

The Essential February 2006 Guide

to TESTING YOUR DISASTER RECOVERY ABILITY

By Paul Robichaux

It doesn't matter how good your disaster recovery procedures and processes are if you don't test them.

This stance may seem extreme, but disaster recovery is so important that the key metric is simple: either your procedures work as intended, or they don't. Finding out how well your processes work to restore your data, and continue your operations, is something you need to do before you need them. Testing and validation help you find problems, shortcomings, or areas of improvement in the resources, software, hardware, people, and plans you

depend on to successfully execute a disaster recovery and to keep your operations going during an outage. This Essential Guide describes what to test, how to test it, and what results to expect.

Validation, Testing, and Deployment

Before we can get into the nuts and bolts of testing, it's important that you understand that under the general rubric of "testing" there are actually three separate phases. Each of these phases has its own requirements, and all are required to comprehensively test your disaster recovery design. If you've ever worked on large-scale construction or design projects for commercial airliners, spacecraft, safety-critical software, or the like, these phases may already be familiar to you.

The first phase is *validation*. This part of the process requires that you find the answers to two questions:

- Do the requirements you have for your Exchange system accurately represent your business requirements? For example, if your business depends heavily on email, and you have an explicit service level agreement (SLA) that says that email will never be down for more than one business day, your Exchange design should reflect that required level of service.

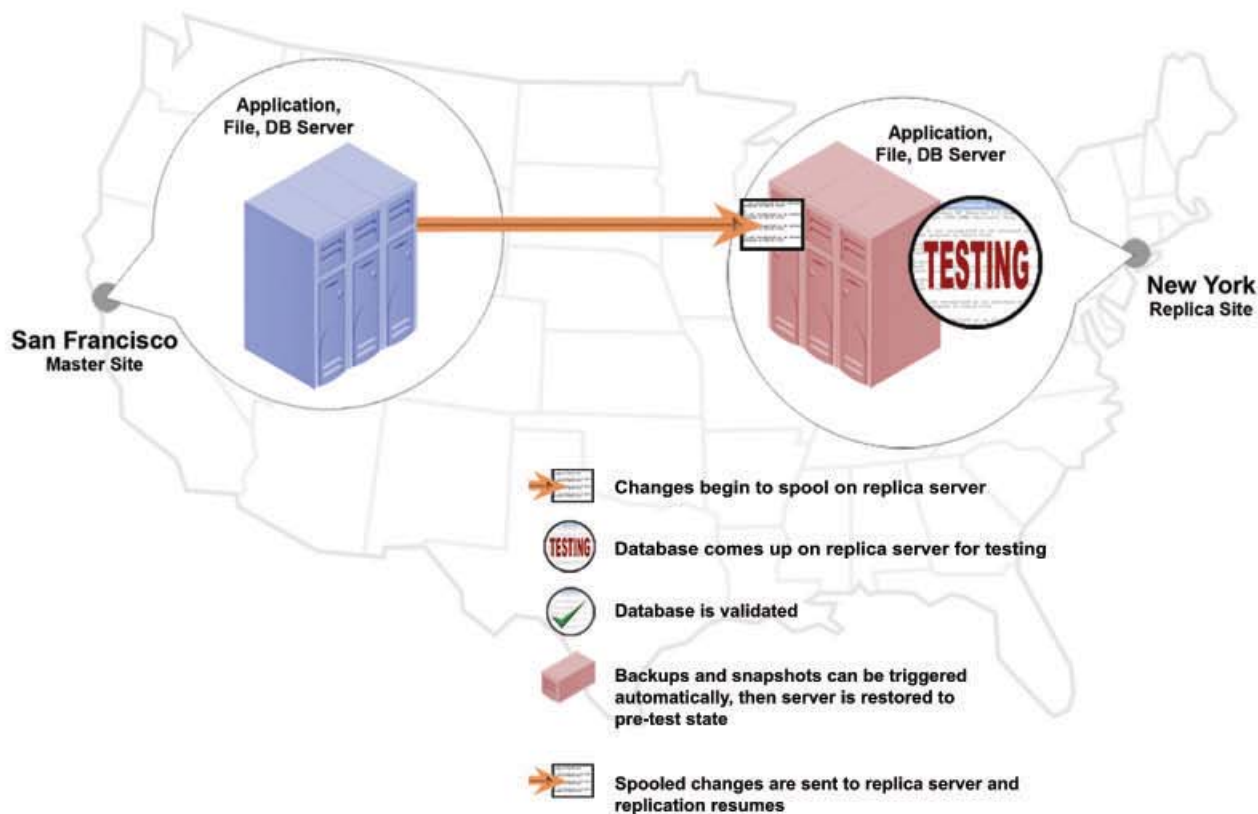
Special Advertising Supplement

This special advertising section was produced by *Windows IT Pro* in conjunction with XOsoft and appears as an insert in the February 2006 issue of *Windows IT Pro*.

sponsored by



"Forget the prayers - go with the proof."



XOsoft™

XOsoft's Assured Recovery allows for in-depth testing of the recoverability of the application on a replica server without any disruption to the production server, to the replication process, or to the automated failover protection mechanisms that are in place.

Visit <http://www.xosoft.com/AR>
for more information

- Can you prove that your system, as designed *and* implemented, actually meets the requirements?

As its name implies, validation is mostly about proof: proof that your design is *correct* (that is, that it accurately meets the business requirements) and *complete* (in other words, that it doesn't leave out any required capabilities). You might have a great design that is correctly implemented but that doesn't meet some of your business requirements—that's

why validation is important, because it helps you catch such situations. Validation can include design reviews and performance testing, both of which also can take place in other parts of the testing cycle.

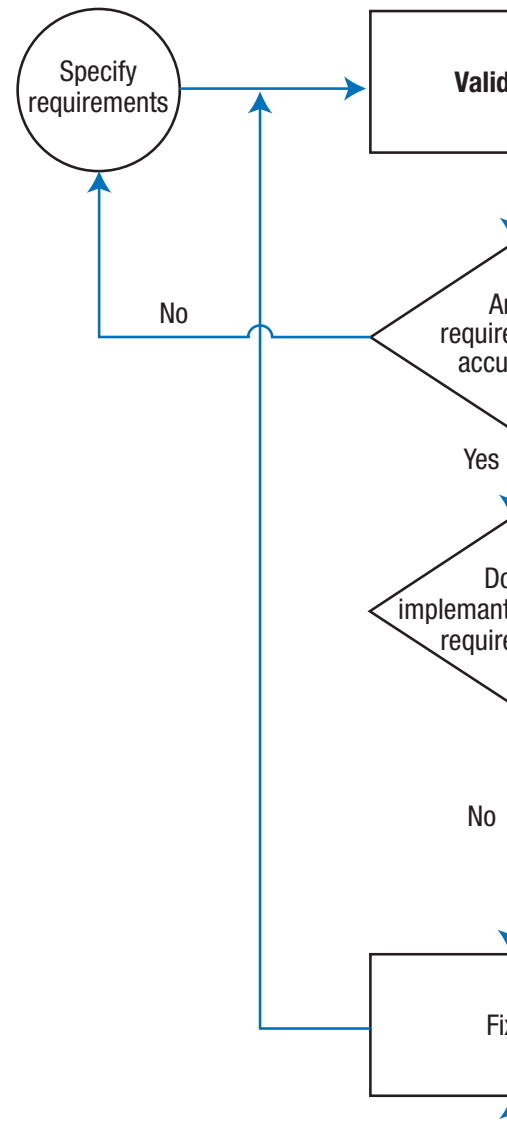
The second phase is *testing*. This phase is what most administrators think of when they think of disaster recovery testing: you try some things and see how well they work. You can actually do this in two ways, each of which has a specific term for explaining what it is. First is *unit testing*, where you exercise individual hardware, software, or process components and see if they do what they're supposed to do. The second is *integration testing*, where you systematically test all the components *together* to make sure that they work as a group. This may seem like a fine distinction, but it's impor-

Building a Test Plan

Your test plan can be as simple as a series of goals, expressed as statements or even as questions. Or it can be much more elaborate; but either way, you should be answering five basic questions:

1. What explicit performance targets do you have to meet? These targets might involve message delivery time, uptime, the time required to complete a recovery, or other targets that make sense according to your business requirements.
2. Are there any implicit requirements that you face as a result of the explicit requirements? For example, if you have an explicit requirement that all recoveries must take less than 10 hours to execute from start to finish, that applies an implied requirement on the size of your mailbox databases—you have to keep them small enough to restore completely within the given time period.
3. What performance data do you already have? If you don't have any data, you'll need to gather some before you can make an effective test plan; thus, the first part of the plan will be to get some data as part of the validation of your existing architecture and design. A related question is how will you get the data: what tools will you use and what performance measurements will you take?
4. How will you decide whether the tests are successful? In other words, what are the acceptance criteria for the testing? A good test plan should specify numeric targets (just like Jetstress does—your Jetstress test will either pass or fail depending on how much latency the disk system encounters while the test runs) so that there's no ambiguity in how you decide whether the tests are a pass or not.
5. What's in, and out, of the testing scope? For example, if you have multiple servers, you might decide that your initial testing should cover only restores of one server at a time. That's fine, as long as you document what you are, and are not, including.

A good test plan will be almost boring in its degree of detail. It should leave virtually nothing open to interpretation; that way, there's no room for debate about how you'll tell whether the tests are successful or not. Of course, the ultimate test of a disaster recovery implementation is doing an actual recovery, and you should definitely include full-up simulated recoveries as part of your testing. This approach is valuable because it shows you where your procedures and people are performing well, and where you might make improvements.



tant—unit testing will tell you things such as whether a particular server’s tape drive works properly, but integration testing will highlight whether you can successfully take a tape that the drive wrote, move it to another physical location, and use it to restore mail data onto a different server.

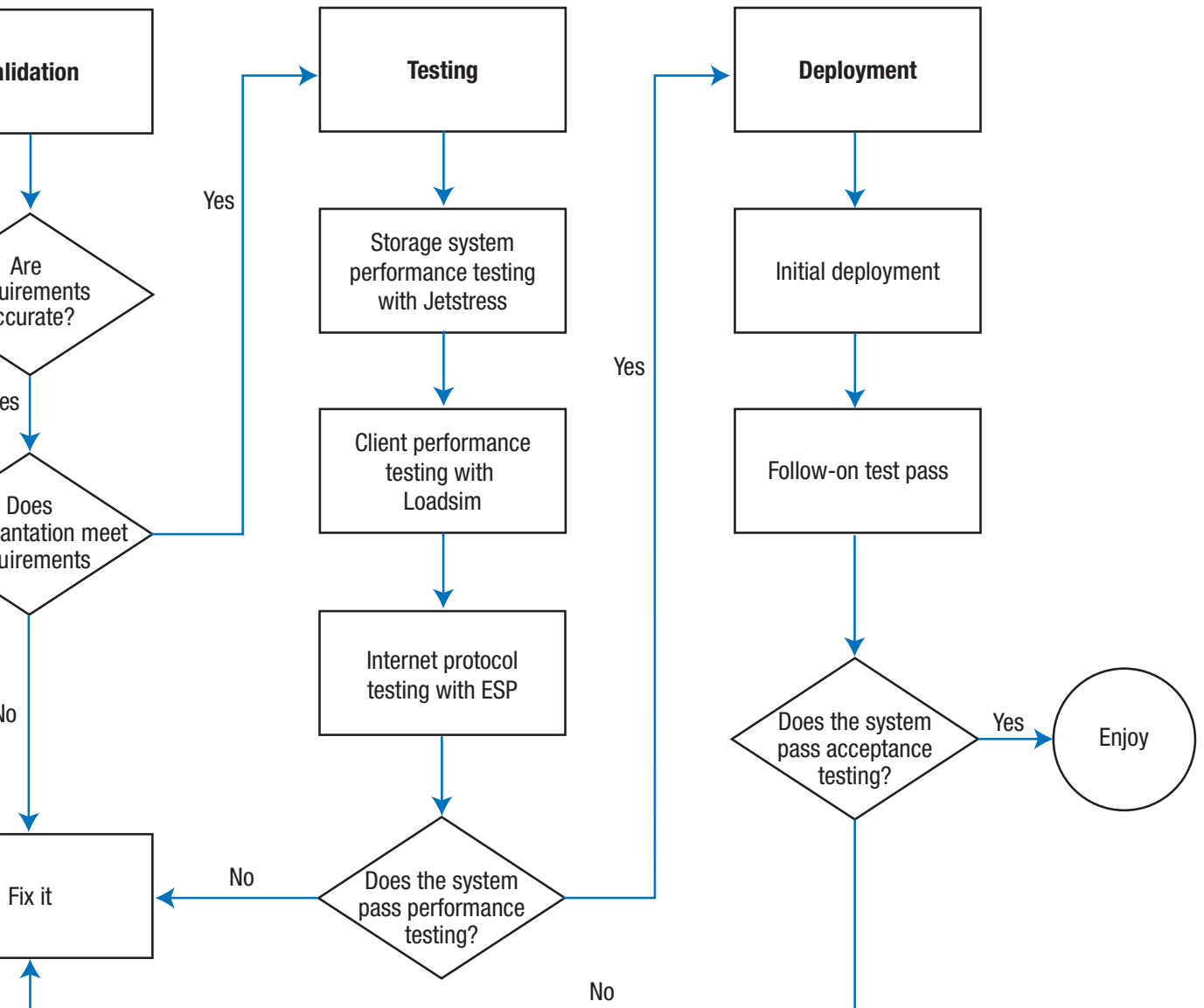
The third phase is one that we all know well: *deployment*. In this phase, you execute your plan, modifying it as necessary. Of course, just because you’ve deployed your solution doesn’t mean that you’re done with testing it; post-deployment testing is a great way to find settings or procedures that may have slipped through the cracks.

This description may make testing sound much more complicated than you’d expect. Testing large deployments can certainly be a time-consuming process; even so, validation, testing, and deployment (VTD for short) are absolutely critical to making sure that your disaster recovery operations work right every time.

Testing Tools

Testing obviously requires tools; without them, you can do a great deal of bench-checking of your design, but you won’t be able to accurately assess it in a real-world environment. You can use four primary tools to test the performance and stability of your Exchange environment:

- The Exchange load simulator (Loadsim) is a familiar tool to many Exchange administrators because it’s been around for a while. You can use Loadsim to see how a given configuration will react under a specified load of MAPI users; Loadsim is smart enough to emulate both conventional MAPI access (as with Outlook XP and earlier) and the cached Exchange mode implemented by Outlook 2003. However, Loadsim doesn’t perfectly imitate real users, and it can only simulate MAPI loads. When you run a Loadsim test, the load it generates is spread evenly across the entire



test interval. That means you get a good picture of how well the system handles a sustained load, but not much information about how it responds to peak loads (i.e., first thing Monday morning, right after lunch, and so on). Loadsim is very valuable as a way of validating configuration changes: gather a baseline of your performance data, then run Loadsim tests, tweaking the test settings until the observed Loadsim results closely match the actual results. At that point, you can take the Loadsim profile, run it against your proposed configuration, and get a solid idea of how the new configuration will handle the load you already have.

- Jetstress generates disk I/O operations in a pattern that accurately simulates the real I/O pattern of a busy Exchange server. It does this by creating an empty database, filling it with random data, and simulating Exchange I/O operations against it. You get to specify the number of I/O operations per second (IOPS) that you want Jetstress to generate; when the test runs, Jetstress tracks whether or not the average latency meets Microsoft's recommendation. Jetstress is a terrific tool for checking the performance of a proposed SAN configuration, provided that you run it long enough. It's also a useful way to try different storage group and database locations to see which one gives you the best sustained performance.
- The Exchange Stress and Performance tool, or ESP, is a counterpart to Loadsim. Where Loadsim does MAPI, ESP handles almost everything else. By writing appropriate ESP scripts, you can jam arbitrary amounts of SMTP, IMAP, POP, NNTP, or WebDAV traffic through your Exchange server to see how it handles the load. The ESP documentation goes into great detail about how to set up ESP tests and how to interpret the results.

- The Windows performance monitor (PerfMon) is a prosaic tool that's been with us since the first release of Windows NT. Despite its humble origins, you can use it to effectively monitor your system's performance by watching key counters (like the *Disk transfers/sec* counter on both physical and logical disks, the *Average disk queue length* counter, the *RPC Average Latency* counter on the information store, the amount of RAM and VM pages being consumed, and the percentage of CPU time in use). Pick a set of parameters to monitor, then watch it over a period of time to ensure that you get a solid baseline.

Conclusion

Planning for disaster recovery may sound like one more round of boring paperwork that keeps you from doing something interesting—but that couldn't be further from the truth. Building a solid test plan for your disaster recovery environment will build your knowledge of your current environment by forcing you to look into all the “dark corners” of your infrastructure; it will improve your actual operational skills by helping you get more familiar with the tools you need to use, and it gives you assurance that when you need to, you can recover the critical data your business depends on. ■

Paul Robichaux is an experienced software developer, author, and a partner at 3sharp LLC. He writes and teaches about Exchange, validates enterprise deployments for security, storage management, and scalability, and he is a Microsoft MVP for Exchange Server.

Testing vs. Production

You've probably heard Robert Burns' line about the best-laid plans of mice and men. Just having a good VTD plan will give you a great start on the road to building a repeatable, stable, confidence-inducing disaster recovery capability. Honing your disaster recovery skills in a test lab is a great idea, assuming you have a test lab, but how do you translate those skills into your production environment? Here are some ideas that may be useful to you:

1. Keep records of what's different. Even a one-page bulleted list of differences between your test and production environments can help make your recoveries more efficient by helping you avoid wasted time.
2. Practice your skills in production, when business conditions permit. For example, you could move mailboxes off a server if you were going to decommission it, or you could recover that server's data to a recovery storage group. These two processes have the same basic result, but only one of them lets you practice working with RSGs in your live environment.
3. If you're using products that support failover, fail over to your alternate environment and live with it for a while. Different vendors recommend different frequencies for this, but all of them recommend that you use their failover solutions to cut operations over to your backup system so that you can periodically shake out any flaws in it.

"Please, PLEASE let this failover work as planned."

In the event of a failover situation, are you sure your replica environment will operate? XOssoft's disaster recovery software has built-in testing, validation and reporting on the recoverability of your replica server environment.

Forget the prayers - go with the proof.



XOssoft™
JUST KEEP WORKING

www.xossoft.com

