12

December 2010

testing experience

The Magazine for Professional Testers

printed in Germany

print version 8,00 €

free digital version

www.testingexperience.com

ISSN 1866-5705

Open Source Tools

Pros and cons

by Bert Wijgers & Roland van Leusden

Before choosing a tool, it is wise to think about your wishes and needs first, then compare a few candidates and finally, try one or two tools. Even more so if the new tool has to interface with other software. Somehow open source tools are often introduced without any prior tool selection process. Apparently the absence of an initial license fee is all the justification that is needed. However, there are other factors influencing the total cost of ownership of a tool. These factors have to be considered up front, otherwise you may find yourself paying too much and getting too little.

In one of our recent projects it was decided to use an open source bug tracker. This decision was entirely based on the tight budget. The bug tracker was installed on an old desktop computer and all went fine until one day the hard disk of the desktop crashed. There was no backup and the information about our bugs could only be recovered by a hard disk recovery service, with significant costs.

In a tool selection process you have to consider factors that will play a role later on, during the implementation and usage stages. Only when all these factors are taken into account the right choice can be made. In this article we'll focus on tool selection for test automation and performance testing. Since interfacing with the application under test is of the utmost importance in these domains, the tool selection process has to include a trial, often referred to as a proof of concept (POC), in which the tool is put to the test. Only when commercial and open source tools are treated equally in the tool selection process you will be able to choose the one that fits best with your situation.

For every test automation or performance test project it is necessary to identify the testing scope. Is it only for your project or for other projects in the organization as well? How complex is the application under test? Is there just one application under test or are there several? Also, test tools need to fit into the overall testing architecture and they should be viewed as process enablers, not as 'answers'. Based on those requirements the tool comparison can start.

Costs and licenses

Apart from the initial license fees, some other costs of open source software are potentially lower. For instance, open source software typically has lower hardware requirements than commercial alternatives. For example, OpenSTA, used for testing simple ASP pages, is reaching up to 3000 virtual users on a load generator of 1Gb RAM on a single 1Ghz P4 processor with Windows2000. LoadRunner, in the same configuration, can handle a maximum of around 1000 virtual users.

The open source label does not imply the software is free of charge. Developers can and do charge for open source software. Open source actually refers to the availability of the source code. This makes it possible to change the software and adapt it to your specific situation.

Open source software is released under different licenses. One example is the Berkeley Software Distribution license that allows anyone to change the source code, without any obligations. Another example is the General Public License (GPL) that states that any changes you make in the source code must be released to the open source community. If a tool is released under yet another license, be sure to check whether it can be used for commercial purposes.

The Linksys saga is an example of what can happen if you use modified code and don't respect the GPL. In 2003 Linksys released the WRT54G wireless router. Some people from the Linux Kernel Mailing List sniffed around the WRT54G and found that its firmware was based on Linux components.

Because Linux is released under the GPL, the terms of the license obliged Linksys to make the source code of the WRT54G firmware available. It remains unclear whether Linksys was aware of the WRT54G's Linux lineage, and its associated source requirements, at the time they released the router. But ultimately, under pressure from the open source community, Linksys open sourced the WRT54G firmware. With the code in hand, developers learned exactly how to talk to the hardware inside the router and how to code extra features the hardware could support.

If you have plans to modify the source code from an open source tool, in order to implement customized features, you should know under which license the tool was released. Where commercial software requires a license fee, open source software might come with obligations of a different kind.

Platforms and features

Mature commercial software usually offers more features than their open source counterparts. These features range from easier installation and better manuals to script examples and fancy reports. Often they support a wider range of platforms. Currently, open source tools can only test web based applications, while commercial tools can also be used with local client server applications. Ever more often, the front end of these local systems uses internet protocols, like HTTP/S, IMAP and POP3, as well. However, the back end is often a good old mainframe that may still be there in ten years time.

One of the most important features of a tool is the support that comes with it. When a problem arises, your vendor will solve it for you. This may take some time, but at least it's off your mind. With open source tools you have to solve the problems yourself. The community may or may not help you, but anyway, the weight is on your shoulders. When choosing an open source tool for test automation or performance testing you should be prepared to spend more time to get the tool up and running. If you underestimate this extra effort, the open source tool might turn out to be more expensive than its commercial counterpart. On the other hand, if you are able to solve any problem that may arise, why pay for features that you don't need?

Current users admit that open source tooling requires more skill to deploy and maintain, compared to turnkey commercial tooling. But after a steep initial investment in acquiring those skills, the long-term costs are lower. Obviously, cutting out costly licenses saves money for other expenses, like training. However, training for open source tooling is harder to find than training for commercial tools. Only a few open source tools have certification programs

With open source, doing a POC becomes even more important, as open source software is not always documented well and might be harder to install. The level of technical skill required to install the tool might be higher since many open source projects do not focus on installation wizards. Once installed, the more popular open source tools update automatically. With others you have to keep track of the updates yourself.

Testers and developers

With the promise that no scripting is required, commercial tools appeal to testers. Since scripting is their core business, developers are not susceptible to this promise. Developers tend to prefer open source tools to automate their unit testing, since they have the skills and knowledge to adjust the tool to their needs.

In an agile software development environment testers and developers cooperate closer than in a waterfall software development environment. Testers help developers to write their unit tests and developers help testers to automate their functional tests. Testing is considered to be an integral part of the software production process and every team member's responsibility. The role of the tester changes from gatekeeper to team player.

Test automation and continuous integration of new code are essential practices in agile projects. Part of the development cycle might be test first. That is, the tests are written before the code. Developers then write code that passes the tests. This code is subsequently integrated in the product and the tests are added to the regression test suite. The new product is then automatically regression tested. Code base and regression test suite grow together in a highly interactive way.

Agile projects tend to use open source tooling for their test automation purposes. This can be explained from the fact that in agile projects test automation is not the sole responsibility of testers but the joint responsibility of testers and developers working together in multifunctional teams.

Objects and protocols

There is a fundamental difference between test automation tools and performance test tools. Test automation tools work at the Graphical User Interface (GUI) level while performance test tools work at the protocol level.

For a test automation tool, the big question is: "Does it recognize all of the objects in the various application windows properly?" Object recognition for test automation tools is never 100 percent. If object recognition is less than 50 percent, the test automation engineers will be forced to perform so many workarounds that the objective of test automation will not be reached. It will take more time to set up and maintain the test automation tool than it would cost to do the regression tests manually.

For a performance test tool the question is: "Does it recognize the client server communication protocols properly?" This question can only be answered by doing a POC in your own environment, with the application under test. Even if the documentation of a tool, commercial and open source alike, says that the protocol is supported, you might find that it doesn't, for example due to version differences.

Tools and infrastructure

A POC is also useful to compare the test results of the various tools. Test tools are also software, and we know that they contain bugs that can strike at any moment. We have done an experiment with Webload, OpenSTA and LoadRunner installed on the same machine doing the same script in an isolated lab, not connected to the corporate network. Using a single virtual user, differences of 30 percent were found in the reported transaction times. This experiment highlights the fact that the tools don't match with each other. How can we ever know if they match with reality?

The differences in transaction times are caused by the different ways in which the tools work and measure response times. We can get a reliable idea about performance, from either tool, based on the differences in the transaction times under different loads. The absolute transaction times may or may not be realistic. At least, the relative performance under different loads should be realistic. Actual measurements support this. In addition, monitoring tools like Perfmon, Rstatd, Tivoli, HPOpenview, to name a few, will assist you to determine the load you put on your infrastructure.

The tool selection should also take the infrastructure of the application under test into account. If you use a load balancer in your infrastructure, the ability to emulate the behavior of different IP addresses accessing a system (IP spoofing) is very useful. Not every tool supports this. Also, if you have different user groups that access the application through different channels (for example WAN, LAN and ADSL), the ability to emulate the behavior of different network infrastructures is needed. Those kinds of abilities are usually only found in commercial tools.

For final validation you really want to get some time on the production hardware before the application under test goes live, especially when the production hardware differs from the hardware in the test environment. So try to incorporate the production environment in the POC. It should not be too hard to get hold of it when the production hardware is not being used, for example in a weekend.

Along the way, a good performance tester will always question the data that the tools produce. From time to time, he will manually interact with the application under test to see for himself what the user experience is while the application is under load.

A test automation tool or a performance test tool is only a small piece of a much bigger test framework. It may be difficult to integrate an open source tool with the commercial software that you are using. If this kind of integration is needed then it should be part of the POC as well. The same goes for commercial tools. Some commercial tools are vendor locked with other test support tools, for defect tracking, source controlling and test management, thus forcing you to buy those tools as well. You may end up paying more than you thought you would.

Vendors of commercial tools are more than happy to organize a POC to show what their tools are capable of. Also with open source there are parties willing to help you, but their goal is different. They don't want to sell the tool, as it is probably free anyway, but they want to sell consultancy services. If such parties don't exist you should organize a POC yourself. Before you commit yourself, make sure the tool is up to the task at hand and fits in your specific situation.

Preparation and commitment

Before you can start using tools for test automation or performance testing your test documentation needs be at a certain level. Improving test documentation may buy you some time for tool selection and will certainly make your automation project go more smoothly. This step should always be done regardless of the tools you intend to use.

For test automation you might have to state the expected results in the test cases. These are often not documented because an intelligent tester understands what to expect. A test automation tool is not intelligent and has to be told explicitly what to expect. For performance testing you need to know what each user is doing and when they are doing it. In other words, you need to have usage profiles.

Another pitfall is the absence of a dedicated and experienced test engineer. Any which way you choose, it requires time and skill to implement and use tools for test automation and performance testing. If the team can not deliver a dedicated and experienced test engineer the test automation or performance test project is bound to fail, no matter which tool is used.

Foolproof tools don't exist. But if you take some time to think before you start, you can at least choose the tool that is right for you. Be prepared to invest in any new tool.

Biography

Bert Wijgers is a test consultant with Squerist. He has a keen interest in testing methods and integral approaches to software development.

Bert holds a university degree in experimental psychology for which he did research in the areas of human-computer interaction and ergonomic aspects of the workspace. He worked as a teacher and trainer in different settings before he started an international company in web hosting and design for which a web generator was developed. There he got the first taste of testing and came to understand the importance of software quality assurance. Bert has an eye for the social aspects of software development. He uses psychological and business perspectives to complement his testing expertise.

In recent years Bert has worked for Squerist in financial and public organizations as a software tester, coordinator and consultant.

Roland van Leusden is a senior test consultant with Squerist. He is specialized in non-functional testing.

He is passionate about testing and always looking for ways to improve the testing process. He is interested in the people he works with and faces them with a pleasant and targeted attitude. Quality and punctuality are very important to Roland. In close collaboration with the customer he searches for the best test strategy in relation to risk, time and the project specifications. Roland has good communicational skills and the ability come up with creative solutions in stressful circumstances.

Roland has in-depth experience with functional and non-functional testing using various open source and commercial tools.