Put a Load on It

Fiona Charles

Originally published on TechWell.com March 25, 2011.

Two that got away

Every tester and test manager with a few years' experience lives with the memory of at least one important bug her testing didn't find.

I think especially about two particular bugs that slipped through testing I was responsible for. One, from the first large-scale systems integration test I managed, made it to production—at least in pilot. My client was a retail company implementing a new reward-points scheme for frequent shoppers. My test team had done an extensive test with every scenario we could think of to try and induce incorrect calculations of customer points and points balances. The bugs we found were fixed and retested, the project declared a great success, and the systems deployed to production.

Everything went fine with the first group of stores to go online. Customers were awarded points for purchases as expected, and their points balances were accurately incremented with new points and decremented for the products they returned and the points they redeemed for reward purchases. But as more stores joined the pilot and data traffic increased between the stores and the central office points system, some wrongly calculated customer balances began to show up. Luckily, the errors favored the customer, so the retailer didn't have regulatory issues from ripping off customers. Still, it was a nasty job to find and fix the problem and then clean up the data throughout the integrated production systems—particularly the aggregated data in the financial systems, where customer reward points were accounted for as a liability.

Could we have found this problem if we'd tested for a long enough time with sufficient data volumes? Perhaps. My customer, the retailer, hadn't wanted to spend either money or time on load testing. But even though production data volumes exposed the problem, I have always suspected that we wouldn't have found it if we had done a traditional load test. The performance test team would not have been looking for functional bugs and, unless the system had actually failed, they likely wouldn't have found any.

The other bug I wonder about was, in fact, detected during performance testing—but purely by accident. My team (again on the vendor side) had completed functional testing of a new custom-developed bank teller application. Then the bank's own performance test team did a load test, with scripts based on scenarios provided by the project business analysts. One of the BAs was in the room as the automated load scripts ran, and she happened to notice a display of numbers different from those she expected at that step. To everyone's horror, the teller system had returned details of Customer B's account in response to an account query for Customer A.

It was a showstopper. Without question, the bank had to delay the planned system implementation till the bug was fixed. It took time to pinpoint the bug. The prime suspect was a caching error, but no-one could find one in the teller application. Nor could anyone

reproduce the problem, in spite of rerunning the load scripts severally and in different combinations. Eventually, a clever programmer hypothesized and subsequently proved that the caching bug was actually in an infrastructure system underlying our custom application. Though that system was in widespread production use in many business domains, the problem had never before been reported.

Catching them

I've thought long and hard about what those bugs mean for my test strategies. (I wonder, too, about the other ones that got away that I don't know about.)

How can we find the functional issues that manifest only under load?

Typically, industry practice with business systems is to separate load testing from functional testing. Different teams with different skills and expertise do their testing at different times, and each evaluates the results against its own criteria. Rarely do we get the two together and evaluate load test results for functional correctness or incorporate sustained load in our functional testing.

We don't always need to do this in pre-release testing. For my retail customer, time to market was more important than strict accuracy (so long as the error didn't cheat customers). In effect, the pilot became the load test and getting the test team to do some routine accuracy checks during pilot helped to mitigate the risk. Although it caused an upset at the time, finding the bug in the pilot was more acceptable for this business than delaying the launch for a load test to be done.

In some contexts, that's a strategy we can consider and discuss with our business customers. In others, as in my bank example, the business impacts of certain kinds of escaped bugs would be too high and we need to look at a test strategy that combines functional testing with sustained load.

We probably won't need to do a comprehensive risk assessment. Normally, in assessing the risks of implementing a software system, we evaluate both the impacts of varied kinds of bugs and the probability of their occurrence. But bugs like the two I've described are notoriously difficult to predict. Who would have guessed that, after weeks of careful functional testing, the teller application could bring up the wrong account for a bank customer?

In reality, the consequences of certain kinds of bugs in systems intended for certain kinds of use can outweigh a perceived low probability of occurrence. We don't need detailed risk assessment to tell us that an emergency services mobilization system had better not send an ambulance to the wrong address. Even if we think that's an unlikely outcome, we should build checks for it into test scenarios where the ambulance service is experiencing a higher-than-expected volume of callouts over a sustained period. Checking every potentially calamitous functional outcome under load should be the minimum when something that the business really values is at risk.

We can identify which outcomes to check by brainstorming with our test teams and business customers, asking, "In the use of this system, which errors would really hurt someone (e.g., customers, the business, or the general public) if they occurred?"

Sometimes, the risks will justify combining complex functional testing with system loads. If a lighter approach would suffice and we're planning a load test anyway, we can expand it to include a check for each nasty outcome. Or, if no load test is planned, we can expand the functional test to incorporate a period of sustained load.

In the end, it's a business prerogative to decide whether or not to pay for load testing. It's our responsibility, however, to give our business stakeholders information that allows them to base their decisions about testing on fact and reasonable inference. It helps to give concrete examples, like the two bugs I've described here or others from your own experience. It also helps to suggest possible outcomes from the system you're testing that could have a major impact if they happened.

As testers, we can start by developing our own awareness that maintaining a separation between functional and load testing may not be serving our stakeholders well. We need to consider more holistic test approaches that better simulate how systems will actually operate i