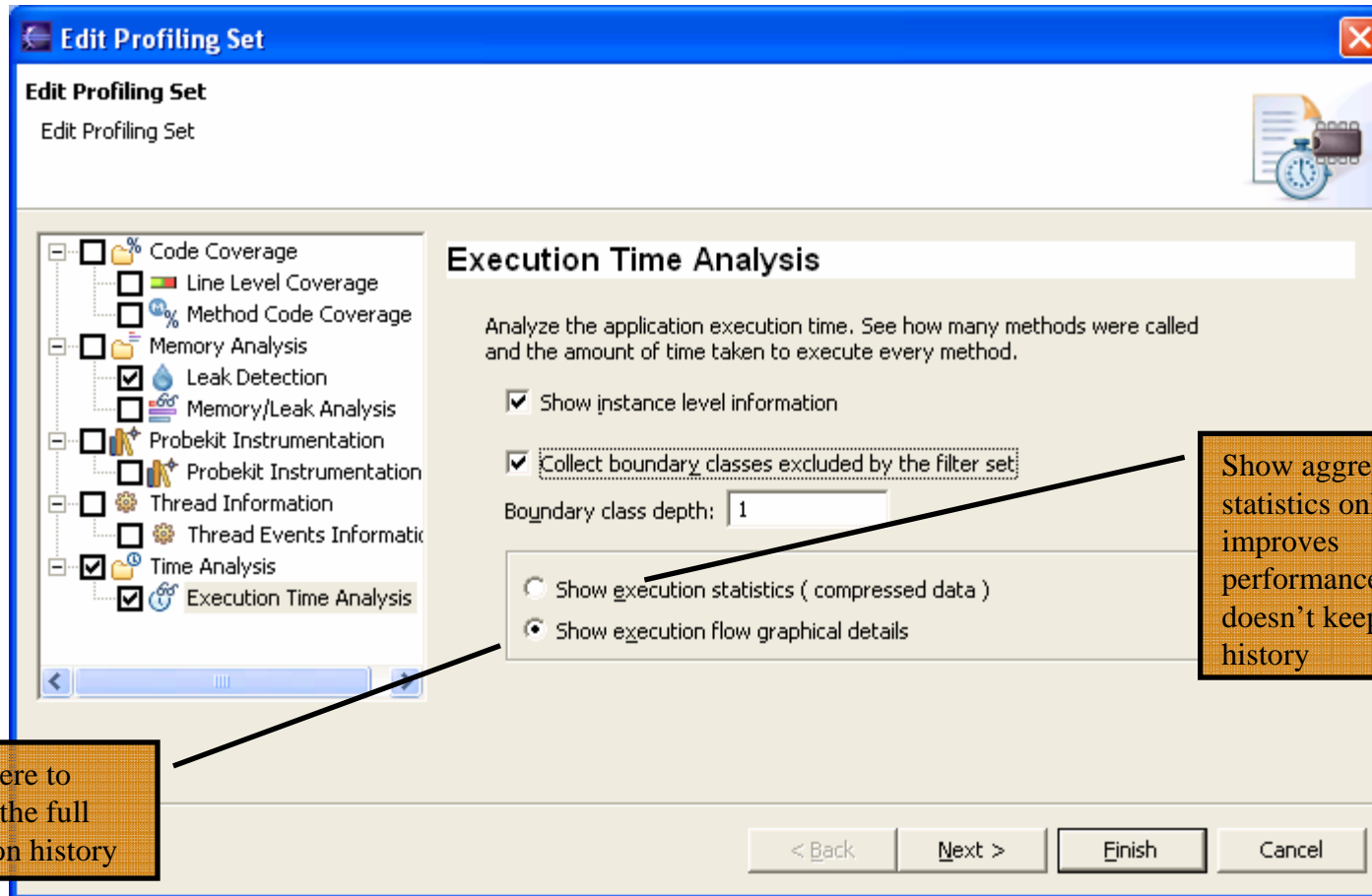
The screenshot displays the Eclipse IDE's Profiling and Logging tool. The top window shows a UML2 Sequence Diagram with four lifelines: main Process bakery..., Thread-0 Process bak..., Thread-1 Process bak..., and Thread-2 Process bak... The diagram shows messages for SecretStuff, addItem, and makeBread. Thread-1 is highlighted with a green box and labeled "Thread holding lock".

The bottom window shows the Thread View for "Thread Contention Analysis [bakery.TestKitchen-4316]". It includes a legend for thread states: Running (green), Sleep (light blue), Waiting for Lock (orange), Waiting for Object (red), and Dead (grey). The timeline shows the execution of main, Thread-0, Thread-1, and Thread-2. A tooltip for Thread-2 at 0.803s indicates it is in a "Waiting for Lock" state, with the lock held by Thread-0.

*Thread holding lock*



# Performance analysis instrumentation



Select here to capture the full execution history

Show aggregate statistics only - improves performance but doesn't keep the history



# Execution history: performance callgraph

Callgraph shows summary of execution history

Details of the selected method are here...

The screenshot displays the Eclipse Profiling tool interface. The central pane shows a Performance Call Graph for 'ThreadsMore.run' at PID:4876. The graph highlights the 'Max Path to Root'. The root node is 'main', which branches into 'TM #0', 'TM #1', 'TM #2', and 'TM #3'. These threads converge on 'ThreadsMore.run', which then branches into several sub-methods: 'ThreadsMore.-clinit-', 'Thread.setName', 'Thread.start', 'Thread.join', 'ThreadsMore.doCP', and 'ThreadsMore.printf'.

The right-hand pane shows 'Method Details' for 'ThreadsMore.run'. It lists various performance metrics:

- Method: ThreadsMore.run
- TM #0: 4876
- Calls
- Base Time
- Cumulative ...
- Avg Time
- Min Time
- Max Time
- Source File

The 'Callers' section includes a pie chart and a table:

Caller	Percent
TM #0	24.445
TM #1	25.488
TM #2	25.208
TM #3	24.859

The bottom pane shows 'Method Statistics' for 'ThreadsMore.run' with a table of method calls:

<Method Names	Class Names	Base Time	Average Base T...	Cumulative Time
toString() java.lang.String	StringBuffer	0.000064	0.000016	0.000064
ThreadsMore()	ThreadsMore	0.000047	0.000012	0.000728
Thread()	Thread	0.000681	0.000170	0.000681
StringBuffer(java.lang.String)	StringBuffer	0.000066	0.000017	0.000066



# Execution History – Sequence Diagrams

The screenshot displays the Eclipse Platform interface with the following components:

- Left Panel (Project Explorer):** Shows a project named 'wca3bgilbert7' with various classes and threads listed, including 'ThreadsMore at wca3bgilbert7 [ PID:4876 ]'.
- Top Panel (UML Sequence Diagram):** Shows a sequence diagram with participants 'ThreadsMore', 'Thread', and 'Class'. Messages include 'toString', 'setName', and 'Thread'. A callout box points to the diagram with the text: "Sequence diagram shows packages, classes, methods".
- Right Panel (Method Statistics):** Shows a table of method statistics for 'ThreadsMore at wca3bgilbert7 [ PID:4876 ]'. A callout box points to the table with the text: "Methods and/or classes can be hidden from the diagram".
- Bottom Panel (Console):** Shows the execution output, including messages like 'Adding Phase.', 'Consuming Phase.', and 'Adding Phase.'.

>Method Names	Class Names	Base Time	Average Base T...	Ci
loadClassInternal(java.lang.String)...	ClassLoader	0.000175	0.000058	
main(java.lang.String[]) void	ThreadsMore	0.000527	0.000527	
println(java.lang.Object) void	PrintStream	0.021751	0.000680	
printMe(java.lang.Object) void	ThreadsMore	0.000575	0.000018	
run() void	ThreadsMore	0.005128	0.001282	
setName(java.lang.String) void	Thread	0.000078	0.000019	
start() void	Thread	0.009117	0.002279	



# Execution History: Execution Flow

The execution flow view shows the call sequence with Threads as vertical columns. You can zoom in for details.

Method Statistics - Jellotime at wca3bgilbert7 [PID:5940]

>Method Names	Class Names	Base Time	Average Base T...
_eu(com.ibm.rational.pd.linelevel...	PKLLMCProbe	0.005708	0.000001
action(java.awt.Event, java.lang....	Jellotime	0.000155	0.000155
add(java.awt.Component) java.a...	Container	0.000292	0.000146
add(java.lang.String, java.awt.Co...	Container	0.000525	0.000263
Applet()	Applet	0.001660	0.001660
Button(java.lang.String)	Button	0.000217	0.000109
checkPackageAccess(java.lang.Cla...	ClassLoader	0.000516	0.000034

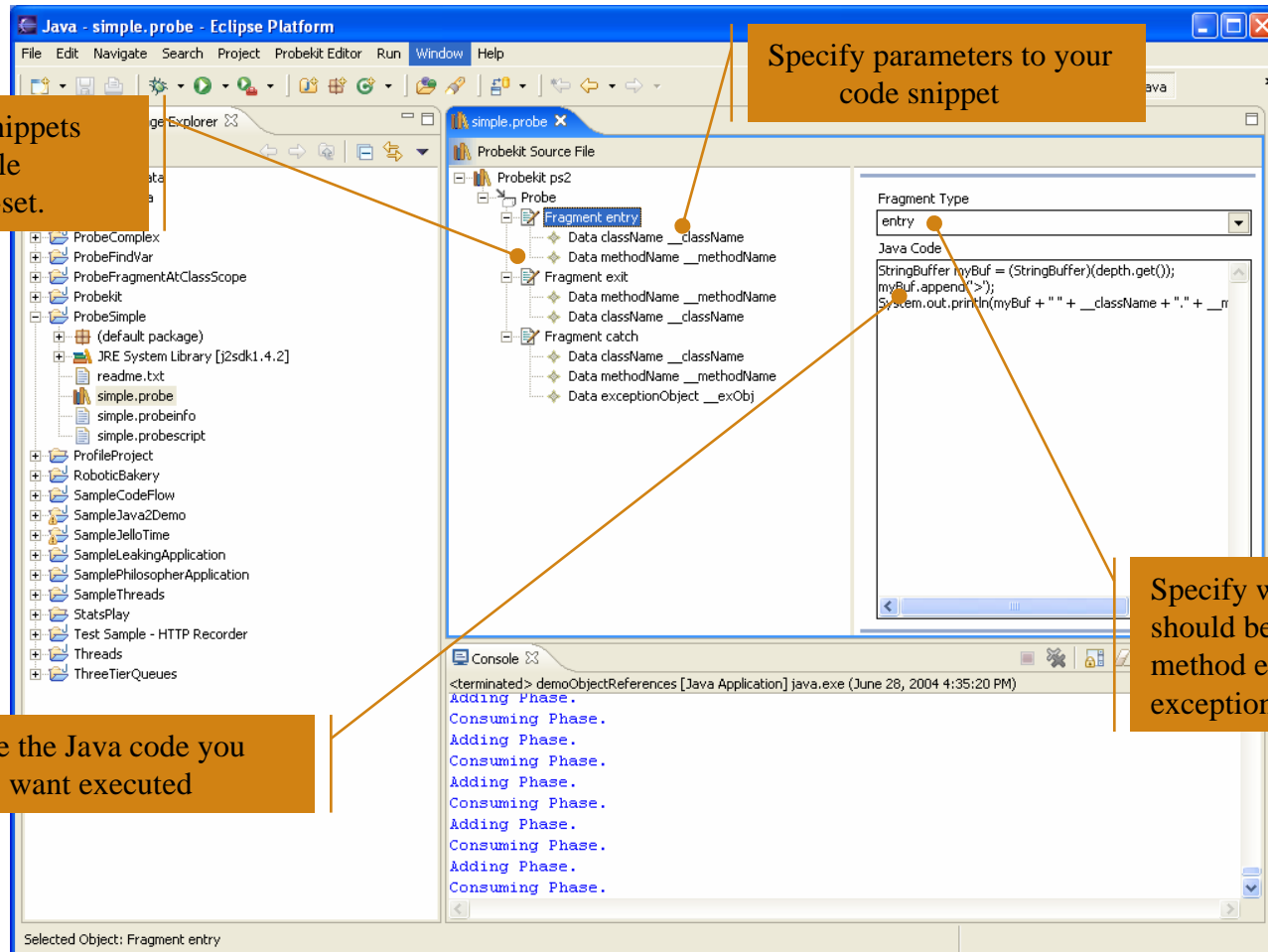


## Probe-Based Analysis

- Probes – snippets of code inserted in your application
  - ▶ Extend runtime analysis to fit your needs
  - ▶ Access to input/return arguments
  - ▶ Define static and thread local objects
  - ▶ Ultimate debugger breakpoint
- How?
  - ▶ Author a probe
  - ▶ Create a Profile Set that includes your newly created probe
  - ▶ Profile using that Profiling Set



# Author a Probe



Create multiple snippets in a probe, multiple probes in a probe-set.

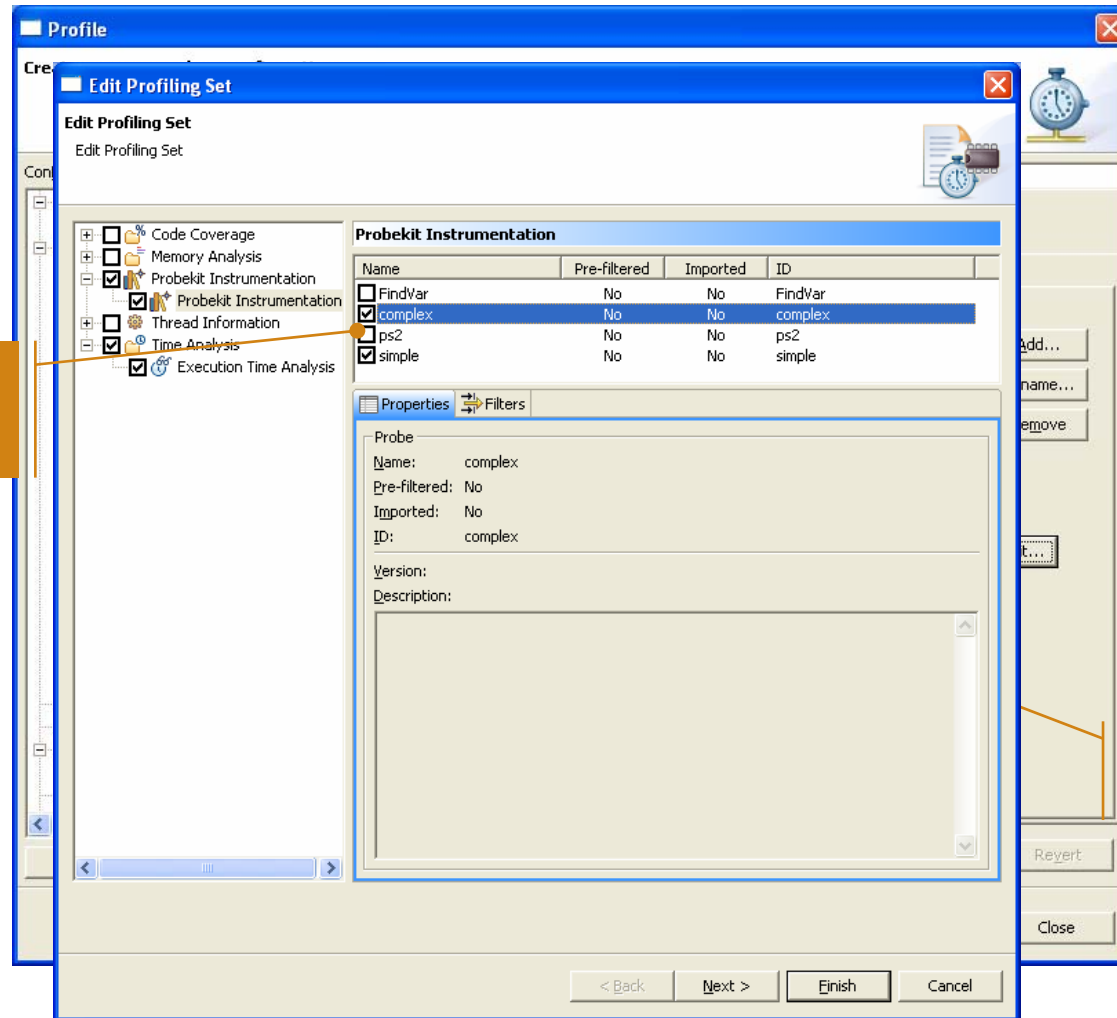
Specify parameters to your code snippet

Specify when the snippet should be inserted: L method entry, exit, exception catch, etc.

Write the Java code you want executed



# Use a Probe



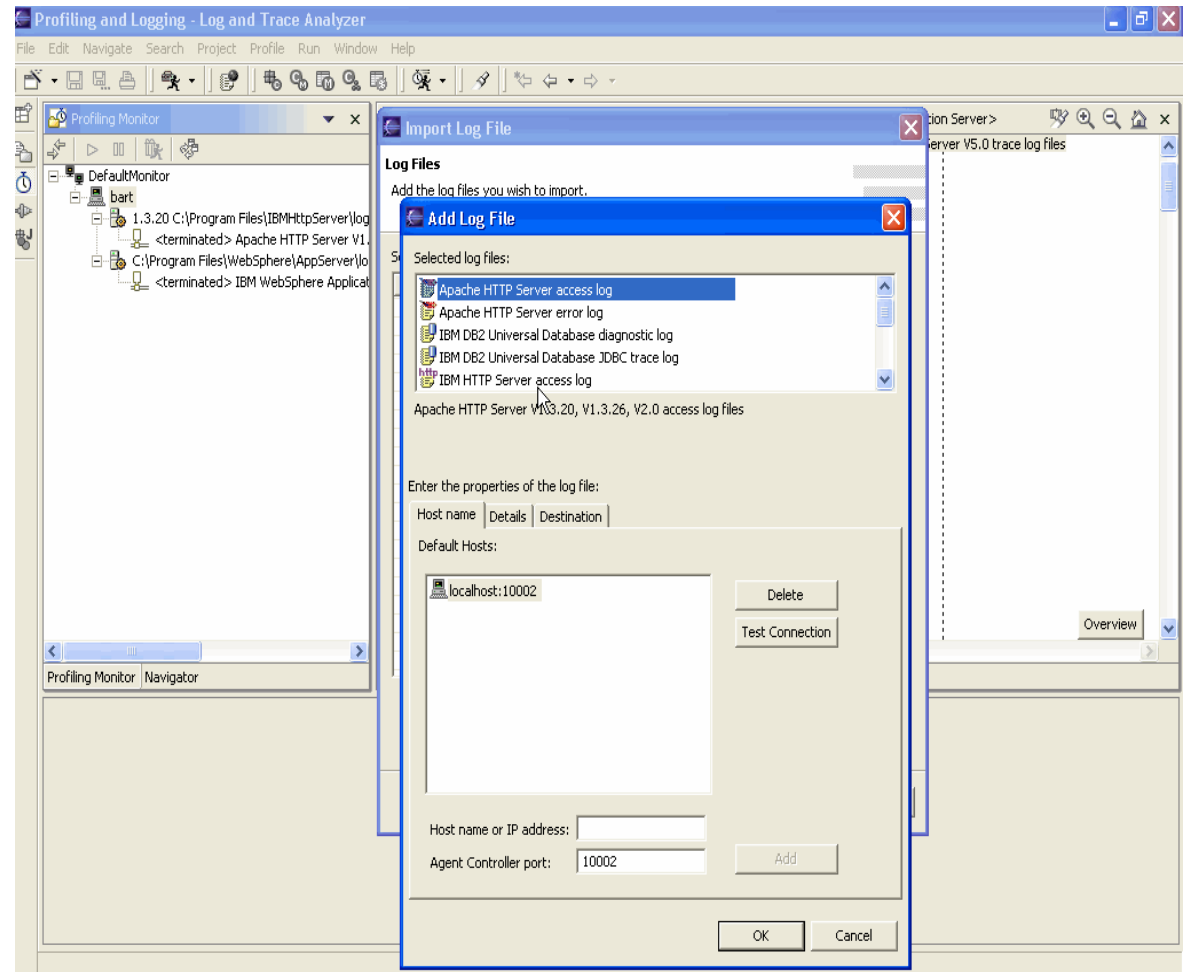
Specify one or more probes as part of your profiling set

Create or edit a profiling set



# Log and Trace Analysis

- **Generic log adapter to converts existing log files into Common Base Events (CBE) format**
- **Built in parsers: Imports existing log files and converts to CBE format on the fly using the Generic Log Adapter technology**
- **Built in correlation engines: Visually displays the correlation between log records using a number of factors**
  - **Sequential Correlation**
  - **Associative Correlation**



## Log and Trace Analysis – Symptom Database

- Used in the analysis of events and error messages that may occur in a log.
- XML file of symptoms, string match patterns, associated solutions, and directives.

The screenshot displays the IBM Rational Log and Trace Analyzer interface. The top window, titled "Profiling and Logging - regex.adapter - Log and Trace Analyzer", shows a UML2 Sequence Diagram with two lifelines: "Access L..." and "Error Log ...". A red arrow points from the "Error Log ..." lifeline to the "Log View" window below. The "Log View" window shows a list of log records with a filter applied: "Filter matched 1396 of 1396 records". The selected record is:

```
DSRA0080E: The Connector:Pool Manager could r
```

The "Property" table for this record is:

Property	Value
localInstanceId	
globalInstanceId	N2CDEF800C7211D68000AC4D2B08B8C6
creationTime	2004-03-02 12:44:37.035000-05:00
severity	50

The "Details localInstanceId" tab shows the following analysis result:

```
Extended message documentation - Explanation: The database back end generated an exception which was caught by the Data Store Adapter and reported to you. User Response: Try to fix the primary problem reported by the database software. [---WAS 5.0 All Versions---]
```

The "Analysis Result" tab shows the following analysis result:

```
Extended message documentation - Explanation: This message indicates that an exception was thrown by the Pool Manager when attempting to allocate a Managed Connection. The exception text should help with deciphering the problem encountered. User Response: If there are no user controlled indications, the message may be due to an error in the internal J2C runtime processing. Contact WebSphere support and provide the data from running collector.bat. [---WAS 5.0 All Versions---]
```

# Code Coverage Analysis

**Code Coverage Analysis in Eclipse IDE**

The screenshot displays the Eclipse IDE interface with the following components:

- Project Explorer (Left):** Shows the project structure for 'wca3bgilbert7'. The 'Line Level Coverage' view is expanded, showing green checkmarks for hit lines and red 'X' marks for missed lines in the 'Jellotime at wca3bgilbert7' package.
- Annotated Source View (Center):** Displays the source code for 'com.queues.TestThreeTierQueue'. The code includes methods like 'initjta()' and 'initjta()'. Green checkmarks are visible on the left margin of the code editor, indicating which lines were executed during the test run.
- Coverage Navigator (Right):** Provides an overview of the coverage across the project. It shows a tree view of packages and classes, with colored bars indicating the coverage status of each element.
- Thread View (Bottom):** Shows the 'Thread Run Overview [Jellotime-232]'. It includes a legend for thread states (Running, Sleep, Waiting for Lock, Waiting for Object) and a timeline graph showing the execution duration of the thread.

Annotations in the image highlight key features:

- Annotation 1:** 'Annotated source view shows lines hit or missed.' points to the green checkmarks and red 'X' marks in the source code editor.
- Annotation 2:** 'Coverage navigator shows overview' points to the Coverage Navigator window on the right side of the IDE.

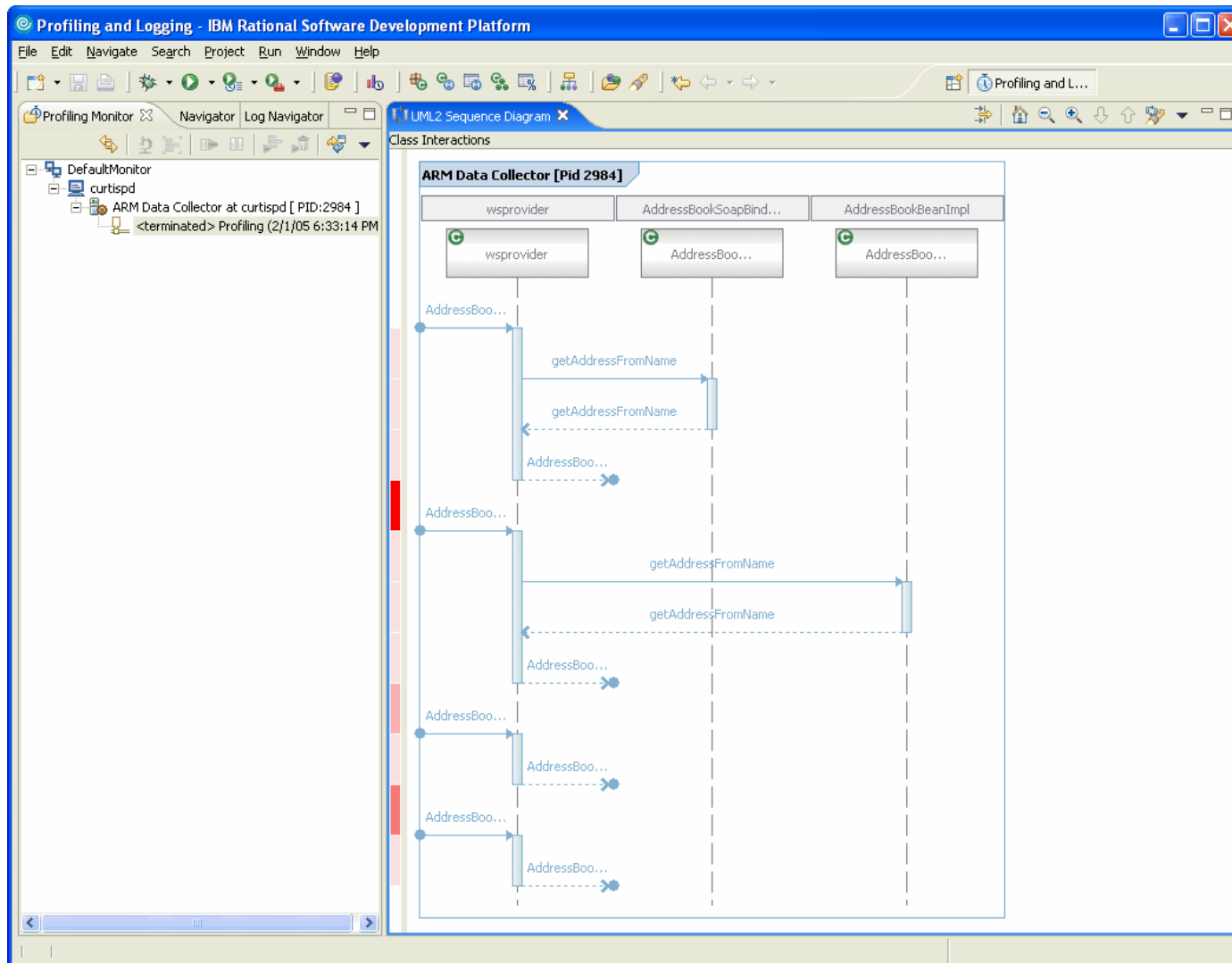


## Distributed Transaction Analysis

- Import application response measurement (ARM) based data collected in production systems or in test labs while during performance testing
- Analyze distributed transactions
- View them as UML2 class interactions between multiple processes or hosts
- Correlate to log views and system performance



# Distributed Transaction Analysis – class interaction between web services



# System Analysis and Correlation

1. The CPU monitoring on the machine where the transaction is running is linked (based on time) with the trace and log views

2. At the point a user notices a performance problem or load...

3. ...the corresponding transaction can be visualized in the UML sequence diagram..

4. ...and the log entries on the machine can be correlated and analyzed using the symptom database

The screenshot displays the IBM Rational Performance Center interface. At the top, a 'Statistical Data' window shows a line graph of CPU usage over time, with a red dot marking a peak at 15:28:45. Below this, a 'UML2 Sequence Diagram' window shows a sequence of messages between objects: 'doGet' from 'ShoppingServlet' to '\_CatalogHome\_Stub', followed by 'create' messages to '\_Catalog\_Stub' and '\_shopping', and 'getItemsByCategory' messages to 'ImageServlet' and 'EJSRemoteStatelessC'. A red bar at the bottom of the diagram indicates a time range. At the bottom, a 'Log Records' window shows a list of error messages, with one selected: 'J2CA0020E: The Connection Pool Manager could not allocate a Managed Connection'. A 'Property Value' table is visible next to the log entry.

Property	Value
analyzed	true
creationTime	2005-02-16 14:35:11.019000-
elapsedTime	0
extensionName	CBCECommonBaseEvent
globalInstanceId	CEE4BF3F22BCB84C1D56FDBB1805A11D9



## Summary

- **Testing solutions must provide:**
  - ▶ Architecture discovery, analysis and control
  - ▶ Code review
  - ▶ Component Test
  - ▶ memory analysis : leak candidates identified
  - ▶ threads analysis: thread and lock interactions
  - ▶ performance analysis: call graph, execution flow, uml2 trace
  - ▶ Probe based analysis: write and deploy multiple probes
  - ▶ Log and trace analysis: generic log adapter, symptom database
  - ▶ Code coverage analysis: summary view and annotated source
  - ▶ Distributed transaction analysis: production or test environments
  - ▶ System analysis and correlation: across multiple hosts or processes

