

BETTER SOFTWARE

LIFE'S NOT A DRESS REHEARSAL
Plan for emergencies now
PAGE 14

CASTING CALL
Hire a tester with an agile attitude
PAGE 16

The Print Companion to  StickyMinds.com



CONFIDENTIAL

Tester PI: Performance Investigator

PAGE 24

Tester PI: Performance Investigator

It's no mystery. You can save time and money when you investigate performance early and validate performance last.

by Scott Barber





Imagine you are reaching the end of a major software development project. Functional testing is in its final phase and so far hasn't revealed any ship-stopping defects. You have planned and developed your performance tests to validate the requirements you were given, and finally the project is entering two weeks of performance requirements validation, which is anticipated to be the last activity before go-live.

Your first performance test demonstrated that at a ten-user load, the system response time increased by two orders of magnitude—meaning a page that returned in one second with one user on the system returns in one hundred seconds with ten users on the system. The second test showed that at a fifty-user load, the system fails miserably with Java exceptions prominently displayed on every requested page. But the system is intended to support 2,500 simultaneous users!

Sound familiar? That is *exactly* what happened to me the first time I came onto a project to do performance testing at the end of development—rather than at the beginning. In this case, it took eight days to find and fix the issue causing the *failure*, not the response time issue, which left four business days to improve response time and complete the performance validation—assuming, of course, that no additional defects further delayed the validation. Not surprisingly, even after we resolved the response time issue, the system was not even close to meeting the performance requirements. In fact, we determined that the corporate network was inadequate to support the additional bandwidth needed for this application! As you can imagine, the product did not go-live on the advertised date.

Think about how different this performance testing effort would have been if there had been a plan to determine the actual capacity of the selected server hardware, to verify the available network bandwidth, to execute some preliminary tests on critical functionality, and to shake out configuration errors in the load balancers when those items *first* became available. In the case of the project above, the chaos would have been completely avoided if there had been such a plan in place. One test, one script, one tester. Four hours, tops, at the beginning of the project and both the

debilitating software defect and the insufficient network bandwidth would have been detected, resolved, and forgotten before anyone had even published a go-live date.

Sadly, late performance testing and finding avoidable show-stopping problems without enough time to react is not uncommon. Many people have similar stories, which lead some managers to briefly consider spending money to bring the performance tester on the project early. But there is a big step from “briefly consider spending” to “spending.” To take that step, managers need more than stories, especially when they already know that it is virtually pointless to validate performance requirements on a system that is still in flux. Every change to the system can cause unexpected performance changes and thus require the validation process to start over. Managers need to know what they will gain from bringing in the performance tester before the software is functionally stable. What value will it add? How will it be planned? What other activities will it impact?

This leads to two basic questions. First, how do we communicate to our managers an approach for early project performance investigation that will give them confidence that we aren't just shooting from the hip, so to speak? Second, how do we demonstrate that our approach will actually reduce the likelihood of late project performance surprises rather than wasting project time chasing shadows that turn out to be nothing more than incomplete areas of the system? I struggled with these two questions for years. No matter how many reported disasters and “early performance testing saved the project” stories I collected, I only succeeded in convincing managers that there is a risk in not detecting performance issues early. I never quite convinced them that you or I may actually be able to mitigate that

risk with something other than dumb luck.

Recently I had a bit of an epiphany that has gone a long way toward answering those two questions. In the middle of a testing project, in the middle of a conversation with the testers—in the middle of a *sentence*—I was struck with a new way to look at the difference between the way investigation and validation relate in performance testing and the way they relate in functional testing. But before I share that epiphany with you, I should probably give you my definitions of validation and investigation as they relate to testing in this discussion.

On all software development projects there is a point at which it becomes important to determine whether the software does what it is intended to do—in other words, “test it.” Depending on the development methodology you follow, this testing may start early or late, be formal or informal, have heavy or light documentation, and be conducted by developers, testers, users, or all three. But no matter which methodology you follow, testing is meant to determine whether the software, as developed, complies with the vision of those who requested it in the first place. So one way or another, tests are conducted. The results of those tests are compared against that vision, whether the vision takes the form of

makes no reference either to outcomes or to expectations. This distinction is the reason we say “investigating a crime scene” rather than “validating a crime scene”—validating a crime scene would violate the concept of “innocent until proven guilty” by implying that the crime scene was being examined with a particular expectation as to what the collected data would mean.

So what makes the relationship between these two activities in performance testing fundamentally different from their relationship in functional testing? In my experience, two factors mark the relationship as different. The first is that some manner of requirement or expectation typically has been established prior to starting functional testing, even when that testing is exploratory (or investigative) in nature. It’s an unfortunate reality that performance requirements are rarely well defined, testable, or actually required for an application to go-live. This means, with rare exception, that performance testing is naturally investigative due to the lack of predefined requirements or quantifiable expectations.

The second factor differentiating these activities is the frequency with which a performance test uncovers a single issue that makes additional validation testing wasteful until that issue is resolved. It is

defects would be uncovered by those tests? How would the tests be designed? How would you ensure that these tests are adding value to the project? How would these tests be scheduled to complement the development process rather than repeatedly derail it?

For me, distinguishing between investigation and validation helped me bridge the gap between anecdotes and the answers for which the managers were looking. Just using the term “investigation” while discussing the value of starting performance testing early seemed to get even the most skeptical managers to pay more attention. Once they were paying attention, my ability to describe a concrete yet nimble approach to that investigation finally convinced them that the dollars they were about to approve for early investigation of performance were going to be well spent.

This approach to early lifecycle performance investigation is not complicated, although it can be a little complicated to explain. It embraces change during a project’s lifecycle, it iterates (not always in a predictable pattern), and it’s not always clear when to make the shift from investigation back to validation tasks. To simplify these potential challenges, let’s follow a linear path through the key elements of the investigation approach and discuss in detail its more dynamic elements.

When viewed linearly, the approach starts with an expression of the intent of the performance investigation. The individuals who have applied the approach have reported that the intent is easiest to both express and follow through on when it is synchronized with project deliveries or milestones. From the expression of intent, strategies are created describing the general approach to investigation enabled by each key project delivery. From each strategy, mini-plans are built for major tests or tasks identified by that strategy. As project deliveries are made, the mini-plans related to that delivery are executed in priority sequence, appropriately reporting, recording, revising, reprioritizing, and improving the application and the overall investigation plan as the work progresses. That makes for one expression of intent

How would these tests be scheduled to complement the development process rather than repeatedly derail it?

specific and predefined requirements, stories on 3x5 cards, or bar napkin sketches. If the test passes, the area of the software that the test exercised is said to be validated. More simply, validation is a testing activity that compares the software under test to the expectations that have been set or presumed for that software.

Since investigation is probably thought of as a testing activity less often than is validation, let’s start with a definition. Investigation: A detailed inquiry or systematic examination. Without going any further, we see an obvious difference between how I’ve defined validation and the way investigation is defined. Validation requires the existence of expectations about the outcome of the testing, but investigation

almost the norm for a single performance issue to lead to a pause, or even a halt, in performance validation testing. This is in contrast to functional testing, where it is fairly rare for a single test failure to essentially disable validation testing of the entire system.

When taken together, these two factors clearly imply that an overwhelming majority of performance tests should be classified as investigation, whether they are intended to be or not. Yet the general perception of many individuals and organizations seems to be “just like functional testing, performance testing is mostly validation.”

Think for a moment how you would plan for a mostly investigative performance testing effort. When would you conduct which types of tests? What types of

per project, one strategy per key project delivery, one mini-plan per significant strategy task, and one set of results per mini-plan. If this sounds like a lot of work, don't worry. This approach is actually extremely efficient, as I hope the following description will demonstrate.

Performance Investigation Expression of Intent

The expression of intent for a performance investigation can often be accomplished in a single work session involving the lead performance tester, the lead developer, the project manager, and a copy of the project plan, which is used to identify key project deliveries. Because we are articulating and recording intent, not creating another plan, we won't concern ourselves with dates; rather we want to identify the sequencing of key deliveries and estimate how much time we can expect between these deliveries. The specific deliveries we are interested in relate to hardware components, supporting software, and application functionality becoming available for investigation. For example, it is often the case on a Web development project that the first delivery that lends itself to performance investigation includes bringing up a Web server to present prototypes to stakeholders. In that case, we would want something similar to this in our expression of intent: "Delivery 1: Web server with static prototype available for X days. Investigate various Web server configurations for response time, throughput, and memory allocation. Investigate available network bandwidth and latency. Investigate impact of firewall, proxy server, and load balancer configurations. During investigation, collect configuration data to aid in validating adequacy of existing network components." Done. We can move on to expressing the intent of investigation for the next key delivery. These examples may or may not be the top priorities for investigation based on the specifics of your project, but I'm sure you can see how learning more about these areas early in the project can add value. Even

if no performance issues are detected, that information will be useful to eliminate variables when trying to pinpoint issues detected later in the project.

Information to include in the performance investigation intent:

- Investigable deliveries
- Duration of each delivery investigation
- Key areas of investigation

See the StickyNotes for an example expression of intent for a Web development project.

Performance Investigation Strategy

After sketching out the expression of intent, the performance tester can immediately begin to draft an investigation strategy for each key delivery. This strategy should be limited to a single page per project delivery and should include pictures or diagrams whenever they are more descriptive than text. When text is the medium of choice, lists typically suffice. While there is a wide range of information that may be included in the strategy, the critical components are the desired outcomes of the investigation and the key tasks anticipated to achieve that outcome. The strategies are living documents that are available to the entire team for review, comment, and revision. Depending on the nature of the project, it may make sense to complete strategies a few deliveries in advance to limit potential rework, or it may make sense to create draft strategies for all of the key deliveries while waiting for the first investigable delivery. The following information might appear in the strategy for our example delivery item: "Measure available network bandwidth and latency while increasing the number and/or frequency of page and/or file requests," and "Under moderate load, measure resource utilization on the Web server using various configuration settings."

Information to include in a performance investigation strategy:

- Intent of the investigation
- Prerequisites of strategy execution
- Tools and scripts required
- External resources required
- Risks to accomplishing strategy

- Data of special interest
- Areas of concern
- Pass/Fail criteria
- Completion criteria
- Planned variants on tests
- Load range
- Tasks to accomplish strategy

See the StickyNotes for an example strategy for a prototype Web site.

Performance Investigation Mini-plan

As a delivery approaches, we finally create what I call mini-plans. Each task in a strategy now gets up to a single page of its own, often including pictures, to spell out the remaining details needed to complete or repeat the task. The mini-plan should be completed far enough in advance to be shared with the team for recommendations or improvements and for necessary resource coordination to take place. Continuing with our example, a mini-plan might define "moderate load," identify the exact resources to be monitored (and who will set up the monitors), and specify the configuration settings expected to be varied and by what degree. Once drafted, the mini-plans can be sequenced for execution by priority as determined by key team members.

Information to include in a performance investigation mini-plan:

- Strategy task execution method
- Specifically what data will be collected
- Specifically how that data will be collected
- Who will assist, how, and when
- Additional information needed to repeat the investigation
- Sequence of tasks by priority

See the StickyNotes for an example mini-plan.

When each delivery is made, the performance investigation begins with the highest priority mini-plan related to that delivery. The most important part of mini-plan execution is to treat it like a one-to two-day exploratory testing session (see the StickyNotes for more information)—modifying the plan intelligently as results analysis leads to new priorities. At the conclusion of each mini-plan

execution, share your findings with the team, then reprioritize the remaining tasks, add new tasks, and/or remove planned tasks based on the new questions and concerns raised by the team. When reprioritizing is complete, move on to the next-highest priority task.

Keys to mini-plan execution:

- Analyze results immediately and re-plan accordingly.
- Communicate frequently and openly across the team.
- Record results and significant findings.
- Record other data needed to repeat the test later.
- Revisit investigation priorities after two days.

The key to successfully implementing the performance investigation approach, of course, is continual communication among team members. As I've indicated above, it's a good idea not only to keep planning documents available to all members and check back with one another frequently, but also to plan time into testing schedules to review and update task lists and priorities. Whether you implement this three-tier approach to planning verbatim or with a healthy dose of customization, or whether you do it using documents, spreadsheets, notebooks, or whiteboards and digital cameras (my personal favorite) is completely irrelevant as long as you keep in mind the goals of the approach: structured yet highly adaptable investigation.

Goals of performance investigation approach:

- Demonstrate to stakeholders that there is a plan of attack.
- Provide managers and stakeholders with progress and value indicators.
- Reassure managers that key information will be captured.
- Provide a structure for capturing information that will not noticeably impact the time available for actual investigation.
- Ensure that the approach is designed to embrace change, not simply tolerate it.

Case Closed

So, realistically, what have we just

done? We started by agreeing that leaving performance measurement, validation, investigation, tuning, etc., until the last set of tasks prior to releasing the application to production is risky. We then acknowledged that trying to validate or predict production performance too early is essentially pointless, and we granted that it is reasonable for managers to resist early performance testing if they believe that is what will happen. Next, made a distinction between late lifecycle validation and early lifecycle investigation to demonstrate to those managers that we are not proposing validating earlier, but instead we are proposing a whole new activity. Finally, we outlined an approach that allows us to investigate without spending a lot of time writing documents that will be rendered invalid with the first test, while giving the managers and stakeholders confidence that this is an activity that will provide results they can relate to specific, tangible value. Many managers will still want to see a return on investment analysis before they actually approve the expense of early performance investigation, but that's a good thing. In my experience, managers don't ask for an ROI until they are mentally ready to spend the money.

Of course, most managers won't have the time to hear you out unless you can get their attention quickly. I think these six words will probably do the trick: Investigate Performance Early; Validate Performance Last. **{end}**

Scott Barber is the CTO for PerfTestPlus, Inc. His specialty is context-driven performance testing and analysis for distributed multi-user systems. Contact him at sbarber@perftestplus.com.

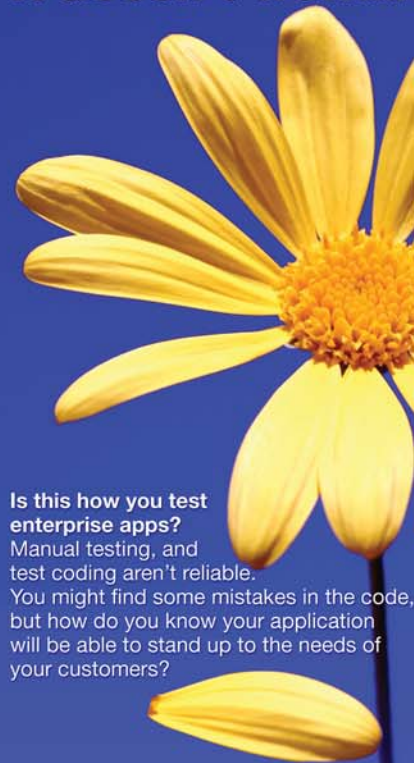


Sticky Notes

For more on the following topics, go to www.StickyMinds.com/bettersoftware

- Example expression of intent for a Web development project
- Example strategy for a prototype Web site
- Example mini-plan
- More on exploratory testing

It works, It doesn't work...



Is this how you test enterprise apps?

Manual testing, and test coding aren't reliable.

You might find some mistakes in the code, but how do you know your application will be able to stand up to the needs of your customers?

The bloom is off the rose for manual test coding. It's great if developers can unit test, but that doesn't uncover the most costly errors until it is too late.

iTKO's LISA automated testing finally gives everyone the *freedom to test* throughout your SOA project. She is an elegant, no-code testing solution that even non-technical participants can use to test components and the entire implementation as it is delivered.

From functional testing, to regressions, to load and performance tuning, there is a single solution that was built to cover your entire development and deployment lifecycle.

Meet LISA,
and fall in love with testing again.



LISA™ from iTKO.

iTKO
www.itko.com