The Lab to Have When Time's Short

For just-in-time testing to work, you've got to be ready for it—before time runs out. By Robert Sabourin

How do you know you are finished? How do you know that the system is ready for real end users? How do you converge the project, delivering the fruits of your labors and the spoils of your blood, sweat and tears? Steve Covey, author of "The 7 Habits of Highly Effective People" (Free Press, 1990), might advise: "Begin with the end in mind…First things first…."

This article talks about how you can always be primed and ready so that testing can start "just in time"—at any time. We will look at how you can cope with testing systems with minimal specifications, no detailed requirements and without any up-front analysis. As a test lead, you will learn ways to prepare for testing unpleasant, unexpected, undocumented and sometimes unwelcome projects. As a tester, you will learn about some techniques that can sharpen your testing skills, helping you to focus on finding important and critical bugs. Developers and project stakeholders will learn some important techniques for collecting and managing testing ideas and for effectively reporting testing progress and system readiness.

Let's say a project has just landed in your lap. There are no requirements, no specifications, no manuals and just a bare-bones Help system. Your job is to provide answers. The questions:

- What would work?
- What would not work?
- What problems would end users experience?
- What do we know?
- What do we not know?
- What do we need to know?

You can take your magic testing wand and, using a blend of Tinker Bell's Pixie Dust and the Jedi Force, presto, you've got it—exactly all the information people need to hear.

However, in reality, if you are a typical test manager, you probably ran out of Pixie Dust on the last project, and you are being somewhat overwhelmed by the dark side of the Force. But take heart—there are many practical things you can do to test the software and provide tons of great, objective information to project stakeholders in a timely and cost-effective manner. All you have to do is apply the five key disciplines of just-in-time (JIT) testing.

JIT testing works well when you don't have the benefit of a detailed test analysis, test plan or test procedures. JIT testing blends systematic approaches to exploratory and analytic testing techniques with risk-based management of testing objectives. During a JIT testing project you:

- Discover new testing ideas on the fly
- Triage testing ideas
- Elaborate your testing opportunities
- Implement your testing opportunities

continued on page 16 ►

Robert Sabourin has more than 20 years of experience leading teams of software development professionals. An adjunct professor of software engineering at McGill University, he often speaks at conferences on software engineering, SQA, testing and management issues.

Equipment, Preparation and Training Are The Keys to Testing on Short Notice

Photograph by Rob Friedman

◄ continued from page 13

• Track testing status

Be Prepared

Before you start a JIT testing project, the testing team should be prepared. A JIT testing team is like an emergency response team—always ready for action, and always prepared for the worst that can be thrown at it.

Imagine you and your team were a squad of firemen prepared to put out fires and save lives. Firemen never know when they will be called upon. Firemen must always have their equipment ready and available. Firemen must always be in great physical shape. They must be strong, brave and of good moral fiber. Firemen must constantly train, honing existing skills and learning new techniques. They must learn to cope with new hazards. Firemen must learn new ways to use the equipment they have. Firemen study how other fires have been dealt with, and they analyze how in the future they might respond better to various emergencies.

All members of the JIT testing team must be in great shape. I'm not suggesting that you run out to the gym or do aerobics, although these do help! I'm suggesting that the testing team members should all be in sync in terms of their basic testing values. Make sure that everyone on the team understands the role of testing in the company. Is testing focused on gatekeeping, or is testing focused on providing objective information about the state of readiness of software? A disconnect about the role of testing could lead to conflict during a project.

Get in sync by looking at what is expected of the testing team. Make sure you know what is needed in the near future. Don't get too attached to any one perspective. On the next project, you may be asked to look into aspects that are totally contrary to what you just completed.

Recently, I contracted to do compatibility testing of two security applications running in a non-conflicting manner on the same computer. Past testing had been done with the security applications running in a conflicting manner. Some testers needed time to grasp the fundamental change in philosophy; they wanted to study failure modes by asking both applications to do similar operations concurrently. The bugs found were recorded and reported, but were for the most part ignored. The client wanted to see what would happen if non-conflicting operations were performed concurrently. Testing therefore should not have been trying to force failures, but rather to see if failures would be observed when users intentionally avoided conflict. The testing would have been more efficient had testers focused on holding the same values well in advance of getting any build to test.

The credo "on time, on quality, on budget" is meaningless unless you are "on purpose." The healthy JIT tester is agile and can turn on a dime. You must make sure that all team members can be interrupted from other responsibilities and enjoy the challenge. A JIT testing team spends plenty of time getting ready to react to change. Team members who do not enjoy adapting and reprioritizing on the fly might be better assigned to a more traditional and predictable project. Some testers prefer completing a detailed analysis before starting to draw up a test plan.

Make sure the right testers are on the right teams. Almost all testers I have worked with really enjoy JIT testing projects—just as long as everyone on the team knows they are on JIT testing projects. If some team members expect to have a detailed planning and analysis phase, they're in for a big surprise.

Equipment and The JIT Test Lab

Even if you do not have any software to test, a JIT test lab must be primed, ready and available. The test lab provides resources, equipment and tools to be used by JIT testers in action.

The successful JIT test lab is separate from the day-to-day desktop workstations of JIT testers. The lab's specific equipment needs will depend on technology requirements, automation needs and the number of JIT testers who will be working concurrently.

There is no "silver bullet" JIT test lab configuration, but there are some important principles to keep in mind. The lab should be set up so that testers can easily reset each computer to a known starting state. If we can set the system to a known state, then we can have more repeatable testing. This is great for bug isolation and confirmation of fixes.

Testers should also be able to change configurations quickly. Different system images should be prepared to provide important combinations of operating systems, third-party applications, Web browsers and Internet connections. Images can be prepared independent of the software being tested.

Computers have different hardware configurations. It's important to have access to some different hardware configurations in order to make sure that your software behaves well in many contexts. I recommend having at least one test system at the low end in terms of system memory, disk space, network bandwidth and processing capacity. You should also have access to at least one system at the high end, with tons of memory, massive disk space, a high-bandwidth connection and a lightning-fast processor. Other test machines should match what you expect users to have, or the recommended hardware configurations.

Vary the hardware as testing continues. There are many bugs that can be better exposed on low-end computers. They sometimes challenge development assumptions about disk and network access timing. Some cool bugs happen when data is 100 percent cached in RAM, or when computers are way too slow and normal delays trigger timeouts.

Data layers can really benefit from a JIT testing approach. Build up a set of JIT test databases in advance. Set up databases to represent different user experiences. Vary the test database during testing. Try having various test databases available that are typical, almost empty, and very full.

Before testing starts, you can prepare the lab so that the system configurations match what you expect end users to encounter. Test with the same permissions as end users. Some bugs slip by because testers work as administrative users instead of as restricted users.

If firewalls are going to be up during live operation, then be sure firewalls are up in the test lab. Work closely with system administrators, network managers and database administrators to make sure the systems under test are isolated from your development environment and match the target environments you expect your customers to encounter.

Make sure your JIT testing lab is a well-oiled machine. It must run smoothly when a project is parachuted in unexpectedly. The lab's equipment should be available at all times. I suggest setting up a 24x7 operation. Platforms and connections should be readily available, even if we do not know what application will be tested or in what manner.

I suggest working with network or system technicians to continually define and build new platform images. Anticipating platform needs in advance is a critical success factor. It takes several hours to install and image a platform, so you should carefully select the platforms needed. Take care to make general-purpose images that are not locked into or dependent on a specific version or configuration of your software.

Be prepared for anything. Manage the JIT test lab and test environments independently of the software being tested. Treat the lab as a service available to—and always ready for—whatever projects developers may throw at them.

Make a grid. I use a chart (see figure, opposite) to indicate which OS, browser, locale, connection and so forth will be available in the lab, independent of what is being tested. Rows represent test systems, and columns represent days, or weeks, or sessions.

I make a different grid for each aspect of the platform being varied. So if we care about OS, browser and locale, I'll create three tables, one for each. I distribute OS, browser and locale in a different pattern on each grid, and I use many techniques to design these grids—for example, random distributions, Pareto analysis, orthogonal arrays and risk analysis.

The lab manager uses the grid to set up systems. When testing is run, you use the systems available on the day of the test. Systems are varied every day, all the time. Many of my clients find that broad JIT platform coverage is one of the great benefits of this approach.

It is also critical to have a bug-tracking tool up and running at all times, ready for new projects and workflows on demand. Make sure all JIT team members have access to the tool and know how to use it to report bugs.

Have master templates readily available for documenting test procedures, test charters and test notes, as well as for reporting test status. Make sure people know where the templates are, and make sure team members are able to adapt the templates to new project needs on short notice.

Training Is Essential

In order for JIT testers to be ready for anything, they need to keep up with the latest techniques. They also need to know a lot about what testing techniques and approaches have been used on other projects.

JIT testers should be encouraged to read about testing and to share experiences with other testers, especially people in other organizations. Many excellent Web resources exist that allow testers to share experiences and to reference useful articles, and many books have been written about testing.

One of the best ways people can learn is by teaching. I encourage JIT testers to give presentations to their peers during "lunch and learn" sessions, which help make team members aware of new techniques and approaches, as well as how to apply old tech-

SAMPLE GRID OF BROWSER OS PATTERN

Day	1	2	3	4	5	6	7	8	9	10
T1										
T2										
T3										
T4										
T5										

Platforms			% Use
	98	Netscape	0.1
	98	Internet Explorer	0.1
	XP	Netscape	0.3
	XP	Internet Explorer	0.5

niques to solving new problems.

Tutorials about technology help testers better understand failure modes and what can break.

The next time you send JIT testers to a conference, ask them to give a presentation about the key points they learned. They will thereby become more active participants in the learning experience.

Sharpen Your Skills

Constantly hone your team's skills. Test, test and then test some more. Never stop testing. The best way for a testing team to remain sharp, focused and ready is to continue testing.

But what should you test? You can test in order to learn. Why not test similar applications from other vendors? You may learn about performance and functional characteristics from competitive products and technologies. You can test old versions of your application to see how it behaves. You can test off-the-shelf software your company is planning to deploy. There are many ways to practice testing in a manner that will be of value to your organization.

Actively try new approaches. Apply new ways to analyze, organize or report testing results. Try using new tools on familiar code.

Learn how things break and how to generate failures. Although the JIT team may have no software in hand, it can still take the time to anticipate the types of problems it may encounter.

I urge JIT team members to learn about bugs from other systems. How do similar systems fail? What types of bugs get inserted into projects? What will end users dream up that will challenge the software? How will hackers try to break through security? What bugs in other programs could possibly interfere with the system?

JIT testers can learn about bugs in other systems by studying relevant network resources, and also by reviewing bug taxonomies. Bug taxonomies are organized collections of bugs and failures. Reviewing bug taxonomies can help you generate great testing ideas and can help you anticipate how to line up new approaches for software coming your way.

There are many great bug taxonomies available to testers; some are listed below in "References."

Conduct Drills

It's critical that fire departments organize drills from time to time to make sure they're ready, and to help the public know in advance how to react to an emergency situation. We, too, can implement JIT testing drills and measure the responsiveness and readiness of the team.

A JIT test drill simulates a real test project. Don't tell the team it's a drill. Offer a deliverable, using a build from a previous project, and then note how long it takes for the team to get the software test ready in the lab.

I like to use a very artificial measure that I call "the time to first bug." I clock how long it takes from when I hand the CD to the test lead to when I get the first high-priority bug reported—a bug we would actually have to fix if this were a real project.

"Time to first test" is also important. If you are indeed ready, you should be able to start testing within minutes.

Drills should not happen too often, and team members should be debriefed after the drill. The point of the drill is to help people be ready for when the real thing happens.

Now You Are Ready

Being prepared is the key to effective JIT testing efforts, ensuring that we can find important problems in a timely and cost-effective manner even when we do not have detailed test plans and have no advance knowledge of the application coming our way.

Now that we have reviewed several ways to be prepared for a project, we can dive in and actually implement JIT testing. In next month's issue of Software Test & Performance, we'll cover day-by-day operations and the five keys to JIT testing, and you will learn strategies for implementing and managing JIT testing projects. ☒

REFERENCES

- "Testing Computer Software," 2nd ed., by Cem Kaner, Jack Falk and Hung Quoc Nguyen (Wiley, 1999)
- Bug Taxonomy and Statistics, by Boris Beizer, as amended by Otto Vinter; based on "Bug Taxonomy and Statistics," Appendix, in "Software Testing Techniques," 2nd ed., by Boris Beizer (Van Nostrand Reinhold, New York, 1990)
- "A Taxonomy of E-Commerce Risks and Failures," Giridharan Vilangadu Vijayaraghavan (Department of Computer Science, Florida Institute of Technology, 2003): http://www.testingeducation.org/a/tecrf.pdf