



White Paper

metagroup.com



800-945-META [6382]

June 2003

Accelerating J2EE Application Delivery

Integrating Performance and Load Testing for Agile Development

A META Group White Paper

“The demand for software quality is increasing as organizations absorb new technologies and deal with schedule pressure. To meet demand and take advantage of business opportunities, software must have high reliability while being delivered in a rapid manner. When choosing tools to drive quality and productivity, it is critical to find those that work in a seamless fashion and that promote fast discovery and coordination across the development team. This report considers the links between load testing and application profiling and how to coordinate efforts across development teams.”



METAGROUP



Accelerating J2EE Application Delivery:
Integrating Performance and Load Testing for Agile Development

Contents

Development Becomes Collaborative	2
Increased Application Complexity	3
Integrating the Testing Cycle	4
<i>Borland and Mercury Interactive.....</i>	<i>5</i>
Mercury Interactive LoadRunner	5
Borland Optimizeit	6
Conclusion.....	6

Accelerating J2EE Application Delivery: Integrating Performance and Load Testing for Agile Development

Integration & Development Strategies

Thomas Murphy

Development Becomes Collaborative

As application architectural complexity rises, it becomes increasingly difficult to develop and deploy applications. In addition, most IT organizations are struggling with high maintenance costs (e.g., 60%-80% of typical IT budgets are spent on operations and maintenance) and difficult integration scenarios. Iterative development processes deliver increased productivity and improved quality by driving collaboration among developers, testers, and end users. Yet tools have frequently lacked support for coordination of efforts, which often reinforces traditional less-productive waterfall development behavior. Development tools are evolving to support more collaborative approaches toward application development that will greatly enhance team productivity.

Traditional development environments meld together code editors, compilers, user-interface designers, and debuggers to create an integrated development environment (IDE) that provides the core functions needed by application developers. Additional development life-cycle products for modeling, requirements, testing, and change management generally exist as separate applications. This enforces a rigid division of labor into separate silos: architects, developers, the QA team, business analysts, etc. Given a waterfall development methodology, with its sequential handoffs between subprocesses, this division was suitable. However, waterfall methods can lead to long project cycles and often produce applications that are overbudget, do not meet expectations, and are not flexible. Because of this (as well as a drive toward smaller projects with shorter lifespans), organizations have shifted to iterative techniques. Many are also exploring “agile” methods (e.g., extreme programming, Scrum methodology, feature-driven development).

Development environments will evolve to become increasingly collaborative and to integrate the application life cycle. This will enable greater productivity, especially when using an agile development method. Borland’s Together/J product (now incorporated into JBuilder) illustrated the value of integrating tools across the life cycle. By combining UML (Unified Modeling Language) and coding into a single

environment, developers can rapidly move between model-driven design tasks and editing source code. With combined editing and modeling, the developer does not have to switch context as models and code evolve. This increases the productivity of the developer and the value of the model.

Increased Application Complexity

Java 2 Enterprise Edition (J2EE) provides developers with many benefits. At the same time, however, it introduces numerous complications for development, tuning, and deployment of applications. Many of these complications are related not specifically to Java but to the increased complexity of an n-tier application architecture versus the client/server architecture. Deployed applications may have components running in different containers, serviced by different virtual machines, and running different versions of class libraries. Numerous options exist for caching and load balancing along with various parameters to control the behavior of the memory management system.

These issues create potential difficulties for testing and tuning. Developers often find that an application seemingly runs fine in the development environment, only to hear that the application does not scale in the pre-production test environment under load. Tools and techniques that may have been sufficient for testing and deploying applications in the client/server environment no longer suffice. Developers increasingly work with unit testing tools to ensure that functional specifications are fulfilled and use various profiling tools to help test and verify application behaviors. Profilers enable developers to understand how the system is using memory, the size and performance of routines, and other low-level performance issues. However, they do not provide support for understanding how the system is performing specific functions such as caching, load balancing, or operation of hardware infrastructure.

Furthermore, the J2EE architecture allows many options (e.g., bean- or container-managed persistence, JSPs, servlets, session beans, message-driven beans, etc.), each of which has implications for the deployment, performance, and scalability of the application. The ability to swap out system services (e.g., message brokers, caching and persistence frameworks) provides a great deal of flexibility but also increases the number of possible configurations. Developers using J2EE for n-tier transactional applications require tools that enable rapid development and testing of potential architectures and components.

Integrating the Testing Cycle

These various options and the resulting deployment complexity mean that a great deal of testing occurs during the development of an application. As developers build the code, they should also create unit tests that validate code correctness. In addition, various functional and integration tests may be performed by either developers or QA teams. These tests help to ensure that the system functions correctly as a whole and meets user requirements. This testing may also include usability tests, especially early in the development cycle as the user-interface elements are designed. Finally, as the application nears completion, teams begin to run performance tests designed to prove the overall scalability of the application suites and the projected load as well as to ensure that the system meets its designed service levels. All too often, these tests are performed very late in the cycle, when it is extremely difficult to identify issues and very costly to re-architect the application and fix problems.

Load testing tools generally can identify high-level issues that are causing problems from a deployment perspective. In some cases, these are issues that can be fixed during configuration of the application (e.g., additional servers deployed, increase in the size of caches), but in most cases, further testing must be performed by developers using profiling tools. Profiling tools provide a much lower level of detail, thereby enabling developers to understand memory usage, call paths, and the execution time of each instruction profile. Running profiles on code can be very time-consuming because of this level of detail.

However, if load testing tools are integrated with profiling tools, developers can rapidly drill down to explore execution issues. This enables applications to run under simulated production loads and, if a bottleneck appears, to step from load testing into application profiles. Because the load testing tool can determine potential problem areas, the developer is able to narrow the scope of profiling activity to reduce the time needed to collect and analyze the data. This also eliminates the back-and-forth steps that can occur when the QA team loads tests, finds a problem, and then must pass information back to the development team for it to try to chase down the source of the problem. Similar to the combination of modeling and code editing, combining load testing with profiling improves productivity by encouraging such testing to occur earlier in the process rather than waiting until the entire application has been put together. This can help to avoid the common situation of finding flaws in the basic architecture that would require the deployment configuration and application architecture to be redesigned.

Borland and Mercury Interactive

Borland has actively sought (as have all leading IDE vendors) to broaden its portfolio through acquisition (e.g., Optimizeit: profiling; Starbase: change management; TogetherSoft: modeling). In addition, it has formed a strong partnership with Mercury Interactive. This relationship provides integration of Mercury Interactive's LoadRunner with Borland's JBuilder IDE and Optimizeit products. Integration of these products enables a developer to start a J2EE server running while attached to Optimizeit and then place the load on the system with LoadRunner. If a performance bottleneck occurs, the developer can easily drill down from LoadRunner's high-level view to locate the more detailed execution information provided by Optimizeit. Because both LoadRunner and Optimizeit are integrated with the JBuilder IDE, developers are supported in working in an iterative "build a little, test a little" fashion. As the application grows, LoadRunner scripts can evolve to test new functional areas. Mercury Interactive has also integrated its TestDirector product with JBuilder to manage scripts. This partnership is an example of the continued extension of the integrated development environment into an integrated life-cycle and development environment (ILDE). This environment offers a programmer life-cycle control panel, where the once independent aspects of development (e.g., modeling, coding, change control, testing, deployment) will come together into an integrated development and life-cycle "portal", and support increasingly collaborative development activities. The entire development tool market is shifting to focus around the development of this ILDE.

Integrating tools provides increased value but still leaves developers working in isolation. Each member of the team may have a personal task list, but there is no ability for the project manager to see the state of these tasks. Adding collaborative capability would enable a coordinated workflow; as a developer completes creation of a component and marks a task complete, the system would add a task to add integration tests. Through 2003/04, vendors will focus on building integrated life-cycle management. By 2005, they will begin to add integrated workflow and project management integration.

Mercury Interactive LoadRunner

LoadRunner allows applications to be performance tested under production-level loads in the pre-deployment test environment. By employing integrated performance monitors, LoadRunner helps to identify and isolate performance bottlenecks across the multiple tiers and components of the application. LoadRunner includes support for the leading servers (application, Web, and database). By integrating with JBuilder, developers can create test scripts within the IDE, and launch load testing scenarios.

Borland Optimizeit

Borland's Optimizeit product line offers a set of tools to profile service and system performance of J2EE solutions. Optimizeit ServerTrace provides a service-level view of a Java application (e.g., JMS, JDBC, JSP, JNDI) and also allows for the collection of virtual machine-specific metrics (i.e., thread-specific system behavior). Optimizeit ServerTrace is also integrated with Borland's Optimizeit Suite, which is included as part of the JBuilder IDE and provides the ability to profile and debug specific thread and memory leak problems.

Conclusion

The dual drivers of iterative development and increased application complexity will demand increased integration of tools across the application life cycle. This integration will enable IT groups to work more efficiently and gain the benefits provided by iterative methods. Market leaders will provide seamless integration that not only boosts direct developer productivity but also enhances collaboration among development, testing, and operations teams. Through use of common tools across these groups, scripts and results can be easily shared and training efforts reduced. As organizations transition from first- and second-generation Web application efforts toward B2B integration via Web services testing, best practices and tools will be critical for ensuring that service-level agreements can be met. Improving developer tool integration will enhance the value of life-cycle products across development teams, resulting in higher productivity and increased software quality.

Thomas Murphy is a senior program director with Integration & Development Strategies, a META Group advisory service. For additional information on this topic or other META Group offerings, contact info@metagroup.com.



About META Group

Return On IntelligenceSM

META Group is a leading provider of information technology research, advisory services, and strategic consulting. Delivering objective and actionable guidance, META Group's experienced analysts and consultants are trusted advisors to IT and business executives around the world. Our unique collaborative models and dedicated customer service help clients be more efficient, effective, and timely in their use of IT to achieve their business goals. Visit metagroup.com for more details on our high-value approach.

