Title:
Scalability testing: 7 steps to success

Word Count:
636

Summary:
Very large scale software systems are a breed apart: scalability is vital to their success. If you leave performance testing to the end, you risk finding your systems will fail to scale and you're left with staggering costs of correction. This article provides a chacklist to help you succeed with scalability and performance testing.

Keywords:
Scalability, Load Testing, Performance Testing, Risk Based Testing

Article Body:
Systems that work well during development, deployed on a small scale, can fail to meet performance goals when the deployment is scaled up to support real levels of use.

An apposite example of this comes from a major blue chip company that recently outsourced the development of an innovative high technology platform. Though development was behind schedule this was deemed acceptable. The system gradually passed through functional elements of the user acceptance testing and eventually it looked like a deployment date could be set. But then the supplier started load testing and scalability testing. There followed a prolonged and costly period of architectural changes and changes to the system requirements. The supplier battled heroically to provide an acceptable system, until finally the project was mothballed.

This is not an isolated case. IT folklore abounds with similar tales. From ambulance dispatch systems to web-sites for the electronic submission of tax returns, systems fail as they scale and experience peak demands. All of these projects appear not to have identified and ordered the major risks they faced. This is a fundamental stage of risk based testing, and applies equally to scalability testing or load testing as it does to functionality testing or business continuity testing. With no risk assessment they did not recognise that scaling was amongst the biggest risks, far more so that delivering all the functionality

Recent trends towards Service Oriented Architecture (SOA) attempt to address the

issue of scalability but also introduce new issues. Incorporating externally provided services into your overall solution means that your ability to scale now depends upon these external system operate under load. Assuring this is a demanding task and sadly the load testing and stress testing here is often overlooked.

Better practice is to start the development of a large scale software system with its performance clearly in mind, particularly scalability testing, volume testing and load testing. To create this performance testing focus:

1. Research and quantify the data volumes and transaction volumes the target market implies. Some of these figures can be eye openers and help the business users realise the full scale of the system. This alone can lead to reassessment of the priority of many features.

2, Determine the way features could be presented to users and the system structured in order to make scaling of the system easier. Do not try and have the same functionality you would have for a single user desktop solution provide an appropriate scalable alternative.

3. Recognise that an intrinsic part of the development process is load testing at representative scale on each incremental software release. This is continual testing, focusing on the biggest risk to the project: the ability to operate at full scale.

4. Ensure load testing is adequate both in scope and rigour. Load testing is not just about measuring response times with a performance test. The load testing programme needs to include other types of load testing including stress testing, reliability testing, and endurance testing.

5. Don’t forget that failures will occur. Large scale systems generally include server clusters with fail-over behaviour. Failure testing, fail-over testing and recovery testing carried out on representative scale systems operating under load should be included.

6. Don’t forget catastrophic failure could occur. For large scale problems, disaster testing and disaster recovery testing should be carried out at representative scale and loads. These activities can be considered the technical layers of business continuity testing.

7. Recognise external services if you use them. Where you are adopting an SOA approach and are dependent on external services you need to be certain that the throughput and turnaround time on these services will remain acceptable as your

system scales and its demands increase. A smart system architecture will include a graceful response and fall-back operation should the external service behaviour deteriorate or fail.