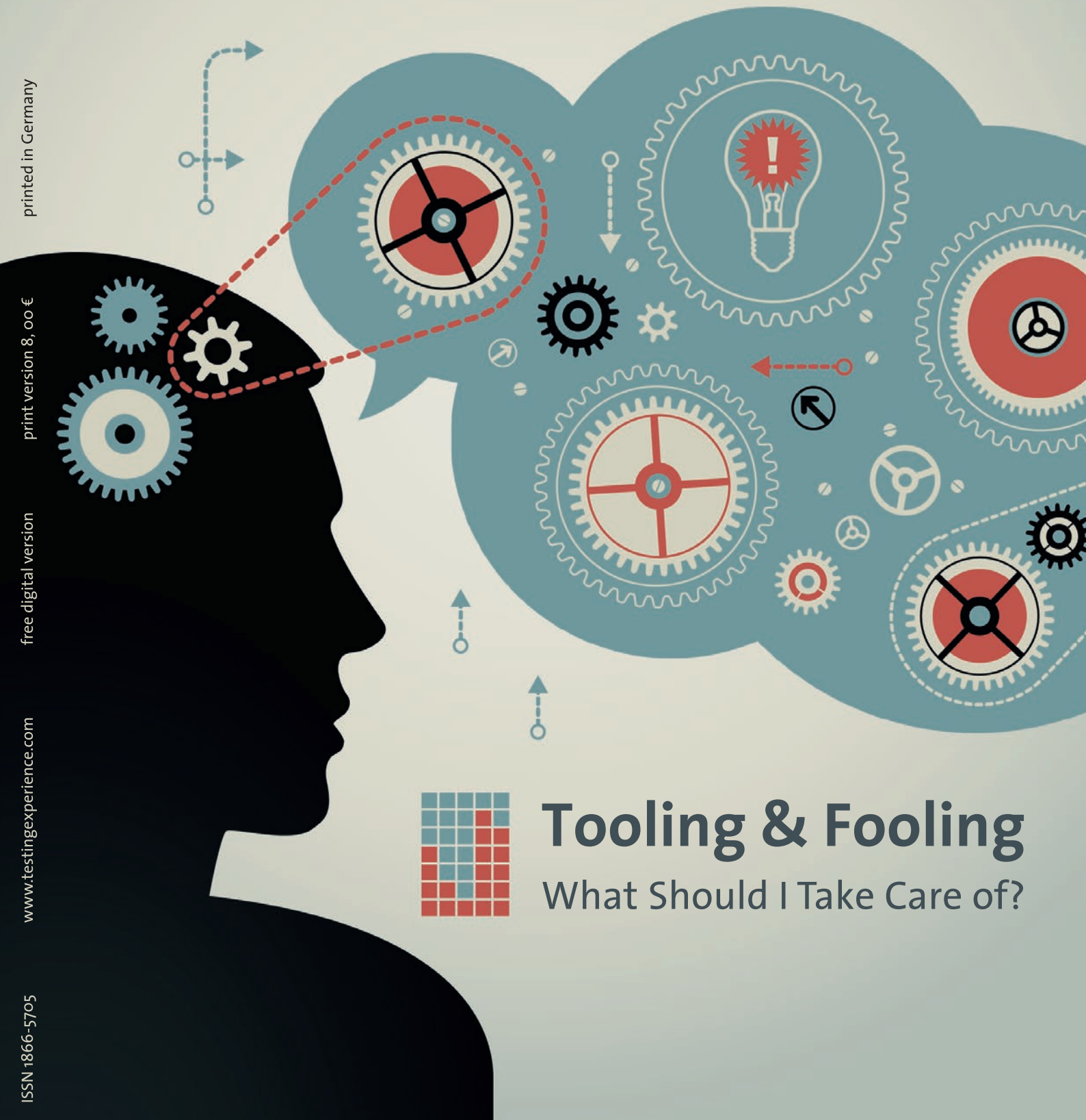
23

September 2013

testing experience

The Magazine for Professional Testers



Tooling & Fooling

What Should I Take Care of?

printed in Germany

print version 8,00 €

free digital version

www.testingexperience.com

ISSN 1866-5705

By Alex Podelko

Load Testing: Which Tool to Choose?

The topic of load testing tool selection always triggers a lot of discussion. Unfortunately, it most often turns into religious wars than objective technical analysis – partially because the outcome will depend upon your specific needs, partially because few people have time to really investigate different tools, partially because vendors are deeply involved and have their own agenda.

Let us first define load testing. The term is used here for everything requiring application of multi-user synthetic load. Many different terms are used for such multi-user testing, from performance, concurrency, stress, scalability, and endurance, to longevity, soak, stability, or reliability. There are different (and sometimes conflicting) definitions of these terms. However, they describe testing from somewhat different points of view, meaning they are not mutually exclusive.

While each kind of performance testing may have different goals and test designs, in most cases they use the same approach: applying multi-user synthetic workload to the system. The term "load testing" is used here, because it better contrasts multi-user testing with other performance engineering methods such as single-user performance testing without the need for a load testing tool.

A typical load testing process is shown in Figure 1.

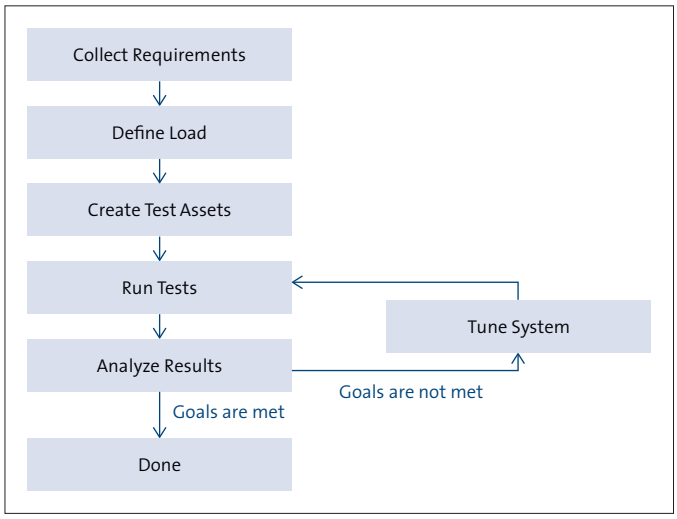


Figure 1. Load Testing Process

You have probably already seen something like this – but here two different steps are shown explicitly: "define load" and "create test assets." The "define load" step (sometimes referred to as workload characterization or workload modeling) is a logical description of the load we want to apply (e.g. users log in, navigate to a random item in the catalog, add it to the shopping cart, pay, and logout with an average of ten seconds of think time between actions). The "create test assets" step is the implementation of this workload and the conversion of the logical description into something that will physically create that load during the "run tests" step. Manual testing may still be an option in a few cases (when load is low and repeatability is not needed) – then it can be just the description given to each tester. But in all other load testing cases, it should be a program or a script.

As far as you need a tool for load testing, the subject of selecting one becomes very important. Moreover, there are attempts (not often without vendor involvement) to present a load test tool as a complete solution to load testing, making the question of selection bigger than it should be. Yes, a good tool in load testing is very important – but it is still just a tool. A carpenter needs good tools, but tools do not make his job; he still needs skills and experience to use them. The same is true in load testing. Let's look at different aspects of load testing tools, keeping in mind that they are only tools to help you do your job – they won't do it for you.

Classifying and evaluating load testing tools is not easy, as they include different sets of functionality often crossing borders of whatever criteria are used. In most cases, any classification is either an oversimplification (which in some cases still may be useful) or a marketing trick to highlight advantages of specific tools. There are many aspects differentiating load testing tools and it is probably better to evaluate tools on each aspect separately.

Here we will discuss some aspects of load testing tools and list some considerations impacting the selection process. The list is far from comprehensive and is provided rather to illustrate the existing issues, showing how the selection process for specific needs may be approached. A few tools are mentioned to illustrate certain aspects, but as there is no intention to provide a deep analysis of all available tools (there are probably a few hundred such tools around), there is no implication that the mentioned tools are necessarily better than others. Let's consider technical aspects first.

Load Generation

There are three main approaches to workload generation and every tool may be evaluated on which of them it supports and how.

Protocol-level recording/playback: this is the mainstream approach of load testing: recording communication between two system tiers and playing back the automatically created script (usually, of course, after proper correlation and parameterization). As far as no client-side activities are involved, it allows the simulation of a large number of users. The tool should support the protocol used for communication between two tiers of the system to be used.

With quick internet growth and the popularity of browser-based clients, most products support only HTTP or a few select web-related protocols. To the author's knowledge, only HP LoadRunner and Microfocus SilkPerformer try to keep up with support for all popular protocols. Therefore, if you need to record a special protocol, you will probably end up looking at these two tools (unless you find a special niche tool supporting your specific protocol). This somewhat explains the popularity of LoadRunner at large corporations using nearly all possible protocols. The level of support of specific protocols differs significantly, too. Some HTTP-based protocols are extremely difficult to correlate if there is no built-in support, so you should look for that kind of specific support. For example, Oracle Application Testing Suite may have better support of Oracle technologies.

Quite often the whole area of load testing is reduced to pre-production testing using protocol-level recording/playback. While it was (and still

is) the mainstream approach to testing applications, it is definitely just one type of load testing using only one type of load generation – such equivalency is a serious conceptual mistake, dwarfing load testing and undermining performance engineering in general.

UI-level recording/playback: this option has been available for a long time, but it is much more viable now. For example, it was possible to use Mercury/HP WinRunner or QuickTest Professional (QTP) scripts in load tests, but you needed a separate machine for each virtual user (or at least a separate terminal session). That drastically limited the load level you could achieve. Other known options were, for example, Citrix and Remote Desktop Protocol (RDP) protocols in LoadRunner – which always were the last resort when nothing else was working, but were notoriously tricky to play back. New UI-level tools for browsers, such as Selenium, have extended possibilities of the UI-level approach, allowing the running of multiple browser per machine (limiting scalability only to the resources available to run browsers). Moreover, UI-less browsers, such as HtmlUnit okr PhantomJS, require significantly fewer resources than real browsers. There are now multiple tools supporting this approach, such as Appvance, which directly harnesses Selenium and HtmlUnit for load testing; or LoadRunner TruClient protocol and SOASTA CloudTest, which use proprietary solutions to achieve low-overhead playback. Nevertheless, questions of supported technologies, scalability, and timing accuracy remain largely undocumented, so the approach requires evaluation in every specific non-trivial case.

Programming. There are cases when you can't (or can, but only with great difficulty) use recording at all. In such cases, API calls from the script may be an option. Sometimes it is the only option for component performance testing. Other variations of this approach are web services scripting or use of unit testing scripts for load testing. And, of course, you may need to add some logic to your scripts. You program the script by whatever means and then either create a test harness to execute it or use a load testing tool to execute scripts, coordinate their executions, and report and analyze results. To do this, the tool should have the ability to add code to (or invoke code from) your script. And, of course, if the tool's language is different from the language of your API, you would need to figure out a way to plumb them. Tools, using standard languages such as C (e.g. LoadRunner) or Java (e.g. Oracle Application Testing Suite) may have an advantage here. However, it is knowing all the details of the communication between client and server to use right sequences of API calls that is often the challenge.

Deployment Model

There were many discussions about different deployment models: lab vs. cloud vs. service. There are some advantages and disadvantage of each model. Depending on your goals and the systems to test, you may prefer one deployment model over another. For example, if you want to see the effect of performance improvement (performance optimization), you may be better off using an isolated lab environment. If you want to do load testing of the whole production environment end-to-end under full load and are not concerned about small variations, testing from the cloud may be more appropriate.

But for comprehensive performance testing, you may need both lab testing (with reproducible results for performance optimization) and

distributed realistic outside testing (to check real-life issues you can't simulate in the lab). Doing both would be expensive and makes sense only when you really care about performance and have a global system – but it not rare, and if you are not there yet, you can get there eventually. If there is a chance you'll face these issues, it would be better to have a tool supporting different deployment models. Whether lab or cloud, another important question is what kind of software/hardware/cloud the tool requires/supports. Many tools use low-level system functionality, so minor differences in OS or browser version may make a big difference – and it would be a very unpleasant surprise if your platform of choice or your corporate browser standard is not supported by the tool.

Scaling

When you have only a few users to simulate, it is usually not a problem. The more users you need to simulate, the more important the right tool becomes. Tools differ drastically on how many resources they need per simulated user and how well they handle large volumes of information. This may differ significantly even for the same tool, depending on the protocol used and the specifics of your script. As soon as you reach thousands of users, it may become a major problem. For a very large number of users, some automation – like automatic creation of a specified number of load generators across several clouds in SOASTA CloudTest – may be very handy.

Environment Monitoring and Result Analysis

Environment monitoring and result analysis are two very important sets of functionality. They are grouped together here for one single reason: while theoretically it is possible to do them both using separate tools, it significantly degrades productivity and may require building some plumbing infrastructure. So while these two areas may look optional, integrated and powerful monitoring and result analysis are both very important. The more complex system and tests, the more important they become.

Integration Support

Integration support becomes increasingly important as everyone talks about continuous integration and agile methodologies. There are some vendors claiming their load testing tools better fit agile processes, but in the best case it means that the tool is a little easier to handle (and, unfortunately, often just because there is not much functionality).

What makes agile projects really different is their need to run large number of tests repeatedly – resulting in the need for tools supporting performance testing automation. Unfortunately, even if a tool has something that may be used for automation, like starting by a command line with parameters, it may be difficult to discover. In case continuous integration is on the horizon (to whatever degree), it is important to understand what the tool provides to support CI.

Of course, non-technical criteria are important, too:

Cost

There are many commercial tools (with dramatically different license costs) as well as free tools. And there are some choices in between: for example, SOASTA has the CloudTest Light edition, which is free up to 100 users. There are over one hundred such different tools. Some free tools, such as JMeter, are mature enough and well-known (BlazeMeter, for example, even provides JMeter-based cloud services). But many free and inexpensive tools are very limited in functionality.

Skills

Considering the large number of tools and the relatively small number of people working in this field, the labor market supports only the most popular tools. Even for second-tier tools, there are few people around and few positions available. So by not choosing the market leaders, you will not be able to count on finding people with this specific tool experience. Of course, an experienced performance engineer will learn any tool – but it may take some time until productivity reaches the expected level.

Support

Recording and load generation have a lot of background sophistication, and issues could happen in any area. Availability of good support or at least an active user community may significantly improve productivity.

In summary, this is, of course, not a comprehensive list of possible aspect of evaluation – rather a few starting points. Unfortunately, in most cases you can't just rank tools on a simple better/worse scale. It may be that a simple tool will work quite well in your case. If your business is built around a single website, doesn't use sophisticated technologies, and load is not extremely high, then almost every tool will work for you. The further you are from this state, the more challenging it is to select the right tool. It may even be that you will need several tools.

Two main takeaways are:

- While all load testing tools look similar, they are actually quite different. And unfortunately, generic descriptions (for example, on the vendor website) are usually useless in understanding the differences.
- Your situation *is* different. A tool may be very good in one situation and completely useless in another. The value of the tool is not absolute; rather it is relative to *your* situation.

And while you may use the aforementioned aspects to evaluate tools, it is not guaranteed that a specific tool will work with your specific product (unless it uses a well-known and straightforward technology). That actually means that if you have a few systems to test, you need to evaluate the tools you consider using your systems and see if the tools can handle them. If you have many, choosing a tool supporting multiple load generation options is probably a good idea (naturally testing it with at least the most important systems prior to implementation). ■

> about the author

For the last 16 years, **Alexander Podelko** has worked as a performance engineer and architect for several companies. Currently he is a consulting member of technical staff at Oracle, responsible for performance testing and optimization of enterprise performance management and business intelligence (a.k.a. Hyperion) products. He blogs at *alexanderpodelko.com/blog* and can be found on Twitter as *@apodelko*.