# Improved J2EE Performance and User Experience with Pre-production Testing

*written by*
*Quest Software, Inc.*

World Headquarters
5 Polaris Way
Aliso Viejo, CA 92656
www.quest.com
e-mail: info@quest.com

Please refer to our Web site for regional and international office information.

Updated—February 2005

# ABSTRACT

Quest Software knows the expectations customers have for IT investments are not always met.

That's why we develop innovative products that help our customers get more performance and productivity from their applications, databases and infrastructure.

Through a deep expertise in IT operations and a continued focus on what works best, we help our customers meet higher expectations for enterprise IT.

With Quest Software, you can expect more.

# CONTENTS

# OVERVIEW

This paper will argue that application performance tuning and analysis should occur regularly, repeatedly and earlier in the application life cycle. What we call an "iterative" approach to tuning and analysis entails addressing performance problems before they accumulate, overwhelm and become costly. Responsibility for tuning must fall to developers, who need to treat tuning and analysis as elements of the development process. This approach opposes a traditional view of tuning as a process separate from construction, or one assigned to a QA team unfamiliar with the code and life cycle construction phase.

We view the need for a shift to this iterative approach to be motivated by:

- Changes in how consumers view performance and reliability
- Changes in customer requirements and in how software is brought to market
- The exploding presence of Java in the enterprise market, particularly in the server-side arena

These motives reflect the growing realization that critical software systems are strategic assets that must be protected and nurtured throughout development and use.

This paper will also illustrate how JProbe Suite and PerformaSure provide a full range of testing procedures that, when utilized throughout the development stage, make iterative testing a best practice. It will show how JProbe Suite's accuracy, breadth, detail and ease of use enable developers to not only determine where problems are occurring, but also refine units of code as they are completed. This mitigates performance problems before they reach a critical level. It will show how PerformaSure, when used in a staging environment, will allow users to identify performance issues across their entire J2EE stack prior to production hand-off.

# INTRODUCTION

As Java and custom Web applications (J2EE) continue to become the Internet's dominant programming environment, the issues of performance and reliability remain a major concern. As the Internet continues to grow, companies are pushed to produce software faster than ever, inevitably resulting in products with more bugs. According to Brian Bershad, associate professor of Computer Science at the University of Washington, "The release schedules are crazy. It's not that the people (developers) aren't smart; it's just that they don't have time to think."

Whether the goal is to create an online trading system, an e-business site, a banking system, a portal application or a customer relationship management system, poor performance can translate into tremendous economic cost. Moreover, end users are beginning to expect the same levels of performance from their online applications as they do from their automobiles and appliances. Performance glitches are simply unacceptable.

By "poor performance", we mean the presence of coding and configuration irregularities that result in such things as inefficient memory usage, performance bottlenecks, thread deadlocks and resource saturation. If these irregularities occur while an application is running, they can lead to frustrating delays, inaccurate transmissions, screen freezes and even application crashes. Unconstrained memory growth, for example, can dramatically slow an application, preventing a stockbroker from executing a trade at a given moment. A performance bottleneck can cause frustrating delays as a customer attempts to pay a bill online.

Missed deployment and delivery dates, budget overruns, failure to comply with industry regulations, interrupted workflow, frustrated customers are all potential byproducts of application performance flaws. However, they can be prevented with the right products and an effective and informed risk management strategy that encompasses best practices in development and QA.

A best practice imperative for managing the risk of performance deficiencies centers on the issue of scheduling. For reasons that we will explore in the following sections, performance tuning and analysis should be integrated throughout the development stage rather than deferred until building is completed. Moreover, performance tuning and analysis should be viewed as aspects of a preventative procedure rather than a purely responsive one. When a coding problem is detected after deployment by an end user, it is not only more costly and time consuming to repair, it can also result in negative user perception. The push to produce cost-effective but reliable software products demands that coding problems be proactively identified and resolved throughout the development process.

This represents both a shift in procedure and in thinking about the overall importance of performance and how applications are designed, developed and brought to market.

In order to illustrate and explain this shift, this paper will address:

- The conditions historically responsible for relegating performance tuning and analysis to Quality Assurance (QA) and acceptance phases for representing it as a one-time, responsive measure.
- The current conditions that make earlier performance tuning and analysis not only feasible but also optimal.
- Procedural best practices for performance tuning and analysis, including timing, personnel and methodology.
- The value of JProbe Suite and PerformaSure in optimizing development resources and time, while delivering higher performing Java (J2EE) applications to production.

In summary, a best practice for performance tuning and analysis is to make it a fully integrated, iterative and systematic process executed during the development and QA stages of the application life cycle.

## THE EVOLUTION OF PERFORMANCE TUNING AND ANALYSIS

Given the dramatic impact of performance flaws, one would expect performance-tuning efforts that maximize J2EE application reliability would be included in the overall life cycle. One would also expect that dependable strategies to ensure application performance and reliability would be commonplace and built into the life cycle process.

In a survey by Quest Software, 91 percent of respondents agreed that analysis should take place during development. Yet 53 percent admitted that this rarely occurs. Why?

Unfortunately, application performance still tends to be viewed as a second-level priority, behind such things as functionality, ease-of-use, compliance with GUI standards, maintainability and development time. Largely because of its lowered priority, performance tuning and analysis is treated as a separate process activity tacked onto the end of the development stage of the application life cycle, and generally confined to QA (in conjunction with load testing). Unfortunately, most companies do not specify performance requirements in enough detail to perform adequate testing. Due to scheduling challenges, the time for manual performance testing along with manual functional testing is greatly compressed and sometimes skipped.

How did this approach take hold? In part, the requirements gathering phase for projects is much more difficult with respect to specifying exact load and scalability requirements. Also, performance and throughput is more important than ever before due to the increased use and availability of these applications worldwide. Unlike functional testing, performance testing requires more developer involvement to determine the root cause of issues and possible improvements. Performance issues are typically more difficult to diagnose than functional issues.

Managers need to understand the impact, cost and disruption caused by bugs caught in production. Evidence is mounting that the detection and removal of a bug before the product goes GA can save from 10 up to 1000 times the cost of repairing it afterwards. Quest has observed that in most cases where performance problems are detected in production, the development team is called in as J2EE experts to help quickly diagnose and resolve the issue. This practice is a major disruption for development managers trying to meet a deadline.

As customers strive for higher levels of productivity, they demand increasingly higher levels of performance and reliability, particularly in the area of Internet applications where the financial implications of unreliable code are magnified.

## Changing Market Dynamics

In addition to the reasons we've described, a shift in how software products are brought to market also favors a revision of the traditional approach to performance tuning and analysis.

Many medium-to-large size companies are looking to outsource portions of their development projects to countries with a lower cost of labor. This outsourcing arrangement can add additional complications when it comes to ensuring a high performing application. Companies need to have a way to validate the performance of applications developed abroad, prior to deployment into production, when the code is released from development. Standard developer-focused solutions, like JProbe Suite, are perfect for the company receiving the outsourcing, but not as appropriate for company doing the outsourcing, as they did not write the code. The company doing the outsourcing needs a higher-level tool that can be used to look across the entire J2EE stack and identify performance bottlenecks—this is the perfect place to use PerformaSure. Clear communication between the company outsourcing code and the company receiving it is imperative to ensure that performance requirements are met. The receiving company should use a tool like JProbe Suite and provide the company who outsourced to them with reports on performance testing progress, discovered bugs, etc.

The market is lowering its tolerance for defects. It is also demanding larger, more complex applications (ones that rely on a great deal of componentized code written by many different authors). Given this tension between a decreasing tolerance for defects and an increasing demand for functionality in a shorter time frame, it is necessary to re-examine the place performance tuning and analysis occupy in the application development life cycle.

Software organizations are responding—partially. They are revising their deployment criteria and moving to smaller, more frequent project iterations with earlier delivery dates. They are also taking advantage of the fact that object-oriented code has an inherent capacity for modularization. This allows for the release of parts of applications as code can be tested prior to the completion of a project. In other words, the need for speed to market and the advent of modularity are pushing tuning and analysis back in the application life cycle. But are they being pushed back far enough? Are they being made truly iterative? And is responsibility for them—and the tools to carry them out—being put into the hands of all developers working on a particular project?

Before examining how object-oriented code (Java and J2EE in particular) encourages the use of an integrated, iterative approach, let us summarize what we have said about the current performance-tuning environment. Because performance tuning and analysis require valuable time and have not been viewed as critical requirements (and because organizations often look to hardware upgrades to solve performance problems) there is a tendency to make tuning a low and expendable priority and to accept the tradeoff of compromised quality. In today's business environment where applications have to be built and deployed quickly to capitalize on the rapid changes in business processes and market conditions, this has taken on the look of an acceptable bargain. On the other hand, the push for speed to market—along with the increasing complexity of applications and the explosive growth of object-oriented languages like Java (J2EE) —has meant that organizations are looking for alternatives. They're trying to find ways to achieve higher performance quality without compromising delivery dates.

In short, the current environment is one of:

- An increasing trend to outsource portions of development
- A growing, opposing tendency on the part of consumers to be intolerant of application bugs
- An awakened possibility of modularizing applications and then deploying parts of them, made possible in large part by the growth of object-oriented languages

## Unique Challenges of Java Development and Performance Management

Java is uniquely suited to address the tension between the market's push for early delivery and its conflicting intolerance of bugs. Object-oriented languages like Java more readily allow for modularization, and thus make earlier, partial deployments possible. This encourages early, integrated, iterative tuning and analysis. However, certain challenges should be highlighted:

The increased use of components and frameworks requires more performance testing to ensure that these components and frameworks perform acceptably for use cases. Tools such as JProbe Suite and PerformaSure provide technical staff with necessary insight into how the components interact and where performance bottlenecks.

The Garbage Collector in the JVM does not eliminate all memory leak occurrences. Development teams need a tool to help them understand what objects are persisting in memory past garbage collections and when and where these objects were created.

The garbage collection system can also cause performance issues if the application generates many short-lived objects.

J2EE development is still relatively new and most companies lack sufficient experience to anticipate difficulties and avoid performance pitfalls. Tools and process can help guide these companies to higher levels of performance and productivity.

Configuration of J2EE Application Servers and JVMs is complicated and can have a profound affect on application performance. Performance tooling in staging and system testing needs to provide visibility into performance bottlenecks at a system, component and transaction level.

Additional performance, threading and scalability issues can result when load is applied. Companies need to ensure that proper scalability testing is done in their staging environment to correctly identify performance issues, as they would occur in the production environment.

Existing Systems Management products lack the ability to monitor J2EE systems and collect the necessary information needed by developers to correctly diagnose and resolve performance problems detected in production.

Most applications servers provide a JMX interface to allow users to monitor the application server, but users will need monitoring solutions that extend this information and provide key performance information at a transaction level.

Early tuning can help identify the methods that are creating these objects, and the developer can then modify the design to reduce their number. Early tuning also means that lessons of efficiency can be abstracted from one method to similar ones, thereby helping developers avoid repeating inefficiencies. In the latter case, that of memory leaks, the garbage collector is unable to reclaim memory as a result of an erroneous reference. Early tuning makes it easier to identify loitering objects and resolve the unintentional reference that retains them through garbage collector cycles, if for no other reason than that the design and implementation is fresh in the developer’s mind.

Java, with its inherent tendency to distance developers from much of the code they employ within an application, generates unique challenges that underscore the need to make performance tuning and analysis essential elements of the development stage. Also, the need for new testing and monitoring solutions is required to ensure maximum uptime and shortened resolution time for problems detected in production.

## Walking the Talk: The Iterative Model in Practice

We noted earlier that, while most managers and developers agree that tuning and analysis should take place during development, few are actually doing it. When thousands of users are simultaneously accessing an application launched from a single server, a small glitch that would be barely noticeable by a user working in a client-side environment becomes a major performance problem experienced by the entire community.

Likewise, the cost of fixing problems at a later stage has become very high, particularly as applications become more complex. As Juergen Brendel, CTO of Esphion, describes, "A major design flaw that can be fixed with just a stroke of a pen during the design stage may require major recoding if discovered when the software is almost finished." Deferring tuning and analysis until the testing phase (prior to deployment) simply poses too great a hazard as problems that surface at this stage tend to be far more severe, costly and time-consuming to repair.

The cost-effectiveness argument—the value of transforming performance tuning from a responsive exercise to a preventative one—is powerful. Early tuning can help teams avoid last-minute disasters that can threaten deployment dates, alienate customers and undermine carefully planned budgets.

When this element is added to the time constraints of software production, constantly shifting customer functionality requirements and decreasing market tolerance for poor performance, a strong case emerges for the use of a highly flexible, iterative and fully integrated performance-tuning process. Development teams can no longer afford to ignore the demands exerted by a changing marketplace and the rising prevalence of object-oriented code in Internet applications. Organizations have to be sure to treat software as a strategic asset that must be nurtured and protected throughout its life cycle.

Companies need a model that addresses the "organic" quality of the development process, one that acknowledges that the determination of requirements is a process of incremental discovery. Such a development model would proceed concentrically: requirements would be established, designs would be formulated, code would be written, problems would be detected, new requirements would emerge, new designs would be formulated and new code would be written—iterating towards a point at which the application fulfills the needs and expectations of the client. However, this does not remove the requirement for a J2EE-centric set of testing tools and monitoring tools in QA and production.

The key to achieving optimal performance is to tune during each cycle of development. This would help ensure that individual class definitions are solid and can be safely relied upon when they are imported into new applications.

## When to Begin Tuning

Within this approach, timing is essential. Tuning and analysis should be integrated throughout the development stage and organized around complete use cases, enabling developers to avoid potentially wasteful efforts while ensuring that errors do not accumulate.

We are not, however, advocating an "early as possible" position—tuning too early may consume valuable developer time on code that will change or is incomplete. It is important to find a compromise between the need to avoid devoting time to tuning use cases that are ultimately discarded and the need to tune early and often. This compromise can be achieved by tying thread analysis, memory debugging and performance profiling to the implementation of each use case. Changes to requirements or fixes can readily be inserted at each iterative stage. Partial product releases are made possible, and sound code based on repetitive identification and resolution of problems is generated, thereby providing a firm foundation on which to build subsequent code. In the interest of time, the identification of the key use cases and their performance requirements is a very important step that must be completed before any performance tuning.

Companies should look for industry-leading tools that allow integration with their build and unit testing system to allow for automation of collection of performance and memory information as part of the nightly build and test process. In addition, QA tools need to be integrated with load testing solutions and production monitoring and performance support tools need to be able to collect deeper performance information when key thresholds affecting end user experience are exceeded.

## Who Should be Responsible

Responsibility for performance must be shared among many team members in the development process. Business analysts must ensure that they accurately relay information about performance expectations and load expectations for the new business solution. Architects must then look at these requirements, look at the company's systems and expertise and design a solution that will ensure success today and in the future. The developers themselves need to take on a greater share of the responsibility for performance of the application at a code level.

Not only does this represent a shift in the timing of the tuning process, it also shifts responsibility for it. Developers should be armed with tools that enable them to make tuning and analysis new elements of the code writing process.

According to the iterative approach, performance tuning and analysis are assigned to the people who best understand the application design, objectives and the code itself. With their first-hand knowledge, developers are optimally equipped to make the best decisions about enhancements. Moreover, by making performance their responsibility from the start, developers are given the opportunity to learn about designing specifically for performance, a valuable skill that can benefit subsequent projects. The result is that developers can ensure a baseline of quality before an application goes into QA testing.

This does not remove the responsibility from the QA team to execute performance and scalability testing, but ensures that the code they are testing will be more robust due to the additional testing completed during development.

## Advantages

Should this approach be used for every project? When determining how early in the life cycle performance tuning and analysis should occur, and who should be responsible, it is important to consider the project's scope and specific requirements. In rare cases, when one is certain in advance that there will be no changes or if the application is particularly simple, a more traditional approach may be preferable. But it must be remembered that, even with a simple project, costs can rise quickly if one has to fix missed errors at a late stage in the life cycle, particularly if those errors become apparent post-deployment.

The greatest advantage to the iterative approach is that, with recurring tuning, performance reliability is built into each use case. In essence, each intermediate milestone release represents an actual product, so that subsequent stages can be viewed as extensions of earlier products. One corollary to this is that it allows a team to release a product to a client much earlier, something we noted to be increasingly desirable. Customers of e-commerce applications, for example, typically want new versions quickly so that they can address needs rapidly. By tuning earlier, and by developing in these smaller chunks, it is easier to achieve timely deployments.

Other advantages include:

- Removing a serialization bottleneck in the development process
- Forcing a degree of clarity from the perspective of designers and architects, essential for overall developmental efficiencies
- Helping developers better understand their own code and the code they are importing
- Focusing attention on early error elimination
- Allowing for actions to counteract risk to be taken at any time
- Focusing attention on reuse options
- Foregrounding a concern with quality and performance

# JProbe Suite

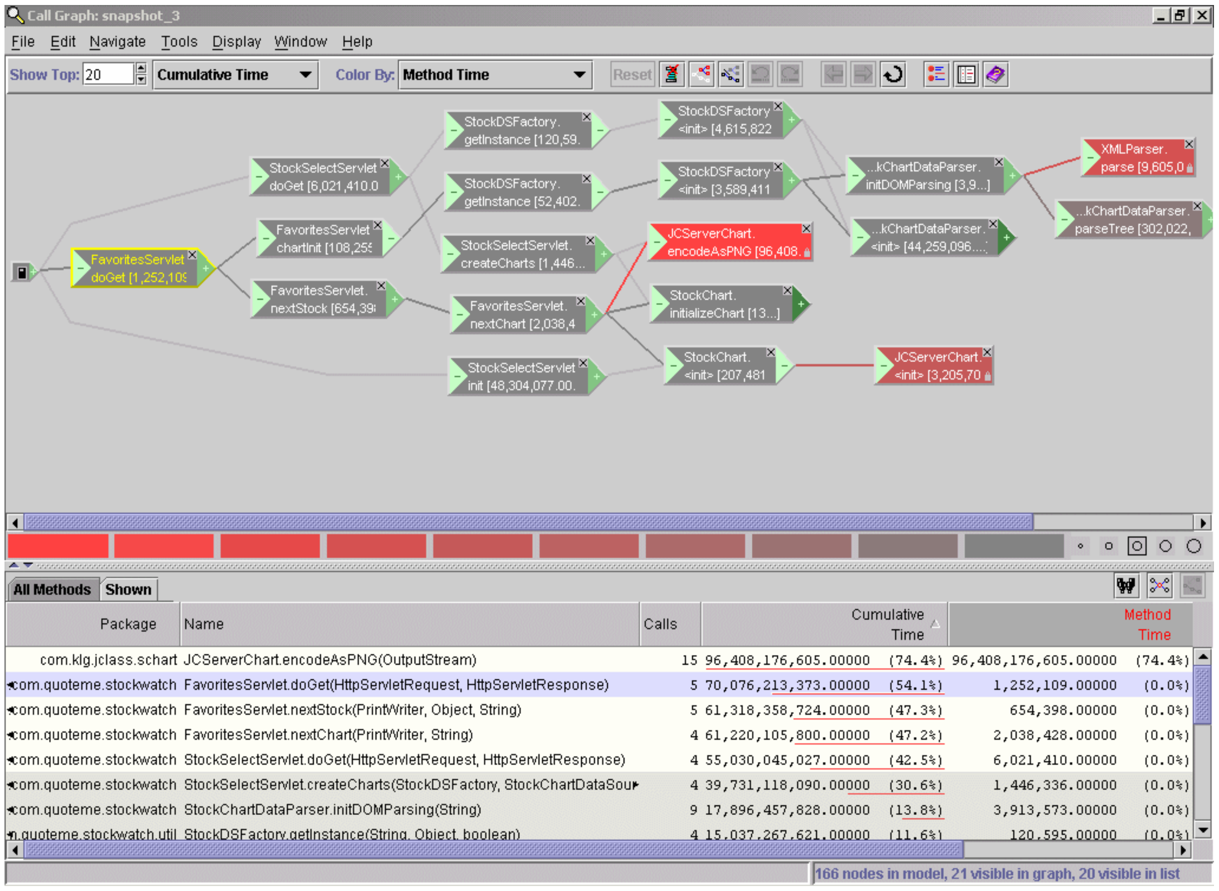
While the advantages of using Java are well known, the unique challenges that we have documented mean that potential savings can vanish quickly if developers are not effectively empowered with the right tools to produce the most reliable, efficient code possible.

JProbe Suite is a complete performance toolkit for Java code tuning. JProbe Suite helps developers diagnose and resolve performance bottlenecks, memory leaks, excessive garbage collection, threading issues and coverage deficiencies in their J2EE and J2SE applications. JProbe Suite is ideal for making early and regular tuning and analysis a best practice.

The tools in JProbe Suite integrate easily with your environment (Application or Web Server, IDE, JDK and operating system). Whether you need to analyze an application running on your desktop or on a remote server, JProbe Suite requires no changes to your application code in order to provide you with the best information you need to diagnose and resolve your java code related performance issues. In fact you can automate the collection of your performance information using JProbe Suite's advanced data collection features and save your valuable time.

## JProbe Profiler

JProbe Profiler helps developers identify and eliminate performance bottlenecks in Java code. It has a visual Call Graph interface as well as unique data collection technology. These features allow JProbe Profiler to provide highly accurate performance diagnostics, with line-by-line precision, at lightning speed. Developers can see exactly where time is being spent as the program runs, and then easily tune their code to dramatically increase the program's speed.


Call Graph: snapshot_3
File Edit Navigate Tools Display Window Help
Reset
StockDSFactory. getInstance [120,59.
StockDSFactory <init> [4,615,822
StockSelectServlet doGet [6,021,410.0
StockDSFactory. getInstance [52,402.
StockDSFactory <init> [3,589,411
...kChartDataParser. initDOMParsing [3,9...]
XMLParser. parse [9,605,0
...kChartDataParser. parseTree [302,022,
FavoritesServlet chartInit [108,25
StockSelectServlet. createCharts [1,446...
JCServerChart. encodeAsPNG [96,408.
...kChartDataParser. <init> [44,259,096...]
FavoritesServlet doGet [1,252,10
FavoritesServlet. nextStock [654,39
FavoritesServlet. nextChart [2,038,4
StockChart. initializeChart [13...]
StockSelectServlet init [48,304,077.00.
StockChart. <init> [207,481
JCServerChart. <init> [3,205,70
All Methods
Shown

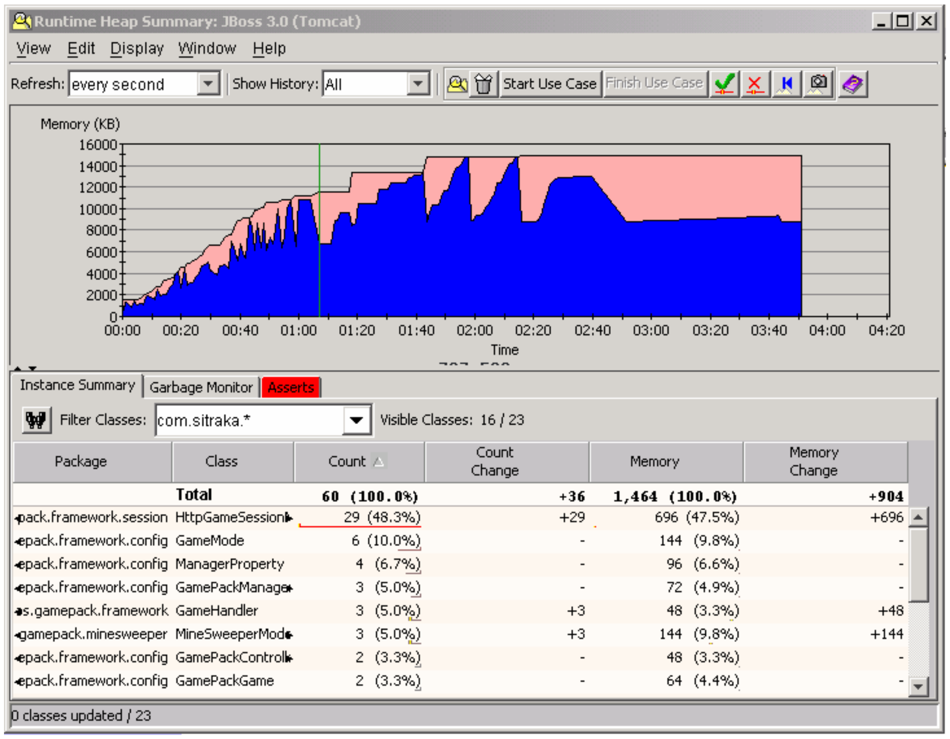
Show Top: 20 Cumulative Time Color By: Method Time

| Package | Name | Calls | Cumulative Time | | Method Time | |
|---|---|---|---|---|---|---|
| com.klg.jclass.schart | JCServerChart.encodeAsPNG(OutputStream) | 15 | 96,408,176,605.00000 | (74.4%) | 96,408,176,605.00000 | (74.4%) |
| com.quoteme.stockwatch | FavoritesServlet.doGet(HttpServletRequest, HttpServletResponse) | 5 | 70,076,213,373.00000 | (54.1%) | 1,252,109.00000 | (0.0%) |
| com.quoteme.stockwatch | FavoritesServlet.nextStock(PrintWriter, Object, String) | 5 | 61,318,358,724.00000 | (47.3%) | 654,398.00000 | (0.0%) |
| com.quoteme.stockwatch | FavoritesServlet.nextChart(PrintWriter, String) | 4 | 61,220,105,800.00000 | (47.2%) | 2,038,428.00000 | (0.0%) |
| com.quoteme.stockwatch | StockSelectServlet.doGet(HttpServletRequest, HttpServletResponse) | 4 | 55,030,045,027.00000 | (42.5%) | 6,021,410.00000 | (0.0%) |
| com.quoteme.stockwatch | StockSelectServlet.createCharts(StockDSFactory, StockChartDataSou | 4 | 39,731,118,090.00000 | (30.6%) | 1,446,336.00000 | (0.0%) |
| com.quoteme.stockwatch | StockChartDataParser.initDOMParsing(String) | 9 | 17,896,457,828.00000 | (13.8%) | 3,913,573.00000 | (0.0%) |
| n.quoteme.stockwatch.util | StockDSFactory.getInstance(String, Object, boolean) | 4 | 15,037,267,621.00000 | (11.6%) | 120,595.00000 | (0.0%) |

166 nodes in model, 21 visible in graph, 20 visible in list

## JProbe Memory Debugger

JProbe Memory Debugger pinpoints the sources of inefficient memory usage by tracking objects that hold references to other objects. Inefficient memory usage results from oversights in reference management that causes failure to reclaim discarded memory. An object that is not being used but persists wastes valuable memory. If memory completely runs out, the ultimate result can be a fatal **OutofMemoryError**. JProbe Memory Debugger's intuitive Memory Usage window allows developers to visualize memory usage, including memory allocation and garbage collection—and to do it in real time. The Garbage Monitor helps developers to improve performance by identifying those methods that generate excessive amounts of short-lived objects.



| Package | Class | Count | Count Change | Memory | Memory Change |
|---|---|---|---|---|---|
| | Total | 60 (100.0%) | +36 | 1,464 (100.0%) | +904 |
| pack.framework.session | HttpGameSession | 29 (48.3%) | +29 | 696 (47.5%) | +696 |
| epack.framework.config | GameMode | 6 (10.0%) | - | 144 (9.8%) | - |
| epack.framework.config | ManagerProperty | 4 (6.7%) | - | 96 (6.6%) | - |
| epack.framework.config | GamePackManage | 3 (5.0%) | - | 72 (4.9%) | - |
| s.gamepack.framework | GameHandler | 3 (5.0%) | +3 | 48 (3.3%) | +48 |
| gamepack.minesweeper | MineSweeperMode | 3 (5.0%) | +3 | 144 (9.8%) | +144 |
| epack.framework.config | GamePackControll | 2 (3.3%) | - | 48 (3.3%) | - |
| epack.framework.config | GamePackGame | 2 (3.3%) | - | 64 (4.4%) | - |

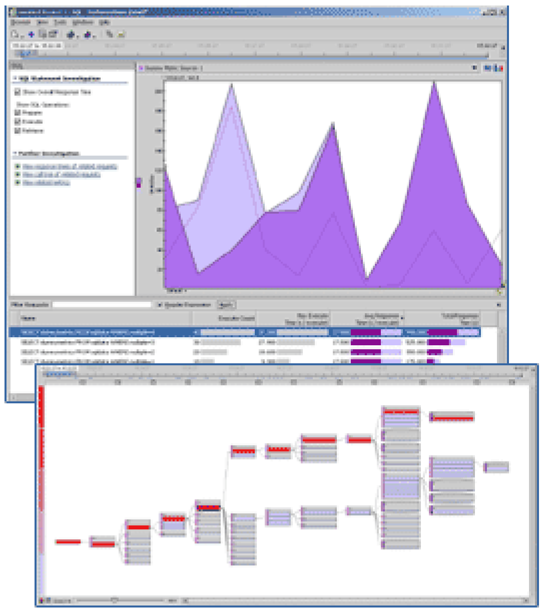
## Quest PerformaSure

While JProbe Suite can give developers great visibility into how the JVM is executing their code, clients developing J2EE applications need visibility into the performance characteristics of their key transactions across the entire J2EE stack. PerformaSure provides users with the ability to correlate system metrics with application server metrics and with transaction-specific performance metrics.

Offering an end-to-end transaction-centric view of performance, PerformaSure is a user-friendly, low-overhead performance diagnosis tool for distributed J2EE applications running in a production or test environment.

PerformaSure's exclusive Tag-and-Follow™ technology traces and reconstructs the execution path of end-user transactions across all the components of a clustered multi-tier J2EE system, allowing application administrators, DBAs, developers and QA to work together to diagnose and resolve performance bottlenecks quickly and easily.

PerformaSure is typically used in conjunction with load testing tools to understand the performance of a user's J2EE solution under realistic conditions. However, every company has performance issues that cannot be simulated in their testing and staging environments. For these issues, users can run PerformaSure, possibly with Quest's other production support tools, in their production environment to capture the information they need to quickly diagnose and resolve application issues.



PerformaSure allows users to easily visualize in which tier the performance bottleneck resides and even recreate the end user transaction path through the application.

## THE METHODOLOGY

Using JProbe Suite to bring best practices to performance tuning and analysis means using it early and regularly. Once a use case of the application has been implemented, the developer should first tune to ensure optimal memory usage. A heap analysis performed using JProbe Suite Memory Debugger will ensure that loitering objects don't stay in the environment longer than they should. Finally, the developer should performance tune with JProbe Profiler. It is at this point, when developers have completed all tuning and have implemented all changes for a particular use case, that that unit of code is passed to QA.

The most efficient methodology for attaining Java code that is reliable, high performing, efficient and free from error is to first ensure code correctness, then efficient memory usage, and finally performance enhancement. These steps bring best practices to performance tuning and analysis by making them early and iterative processes performed by those who know the code the best: the developers themselves.

PerformaSure is best used during QA/testing, most likely in a staging environment, to allow performance tuning of the entire application (looking at code, application settings and database performance related issues). Users also need to review their production environment to ensure that they have the right tools to monitor and collect performance information on their J2EE application, because performance issues do still occur in production even despite best efforts to identify and remove them during the development and testing phase.

## CONCLUSION

An investigation into an Airbus A320 crash that killed 87 people revealed that a computer design flaw was partly to blame. An $80 million satellite was lost in 1993 after a software error caused its thruster rockets to consume its entire fuel supply.

Not all instances of performance flaws generated by faulty code result in such extreme scenarios, but they all have consequences. As the market demands increasingly complex applications, and as the reliance by enterprises on Internet-based applications for mission critical activities rises, facing those consequences has to be an overriding concern for all development teams. And, the need to treat software as a strategic asset has to be a priority for organizations.

The most important step that companies can take is to shift their view of performance tuning from a responsive process to a proactive, anticipatory, preventative process. Ensuring that errors don't accumulate or get buried is key for keeping development costs down and ensuring users remain connected and satisfied.

Companies must embrace an iterative approach to performance tuning and analysis in which these practices are performed early and regularly throughout the development stage of the application life cycle. They must also ensure that solutions used during QA and production can collect the information required to help development teams quickly isolate the root cause of issues, ensuring greater uptime By placing the right tool in the hands of the right people at the right time, companies will improve their Java application performance and availability. Following this model not only ensures higher Java and J2EE application performance and reliability; it also permits a more rapid product release to clients.

## ADDITIONAL RESOURCES

For additional information on JProbe Suite, please visit: http://www.quest.com/jprobe

For additional information on PerformaSure, please visit:
http://www.quest.com/performasure

## TRAINING

Quest Software's Java performance experts are available to help. For more information visit http://www.quest.com/services or e-mail training@quest.com.

# ABOUT QUEST SOFTWARE, INC.

Quest Software, Inc. delivers innovative products that help organizations get more performance and productivity from their applications, databases and Windows infrastructure. Through a deep expertise in IT operations and a continued focus on what works best, Quest helps more than 18,000 customers worldwide meet higher expectations for enterprise IT. Quest Software can be found in offices around the globe and at www.quest.com.

## Contacting Quest Software

| | |
|---|---|
| Phone: | 949.754.8000 (United States and Canada) |
| Email: | info@quest.com |
| Mail: | Quest Software, Inc.<br>World Headquarters<br>5 Polaris Way<br>Aliso Viejo, CA 92656<br>USA |
| Web site | www.quest.com |

Please refer to our Web site for regional and international office information.

## Contacting Quest Support

Quest Support is available to customers who have a trial version of a Quest product or who have purchased a commercial version and have a valid maintenance contract. Quest Support provides around the clock coverage with SupportLink, our web self-service. Visit SupportLink at http://support.quest.com

From SupportLink, you can do the following:

- Quickly find thousands of solutions (Knowledgebase articles/documents).
- Download patches and upgrades.
- Seek help from a Support engineer.
- Log and update your case, and check its status.

View the ***Global Support Guide*** for a detailed explanation of support programs, online services, contact information, and policy and procedures. The guide is available at: http://support.quest.com/pdfs/Global Support Guide.pdf