Admit Your Issues Of Abandonment


Scott Barber

Do you have abandonment issues? No, not the kind that cause you to have anxiety attacks every time your spouse goes to the store without you.

I mean the kind that caused Alberto Savoia to write in his 2001 STQE (now Better Software) article "Trade Secrets from a Web Testing Expert": "When you adopt concurrent users as a load testing input parameter and fail to account for user abandonment you run the risk of creating loads that are highly unrealistic and improbable…You should simulate user abandonment as realistically as possible. If you don't, you'll be creating a type of load that will never occur in real life—and creating bottlenecks that might never happen with real users. At the same time, you will be ignoring one of the most important load testing results: the number of users that might abandon your Web site due to poor performance. In other words, your test might be quite useless."

We know that people who surf the Web abandon Web sites, and we know that they do so for a number of reasons. Think about your own surfing habits: Have you ever gotten tired of waiting for a Web page to load and exited the site before completing the task you had in mind? No doubt you have. Now think about your performance tests. Do your tests account for that behavior? If you are using default settings on your load generation tool, I'd wager that they don't. In fact, I'm willing to wager that if you didn't hand write a custom abandonment routine for your load generation tool, at best the tool isn't accounting for abandonment at all, and at worst the tool is accounting for it, but not in the way you think.

One of the great things about most Web sites is that if the load gets too big for the system/application to handle, the site slows down, causing people to abandon it, thus decreasing the load until the system speeds back up to acceptable rates. Imagine what would happen if once the site got slow, it stayed slow until someone "fixed the server." Luckily, abandonment relieves us of that situation, at least most of the time. Assuming that the site performs well enough with a "reasonable" load, performance is generally self-policing, even if at the cost of losing some customers/users. So, one reason to correctly account for user abandonment is to see just how many "some" is. Another reason is to determine the actual volume your application can maintain before you start losing users. Yet another reason to account for user abandonment is to avoid simulating, and subsequently resolving, bottlenecks that, realistically speaking, might not even be possible.

Consider that if we don't account for abandonment at all, our test may wait forever to receive the page or object it requested. When the test eventually receives that object, even if "eventually" takes hours longer than a real user would wait, the test will move on to the next object as if nothing were wrong. If the request for an object simply isn't acknowledged, the test skips it and makes a note in the test execution log with no regard as to whether that object was critical to the user. Now, I can think of no value that this scenario adds to a performance-testing effort, unless there is a need to show some stakeholder that, "Under the conditions [the stakeholder] specified, the average page-load time was roughly 2.5 hours." Unfortunately, we do occasionally have to demonstrate such flaws in logic physically by generating ridiculous and meaningless numbers, but that is all such data gathering provides—a demonstration, not a performance test. The point of a performance test is to gather information that will help us deliver a quality, well-performing application, not to prove that the application could sustain an impressive volume of user traffic *if* those users were willing to wait an improbable amount of time to get page responses.

Certainly there *are* some cases wherein not accounting for abandonment is an accurate representation of reality. Maybe a Web-based application has been exclusively created for an audience that has no choice but to wait. For example, a client of mine had a policy that all employees must enter the hours they worked that week between noon and 5 p.m. every Friday (with a few trivial exceptions). With roughly 3,500 people accessing this system during a five-hour period on top of the normal traffic, the system got very slow. Under other circumstances, users might abandon the site and try later or go somewhere else, but in this case, the users didn't really have a choice, so accounting for abandonment in our performance testing wasn't a valuable use of our time; it was more valuable to determine just how long the employees would have to wait under different usage scenarios.

So what about the implications of accounting for abandonment inaccurately (which is unfortunately what most load-generation tools do by default)? Many tools assume that all users abandon at a predetermined time (commonly 240 seconds) after issuing a request and then carry on requesting the following page as if nothing happened. This kind of improper accounting for abandonment can cause results that are even more misleading than if abandonment hadn't been modeled at all. To support this statement,

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

let's consider a couple of real examples of default settings from popular tools and spell out their side effects:

Default scenario A: "At 240 seconds, stop trying to get this object, log a message and move on to the next object." What if the application needed this object to load successfully to continue to the next step in the process? In this case, the tool is now "forcing" the application to respond to requests that a (non-malicious) real user couldn't even create. This situation skews page and object load times, because even though you don't actually know how long the object would have taken to load, 240 seconds is used in the calculations as if the download were successful. Worse than that, the subsequent errors caused by the forced page requests often mask the real issue (an object request timing-out), making it appear as if the test script is flawed—and none of this addresses the additional, terribly unrealistic load applied after the time-out that is likely to skew your results dramatically.

Default scenario B: "Just log when people would have abandoned for analysis but don't actually exit the test user." While this may be useful during early testing, it paints a very inaccurate picture of the actual abandonment rate for a laundry list of reasons. For example, once a simulated user encounters one page slow enough to abandon, that user typically encounters subsequent pages slow enough to abandon if that simulated user is not exited. This results in statistics showing an artificially high number of pages slow enough to abandon, even though only one actual user (or a few) would have abandoned. And once again, there is the matter of the additional load being applied by these simulated users who should have abandoned the site.

Default scenario C: "Immediately exit the user when an object doesn't load in X seconds." This may seem like a good alternative to scenarios A and B, but it might actually be worse. A typical Web page contains between eight and 60 separate objects, many of which a user doesn't actually care about; be they graphics, ads, or other non-text objects. In this scenario, the test may show every user abandoning over a graphic that a real user might not even notice is missing, let alone abandon over.

Note that I'm talking about grossly misrepresenting real abandonment characteristics. Modeling the abandonment range incorrectly by a few seconds isn't going to cause this kind of problem. However, abandoning because a decorative graphic didn't load in a certain time or not abandoning when search results took 12 minutes to return will probably distort your results significantly. Your abandonment model doesn't need to be perfect, but it does need to be reasonable.

It's also worth noting that if you don't account for abandonment at all and virtually all of your pages load faster than your performance goals, then your test would be perfectly accurate (in terms of abandonment simulation)—by accident. While personally, I'd rather be right by accident than wrong, there's no reason to settle for being correct by accident, when, in most cases, it only takes a few extra minutes to include abandonment in a performance test. I think it's worth it to gain the confidence that the results are honestly accurate as opposed to accidentally representative.

Certainly there are some cases wherein not accounting for abandonment is an accurate representation of reality.

I mentioned that logging potential abandonment without exiting the simulated user may be useful during early testing. This is true for several reasons.

For example, suppose you have an abandonment model that says all users will abandon if they encounter a page-load time of 30 seconds, but while your site is under development it's taking an average of 45 seconds to return a page, even at very low user loads. You'll still want your scripts to run all the way through to gather information and create system logs to help track down the reason the times are so slow. In this situation, abandoning all of the simulated users when they request the home page gives you no information to help tune the system. Use your best judgment early in testing about whether to just write abandonment data to the log or actually exit users when they reach the abandonment threshold for a particular page or object, but always exit users when you're executing a test intended to predict the experience of real users in production. If you don't have any empirical data to draw from or aren't sure how to calculate an appropriate abandonment threshold, use your own abandonment patterns as a starting point. As Savoia put it, "It's important to realize that even the most primitive abandonment model is a giant leap in realism when compared to the commonly used...time-out [values]."

In closing, I've included a few heuristics that I've found useful when interpreting the results of tests that account for user abandonment.

1. Check the abandonment rate before you evaluate your response times. If the abandonment rate for a particular page is less than about 2 percent, look for and handle outliers.

2. Check the abandonment rate before drawing conclusions about load. Remember, every user who abandons is not applying load. The response time statistics may look good, but if you have 75 percent abandonment, your load is roughly 75 percent lighter than you were testing for.

3. If the abandonment rate is more than about 20 percent, consider disabling the abandonment routine and re-executing the test to help gain information about what's causing the problem.

Realistically, if you are performance testing a Web site, it's really not a question of whether or not you have abandonment issues, but rather a question of whether or not you have admitted to your abandonment issues and how you have chosen to deal with them.