Changing the Ground:
Testing an Infrastructure Upgrade

Fiona Charles

Originally published on Techwell/Stickyminds March19, 2012.

Every experienced tester has a painful story about the infrastructure upgrade that blew up a development project—or worse, a critical production system. Remember the minor operating system upgrade that was going to be so “transparent” that it wouldn’t need testing? Or that “trivial” compiler upgrade?

I’ve lived through a few of these, not one of which took less than three times its planned duration. Hard experience teaches testers to be wary of any infrastructure upgrade. Nevertheless, we don’t have to test everything in the affected applications. Instead, we need to identify and address the actual risks in our testing.

One of my consulting clients urgently needed to upgrade the unsupported operating system under its mission-critical merchandising system. That would force them also to upgrade versions of their database engine, compiler, and report generator for mutual compatibility.

The retail business depended daily on the merchandising system and required frequent enhancements to it. Taking time for a platform upgrade that would add no business value was a big concern, and IT was worried. How could they ensure that the merchandising system would function identically after the upgrade? How could they complete the essential upgrade work before the business resumed clamoring for critical enhancements? The proposed testing alone was going to require more time than the business could afford.

IT management asked me to review the upgrade test strategy with two goals: ensure that it adequately addressed the risks, and find any possible ways to minimize the required testing time.

Everyone agreed that good testing would be critical for success, but the test team was not well positioned to tackle the upgrade. There were no predesigned regression tests for the merchandising system and no automated scripts for any of the applications. Nine of the ten team members, including the test manager, had worked for the company less than a year, and only one had any previous retail domain experience. The team was still learning the applications.

Understandably, the test manager had approached the infrastructure upgrade conservatively, with a test strategy to develop and execute a comprehensive black-box regression test suite. Primary focus would be on the merchandising system, and the test would also cover major flows through the entire enterprise integration. Analogous to

many Y2K projects, the team would capture “before” results for comparison after the upgrades.

I saw long-term advantages in this strategy. It would give the team a good basis for regression testing future enhancements and infrastructure upgrades impacting the merchandising system. It would also be a tremendous learning opportunity, allowing the testers to build deep knowledge of their company’s most critical systems. But it would be costly in the short term. The test development work alone would be a mammoth effort and—even less acceptable in a fast-moving business—testing would take a long time, during which no major application upgrades could be applied. Only a few testers would be available for much of the work; most were fully committed on other business-critical projects.

How could all that work and time be justified for this project alone? There had to be faster and cheaper ways to address the risks the multiple upgrades posed to the merchandising system.

As I explored the project with the project manager, test manager, DBA, and other members of the upgrade team, it was clear that everyone was operating on assumptions of waterfall sequencing, and project responsibilities sharply divided according to specialty. The test team felt responsible for all the testing. The test strategy was based on employing only the testers’ existing skills, and waiting until late in the process for access to systems to test.

I suggested reframing how they thought about the project. If you think of an infrastructure upgrade as primarily a testing project, then it would make sense to adopt a whole-team approach. Why not explore as a whole team the most effective and efficient ways to test throughout the project?

Even if they decided that the test team should document the test strategy and coordinate test development, testing shouldn't necessarily assume ownership of the tests. With infrastructure upgrades, there will often be other specialists better placed to design tests and find problems. So I suggested the project begin by assembling a cross-functional team of techies and testers to have overall responsibility for deciding the test strategy and testing the upgrades. It would need at least one representative from each technical team involved in any one of the upgrades: product and technical architecture, the DBA team, etc.

The cross-functional team could take a risk-based approach to the central test design based on answers to these questions:

- What is most likely to go wrong from a technical point of view with each upgrade? What kinds of problems would have unacceptable impacts for their business? How and where could those problems manifest? Which techniques would they need to catch each problem? Which data, and how much? Where could analysis or knowledge of the code preclude a need to test?

- Where is the earliest point in the upgrade process that someone on the cross-functional team could detect and fix each kind of problem they had identified?

- For which problems would they want extra insurance from additional testing later in the process?

The team could then design tests targeting each of the risks and decide who would be best placed to do each one. They could identify where it would be useful to pair testers with DBAs or programmers.

In addition, I suggested they supplement their central test design with:

- Sanity checks that would expose catastrophic problems quickly in each major functional area

- Some key scenarios covering things that they didn't think could happen, but which would critically harm their business stakeholders if they did

To find out the technical risks, I recommended two principal avenues. First, research each upgrade, and if possible, combinations of upgrades:

- Ask the vendors.
  Look on the web at technical forums and other resources. Someone out in the world has done what you're going to do and found problems with it.

Then, conduct a workshop with the merchandising system maintenance team and the technical experts responsible for each upgrade. Ask what kinds of errors this type of upgrade might introduce:

- For any processing modes that could change, which application technical components depend on the existing modes and could break or behave unpredictably with the upgrade?

- How and where could each change or error type manifest?

For the business impacts (problems that would hurt their users), we looked at another set of questions to ask and did a pilot workshop with a group of stakeholders.

Although this was not an agile shop, the project team embraced the whole-team approach because it targeted the upgrade risks directly and engaged each specialist's expertise where it would be most effective in the analysis and testing. There was no need for a lengthy "test phase."

It turned out to be a very successful strategy. The team found and fixed several problems in their applications that were illuminated by the upgrades before they went live, and they spent far less time and money than they would have with the comprehensive black-box regression test they'd originally intended.