A Good Idea Whose Time Has Come

I have a passion for performance testing. I came by this passion accidentally, but that's a story for another day. Here, in this first instance of the Peak Performance column, I'd like to talk about the evolution of modern performance testing.

Scott Barber

For the sake of illustration, let's draw an analogy between the evolution of modern astronomy and modern performance testing. For argument's sake, we'll say that modern astronomy began around 1500 with the work of Nicolaus Copernicus (www-groups.dcs.st-and.ac.uk/~history/Mathematicians/Copernicus.html), and we'll equate that to 1995, when AOL membership expanded from 2 million to 4.5 million subscribers.

Following our analogy, Alberto Savoia (www.agitar.com/company/000009.html), whom I've often referred to as the father of modern performance testing, could be equated to Copernicus, who is often considered the father of modern astronomy. Savoia, like Copernicus, published several articles and papers that at their time were considered "interesting, but fundamentally irrelevant." Savoia's papers, published in the late '90s, in my opinion later became the foundation of current performance testing theory and practices.

However, from the time of Savoia's publications until very recently, the software testing industry as a whole simply hasn't been ready to hear about, believe in or pay for the inherent complexities of high-quality performance testing. Most of the industry has believed that if an application was built well, performance testing wasn't a necessity, in much the same way that that functional testing was felt to be a safety net for poor development in the mid-'90s.

This is not entirely different from the time between Copernicus and Newton (www.newton.cam.ac.uk/ newtlife.html), who developed the mathematics and principles of physics to support Copernicus's writings. Possibly the acceptance of the value of Savoia's work is related to the writings of such individuals as Connie U. Smith (www.perfeng.com) and Daniel Menascé (cs.gmu.edu/faculty/MenasceHome.htm), who both published widely read books, articles and presentations on the subject between 1998 and 2002.

The major difference is that instead of Copernicus' 150 years, it took a mere five years before the software industry as a whole started viewing performance testing as a good idea. Unfortunately, it was still seen as an optional set of tests that were "nice to run" during the two-week period immediately prior to going live.

"After all," I heard executive after executive say, "for what it would cost me to buy or lease a tool and bring you in for two weeks, I could double the number of servers hosting the application. I'd rather just save the money in case I do have a performance problem. Then I'll just use those dollars to add hardware."

If you were an architect or a performance tester around that time, I'm sure you understand the frustration caused by such statements. The frustration was due to the fact that once file servers, application servers and databases are added behind the Web server, for example, the paradigm of "just add another server" to improve performance—mostly valid for static graphics and text-based Web sites—breaks down. Unfortunately, it seems that not everyone got that message.

So, even two years ago, performance testing was typically an add-on to a project, not just part of how to build quality software. I will grant that the organizations that were doing performance testing as an integral part of their development process were starting very early in the project, but many of them still seemed to think that a performance tester was just an automated functional tester using a different tool.

For instance, it was generally believed that record/playback was an adequate method for creating performance test scripts, and when someone mentioned that those scripts needed to be edited by someone with a programming background to produce reliable results, they were met with looks of confusion and disbelief.

Further, it was popular when creating application workload models to simply "pick the top *x* test cases that are currently passing the automated functional tests and just rewrite those in the performance tool. That should be good enough."

Even more surprising in retrospect was that the apparent best practice for handling a performance issue seemed to be: "Observe poor performance. Enter defect. Defend tools and scripts to developer. Defend tools and scripts to lead developer. Defend tools and scripts to project manager. Close defect due to the belief that the results are incorrect. Run the same test later on a new build. Notice that the issue is resolved. Ask the developer if he or she changed anything. The developer admits to finding and fixing something

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

that could have effects like that…after the defect was closed." Sound familiar?

After about six years of trying to convince software development teams that accurate modeling of production users really makes a difference in their results; that starting performance testing and test planning early on really does pay off; that adding more hardware actually solves only a minority of performance problems; that neither "fast" nor the "8-second rule" constitute appropriate performance goals; and that a performance tester who is a mid-level everything and can interact effectively with the developers adds value to the team, I figured that I was in the wrong business and took a regular 9-to-5 testing job.

I never lost my passion for performance testing, though. I did a little writing, took some side gigs, even founded a workshop where performance testers get together and share ideas and methods for handling particularly tricky situations. Yet all of the stories, e-mails, forum postings and complaints by everyday Internet users kept demonstrating that performance testing was still not being taken seriously enough.

I resigned my position as a test manager today. I really had planned to be a test manager for several more years, but something has changed in recent months. I've been hearing questions like:

"We're about to kick off a development project; what *are* the performance testing tasks we should be thinking about right away?"

"I know we can't just extrapolate our response time and load values from the test environment to the production environment, so how *do* we find out what the production environment can handle?"

"What can I show to the development manager that will make him feel better about the performance tester and the developers working together?"

"When will we finally get some load-generation tools that have a *real* IDE? I just can't write enough code in these crippled languages to simulate my users accurately enough!"

"Can you share with me your approach to determining appropriate, meaningful and testable performance goals and requirements?"

I was pleasantly shocked.

Suddenly it seemed like the industry had started putting the focus on performance testing that for years it has deserved. Does this mean that we've reached the astronomical equivalent of the early 1900s, when Albert Einstein's (www-gap.dcs.st-and.ac.uk/~history/Mathematicians/Einstein.html) theories changed the way we all conceive light, gravity and other principles that keep the planets in motion? I believe we have.

Suddenly it seems like the industry has started putting the focus on performance testing that for years it has deserved.

All of that is to say that it is my belief that folks are ready to hear and react appropriately to the lessons we performance testers have learned in the past, and that we are ready to make the kind of fantastic advances in performance testing that have been made in astronomy over the past 60 years or so. The performance testing tools that are coming on the market this year have finally taken many of these lessons into account and are being designed and marketed to accommodate not only the "record/playback" testing that is valuable at certain times, but also the more complex testing that requires reasonably heavy scripting and direct interaction with the development team.

With this increased awareness—plus the growing education of managers and executives about the issues surrounding performance testing and the new tools enabling testers to more easily implement better tests—quality performance testing is poised to become a mainstream part of software development.

In this column, it will be my intent to share as many of the lessons that had been put on hold as I can, in digestible nuggets that I hope will be timely, valuable and applicable to performance testers new and old, as well as to the managers and executives they work for and the developers they work with.

My hope is that I can provide you with resources that you will read and hand to others on your team with a sticky note attached that says "Please read, Re: yesterday's conversation about performance testing." If I'm successful, the information will provide you with the support you've probably been looking for to advance the state of performance testing on your team.

In the next issue we'll start digging into those lessons learned—lessons about evaluating and minimizing performance-related risks; using the tools at your disposal to improve the quality and analysis of your results; organizational and process barriers to improving performance testing; and the most critical skills for a performance tester to have.

In the meantime, I'd like you to take the following from this column: All indications are that folks are ready to put quality performance testing into their plans, budgets and processes. I urge you to pull your own lessons learned from the shelf, dust them off and start sharing them. I'm willing to wager that they will be much better received now than they were when you put them on the shelf in the first place.

So, I wonder: What will turn out to be the performance testing equivalent of the first person landing on the Moon? ☒