Good Test Tools And How To Pick Them

The other day I received my December issue of Software Test & Performance magazine containing the tally of our votes for our favorite test tools. As I read through our favorites, I decided that now is as good a time as any to talk a little bit about performance testing tools.


Scott Barber

Like each of you, I have my personal favorites and biases when it comes to tools and the contexts I prefer to apply them to. With that said, I will make a significant effort to distinguish between data and personal opinions.

As a performance testing consultant for the majority of the past six years, I have had the unique opportunity to work with most of the mainstream commercial tools, an amazing number of free and open-source tools, and more than a few "homegrown" tools (some from my own "garden"). I have met only a few people who have had the opportunity to work with such a wide variety of tools—which actually supports a point I make below.

A Focus on Load Generation

Before we get started, I'd like to point out that we're going to discuss only load-generation tools in this month's column, not resource monitors, profilers or optimizers.

While I agree that these other categories of tools, as well as many others, can add a lot of value to performance testing efforts, there are too many of them and their technologies are too specific for me to even attempt to discuss them all at the same time.

Also, as far as I am concerned, no matter how a tool is packaged, if it generates load, it is a load-generation tool. Anything else it may do—even if those things are valuable and the tool does them well—is an add-on to this core functionality.

Calling it a load-generation tool establishes the expectations that if the tool is used well to implement a modeled load, the resulting generated load bears some resemblance to that modeled load; the load encompasses appropriate data variance; and the load at least reasonably simulates actual user traffic.

I often hear individuals dismiss a load-generation tool based on ancillary issues, such as what the default reports look like, how the tool interfaces with a particular test management tool, or a lack of syntax highlighting in the IDE.

Now, I, like you, have strong preferences about these features, but I'd rather spend a little extra time reformatting a report, entering some data into another system, or editing my scripts in another IDE than fight with the tool to get it to generate load acceptably in the first place.

Let's face it, if I'm not generating the right load, I shouldn't be looking at results!

So when I evaluate a load-generation tool, I am focusing entirely on its ability to efficiently and accurately generate the desired load against the specific application I'm currently testing, and to collect data from the tests in some format that I can get to.

As a performance tester, that is what really matters to me.

If several tools can do those things, and a manager or stakeholder is willing to pay for additional features, I think that's great. If, however, there are several tools that meet my minimum criteria, I'll recommend the one with which I can generate the most realistic load over the one with the coolest add-ons.

Defining What Is Best

Before digging into any particular tool, I think that it is important to remember that no matter what I, Software Test & Performance readers, market analysts, salespeople, our bosses or our clients say, the "best" tool for the job is the one with the following attributes: It is available/permissible for use; it is capable of doing the job that needs to be done; the individuals who will be using the tool are very good at getting the tool to do what needs to be done; and they *like* to use it.

In my experience, performance testing efforts using homegrown or open-source tools with generally limited or inferior capabilities but driven by individuals who know the tool very well have a much higher rate of success than efforts using top-of-the-line commercial tools driven by vendor-certified individuals with little practical experience. The notion that a "better" but untried tool will increase performance test effectiveness because of additional capabilities is simply not consistent with my experiences.

Nor do my experiences support the notion that a tool that "automagically" handles script customizations at the click of a button, rather than requiring manual code modification, actually saves time or effort in the long run.

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.

When a tester customizes a script by clicking a button, the resulting change to the script is implemented in the manner the tool developer guessed would be most common, which is not necessarily what the tester had in mind.

The truth is that "most common" is not nearly as standardized as most people would like to believe—and when the developer's idea of "most common" doesn't apply to your specific application, it is typically *much* harder to work around the automatic customization than to have simply written it yourself to start with.

In the end, no tool that you download or purchase was built specifically to test your application, making it significantly more realistic to expect you to make manual code customizations than to expect the tool always to do it for you.

Three Exciting Things

Three things happened this year that have me excited about load-generation tools for the first time in quite a while.

The first was IBM Rational's release of its brand-new load-generation tool, Rational Performance Tester, or RPT. If you're not familiar with this tool, you should know that this isn't just another Rational Robot patch—nor is it really a Robot replacement, though there are certainly places where Robot is being used that can potentially benefit from switching to RPT. It is an entirely new tool, built from the ground up. The scripting language is Java and the IDE/user interface is Eclipse, but that isn't the exciting part.

The exciting part is that it was built to target companies that develop using IBM technologies (that is, are already doing software development using Eclipse). That means that *finally* there is a tool on the market that puts the performance tester and the developer in the same environment, with the same IDE, the same programming language, and even the same source control if they so choose.

Think about what that enables. If you are a performance tester in that environment, you never again need to go further than your favorite developer's cube to get some help with the performance script. You can make your tests available to the developers such that they might not even have to minimize a window to see if what they just changed helped or hurt performance. RPT is still a first-generation tool, and we'll have to wait and see if the Eclipse developers and testers really make use of the power this extends to them.

That said, the simple fact that the builder of a load-generation tool finally recognized that developers and testers sharing an environment and a programming language was an idea whose time was long overdue makes me happy. Check it out at www-306.ibm.com/software/awdtools/tester/performance.

Finally there is a tool on the market that puts the performance tester and the developer in the same environment.

The second thing that happened was that I finally got to use OpenSTA on not one, but two enterprise-grade performance testing projects.

For the past few years, I have claimed, based on evaluations and experiments but no full-scale projects, that for testing applications that use the HTTP protocol, OpenSTA is at least competitive with the expensive tools. Last summer, though, I finally got to work on two separate projects where we gave up on the pay tools we had started with in favor of OpenSTA because, in those particular circumstances, it just worked better.

For those of you who aren't familiar with OpenSTA, you can read about it and/or download it at www.opensta.org. Both the developers' and users' communities are active on the Web, and the documentation is fabulous for an open-source tool (honestly, it's pretty good even by commercial standards).

If nothing else, it's good to know that there is an option out there for small businesses or departments to performance test their Web sites without breaking their annual budget on a load-generation tool.

Finally, in early November, Microsoft released the 2005 edition of Visual Studio.

If you haven't heard, one of the versions of VS 2005 is Visual Studio Team System, and one of the components of Team System is a load-generation tool. This is exciting news.

It was cool when RPT came out, but now I can performance test right in Visual Studio. My absolute favorite part about this is that if I am performance testing in a Visual Studio shop, I can write my performance tests in any of the .NET languages.

Oh, and did I mention that we can tie into any performance unit tests the developers have written? And how about the neat trick where I can get a code coverage report for my performance scripts?

I guess those things count as "add-ons" for a load-generation tool, but I couldn't help mentioning them. As I said, once the basics are covered—and here they certainly are—perks are always welcome!

So in a period of a little more than six months, my load generator's tool box has grown by three tools, five fully featured programming languages, two developer-grade IDEs, a whole bunch of what are now widely reusable custom functions, and the chance finally to sit next to the developers while we are doing performance testing without having to waste most of the often-limited time we get to spend together explaining an unfamiliar UI.

This is all thanks to a couple of fortuitous opportunities and at least a few load-generation tool publishers that are delivering their new tools in the format that many of us performance testers have been requesting for years.