

CHOOSING A LOAD TESTING STRATEGY

Why and how to optimize application performance

Contents

Executive summary	3
What is load testing?	3
Why load test?	4
When to load test?	6
Strategies for load testing	10
Testing in the real world	16
The strategy for success	17
Load testing with Borland® SilkPerformer®	20

Executive summary

Poor software quality carries with it tremendous costs to an organization. Today, virtually every business depends on software for the development, production, distribution and/or after-sales support of products and services. According to a 2002 study conducted by the United States Department of Commerce, National Institute of Standards and Technology (NIST), poor software quality in the United States alone results in a cost to the U.S. economy of \$60 billion per year. ¹

Load testing is an important component in optimizing software quality, and when done properly can help mitigate the costs associated with poor quality. We will explore the importance of load testing, when in the software development process to load test, and how optimal performance can be achieved by proper load testing. In addition to discussing the various strategies for implementation, we will explore the very real benefits load testing returns to the organization.

Finally, we explain the proper strategy for realizing the full benefits of load testing - including best practices, planning for optimal performance, the suggestion that QA should really be viewed as Production Engineering, and how to extend performance assurance into production.

What is load testing?

Load testing is the systematic exposure of an application to real world, expected usage conditions in order to predict system behavior, and to pinpoint/diagnose errors in an application and its infrastructure before it is deployed. Load testing is used to analyze the following three aspects of an application's quality of service:

- Performance (response times)
- Scalability (throughput)
- Reliability (availability and functional integrity)

There are many types of load testing, each of which performs a given function. For example, some load tests might test peak stress levels for an entire application by simulating the maximum expected number of users of the application. Others might maintain a "steady state" number of users for days, as a way to uncover memory leaks. Still other load tests might stress a single application component, e.g. a middleware component such as a Web service, in order to verify its performance singly, before testing performance of the overall composite application. Regardless of the specifics, all load tests aim to make an application more reliable by identifying where, when and under what circumstances the application breaks.

¹ United States Department of Commerce, National Institute of Standards and Technology (NIST), "The Economic Impact of Inadequate Infrastructure for Software Testing", May 2002

Why load test?

Load testing is the complement to functional testing, which validates that an application provides the required functionality. Functional testing can validate proper functionality under correct usage and proper error handling under incorrect usage. It cannot, however, tell you how much load an application can handle before it breaks or performs improperly. Finding the breaking points and performance bottlenecks, as well as identifying functional errors that only occur under stress requires load testing.

All business decisions are ultimately monetary judgments, and the justification for load testing is no different. In this case justification is achieved through the following two major benefits:

- Reducing the number of application failures, in terms of both performance and actual errors or even outages
- Lowering costs for the application infrastructure

Reducing the number of application failures

Load testing, implemented properly and performed throughout the development cycle, will significantly reduce the number of failures experienced by an application. The number of avoided failures will vary according to application complexity, developer skill and application usage, but if a tool prevents even a handful of such failures, it can pay for itself several times over. This is especially true if the application is a mission-critical and/or customer facing one used to generate revenue or interact with an external audience in some manner.

According to Gartner, the average cost of unplanned downtime for mission-critical applications is \$100,000.² However, it is critical for businesses to understand that the true cost of failure goes far beyond missed revenue opportunities during an outage. To fully recover from a failure, businesses must “clean up” after an outage, a process that can cost many times the lost revenue.

To make this point more relevant, simple calculations can be performed to illustrate the costs inherent in poor performance. Besides the obvious hard costs, there are also softer costs such as brand image, customer satisfaction, competitive gain and other factors. These are some of the costs of poor performance.

Immediate lost revenue or productivity (revenue per minute) x (minutes of downtime)

A customer facing Web application that transacts \$100,000 of business per hour and is down for 30 minutes would incur direct revenue losses of \$50,000, should an outage occur. If alternate transaction channels (e.g., a traditional call center) are available, businesses should subtract any overflow business received in the call center from the lost revenue and add any extra costs. For example, if the same organization handled an additional 100 orders, each worth \$50, in their call center during the application outage and each call carried \$5 more overhead than a Web transaction, the total lost revenue would be \$45,500 (\$50,000 - \$5,000 + \$500). The greater a company’s dependence on the Web, the greater the potential revenue impact of an outage. From this example, one can see how revenue losses multiply quickly the longer the application is unavailable.

eBay, in its early days, lost \$3-\$5 million in revenues and suffered a 26 percent decline in their stock price as a result of a single, 22 hour site outage.³ eBay understood quickly the very real impact of poor performance on their bottom line and today utilizes a very robust internal performance center to thoroughly test software prior to deployment.

² “Quality Development is Crucial to Emerging Technologies”, Theresa Lanowitz, Gartner Group, April 23, 2003, Research Note K-19-8371
³ Source: Gartner Analysis

If an internal application, for example an ERP system, goes down, employee productivity is lost immediately. If the employees affected are dependent on the application in order to perform their jobs, the costs for downtime quickly grow. On top of the loss in productivity, the costs incurred to the business have to be considered. For example, if an order processing system goes down at the end of a quarter due to high loads, orders may not be able to be processed and revenue may not be recognizable within the quarter. Order processing system availability was widely reported to be an issue for Hewlett-Packard's Enterprise Servers and Storage Group in August 2004, at the end of a quarter. Carly Fiorina, HP's CEO, attributed a quarterly loss of about \$400 million in revenue and \$275 million in operating profits to the glitch in HP's order processing systems.⁴

Customer abandonment

A Jupiter Media Metrix survey of more than 2,000 users (shown in Figure 1 – Jupiter Media Metrix Consumer Study, below) found that a full 46 percent of users permanently or temporarily abandoned their relationship with a preferred site because of technical difficulties or performance problems.

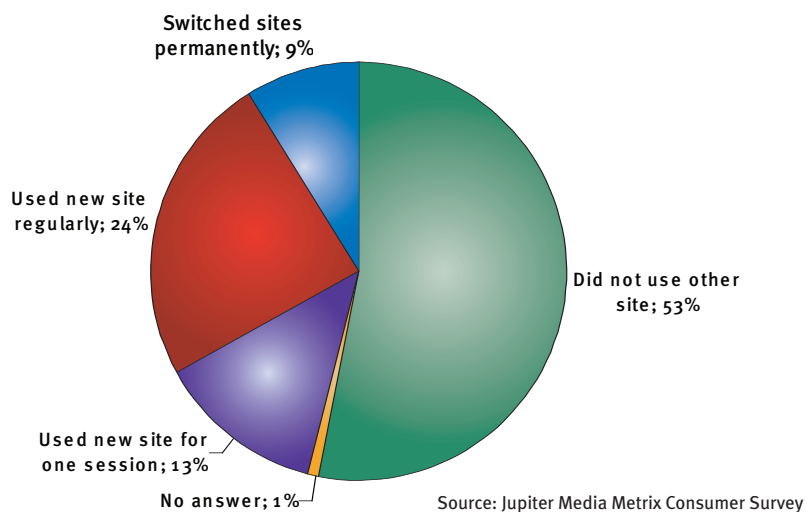


Figure 1 – Jupiter Media Metrix Consumer Survey: 46 percent of users have temporarily or permanently abandoned a preferred site as a result of a technical or performance problem⁵

As shown, 13 percent of users left the site for only one session, 24 percent established a parallel relationship with a competitive site and nine percent abandoned the site permanently. Re-establishing relations with the first 13 percent of users should be fairly inexpensive – an apology email may be sufficient – but winning back the 24 percent of users who surrendered their exclusivity could mean expensive measures such as competitive discounting, free shipping or other incentives. In one such instance, after a daylong service outage, AT&T issued \$40 million in rebates to its customers.⁶ Finally, the nine percent who left permanently are gone forever, and must be replaced through standard, costly customer acquisition initiatives. It is important to remember that a site needn't fail outright for customers to abandon it. One research study discovered that for certain users, even a 10 percent degradation in response times led to a five percent increase in customer abandonment.⁷

⁴ "HP Puts Part of the Blame on SAP Migration", Marc Songini, Computerworld, August 17, 2004: <http://computerworld.com/softwaretopics/erp/story/0,10801,95276,00.html>

⁵ "Testing Tools and Methodologies", Foster and Allard, Jupiter Media Metrix, January 1999

⁶ Source: Gartner Analysis

⁷ "Tying Performance to Profit", Peter Christy, Jupiter Media Metrix, June, 2001

Internal end-users who aren't given a choice of services may simply refuse to use an application rather than tolerate sub-standard performance. Therefore, software performance optimization is vital for satisfying customer usability needs—whether internal or external customers—and keeping up with (or surpassing) competitors.

Incident-specific costs Depending on the type of application and failure, businesses may incur substantial costs in other areas, as well. Finding thousands of dollars for immediate system repairs and upgrades can be painful. If a business is obligated to provide a certain level of service to its customers, there may be financial or legal costs. If the failure becomes public knowledge, PR and marketing budgets may have to be realigned to defend against negative press or attacks by competitors. If a denial-of-service attack or other assault was responsible for the failure, a security assessment and response could cost hundreds of thousands of dollars, as well.

Lowering costs for the application infrastructure

Applications that are not tuned for optimal performance may require a lot more system resources than those that are. Further, if an application's system requirements are not known up-front, underlying system infrastructure often gets oversized to ensure performance requirements are met. This results in both unnecessary hardware and ongoing maintenance costs – and such costs can quickly become significant. Unnecessary investments can be avoided with adequate load testing through application tuning and capacity planning. As an example, one well-known provider of accounting software was able to triple application performance utilizing their existing infrastructure—and thus significantly save expense they would have incurred to purchase additional hardware—as a direct result of load testing activities.

When to load test?

The earlier load testing starts in the application development process, the earlier software defects or architectural problems will be found. Knowing that the cost for correcting issues grows exponentially for each downstream phase of the development lifecycle in which they remain undiscovered,⁸ it is imperative that load testing start as early as possible. Given today's typical multi-tier applications with separate tiers for presentation, business and data logic, as well as the often necessary layer for the integration of legacy applications, the following phases in the development process are depicted below, in Figure 2 – Phases in Which Load Testing Activities Can Be Leveraged:⁹

1. Component-level stress tests
2. Infrastructure load tests
3. Architecture load tests
4. End-to-end load tests

⁸ Software Engineering Economics, Barry Boehm, Prentice-Hall, 1981

⁹ "Best Practices for Web Application Deployment", Ernst Ambichl, Segue Software, Keynote Total Performance Management Symposium, March 18, 2004

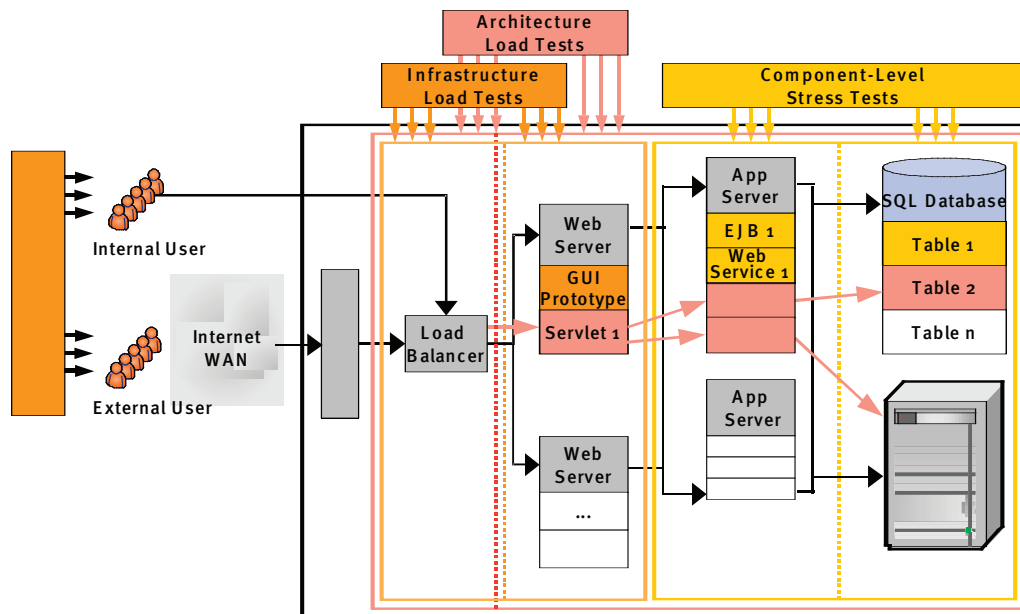


Figure 2 - Phases in which load testing activities can be leveraged

Component-level tests

Unit testing is already a well-accepted activity to verify functionality of distributed software components early in the development cycle. However, software application components living on the server (for example, Enterprise Java Beans™ (EJB™s) used to host business logic or Web services providing access to data maintained in legacy applications) are accessed by multiple clients at the same time. Hence, classic unit testing is not sufficient. Only a stress test on these components (as shown in Figure 3 – Component-Level Stress Tests) is able to issues easily and cost efficiently identify typical problems such as deadlock situations, synchronization problems, memory leaks, performance problems or architectural issues. However, diagnosing those issues within a classic end-to-end load test just prior to deployment is more difficult and, at that late stage, it is also much more expensive to resolve any issues identified.

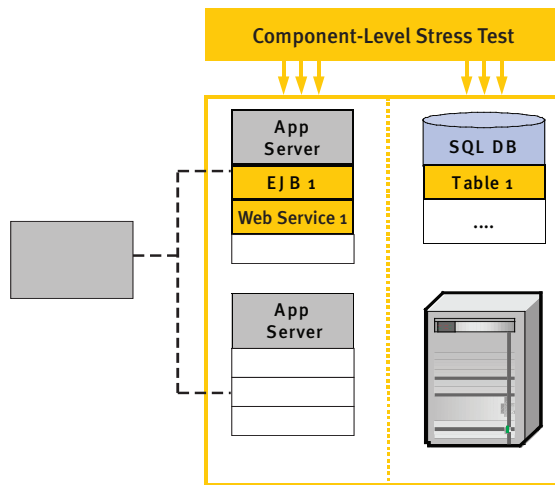


Figure 3 - Component-level stress tests

Compared to all other load testing phases, component-level tests are unique, because they are usually done when there are no clients yet available from which to record a test script. Hence, test scripts must either be reused from unit testing or built manually from scratch.

Infrastructure load tests (benchmarking)

Usually, decisions about the hardware/software infrastructure for a software project are made early in the project. Nevertheless, the chosen infrastructure—which may include a load balancer, Web, application and database servers, related server hardware and operating system—is a major factor in determining the performance, scalability, reliability and costs of the application to be deployed. Hence, all the available infrastructure options should be carefully evaluated, weighing both performance and cost.

Official benchmark numbers often do not help in such an evaluation as they are typically only available for single components – they can't take the architecture of the application into account. In contrast, early load tests and benchmark results for the various application infrastructure options (shown in Figure 4 – Infrastructure Load Tests) can provide valuable guidance about which option to choose. They also can be used to influence decisions about the application architecture (e.g., either J2EE or .NET). Also, understanding the effects of the various system configuration settings provides early information for system tuning. Finally, knowing the relevant performance indicators provides a valuable benchmark for later end-to-end load tests and application monitoring.

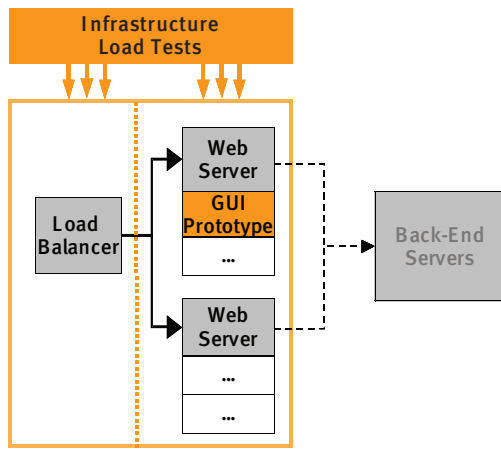


Figure 4 - Infrastructure load tests

Architectural load tests (benchmarking)

Early load tests on the application architecture can be run to verify that application components residing in the various tiers work together as expected.

Using an “All-Tier” prototype that includes a small subset of the complete functionality touching all tiers (such as shown in Figure 5 – Architectural Load Tests), tests can be performed early in the development process in order to detect design flaws. As stated previously, errors found early in the development process are far less expensive to resolve than those found in later phases. Even different design alternatives, such as the distribution and replication of the presentation, business and data logic, can be evaluated.

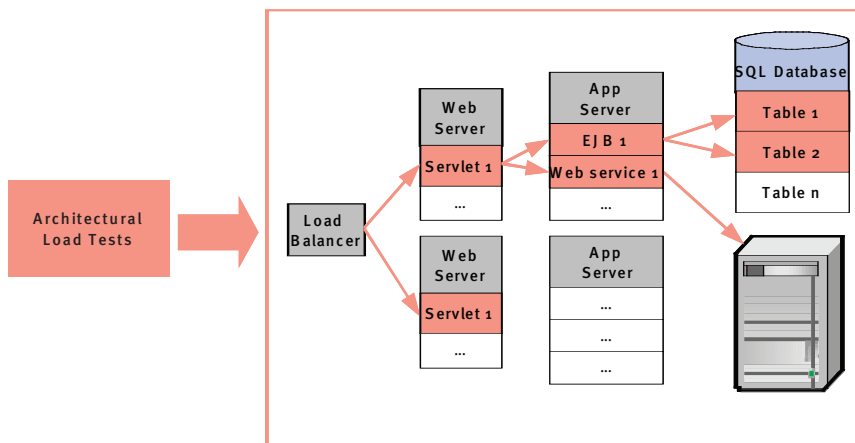


Figure 5 - Architectural load tests

End-to-end load tests

End-to-end load tests (shown in Figure 6 – End-to-End Load Tests, below) are classic load tests that analyze the entire application under various realistic, end-user workload scenarios that may last several hours or even as long as several days.

These kinds of tests are usually performed in staging environments and the results are used to answer questions like:

- Are there functional errors that only occur under load conditions?
- What system capacity is needed across all tiers?
- Will the application meet defined service levels?
- Is the application tuned for optimal performance?
- Has there been any negative performance impact as a result of error resolution/tuning activities or from the introduction of additional functionality?
- Is the application ready for full deployment?

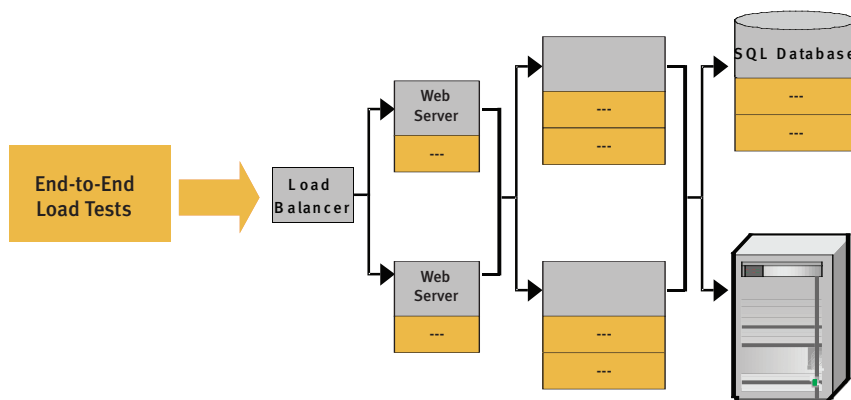


Figure 6 - End-to-end load tests

Strategies for load testing

Having now defined load testing, explored the reasons it is important and discussed when to load test, let's explore the various load testing strategies available to an organization. There are a number of strategies businesses use for load testing, though only a few are viable. Typically, the load testing strategy is driven by available budget, rather than the business need or criticality of the application. This is not ideal – because, as with anything, you get what you pay for.

This section independently assesses each load testing strategy and provides the pros and cons of each. The main strategies employed for load testing are:

- Manual load testing
- In-house developed load testing applications
- Open-source load testing tools
- Testing frameworks integrated within IDEs
- Web-only load testing tools

- Hosted load testing services
- Enterprise-class load testing solutions

Manual load testing

We have already covered many of the reasons for load testing and illustrated some of the very real consequences that can occur when an application does not undergo load testing prior to deployment. It is worth mentioning, however, that businesses performing manual load testing could be lumped into the same group as those who do not test at all. Manual load testing is an inefficient use of time, money and resources. Manual load testing does not produce repeatable results, cannot provide measurable levels of stress on an application and is an impossible process to reliably coordinate. It is also not a process that is easily scalable. Finally, manual load testing cannot automatically leverage tools to aid in the diagnosis and root cause identification of performance issues. The limitations of manual load testing are illustrated in Figure 7 – Manual Load Testing is Problematic.

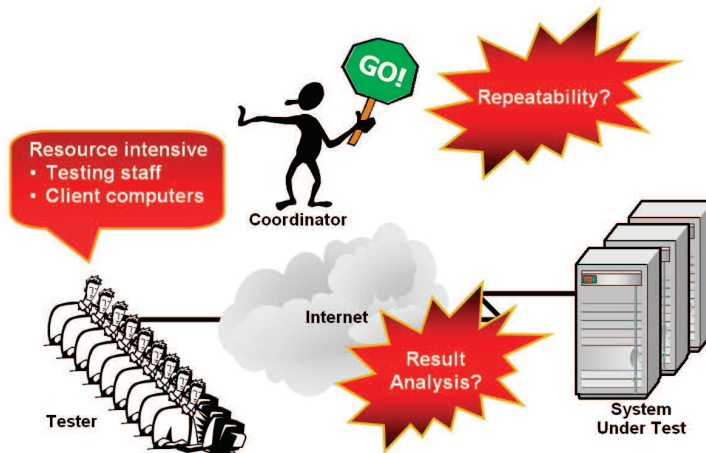


Figure 7 - Manual load testing is problematic

For an example of the limitations involved in manual load testing, take the case of a retail Web application seeking to support a modest volume of 50 concurrent transactions. Testing this level of usage in a manual load testing environment would require 50 individual users on 50 computers, initiating tightly-scripted transactions at the same time. The users would have to proceed through the transaction at fixed speeds while somehow simultaneously logging every error found. When a test needs to be rerun, the same 50 users would have to perform exactly the same test with exactly the same timing. Even with five users, this is not practical, and load tests involving hundreds or thousands of virtual users would be impossible (as well as extremely expensive) to perform. Also, once a problem is found, diagnosing its root cause without any diagnostic capabilities to assist the performance tester is another obstacle encountered with manual testing. Despite these limitations, many businesses insist that “having everyone test the application before it goes live” is sufficient. Unfortunately, applications deployed in this manner will not ultimately be reliable.

Home-grown testing applications

Many development managers understand the value of load testing but lack the budget necessary to procure a load-testing application. Sometimes a decision is made to build a load-testing application in-house. While well intentioned and better than nothing, this approach also has a number of drawbacks.

Application coverage

In most homegrown testing applications, scripts are written to test very specific functions. For example, a developer might write a script that makes rapid, concurrent updates to a database table. Written properly, these “point solution” scripts can test an application’s ability to handle a single action, but they do not easily measure an application’s ability to handle the complex mix of transactions created by a live user. The only way to adequately test real usage is to model, record and play back different, concurrent user scenarios that are representative of the anticipated user load. Unless the developers undertake the tremendously expensive process of building a load-testing solution from the ground up, meaningful testing is nearly impossible.

Tool specificity

Since homegrown scripts are built to test a specific function within a specific application, they cannot easily be altered to test other functions, applications or technologies. The process of altering or porting these scripts to fulfill another need can often take as long as writing an entirely new script from the ground up.

Staffing specificity

Each script is typically a self-contained unit with unique logic, error-reporting mechanisms and even unique authors. This makes it extremely difficult for business users — who are most knowledgeable about application functionality — to understand and work with the code. So while a commercial load-testing application might allow a business analyst to model user behaviors through a graphical user interface, with an in-house developed tool that task is left to the developer — who may not be the person most knowledgeable about application functionality. Even other developers may have trouble understanding a piece of code if the original author did not comment it properly, leading to serious knowledge gaps if a developer should ever leave the company, mid-project.

Poor integration

As mentioned above, there is generally little coordination among disparate testing scripts, and it is extremely difficult to produce any standardized reporting across tests. Even for a single test script, it is difficult to conceive of the possible ways a business user or QA manager might want data formatted and reported.

For these reasons, homegrown testing solutions are generally not practical. By the end of a development cycle, they have often cost more than a packaged solution, and ultimately, the organization may be deploying an application that is still not ready for deployment. Further, the lack of reusability in a homegrown testing solution renders the testing process almost irrelevant when future releases or entirely new products need to be tested. Development managers who feel a homegrown solution is “all I need right now” should reconsider and look for ways to cost justify a load-testing solution. As we have already seen, the ROI to an organization provided by a load-testing solution is very real when weighed against the risks of poor performance.

Open source load-testing tools

There are several load-testing tools available in the open source industry free of charge. Only a few of them provide a script recorder that allow testers to easily record user actions into a script through a GUI and replay the scripts simultaneously as a load test. Many others offer a very limited scripting interface, which is also not extensible. While these tools may satisfy very basic needs for simple Web application testing, they lack many capabilities that are absolutely necessary for load-testing mission critical applications. According to eWeek,¹⁰ “the main weaknesses in these tools are their lack of advanced scripting options for testing complex applications and their often-limited reporting options.”

Beyond scripting and reporting, those tools have many other shortcomings that prohibit their adoption for testing mission-critical applications. The lack of support for non Web/HTTP technologies makes their use impossible for testing many standard enterprise application environments, such as ERP/CRM applications, Citrix deployments, client/server applications, distributed applications with a native Windows® or Java™ client and wireless or custom TCP/IP based applications, such as a mainframe accessed by a terminal. Also, many lack the capability to provide early component-level stress tests of middleware or database implementations. Such early testing can reduce the cost per defect uncovered significantly, as compared to testing later in the development process.

Due to the lack of high-level scripting APIs, scripts also become very long. This, in turn, makes the scripts difficult to maintain. Many of the open-source tools are also based on Java, a technology that carries significant performance overhead. This, in turn, leads to higher hardware/resource costs for maintaining multiple load-testing driver machines.

Open-source tools typically cannot accurately emulate a Web browser user session due to a lack of functionality in several critical areas, such as: modem speed emulation, caching, full functional cookie support, IP spoofing and user-based connection handling (threading). This incomplete functionality leads to inaccurate test results, which makes their usage in Web load tests — especially for mission-critical applications — questionable.

Open-source tools are generally point solutions that focus only on stressing a Web site or application. They do not offer an integrated solution for functional/regression testing, application performance monitoring and management, and overall test management. Finally, open-source tools also lack a supporting ecosystem to provide the necessary technical support and services to help organizations successfully implement load testing.

Testing frameworks integrated into IDEs

A new trend in the software industry is that of “Mega-IDEs”. Several integrated development environments (IDEs) have started to include tightly integrated testing tools – including load testing. The most well known examples are Microsoft® Visual Studio® Team System/Test in the Microsoft .NET world and Eclipse/Project Hyades in the Java world. Their strength is that they provide “test-driven development” within their respective IDEs. Developers do not have to learn to use a new tool in order to perform testing – they can work within an already familiar environment.

However, the Mega-IDE’s greatest advantage is also their greatest disadvantage: they only provide a development centric view. They assume developers are performing all coding, unit, load and production testing activities. Their view of the application lifecycle maps only the perspective of a developer – QA and IT needs are left out.

Colin Doyle, Integrity Solution product manager at software configuration company MKS Inc., a member of the Eclipse/Hyades project, said, “No QA or senior manager is ever going to fire up Eclipse or Visual Studio. They are developer frameworks.”¹¹

¹⁰ “Open-Source Testers Offer Low-Cost Alternatives”, Jim Rapoza, eWeek, August 11, 2003: <http://www.eweek.com/article2/0,3959,1216342,00.asp>

¹¹ “Moving Toward the Mega-IDEs - Tool Makers Offer Their Support, But Will Everyone Want Them?”, Jennifer deJong, Software Development Times, September 15, 2004: <http://www.sdtimes.com/news/110/story3.htm>

Additionally, these testing frameworks only offer support for their respective application environments — .NET or Java — and provide very rudimentary functionality for load testing Web applications. None of them provide support for any other application environment.

While these testing frameworks may be a good solution for small projects where developers are also performing load-testing activities, they clearly do not fit the needs for load-testing mission critical, heterogeneous or legacy application environments. Also, none of these testing frameworks can cover all aspects of testing – e.g. functional testing, as well as application performance testing in an integrated fashion. And the testing capabilities they offer certainly don't extend into production.

Web-only load-testing tools

There are a couple of special-purpose tools on the market for load testing Web applications which cost significantly less than an enterprise-class load-testing solution. Of course, the limitation in testing only Web applications can have serious drawbacks. Once the need for load testing non-Web applications comes up, which is fairly common in today's heterogeneous application environments, a new tool must be procured in order to complete the task. However, introducing a new tool means introducing other new costs above and beyond the cost of the tool. Employees must be trained on the new tool, maintenance must be paid and implementation services may be needed. Existing knowledge of the Web-only load testing tool cannot be reused for other non-Web projects; this means a significant cost overhead, compared to using a single enterprise load-testing solution to cover all the different application environments and technologies.

However, even when just a Web application needs to be load tested, Web-only load-testing tools share a couple of significant disadvantages compared to an enterprise-class testing solution:

Limited scalability and accuracy

Web-only load testing tools all use a common approach: driving Microsoft's Internet Explorer® for replaying test scripts as virtual users. As a result, Web-only load-testing tools are either accurate or scalable. However, they cannot be both accurate and scalable because of technical restrictions – Microsoft did not architect the Internet Explorer library for multiple concurrent usage. Thus, Internet Explorer is not an ideal foundation upon which to build a load-testing solution. Generally, such tools achieve high accuracy when driving the complete GUI-less instance of Internet Explorer. However, in this mode (thick replay mode) they are not even able to run a few concurrent virtual users, because the resource footprint is simply too high. This means that thick replay mode can only reliably be used for functional testing, not for load testing. To achieve a somewhat higher level of scalability, one must reduce the resource footprint per virtual user. The only way to do this is to directly drive Internet Explorer's Internet/HTTP communication library (WinInet.dll), thus bypassing all the overhead that makes up the complete Web browser, including connection pooling, caching, cookie management, etc. However, the accuracy of such an approach is questionable, as it forces different virtual users to share common resource pools which is not an accurate emulation of real world usage. Thus, again, load test results are not an accurate representation of the production environment.

Limited browser compatibility

Web-only load-testing tools work with Microsoft's Internet Explorer; other Web browsers such as Mozilla, Firefox, Netscape or Opera are not supported at all. Hence, a realistic user distribution in the test modeling may not be achieved.

Limited reusability

One of the biggest weaknesses of Web-only load-testing tools is generally found in the limited functionality of their scripting language. Only focused on a visual scripting approach, they lack many basic concepts that every programming language

provides such as modularization and scripting (e.g., loops, if-else, etc.). This makes reuse of scripts for other projects very difficult, if not impossible, to achieve.

Lack of support for component-level testing

Web applications are often built using distributed components such as EJBs, .NET middleware components and databases. The earlier such server components can be tested to verify functionality and performance under realistic server conditions, the lower the costs for correcting any defects found. While enterprise load-testing solutions are fully capable of performing such early load tests — because they directly support the client interfaces of technologies such as Java RMI, .NET Remoting, or ODBC — Web-only load-testing tools are not.

Hosted load-testing services

Hosted load-testing services load test Web applications from a remote server, typically over the Internet. This approach requires no special hardware, software or staff on the organization's part, and can be an excellent way to obtain an introduction to the benefits of load testing.

The only caveat in using hosted load testing as a regular part of the development process is that load testing should be a constant process. Load testing should be repeated often throughout the development cycle on both individual application components and entire application environments. Some application components are not accessible via a public network, and testing them through a hosted solution is therefore difficult or impossible. Even those applications that are Web-accessible still need to be tested frequently; frequent testing utilizing a hosted service can quickly become an expensive proposition. Additionally, since the behavior of the Internet is unpredictable, hosted load tests may not be as repeatable as those run internally. Hosted load tests can provide a proof-of-concept that validates internal testing, but to truly provide adequate coverage, load testing must happen early and often from the inside – not solely in production, once an application has been deployed.

Enterprise-class load-testing solutions

The enterprise testing solution is so-named because it provides the only type of load-testing support that adequately meets the needs of enterprises. More than just a capture/playback load-testing tool for Web application environments, an enterprise testing solution is highly scalable and is able to load test heterogeneous application environments at a reasonable cost. By also supporting component-level stress tests, enterprise-class load testing can help detect performance defects earlier in the software lifecycle, thus reducing the cost to fix them.

Visual test design tools not only flatten the learning curve, but also provide ease-of-use for non-development personnel. Full flow control over virtual user activity and support for modularization through an advanced scripting language provide flexibility and high reusability of test scripts for multiple test scenarios. High-level script APIs above TCP/IP for supported application environments guarantee high maintainability of test scripts. Accurate user emulation and error detection ensure valid test results that accurately predict application behavior in production. Powerful reports help you analyze load test results quickly, while advanced diagnostic tools pinpoint the root cause of a problem, enabling prompt resolution.

Vendors that provide enterprise-class load-testing solutions not only offer the load-testing software itself, but also the option to buy complementary tools for covering the complete software application lifecycle, such as test management, defect/issue management and application performance management. Therefore, quality can be addressed holistically, throughout the entire software application lifecycle. Additionally, the vendor will typically offer training and consulting services to ensure a

successful implementation of the load-testing solution. Amongst enterprise load-testing products, there are certainly notable differences, but enterprise load testing is far more useful than the other options discussed previously.

Testing in the real world

The one constant objection that has inhibited the adoption of an enterprise load-testing solution is price. While an enterprise load-testing solution can quickly show very real ROI to the organization — often within a single development cycle — many managers have not planned for such an investment.

The importance of testing to the organization

Testing is still all too often an afterthought. Timelines are short, budgets are inadequate, and in many organizations Quality Assurance (QA) enters the application development process only at the very end. Numerous studies have shown this is not the optimal way to produce software – it is the most expensive and least apt to result in the deployment of a high-quality application.

Business project managers want their projects to succeed, and usually only short-change testing because they lack budget and quantitative information to substantiate ROI returned to the organization by testing activities. In the absence of quantitative metrics, bringing QA in during product planning or spending tens of thousands of dollars on test tools can be a waste of time and resources. Likewise, holding up a product launch because a tester has a “feeling” something isn’t working can incur resource and opportunity costs to the organization. One way to prove the value of proper testing is to use a testing tool suite that improves the product and provides measurable value to the business manager on a regular basis in the form of metrics to document quality levels in the application. This includes hard, reproducible evidence of application defects and system status reports containing quality metrics that can be used to make informed decisions about resources, spec revisions, and launch dates.

Getting the cart behind the horse

An enterprise load testing solution can easily be justified after it has been implemented. But how can QA managers find the money for a load-testing solution in the first place? Here are some suggestions to assist managers in procuring a load-testing solution:

Find the right budget(s)

Rarely will a single project budget allow for any unplanned additional expenditures. For instance, a new lead generation application for marketing would probably not include budget for a new application server, even if the application server would help the cause. However, investment in a testing solution has broad applicability and can provide utility to nearly all software development projects. Therefore, organizations should try to spread the cost of a testing solution across multiple projects. Business project managers may be more willing to sponsor a fraction of the solution cost to get the reliability it can provide.

Start modestly and grow

Incomplete testing is often worse than no testing at all. Even if a budget does not allow for an enterprise load-testing solution, businesses should avoid the temptation to purchase a low-end tool. Enterprise-class load testing solutions offer value that low-end packages do not, so there will never be price parity. Starting with an enterprise load-testing solution that can be upgraded as performance testing becomes more and more embedded in the application development process allows a business to begin testing the “right” way – with the flexibility to migrate to more comprehensive functionality later, in a cost-

effective manner. Most load-testing vendors price their solutions by the virtual user. Even a small number of virtual users, used early in the development process, can often cost-justify an automated load-testing solution. Further, budgets might open up for future upgrades, particularly as the organization realizes the benefits of performance testing. Some vendors offer entry-level load testing solutions at a lower price range – but be careful that you are not locked into a low-end solution. Starting with an entry-level offering only pays off when all the solution knowledge and scripts can be re-used in the enterprise-class tool, once an upgrade is desired.

Log your costs

Regardless of your situation, remember to document every testing-related cost you incur. Over time, manual efforts, home-grown script design, missed defects that make their way into production and the like add up, and after a well-documented development cycle without a load-testing solution, the case for acquiring one can usually be made.

The strategy for success

Load and performance testing is an indispensable activity for optimizing the quality of a software application. Companies that run mission-critical applications cannot afford the high cost of application failures and unnecessary infrastructure maintenance overhead – particularly if the applications are visible externally and used for revenue generating purposes. Additionally, failures experienced by external audiences leave an organization even more vulnerable to competitive pressures. To fully leverage the benefits of load testing, it is important that an organization view quality holistically, across the entire software application lifecycle – including the production environment. As explained previously, the earlier load testing starts in the application life cycle, the higher its return on investment.

Plan for optimal performance

It is important to note that the goal of load testing shouldn't be to build the fastest application around – the goal must be to optimize an application, balancing both cost and performance. To optimize the performance of an application you need to know the service level at which the end-user is satisfied. Usability studies and end-user behavior analysis during the beta, staging and production phases help identify optimal performance. Each application, module or page should be evaluated and tuned separately for optimum cost/value. Below, Figure 8 – Optimum Performance, illustrates how time and money can be wasted once the tuning of an application extends beyond the point at which end-users are satisfied during peak loads.¹²

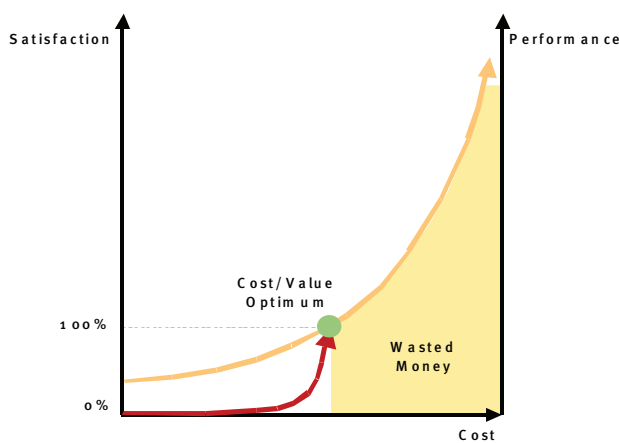


Figure 8 - Optimum performance

Figure 9 - Optimum Performance-to-Tuning Cost Ratio per Module (below), illustrates how adhering to the 8-second-per-page rule doesn't automatically lead to success. It's necessary to analyze and optimize applications module-by-module and page-by-page. Frequently used pages with minimal end-user interaction (for example, the clicking of a single hyperlink) require better response time than do infrequently used pages that involve complex input.

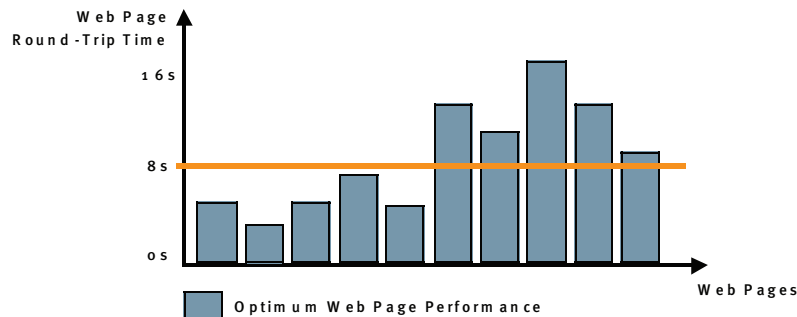


Figure 9 – Optimum performance-to-tuning cost ratio per module

View quality assurance as ‘production engineering’

In the manufacturing industry, products would never be brought into production directly from the product design studio without some form of production engineering in between the two functions. Yet, in the software industry, it is not at all unusual for an application to go through minimal testing — the equivalent of production engineering — prior to its deployment into production. Insufficient testing and poor communication between development and production result in many application performance issues, post-deployment.

We have referred to Quality Assurance as the gatekeepers for software quality throughout this paper. However, a recent Best Practices brief published by Jean-Pierre Garbani, Forrester Research, proposes that QA be viewed as Production Engineering. Garbani’s concept definitely has merit and adopting a view of QA as Quality Engineering — even if not adopting the title — can result in improved quality and ROI for an organization. Let’s explore this concept a little more closely.

Most Quality Assurance groups have a very close proximity to development, but lack a similar strong connection to production. While the close connection with development is appropriate as far as functions and features are concerned, the weaker connection with production is hardly the best way to verify the compatibility of a new application with the production environment and ensure that quality and performance expectations will be met. Thus, QA should be transformed into a Production Engineering group, which also focuses on Performance Assurance, as shown below in Figure 10 – Quality Assurance Becomes Production Engineering. The authority of QA should extend to rejecting any application that does not meet production standards – whether in design, architecture, configuration or performance areas. The role of this group goes far beyond purely testing: it is the essential bridge between development and production.¹³

13 “Performance Management and the Application Life Cycle”, Jean-Pierre Garbani, Forrester Research, February 11, 2005

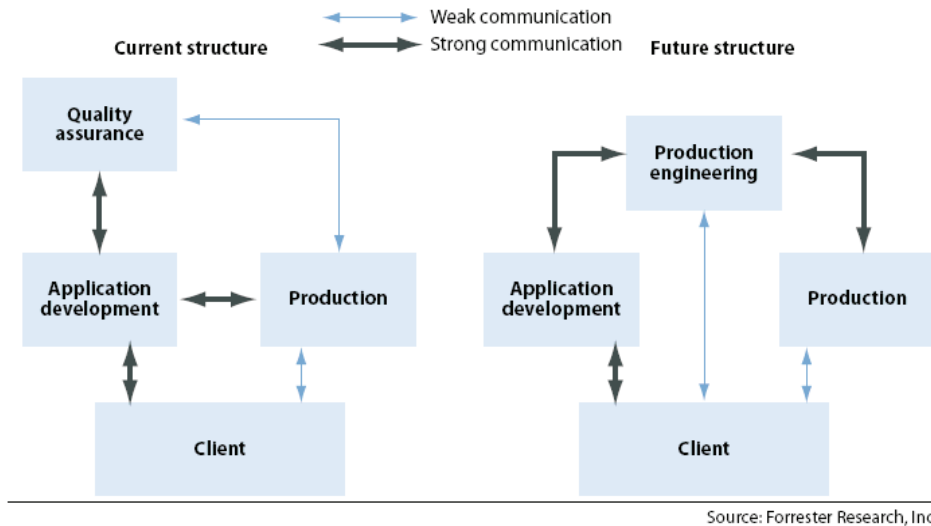


Figure 10 – Quality assurance becomes production engineering¹⁴

Extend performance assurance into production using application performance management and closed-loop testing

Application Performance Management (APM) is the counterpart to load testing, once the application is deployed into production. APM measures application performance, availability and accuracy under actual loads in the real world, and provides diagnostic tools to proactively identify and quickly resolve problems once they occur.

As pre-production load testing and APM have a lot in common, assets developed in earlier load testing phases should be reusable later in production. Similarly, feedback (errors, information, etc.) from production should be able to be fed back into pre-production for load tests that reflect real-world usage conditions. This process is called “closed-loop testing” and is shown in Figure 11 – Closed Loop Testing.

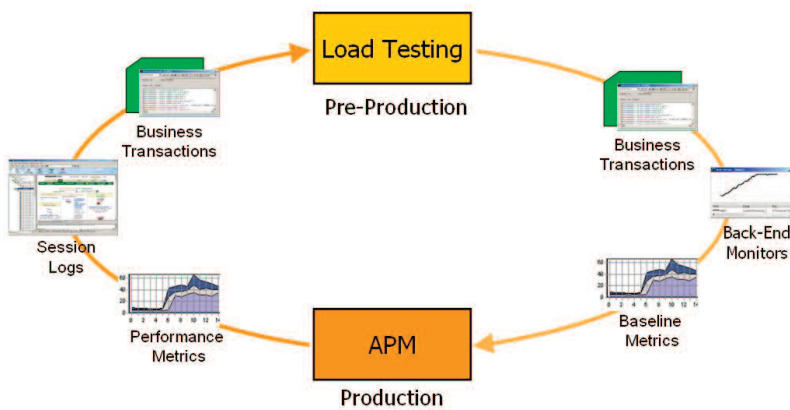


Figure 11 – Closed-loop testing

Closed-loop testing not only provides a way to leverage already existing test assets used in pre-production testing for APM, but also for transferring knowledge between R&D, QA and production. When the feedback loop drives production assets back into QA, it also allows for the validation of load-testing results through real-life usage data. Furthermore, baseline metrics derived from pre-deployment load testing can be used as threshold values that proactively flag a potential performance problem once the application is deployed.

Additionally, the closed-loop approach increases load testing efficiency. Performance metrics measured in production can become performance standards for the next software release cycle. Captured input values from real-users can be used as test data for realistic data-driven testing, moving tests closer and closer to real-world usage scenarios. The result is better test coverage, more realistic test scenarios and more accurate performance metrics. Recorded sessions of actual users encountering an error situation can be fed directly back into pre-deployment and used as load-testing scripts, cutting time needed for re-creating — and resolving — an error situation.

Load testing with Borland® SilkPerformer®

Borland SilkPerformer is a powerful, automated load and performance-testing solution for maximizing the performance, scalability and reliability of enterprise applications. With SilkPerformer, companies can accurately predict the breaking points in their application and underlying infrastructure before deployment. SilkPerformer can simulate hundreds or thousands of simultaneous users working with multiple computing environments and interacting with various application environments such as Web, client/server or ERP/CRM systems. Even with all of its robust functionality, SilkPerformer's visual approach to scripting and root-cause analysis makes it simple and efficient to use.

Borland® SilkPerformer® Lite, an entry-level, Web-only load testing tool fully architected on SilkPerformer technology, makes an initial implementation of load testing very affordable. The same is true for SilkPerformer Component Test Edition, which supports early testing of remote software components, under concurrent access. As load testing requirements grow, an organization can migrate seamlessly to the enterprise edition of SilkPerformer and fully utilize existing product knowledge and test assets already created.

Borland SilkPerformer plugs into Borland® SilkCentral® Quality Optimization Platform. SilkCentral is a quality platform that allows companies to achieve the structure, control and visibility necessary to meet software quality challenges – and ultimately enables them to deploy higher quality applications. SilkCentral is comprised of integrated test management, issue tracking and APM solutions. SilkCentral provides organizations the ability to define, measure, manage and evolve the quality of their business-critical, enterprise-level applications across the entire application lifecycle.

Borland SilkCentral spans the complete application lifecycle – integrating the testing and monitoring engines, management modules, open interfaces and reporting capabilities needed to build quality each step of the way. It also creates dramatic efficiencies by enabling users to leverage all quality assets (i.e., test plans, scripts, infrastructure monitors, metrics and results) throughout application development, deployment and on-going APM and back again, via closed-loop testing. Finally, SilkCentral delivers the collaborative framework, Web-based access and shared information repository needed to accelerate application development/deployment and meet critical time-to-market goals.