The Five Keys To Successful Just-In-Time Testing

Photo Illustration by Alvaro Heinzen

Save Precious Time By Implementing These Interdependent JIT Testing Keys In A Parallel Manner

This article, the second of two parts, describes how to discover, triage, elaborate, implement and track testing ideas and bugs in a systematic manner to unlock JIT testing success. By Robert Sabourin

To implement successful just-in-time (JIT) testing, thorough preparation is necessary. It ensures 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.

Last month, in "The Lab to Have When Time's Short," I addressed how to prepare for JIT testing projects and described how to organize a testing lab to ensure that testers will be ready for just about anything a development team can throw at them. Now it's time to explain the day-to-day operation of JIT testing and present the five keys to achieving success. In this article I'll describe how to triage testing ideas and bugs in a systematic manner to execute JIT testing projects successfully. You will learn the process and understand how to make effective decisions along the way.

The five keys to JIT testing are:

- Discover
- Triage
- Elaborate
- Implement
- Track

These keys are implemented in parallel, and they are interdependent. Don't treat a JIT testing project as a series of events taking place in sequence. I prefer to consider a JIT testing project as a collection of processes that are being responded to. Events occur, and the JIT testing team reacts to them, responds to them and in some cases causes them.

Discover

JIT testing involves discovering testing ideas on the fly. As soon as you get any information or insights about the software to be tested, you can start building a collection of testing ideas.

The JIT test lead will manage a list of testing ideas. No testing idea is too naive, too silly, too big or too small. Frankly, you should collect any testing idea and continue to do so throughout a project. Encourage all people involved in the project in any way to feed testing ideas to the JIT test lead.

The list of testing ideas is constantly changing. A project will generally start with an empty list. I like to kick off a new list by holding a test idea brainstorming session. Here is one approach I used recently:

Invite about five different stakeholders, each representing a different role in the project. Examples of stakeholders include:

- Writers
- Technical or customer support staff
- User education staff
- Project or program managers
- Product managers
- Business analysts
- Developers
- Testers
- Users
- Business stakeholders

Ask each person to prepare a list of things they are worried about, or concerns that they may have. Focus on a maximum of five items in any of the areas shown in Table 1.

Have a brainstorming meeting in which the ideas are collected and logged. Don't spend time discussing

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 speaks at conferences on software engineering, SQA, testing and management issues.

1. PROJECT CONCERNS

Usage scenarios	What do people really expect to do with the system?
Functionality	What is the system really capable of doing?
Data	What information does the system process, produce or manage?
Requirements	What needs is the system expected to fulfill (written or implicit)?
Failure modes	How does the system react to error conditions, invalid data and faults?
Quality factors	What characteristics should the system have for it to be successful (usability, performance, scalability)?
Creativity	Mind maps, lateral thinking, analogy, games
Experience	Real issues based on past experience of your team or others

any idea except as required to correctly and clearly log the idea. The test lead acts as scribe and moderator of this meeting. During this meeting, many new testing ideas are generated.

After the meeting, the test lead consolidates the list and groups the testing ideas in a manner that makes sense for the project. The list of ideas can then be circulated for review and used as a source of new testing ideas, and maybe as a source document for further brainstorming sessions.

Sometimes the JIT test lead must look in the most unexpected places for great testing ideas. I like to look at customer service e-mail logs to get ideas about what can really go wrong when a system is in use.

Old and out-of-date documentation may be a good source of testing ideas. Even though the document is out of date, it can still give you notions of what may have been important to project stakeholders at some time in the past. This raises the question of whether or not the matter is still of importance.

The JIT test lead manages the list of ideas carefully, but with great objectivity. Don't discard an idea because it sounds too simple or too innocent. Sometimes the simplest ideas lead to discovering the most significant bugs.

Once you've compiled a list of testing ideas and assigned a person to be responsible for continuously collecting new ideas and adding them to the list, testing can begin.

Triage

Which testing idea do we want to implement first? Remember the opening credits of the TV Series "MASH?" The theme music "Suicide is Painless" is playing as helicopters bring wounded soldiers in from the front. Medics greet the choppers and immediately assess who will get treated with the scarce resources available. Medical triage is taking place. Whom do we treat right now? Who can wait? The decision is not an obvious one. Should we treat the patient closest to death? Should we treat the patient bleeding the most? Should we treat the patient who risks infecting others? Should we treat the patient in the most pain? Should we treat the patient who could help us win the next battle, or even the war?

Medical triage focuses on making the best decisions possible, and quickly. Which patient can you help the most with the resources available?

In the context of JIT testing, triage is the process of ranking test ideas and bugs in terms of importance or priority. The concept of criticality is important, but it is not the only consideration. Triage is practical. Triage is based on reality.

In JIT testing, two types of triage must take place frequently: test idea triage and bug triage. I urge JIT test leads to perform triage at least once a day, and more frequently if required. Ideally, triage takes place as soon as a test idea is discovered or when a bug is encountered. Practically speaking, though, triage takes place when a few bugs or testing ideas are collected.

Triaging Testing Ideas

Before you get started, it is a good idea to make sure testing ideas all involve about the same granularity and are described to about the same level of detail. Clarify ideas so that you will understand the risks the idea would expose, as well as the value of information provided as a result of whether the test passes or fails. Testing just for the sake of testing is great for practicing skills, but it probably should be given a slightly lower priority than testing for the sake of gathering information directly relevant to the go/no-go decision.

The JIT test lead should try to ensure that testing ideas are about the same size. I like to use a simple measure for sizing test ideas based on the time it takes to implement them. On my scale, the time should range from less than an hour to two weeks. Larger test ideas should be split up; any smaller test ideas should be merged.

I always size JIT testing ideas assuming that the test passes. If the test fails, it could take well over twice the amount of time. Actual JIT testing time depends on the amount of bug isolation, analysis and reporting required.

For each testing idea, we should try to understand the impact. The JIT test lead should estimate the benefits and consequences of implementing the testing idea, as

2. BENEFITS SCALE

Benefit of implementing a testing idea	
High	Provides critical and timely information that we must have before shipping
Medium	Provides important information that may have an influence on the decision to ship
Low	Provides information that is of interest but is unlikely to block product shipment
Benefit of not implementing a testing idea	
High	We could do more valuable things with the available resources
Medium	We could get the information we need by other means
Low	Stakeholders would be missing critical information to make the ship decision

well as the benefits and consequences of not implementing the testing idea. I try to keep this as simple as possible. To establish benefits, I might use a scale like the one shown in Table 2.

As for consequences, I might use a scale such as the one shown in Table 3.

Simple scales are best. In order to estimate these values, I suggest first assigning benefits to Medium and consequences to Normal. Systematically review all items to see if you should change the values. Record the reasons; you may not remember why one idea was Minor and another was Significant a few weeks from now, even though it is obvious today. It's a good idea to consult with at least two stakeholders when deciding about the benefits and consequences. I recommend consulting with someone who can identify the technical risks, such as a development lead, and someone who can identify the business risks, such as a product manager.

When collecting this information, we must also keep track of the credibility of values. Several different scales of credibility could work. Tom Gilb defined one of my favorites in his book "Competitive Engineering" (Butterworth-Heinemann, 2005), offering a range of credibility from 0—a wild guess—to 1.0, or perfectly credible.

Before we decide whether an idea should be implemented, we should make sure our understanding of the benefits and consequences of implementing or skipping the idea are based on credible information. A High benefit that has no credibility should not be used to make a decision. I do not include an idea until I have confidence in my understanding of its potential impact.

You should collect test ideas continuously and do test triage frequently; I recommend that these be done daily. Work with peers representing technical or development and product management or customer stakeholders.

Try to keep triage meetings short. During the triage meeting you are prioritizing testing ideas. You decide which ideas to implement, which ideas not to implement, and in what order they should be implemented. The order of testing is often dictated by logistics or operational concerns. You may need to load data before deleting data, so testing the Delete function might be better done after testing the Load function, even if Delete has a higher priority. The JIT test lead must be practical and able to react to change.

Triage gives a clear indication as to what not to test. Triage does not give the exact roadmap for running a test. As in medical triage, the first test run will be that of the highest priority, and one that can be run because we have the skills and resources available to do it.

The JIT test lead must reject testing ideas when the cost/benefit ratio does not make any business sense for the project. The test lead should consider alternatives such as implementing part of a test idea, a different idea with the same benefit, or more of the idea when a significantly higher benefit can

3. CONSEQUENCES SCALE

Consequence of implementing a testing idea	
Significant	This testing idea would be extremely expensive to implement or it would not allow other important activities to take place
Normal	The costs are typical of similar testing opportunities
Minor	Very low cost; no impact on other activities
Consequence of not implementing a testing idea	
Significant	We would be missing information required to make the decision to ship the product
Normal	Neutral position; if we did the test we would find the information useful, but we could probably ship without knowing and react to related concerns at a later date
Minor	Does not add or take away information, even though the results might be interesting

4. ELABORATING TESTING APPROACHES

Testing Approach	Elaboration
Scripted	A test procedure or automated test script is created. A strategy to assess correctness is defined. Parameters to observe are established. Preconditions are determined. Platform requirements are stated. As appropriate, test cases are defined and test data is collected.This can apply to testing functionality, capabilities, usage scenarios or data.
Exploratory	Systematic exploratory testing involves concurrent test planning, test design and test execution. An exploratory test charter is defined. The scope of testing is defined. Templates and tools for capturing testing notes and observations are established.
Experiments	Generally used to expose the system to harsh or stressful conditions. Testing experiments are run in close collaboration with development, system administration or database management teams. I use the same basic structure as a high school chemistry experiment. I start with an objective, make a hypothesis, define the equipment required and the testing methods, and then stimulate the system and observe the behavior as the system operates. I try to confirm or refute my hypothesis. I try to account for my observations. Expertise is required to stimulate the system in a meaningful way and observe useful characteristics while the test is running.
Implicit	A test idea can be implemented implicitly by making it a natural part of some other testing activity. For example, one testing idea may be to see how the software behaves when using different Web browsers. You do not have to make a special test to see how the software behaves when the browser is changed. Instead, you can vary browsers while other testing continues. Another common example is printer support. You can try using different printers during different parts of functional testing. Implicit test ideas give valuable information at minimal or no incremental costs.

be achieved. In the triage meeting, make sure you consider any changes in the project's business or technical context. If you do this daily there should be little change, and when there is change you can react quickly.

New customers have a tendency to shift priorities. Technological uncertainties are discovered as testing and development progresses. Triage should consider new test results, new bugs identified and new testing ideas. If the context has changed, then past priority decisions may no longer be relevant. The JIT test lead must review past decisions whenever context changes.

As test ideas are triaged, so, too, must bugs be triaged. It's an important part of JIT testing to ensure that the bug priority, severity and reporting schemes are well established in advance of a project's start. The JIT lead must make sure that all team members adhere to the bug management workflow. I generally recommend that bugs be triaged immediately before any new test ideas are triaged. Bugs will influence the priority of testing, so it makes sense to look at the bugs first. A possible agenda is:

1. Business context changes
2. Technical context changes
3. Test results
4. Bug triage

5. IMPLEMENTATION WORKFLOW

Kickoff	**Confirm that the test objective and context are understood** The JIT test lead meets with the tester and ensures that the test objective is understood, business and technical contexts are known and deliverables of the test are agreed to. At this point it is important to make clear what the focus and purpose of the test are.
Prepare	**Ensure that hardware, software and tools are OK** The JIT tester makes sure all facilities and tools needed to run the test are available and operational.
Run	**Test** The test is run. It may be an experimental, scripted or exploratory test. In each case, different deliverables are prepared. Generally, results will include the pass-fail status and a summary of bugs found. Some bugs may be "on the rails," or a clear failure based on the testing objective and purpose, but some bugs may be "off the rails," that is, bugs that are observed in the system under test but not directly related to the objective or purpose of the testing idea.
Complete	**Wrap up, report bugs, collect notes and data** The tester wraps up testing and leaves the system in a controlled state. Make sure all data and images are captured. The tester reports all bugs and ensures that testing notes are clear and complete.
Review	**Review results with lead** The tester reviews the results with the JIT test lead immediately after completing the test. The JIT test lead needs to gain insight into the state of the system under test. The JIT tester must know about the nature of bugs found, what works, what fails, what is shaky and what new opportunities and testing ideas may have been identified. It is at this review session that the JIT test lead confirms notes and makes sure that bug reports are clear and understandable to others. (Make sure information is crisp and relevant; this is sometimes called "bug improvement.")
Follow up	**Reassess goals and react to new info** The JIT lead follows up by revising the test status information and adjusting test priorities. New testing ideas are added to the collection. A new test triage may be called for, especially if important new information is brought to light. The project's status is communicated to all stakeholders. The JIT lead must react to the new information immediately.

Elaborate

Once we have decided which testing ideas are important, and we know which testing idea to implement next, it's time to assign a testing idea to a tester. Test assignments come in a few basic forms. In my experience the bulk of testing ideas are implemented as:

- Scripted tests
- Exploratory tests
- Testing experiments
- Implicit tests

Before a test idea can be run it must be "elaborated." I define the "elaboration" state of a test as a way to see if the test is defined enough to be implemented by a qualified tester. How a test is elaborated depends on the type of testing used to implement the testing idea, as shown in Table 4.

In advance of testing, you should have good examples available for each of the possible testing approaches. I do not suggest you use the examples as templates, but rather as inspiration when you're confronted with a new JIT testing opportunity.

Implement

Although implementing a testing opportunity is where the real work of a JIT testing project actually takes place, this stage probably requires the least amount of description. I try to manage JIT testing work so that there is frequent interaction between the JIT test lead and individual testers. A typical workflow is shown in Table 5.

Track

I encourage the JIT test lead to provide all project stakeholders with a steady stream of "massively objective" information about the state of the JIT testing project and the state of the system being tested. My experience is that JIT test results should be communicated in real time as the project progresses. Information about test status should be at the fingertips of all key stakeholders.

I usually summarize JIT project status with information about test ideas, test results and a bug summary.

I prepare a list of implemented test ideas using a spreadsheet or database program. For each test idea I note the

6. SAMPLE TEST IDEA TRACKING SPREADSHEET

Testing Idea		Testing Approach	Elaborated	Test Results						Run Count
Identifier	Summary			Build 00A	Build 00B	Build 00C	Build 00D	Build 00E	Latest	
TID0010	Test idea 001	Script	Yes	NEVER	PASS	-	FAIL	-	FAIL	2
TID0020	Test idea 002	Explore	Yes	NEVER	-	PASS	-	-	PASS	1
TID0030	Test idea 003	Experiment	In Progress	NEVER	-	-	-	-	NEVER	0
TID0040	Test idea 004	Implicit	Yes							
TID0050	Test idea 005	Implicit	Blocked							
TID0060	Test idea 006	Experiment	No	NEVER	-	-	-	-	NEVER	0
TID0070	Test idea 007	Explore	Yes	NEVER	-	PASS	FAIL	PASS	PASS	3
TID0080	Test idea 008	Script	Yes	BLOCKED	-	-	-	-	BLOCKED	0
TID0090	Test idea 009	Script	Yes	PASS	PASS	PASS	PASS	PASS	PASS	5
TID0100	Test idea 010	Explore	Yes	NEVER	BLOCKED	PASS	BLOCKED	PASS	PASS	2

elaboration state and the test results. Table 6 shows a sample spreadsheet used to track testing ideas.

This example project is tracked on a build-to-build basis. Some JIT testing projects are tracked on a daily or weekly basis.

The "Testing Approach" column is used to record how the testing idea will be implemented.

The "Elaborated" column indicates the elaboration state of the testing idea. If a testing idea is ready to be run, then the elaboration status is set to Yes. If a testing script or procedure is being designed, then the elaboration status is set to In Progress. If more information or system access is needed, then the elaboration status is Blocked.

The "Test Results" section includes Never, Pass, Blocked and Fail. Never means a test has never been run on any build. Pass means the test was run and passed; it means no on-the-rails bugs, although some off-the-rails bugs may have been reported. Passing a test does not mean the software is bug-free. Blocked implies we could not run a test. A dash (–) indicates we did not run a test. Note that not all of the tests are run on all of the builds. Fail means a test has exposed on-the-rails bugs.

The "Latest" column indicates the status of the test the last time it was run. It may have passed on one build and then failed on a later build.

The "Run Count" column indicates the number of times a test was run.

The JIT test lead also reports on the totals. In this case we have one fail, four passes, two tests ideas have never been run and one test idea that is blocked. Two of the test ideas are implemented but are implicit rather than explicitly run (for example, by varying platforms or system hardware).

Bug Summary

The JIT test lead must keep project stakeholders aware of the number of open bugs on the project. One way to illustrate the bug status is to provide a trend curve indicating the number of high-priority bugs over time. Every day, the JIT test lead records the number of open bugs that must be corrected before the project can be shipped.

A Handoff From Developers

It's a good idea to set up a simple protocol with the developers to ensure that the system has structural integrity before accepting it into the JIT testing cycle. Each build is given a simple smoke test by the development team, in the development environment, before being handed off to the testing team.

The testing team runs exactly the same smoke test in the testing lab as soon as the build is delivered. If the smoke test passes in the development environment but fails in the testing environment, then there is probably a build integrity problem that must be resolved before accepting the new build into test.

The testing team should continue to test with the latest build. JIT testing does not stop because a broken build is delivered.

Ending a JIT Testing Project

The nature of a JIT testing project is that we gain insights into the system as we test it. We always have a list of testing ideas along with knowledge of the test elaboration status, the pass-fail status and the open bug status. The decision to ship is based on this information. The role of the JIT testing team is to provide the information to help stakeholders make an informed decision.

You will never be able to run all the test ideas you have collected. You will never be able to show that a piece of software is bug-free, and no amount of testing will ever expose all defects. If you begin with the end in mind, you know at the beginning how you will end. The end of a JIT testing project is a result of triaging testing ideas and bugs.

We know we can ship when the bugs that remain are the bugs we can live with. We know we can ship when the tests we have run give us confidence in the information we have. And we know we are willing to live with the consequences of not running the tests that remain and all of their possible undiscovered bugs.

At least for now. ☒