FEATURES

Embarcadero® J Optimizer™

EMBARCADERO TECHNOLOGIES®

Feature Matrix

OPERATING SYSTEMS
Microsoft Windows XP SP3
Microsoft Windows Vista SP1
Mac OS X (10.5)
Red Hat Enterprise Linux 5
Solaris 10 SPARC (J Optimizer Agent only)

TOUCH-POINT PLUG-INS
Eclipse 3.3.2, Java and Java EE Editions
Eclipse 3.4.1, Java and Java EE Editions
Genuitec MyEclipse 6.6
Genuitec MyEclipse 7.0
IBM Rational Application Developer 7.5

APPLICATION SERVERS
Apache Geronimo 1.1.1
Apache Geronimo 2.0
Apache Tomcat 5.0
Apache Tomcat 5.5
Apache Tomcat 6.0
BEA WebLogic Application Server 9.2 MP3
BEA WebLogic Application Server 10.1 MP1
IBM WebSphere 6.1
IBM WebSphere 6.1 with EJB 3 Feature Pack
JBoss 3.2.6
JBoss 4.0.5
JBoss 4.2
JBoss 5.0
Oracle Application Server 10.1.3.3 (Agent support only)
Sun GlassFish V1.1 UR1
Sun GlassFish V2.0 UR2
Jetty 6.1 (Agent support only)

CODE PROFILING AND PERFORMANCE OPTIMIZATION
Complete code profiling and performance optimization capabilities for identifying and solving code-level performance issues.
Improve performance and reliability of any Java code: Java applications, Java EE applications, servlets, applets, EJBs, JavaBeans, JSP applications, and Java Tag Libraries.
Easily connect to a remote Java process using the J Optimizer Agent to test a program running on a remote machine.
Developers have the option to install J Optimizer touch point plug-in into an existing Eclipse or Eclipse-based product and make it profile ready.

MEMORY AND CPU PROFILING
High-level performance-related data displayed in real time in order to determine whether a performance issue is related to CPU, memory, or both.
Automatic Memory Leak Detector monitors the evolution of memory usage over time for the immediate identification of potential memory leaks.
Real-time monitoring of object allocations to understand how the profiled program uses the virtual machine memory; Allocation Backtrace View enables identification of the code or part of the program responsible for object allocations.
Object Size Display automatically computes and displays, in real time, the amount of memory being consumed by all instances of a class; then sort and view by object size to prioritize objects consuming the most memory.
Reduced reference graph provides a transitive closure of the full reference graph to display only references that should be removed in order to free the object for garbage collection.
CPU Profiler measures pure CPU usage or time usage during a profiling session, with option to use sampling-based or instrumentation-based profiler.
Display profiling information per thread and thread groups, with color highlighting of threads that were busy during profiling session.
HotSpot Display lists methods where most time was spent, to help identify bottlenecks due to single methods.
Scalable call graph visually isolates critical code. Select a string allocation and highlight the flow of a method call to see where memory and time are being spent.
Automatic Application Quality Analyzer supports performance-error prevention and coding standards by automatically detecting VM-level performance bugs.
Export views in XML, HTML and CSV format.

THREAD DEBUGGING
Real-time display of the progress of all threads running within the virtual machine.
Understand thread contentions for a monitor with the detailed panels of the Contention View.
Wait state monitoring to understand why a thread is not making progress with the Waiting View and I/O Waiting View.
Identify and correct excessive locking where a thread enters and holds monitors using the Monitor Enter View reports.
Analyze deadlocks using graphical view of the relationships between threads and monitors to quickly understand deadlock situations.
Predict deadlocks with Monitor Usage Analyzer which generates full list of warnings and errors that might lead to deadlocks and performance bottlenecks, such as lock order warnings, lock and wait warnings, and lock and I/O wait warnings.

CODE COVERAGE

Code Coverage helps identify dead code, frequently used code, and unloaded classes in your Java applications.

Class Coverage View lists all classes and the real-time test results for each class. You can see how many lines of code have executed, and you can apply filters to test only certain classes.

Real-time display of all classes and interfaces used by the tested program and real-time percentage of lines covered per class.

Method Coverage View displays the source code for methods in a selected class, as well as statistical information about the number of times your method was called while your test application was running.

Source Code Viewer shows lines of code that have never been executed, making it easier to spot dead code.

Option to display the interfaces that have been loaded by the virtual machine and those that have not been loaded.

Batch-mode support to easily include code coverage in any batch-mode testing process.

AUDITS AND METRICS

Metrics evaluate object model complexity and quantify your code.

Metrics results can be viewed graphically in either bar charts or Kiviat charts format.

Metrics can be used to create reports and compare the overall impact of changes in a project.

Audits provide information on design issues, naming conventions, and other items which can impact code quality.

Audit results can be displayed with descriptions of what each audit looks for and how to fix violations.

Includes a group of audits known as "Bad Smell Audits" that detect some issues or convention violations in source code (misplaced classes, attributes and methods, wrong inheritance usage), which require some code refactoring.

Create specific QA Sets for source code audits and metrics, and save them to a local file system, or as part of a project which can then be shared with the team.

REQUEST ANALYZER

Profile the performance behavior of your Java EE application code across the following Java EE components: JDBC, JSP, JNDI, Enterprise JavaBeans, and JMS containers.

Improve performance and reliability of Java EE-related application code earlier in development with drill-down performance information for Java EE components.

Visual interface simplifies the complexity of Java EE application interactions using graphical representation.

System Dashboard view provides a graphical display of the application time spent in Java EE components and total number of requests. Shows the percentage of use for each server module to quickly detect any major component-level performance issues.

System Composite view displays all of the Java EE events that have occurred in an application, in real time, in their proper hierarchy. Hierarchy shows the relationship of events in terms of which events spawn others.

SNAPSHOTS AND PROGRESS TRACKER

Snapshots capture all the data from a particular test run, which can then be opened for analysis in the product that generated it, such as Profiler, Code Coverage, or Request Analyzer.

Continuously monitor and measure the impact of performance changes by comparing visual snapshots.

Progress Tracker enables you to visually compare, monitor, and measure the impact of performance changes by comparing visual snapshots from Profiler, Code Coverage, and Request Analyzer snapshots.

Generate reports that can be exported in PDF and HTML format.