Benchmarking Bug Detection Tools

Barmak Meftah • Vice President, Engineering, Fortify Software •
barmak@fortifysoftware.com

The best benchmarks serve non-experts by producing simple, easy-to-compare results regardless of the complexity of the solution. I propose the following three principles for creating a benchmark for bug detection:

1) Take the user's perspective. A bug detection benchmark should measure the following properties of a tool above all else:
- Ability to detect an explicit set of bugs that the community deems important.
- Ability to produce output that can be consumed efficiently (often expressed as "give me few false alarms").
- Performance (both capacity and execution time).

2) Name the scenario. Consider the TPC-C benchmark, the most widely used standard for evaluating database performance:
"In the TPC-C business model, a wholesale parts supplier (called the Company below) operates out of a number of warehouses and their associated sales districts… Each warehouse in the TPC- C model must supply ten sales districts, and each district serves three thousand customers. An operator from a sales district can select, at any time, one of the five operations or transactions offered by the Company's order-entry system." (From http://www.tpc.org/tpcc/detail.asp)
For bug detection tools, interesting scenarios might include:
- A network server that speaks a standard protocol (http, ftp, smtp, etc.)
- A set of device drivers
- A web-based enterprise application
- A privileged system utility.

3) Start with documented bugs (and their solutions). The body of open source applications is large enough and rich enough that a bug detection benchmark should never suffer the complaint that the included bugs do not represent realistic practices or coding styles. The benchmark code need not be drawn verbatim from open source applications, but the pedigree of a each bug should be documented as part of the benchmark. Because false alarms are such a concern, the benchmark should include code that represents both a bug and the fix for the bug.

Creating a benchmark is hard. It requires making value judgments about what is important, what is less important, and what can be altogether elided. These judgments are harder still for people who value precision, since computing a benchmark result will invariably require throwing some precision away. We should not expect to get it all right the first time. Any successful benchmark will inevitably evolve in scope, content, and methodology. The best thing we can do is make a start.

Benchmarks should not be made to serve experts or connoisseurs. The target audience for a benchmark should be non-experts who need a way to take a complex topic and boil it down to a simple one. Domain experts are drawn to building benchmarks partially because they help focus the world's attention on their problem. Invariably this attention causes the experts to put a lot of thought and effort into how to come out on top in the benchmark rankings. The winner of the 2nd place trophy will always bemoan the benchmark: "it ignores the subtlety of the topic, the area where my approach truly shines." But if the benchmark is good, these complaints will be quickly forgotten. The benchmark will focus research on problems that benefit the field.

Experts appreciate benchmarks because they can draw the world's attention to problems they consider important. Invariably, the experts focus on achieving a top ranking in the benchmark, rather than on the overall usefulness of the solution that is being benchmarked. This creates a situation in which everyone except the winner of the benchmark bemoans the fact that the benchmark did not accurately measure the true robustness of their solution.

The best benchmarks serve non-experts by producing simple, easy-to-understand results regardless of the complexity of the solution. This talk discusses three principles for creating a useful and easily understood benchmark for bug detection tools.