Now That We're Web Testing, How Do We Get It Right?

As the Internet's prominence in business and culture has mushroomed, applications on the Internet have evolved in complexity and scale. Companies must build websites for scalability and rigor, able to withstand expected (and unexpected) spikes and peaks in load. Moreover, the time to market has telescoped. As web applications become increasingly mission-critical, errors can mean disastrous strikes to a company's reputation and stock price, as well as exposure to potential legal and financial liability.

Since companies now realize that errors in web application performance and functionality can be insidious, occurring as a result of multiple causes, and risky and costly to fix, they are becoming more proactive in their web testing. The question then becomes not whether a website is tested, but how well was it done?

The benefits of developing high quality testing processes – and the risks of failing to do so – are clear. One analyst report found that delivering a positive customer experience on a website can lead to a greater increase in incremental revenues than simply driving traffic to the site. The report claimed that one dollar spent on advertising yields less than $5 in incremental revenues, compared to one dollar invested in creating a positive customer experience, which can yield over $60 in incremental revenues.

To truly facilitate and accelerate the successful implementation of high-performing web applications, developers and quality assurance (QA) departments must establish best practices and methodologies for web testing and analysis. For example, to assure confidence in application deployment, in shorter project timeframes, testers must take a realistic and an integrated approach to testing. Testing as early and as often as possible is essential. Equally important is fostering collaboration among teams of designers, developers, QA personnel and production support teams throughout the application development lifecycle. This white paper outlines some of the basic practices.

Test Realistically

To increase confidence in your applications' integrity and reduce risk, you can and should start by simulating concurrent Internet users as realistically as possible. For example, a retail site should mix many "window shoppers" with some purchasers and a few administrators. Each role will stress the application differently, giving you a realistic view of how your users will experience your application. Good automation tools will also be able to simulate real-world variables at run time, such as different levels of SSL encryption, multiple client types, variable "think" times, and the effect of slow line speeds.

Such flexibility will add the most value to your test in terms of realism, information learned, and confidence gained. The economic advantages of testing with an integrated, flexible solution cannot be denied. Automation can identify problems sooner, reproduce them faster, and resolve issues earlier. It allows you to focus

your two scarcest resources, skilled workers and time, on application areas where the risks are real and manual testing more appropriate.

In designing a realistic series of scalability tests, you need to determine what your driving factors are. For example, when determining the number of users to simulate, you should build your test based on your historic traffic numbers (if you have them), or your expected visitor profile if not. One industry rule of thumb is to stress your application with three to four times your historic peak load, since a new promotion or product (or an outage at a competitor) can easily attract significantly increased traffic – even if your company isn't advertising on next year's SuperBowl.

You should also aim to establish goal-oriented test objectives. There are hundreds of metrics that can be measured, but which are important to your company? Page load time (the basic standard should be no slower than 8 seconds)? Transactions per second? Throughput? Can you correlate these values with system metrics such as the CPU utilization on your servers? Ensure that you are collecting enough information to effectively diagnose a scalability problem. Bear in mind that your performance testing, while it may be focused on the end user's experience, needs to uncover problems further back in the system. It does no good if the system performs well, but uses so much server memory that it crashes your servers after a few days in production.

Finally, a critical but routinely neglected aspect of scalability testing is to verify your application's data integrity while verifying its performance. Both should be validated under load for every individual user. After all, what good is a speedy response from your web server if it is only delivering a "busy" message back to the user – or, worse yet, delivering subtle data errors? Bad data delivered to users can surely damage your reputation and stock price just as much as a poorly performing site.

"Test Early, Test Often"

Industry studies have repeatedly shown that the earlier problems are found, the more readily and cheaply they are solved. Scalability problems can often lie at the heart of a system, and as such may be difficult, time-consuming and costly to fix. Planning scalability tests as the last check in an application's development potentially exposes you to tremendous risk and change at the very time you are trying to eliminate both.

This risk is present regardless of whether you test your software internally or outsource the work to a third party or service. Leaving such tests to the end of the development cycle leaves you very vulnerable either way. Third-party solutions are, however, a great complement to internal scalability testing if used wisely. For example, you can outsource peak load stress testing after your internal load testing has been completed. Such peak tests may require more hardware and software resources than you have available. The trick is not to avoid peak load testing; but rather to find the most practical way for your organization to accomplish it.

The QA process typically involves testing at the end of an application's development phase, when the bulk of the coding has been completed and design decisions finalized. Yet to minimize the risk of major structural problems being discovered late in the day, you should perform scalability testing well before the entire system is complete. This makes it possible to flesh out all the scalability issues in each

application tier, subsystem or component well before final integration takes place. Systems tested using this type of methodology are more likely to scale successfully, since individual tiers and components will have already passed their own independent scalability verification test suites prior to incorporation into the final system.

To enable this change, two key elements need to be in place. First, the automation tool you deploy needs to be technically capable of testing the core components of your back end systems without requiring the web front end to be available, from Java beans and ActiveX components up to entire layers of your application. Your development processes may therefore need to take into account test automation requirements, so that the appropriate software interfaces are published and available for test engineers to interact with.

Second, you need to ensure that scalability testing is part and parcel of every developer's unit test plan, and that your QA function can scale these component and tier tests to the level required by your final, integrated system. You should therefore select a tool that is economical to distribute across tens of desktops, and update your development processes to reflect this change of approach. It is also important to remember that functionality and scalability testing are not separate testing activities, but ones that need to be performed concurrently. If you start your teams thinking that functional testing is really a single user scalability test, then the benefits of the mindset change will be substantive and immediate.

Test Throughout the Lifecycle

Timing is everything when developing for the Internet. But the key to making this realistic, early, and iterative testing approach work efficiently is to share testing resources cross-functionally around the project lifecycle. If your development team is spread out, ask yourself whether the testing resources can be accessed from all parts of the organization. How can your teams collect correlating performance data, communicate performance results, and collaborate on application upgrades? Individual developers can become more effective if they are enabled to log into shared resources such as hardware RAM, test agendas, and scripts.

This is particularly true as testing moves further upstream into development organizations, following the approach outlined above. Being able to share testing resources across teams, departments and locations can raise the return on your investment in an automated software quality solution. If QA's test lab can be made available remotely to developers while still being managed centrally, then all the groups benefit.

Testing with a community solution can help identify performance bottlenecks and integrity issues very early in the process. Design decisions can then be modified while the implementation is still sufficiently pliant to enable changes to be made in a timely, affordable and controlled manner. Once these periodic tests are completed, the test scripts can then be passed on to the regular QA department for full-scale load, stress, functional and regression tests -- using the same technologies adopted by the development group. Any problems discovered can then be easily reproduced by the developers and fixes initiated.

Similarly, after moving an application onto its production environment, the same scripts used by development and QA can again be used by the IT function to verify that the production environment has not introduced any additional scalability problems. By using common resources to bridge fragmented working teams, you will likely reach this stage with significantly higher confidence in your application's capabilities than previous iterations.

Finally, by sharing testing resources, including hardware and lab environments, it is also possible to maximize your investment in the systems required to generate such tests – an important part of the economics of scalability testing. .

Where Do You Start?

Your organization's very credibility depends on how the outside world perceives it. More often than not, these defining interactions are moving to the web. While embarking on scalability testing may seem like a tall order, it is important to weigh the risks of not undertaking this work. Do not wait until your website goes down before you seriously consider scalability solutions.

An automated testing solution, regardless of its focus, will not in and of itself make up for inherently risky internal software engineering practices. If anything, these solutions will serve to emphasize them. It is therefore important that you view automation as one critical important piece of the wider software quality puzzle. Take what steps you can now – even if this means starting scalability testing at the end of the cycle – as long as you have a coherent plan for the future. And, most importantly, make performance part of the design criteria before development even starts.

You should also communicate the benefits of adopting these approaches to all the functions in the process. Developers need to understand that early scalability testing will reduce the amount of time they spend fixing bugs after they release to QA. QA engineers will spend more time doing important, thorough testing instead of mundane, mechanical regression tests. Production teams will gain a better understanding of the infrastructure required to deploy the application. And executives will buy in to economic risk management.

Running Your Own Tests

Ultimately, each e-business must assess its own needs, objectives and goals. We at RadView Software do not demand that our customers adhere to a strict set of testing guidelines, but rather we encourage them to take into account their own particular business drivers, testing objectives, and requirements, and craft the most appropriate testing scenario from there. Only in doing so can companies truly can generate customized web application verification, and ensure that their web applications achieve the highest level of quality.