Automating Performance Tests: Tips to Maximize Value and Minimize Effort

Created for:
Test Automation Day
Zeist, NE
23 June, 2011

Scott Barber
Chief Technologist
PerfTestPlus, Inc.
Scott Barber

CTO, PerfTestPlus, Inc.

sbarber@perftestplus.com

www.perftestplus.com

Co-Founder:

Workshop On Performance and Reliability

www.performance-workshop.org

Co-Author:

Beautiful Testing

oreilly.com/catalog/9780596159825

Performance Testing Guidance for Web Applications

www.codeplex.com/PerfTestingGuide

www.amazon.com/gp/product/0735625700

© 2006-11 PerfTestPlus, Inc. All rights reserved.
Bad performance test automation leads to:

- Undetectably incorrect results
- Good release decisions, based on bad data
- Surprising, catastrophic failures in production
- Incorrect hardware purchases
- Extended down-time
- Significant media coverage and brand erosion
More than other test automation…

Performance test automation demands:

- Clear objectives (not pass/fail requirements)
- Valid application usage models
- Detailed knowledge of the system and the business
- External test monitoring
- Cross-team collaboration
Avoid bad performance test automation

Unfortunately, bad performance test automation is:

- Very easy to create,
- Difficult to detect, and
- More difficult to correct.

The following 10 tips will help you avoid creating bad performance test automation in the first place.
Tip # 10: Data Design

• *Lots* of test data is essential (at least 3 sets per user to be simulated – 10 is not uncommon)

• Test Data to be unique and minimally overlapping (updating the same row in a database 1000 times has a different performance profile than 1000 different rows)

• Consider changed/consumed data (a search will provide different results, and an item to be purchased may be out of stock without careful planning)

• Don’t share your data environment (see above)
Tip # 9: Variance

• Static delays yield unrealistic results (a range of +/- 50% is typically adequate)

• Delays between each page should be different (users do not spend the same amount of time on every page)

• Script multiple paths to the same result (not every user will take a direct path to their desired result)

• Don’t let every path run to completion (not every user will finish what they started)
Tip # 8: Object-Orientation

- Separate scripts for every path is unrealistic (this can lead to a 1:1 ratio of scripts to simulated users)
- Many paths have overlapping activities (without OO, a change to single webpage can lead to dozens of script edits)
- Script maintenance is difficult enough (building OO scripts can make maintenance up to 10x simpler)
- Makes custom functions viable (code once, reuse over and over – even on future projects)
Tip # 7: Iterative/Agile

• Writing performance scripts is development (if you don’t treat it as such, you’ll regret it at execution time)

• Code some, test some (formal development practices are not generally necessary; applying sound principles is)

• The application will change, so will scripts (it’s more efficient to keep up with changes build-to-build than all at once)

• Use configuration management (when scripts work against a build, check them into the CM system with the build – roll-backs happen)
Tip # 6: Error Detection

• Tools have weak error detection (particularly if your site has custom error pages/messages)

• Error pages tend to load *very* quickly (a test that has 50% undetected “page not found” errors will have fantastic performance results)

• Custom functions are often needed (yes, this means writing real code – get help if you need it)

• Don’t believe your performance results until you check the logs (see above)
Tip # 5: Human Validation

- Building scripts that *seem* to work is easy (building scripts that *do* work can be hard... Check logs & use the application manually while tests are running)

- Performance test results can be misleading (reported response times aren’t always similar to what users see – get humans on the system while it’s under load)

- Numbers don’t tell the whole story (4 seconds may sound good, but users may experience 8 seconds outside your firewall)

- Users like consistent performance (get users on the system, then inject load – pay attention to their responses)
Tip # 4: Model Production

• Results are only as accurate as your models (focus on how the system *will* be used, not how someone *hopes* it will be used)

• Use multiple profiles/models (usage patterns can vary dramatically over time – the same volume of traffic in a different pattern can change performance remarkably)

• Don’t extrapolate results (when the environments don’t match, don’t guess what production will be)

• Limited beta releases are the best way to validate models before it’s too late
Tip # 3: Reverse Validate

- Released does not mean done
  (almost everyone pushes a patch shortly after release 1 – plan on it)

- Check your model against production usage
  (typically at the end of week 1 and month 1 are good)

- Re-run in test environment with revised models
  (you may be surprised at how much the performance results differ)

- Compare results from re-run against previous runs *and* production
  (this is the only way to validate your predictions and/or improve future predictions)
Tip # 2: Tool-Driven Design

• The tool was not made to test your application (expect to need to accomplish some things that the tool doesn’t make easy)

• Do not limit your tests to what is easy in the tool (it is frequently the things the tool doesn’t handle that causes performance problems)

• Don’t be afraid to use multiple tools (sometimes it's simply easier to launch two tests from different tools than it is to get one tool to do everything)

• Tools are supposed to make your job easier (if it doesn’t, get a new tool)
Tip # 1: Value First

• Sometimes the best automation is no automation (spending a week to script a difficult rare activity is not a good use of time – do that activity manually during test runs)

• Don’t fall in love with your scripts (applications change, treat your scripts as disposable – it’s often more efficient to re-record than to debug)

• Make custom code reusable (taking the time to make custom functions reusable will save time later)

• Before choosing to build a complicated script, ask “Is this the most valuable use of my time?”
Summary

10 – Design Data Carefully
9 – Build in Variance
8 – Employ Object-Oriented Orientation
7 – Apply Iterative/Agile Approaches
6 – Incorporate Error Detection
5 – Include Human Validation
4 – Model Production Usage Patterns
3 – Reverse Validate with Production
2 – Avoid Tool-Driven Test Design
1 – Value First; What *not* to Automate
Contact Info

Scott Barber
Chief Technologist
PerfTestPlus, Inc

E-mail: sbarber@perftestplus.com

Web Site: www.PerfTestPlus.com