Better Testing, Better Risk Reduction

When your tests don't measure your system the way your users will perceive it, you can be headed for big trouble. When you test, you should measure how your application is going to feel to your users. These measurements give you confidence both in the programs you've tested, and in the diagnostic process you'll use later in production.

by Cary Millsap

Any time you change your system, you risk creating a performance problem that could disrupt your business. Good testing reduces the risk of such disruptions.

Good tests measure how the application is going to feel to your users. The Oracle Database *extended SQL trace* feature is the best way to do it.

If you can't test all the features in your application, you should at least test the features your business absolutely depends on.

Tracing programs gives you two benefits: increased confidence in the performance of the programs you trace, and practice in executing the performance diagnostic process you'll need in production.

Problem

Your project is over, and now people are *not* happy. Maybe it was just a patch. Or maybe it was a hardware or software upgrade, a migration to the cloud, or a brand new application implementation.

What you did was supposed to help, but somewhere it went sideways on you. Programs that used to run in less than a second are taking 20+ seconds now. One report that used to run in 10 minutes is now taking more than an hour. There are ten programs like this.

So now everybody's watching you. Emotions are raging. The conference room is called the "War Room" now, and you're on the phone three hours a day now where twenty people shout ideas over each other. You're hearing the term "silver bullet" all day long. Why can't you find it?

Plan

The problem with the "silver bullet" plan is that there probably isn't one. It's possible that there's one magic fix lurking in your system that will make everything better, but in three decades helping people solve these kinds of problems, I don't remember ever actually seeing one. Not once.

Whether you have a silver bullet or not is not really the right question. The right question is, why are these ten programs slow? Until you know differently, it's possible that all ten programs are slow for ten distinct reasons.

What's the quickest way to find out? Trace them, with the Oracle Database *extended SQL trace* feature. This feature, unlike any other feature in the database, gives you a call-by-call account of everything your program did that spends time. Trace the most business critical program first, so you can solve your most important problem the soonest.

If each program is slow for a different reason, then you'll find all ten reasons. If all the programs are slow for the same reason, then you *do* have a silver bullet, and you've just found it! Solving the first program's problem will solve all the others' problems, too.

Once you know how a program spends its time, you've accomplished the first diagnostic step. Your job becomes sniffing out wasted time and eliminating it. Maybe it's a problem with an index, or a parameter, or the way your code is written. Maybe it's how your storage subsystem is configured. Maybe the new system just can't run that program as fast as the old system did. It happens. There are thousands of ways a

program can be slow, but your trace data will show you exactly where you need to focus your attention for the program you're fixing right now.

Analysis

Tracing is an awesome way to *fix* problems, but wouldn't it be great to *prevent* them? Of course it would. But how? If you could go back in time, what would you do differently? The answer is *better testing*.

You might have had a big test plan. If your team is like most, they probably looked at some AWR reports. They probably confirmed that CPU and wait times and your top ten SQL statements on your "after" system all look fine.

But those ten programs: you probably didn't check those. If you had, then you wouldn't be surprised today that they're slow. The production problem you're facing right now was caused by how your tests were defined. Tests that measure one thing (resource consumptions on AWR reports) aren't very good at guaranteeing some other thing (programs running quickly).

What you need to test is how each change affects your program durations.

Solution

In a good test, you will measure what users will feel when they use the application after the change you're making. The test plan is simple:

for every feature in your application
  trace the feature on the "before" system
  trace the feature on the "after" system
  if a duration is unacceptable, then fix it

Sounds good except for one thing, right? "For *every* feature in your application"? Literally *every* feature? What if your application has thousands of features?

The answer is, if you can automate it, then yes, why not? Method R Workbench, with its ability to process tens of thousands of trace files in just minutes, makes full-application testing possible.

If you don't want to test every feature in your application, then at least test the most important features that your business depends on. For example, if your company sells things for a living, then your BookOrder, PickOrder, ShipOrder, and BillOrder features better be rock-solid. So you should definitely test those.

The test plan gives you two important assets for each feature you test:

» A feature's trace on the "before" system gives you an objective baseline against which you can measure the effectiveness of your project. Having the "before" trace makes it much easier to troubleshoot the "after" trace in case there's a problem.

» A feature's trace on the "after" system gives you objective evidence of whether the feature is acceptable. If it is not, the trace contains all the details you'll need to diagnose where it wastes time. The "after" trace also serves as a baseline for troubleshooting in the future.

If your application consists of 2,000 features and you test only BookOrder, PickOrder, ShipOrder, and BillOrder, doesn't that leave you exposed to being blindsided by some other feature that you didn't test? Yes, it is entirely possible that some of the 1,996 other features will surprise you.

So, if you can't test *all* of your features, is it worth testing *any* of them? Absolutely.

» Testing your application's most important features makes it far less likely that your project will end up disrupting your business. Yes, your application may have 2,000 features, but your four most important features may account for 80% of your workload.

» When you trace during testing, you learn how to be faster and better at responding to unanticipated performance problems that may occur later in production.

Tracing shows you how your programs spend time. It's how you can measure what your users feel when they use your application. Learning how to trace can radically improve the quality of your performance testing and your production operations. It's how you can keep yourself out of the next ten-program catastrophe, and it's how you can extricate yourself if you happen to be in one.

Technology

Method R Workbench is easy-to-use, high-precision *Oracle time measurement software* for software development, code reviews, performance tests, concept proofs, hardware and software evaluations, upgrades, troubleshooting, and more—for Oracle developers, DBAs, and decision-makers in every phase of the software life cycle.

◇ Method R™
method-r.com
info@method-r.com