Position Paper: A Call for a Public Bug and Tool Registry

Jeffrey S. Foster
University of Maryland, College Park
jfoster@cs.umd.edu

The foundation of the scientific method is the experimental, repeatable validation of hypotheses. However, it is currently extremely difficult to apply this methodology to tools and techniques for finding bugs in software. As a community, we are suffering from two major limitations:

1. It is extremely difficult to quantitatively measure the results of a tool. We can easily count the number of warnings generated, but on a large code base it is extremely difficult to categorize all of the warnings as correct or as false positives, and to correlate them with underlying software bugs. One warning may correspond to multiple underlying bugs, or one bug may yield multiple warnings. Even identifying separable "bugs" can be problematic, since code in isolation can be correct in some sense but interact incorrectly with other code.

   Moreover, we have almost no objective way of measuring false negatives, except by comparing with other tools. But even this approach is problematic: In a recently published paper [1], we compared a number of different bug finding tools for Java, and we found that even when the tools looked for the same kind of error (e.g., concurrency errors or null pointer errors), they report different warnings in different places, most likely due to differing design tradeoffs between false positives and false negatives. Finally, we often have no independent information about the severity of bugs, making it difficult to decide if one error is more important than another. For example, null pointer errors are bad, but they often cause a program to crash at the dereference site, making identifying at least the crash site easy. Whereas race conditions that corrupt internal program logic are extremely difficult to track down, yet may not even violate standard type safety.

2. It is extremely difficult to reproduce others' results. If tools are run on open-source software, then it is easy to get the code for comparison. But it may be hard to duplicate the experimental configuration (did it compile with a different set of header files or different flags?). And even the most thorough description on paper omits by necessity many small details that can have a significant impact on the behavior of a tool. When tools are run to produce the results in a paper, they are often not made publicly available, and often if they are made available later, they may have undergone changes from the original description in the paper. Finally, reimplementing tools and reevaluating them is most likely not considered publication-worthy in top conferences—at least, not without a strong result, like showing that previous claims about the technique are false. Some seemingly unimportant details can also make reproducing experiments extremely difficult, even if the tools are made available. Changes in compilers and runtime environments over time can make building the tools difficult. Moreover, small changes in the tools' target can render them unusable. For example, in [1], we found that language drift over time made it difficult to apply the different tools—tools that ran correctly on Java 1.3 code did not accept Java 1.4 code, often for minor reasons. Even without an "official" change, tools sometimes need to be tweaked—we have found that handling new features of gcc required (minor) changes to a C front end.

Given this state of affairs, I propose that our community should adopt two policies. First, we should develop an (initially) small, shared registry for storing open source code, in raw and possibly preprocessed/munged form, along with detailed information about bugs. By keeping the registry small (at least for a few years), we could focus effort on a set of programs, making it feasible to do much more detailed evaluation of false negatives in particular. It would also be useful to get bug information from developers. One solution might be to develop or co-opt a freely-available bug tracking system that both provided advantages over current solutions and encouraged developers to report bugs and patches in a form amenable to automated extraction. (I do not know what such a system would look like.)

Second, we should strongly encourage researchers to make their tools publicly available for comparison purposes. For developers of open source software, this would be easy, but we should still cultivate a community in which the tool is made available when the paper is published. For developers of proprietary software, the registry would provide a standardized test bed, and the raw output of the tool would be made public at publication time.

References

[1] N. Rutar, C. B. Almazan, and J. S. Foster. A Comparison of Bug Finding Tools for Java. In *15th International Symposium on Software Reliability Engineering*, pages 245–256, Saint-Malo, Bretagne, France, Nov. 2004.