The Open Source Proving Grounds

Ben Liblit
Computer Sciences Department
University of Wisconsin-Madison
<liblit@cs.wisc.edu>

Open source software is an attractive target when developing and evaluating software defect detection tools. This position paper discusses some of the benefits and challenges I have encountered as a researcher interacting with the open source community.

The Cooperative Bug Isolation Project (*CBI*) explores ways to identify and fix bugs in widely deployed software. We use a mix of lightweight instrumentation and statistical modeling techniques to reveal failure patterns in large numbers of end user runs. The ultimate validation of the CBI approach comes when real data from real users helps us find real bugs in real code. Thus, field deployment is a key component of this project. CBI offers instrumented binaries for several popular open source programs for anyone to download and use.

An Open Marketplace of Code and Ideas

Most open source projects expose their entire development process to the public. This includes much more than just source code. Certainly, I can grab millions of lines of code at any hour of the day or night. However, the more disciplined projects also have revision histories, design documents, regression test suites, bug tracking systems, release schedules... all the trappings of "real" software development, free for the taking. All of this comes with no nondisclosure agreements, no lawyers, and no limits on publication of potentially embarrassing defect data.

Furthermore, open source software has gained enough market- and mind-share that it is seen as realistic. Scientific progress demonstrated on open source software is assumed to apply equally well to proprietary systems and to software engineering in general. In the marketplace of research, open source is now considered legitimate.

Openness also facilitates feedback of defect findings to project developers. As CBI discovers bugs in target applications, we report them using the same bug tracking systems used by the developers themselves. Our bug reports must compete with all others for developers' attention. If our reports are clear and describe important bugs, we gain credibility and our patches are accepted gladly. Uninformative or unimportant reports languish. Thus, observing how developers respond to the information we provide is itself an important part of evaluating our tools. The transparency of open source projects makes this process much easier to observe.

At the same time, open source licenses mean we don't need developers' permission, and there's not much they could do to either help or hinder our work. As one developer put it, if there's even a tiny chance that CBI will find a single bug, he's all in favor of it, and in any case it costs him nothing to let us try. A disadvantage stemming from this openness is that open source *distributors* are only loosely connected to many open source *developers*. It has been difficult to convert developer enthusiasm into a truly large scale deployment backed by any commercial open source vendor such as Red Hat or Novell. Ultimately the challenges here are no different from the challenges faced in partnering with any large company. Even so, it can come as a surprise that the lead developer on a project has little or no influence on a third-party Linux vendor that does not pay his salary and simply downloads his source code just like everyone else.

Finding a User Community

Many open source projects feel that because they provide source, they are under no obligation to provide compiled binaries. However, these applications can be quite complex with many dependencies that make them difficult for end users to build. For research that includes dynamic analysis of deployed software, this source/binary gap creates an opportunity for barter with end users. I spend the time to build clean, tested, installable binary packages of open source applications for CBI. In exchange, the users who download these packages agree to provide me with the raw data I need to do my research. Several people have told me that they use CBI releases simply because they are the easiest way to get the desired applications installed and running on their machines.

This approach to finding users has some disadvantages as compared with piggybacking on a commercial release. It is easy to get a few users this way, but hard to get truly large numbers. All the traffic CBI accrues from download links cannot come close to what Microsoft would get by instrumenting even a small fraction of, say, shrink-

wrapped Office 2003 CD's. Anything requiring an explicit download and install naturally selects for a more technical user base whose behavior may not be representative of software usage patterns in general. In spite of these factors, the relative ease with which one can get a small user community makes binary distribution of open source applications an attractive option for research involving deployed software.

Effects of Shortened Release Cycles

Open source projects typically release new versions earlier and more often than their commercial equivalents. This can be an advantage or a disadvantage from a research perspective. Early releases make projects' "dirty laundry" more visible. When hunting for bugs, early releases from young open source projects are a target-rich environment. Feedback provided to developers can be incorporated quickly; one project posted a new release specifically in response to bugs reported by CBI. As noted above, enthusiastic developer response is strong validation for any defect detection tool.

On the other hand, it can be difficult for software quality researchers to keep up with a moving target. CBI depends on accumulating many examples of good and bad runs. If new releases come out every few weeks, there is little time to accumulate data for any single snapshot of the code. However, if we stop tracking each new release, then users eager to try the latest and greatest may wander elsewhere. With access to many binary providers, as well as the source itself, users have no strong reason to stay in one place. Commercial providers have a more captive audience. Combined with longer release cycles, this gives researchers operating in a commercial environment more time to collect data for analysis.

Conclusions

In hindsight, I have found that working with open source makes it easier to get started, but perhaps harder to get finished. The barriers to entry are low, making it very easy to try out any crazy scheme that comes to mind. A tool can sail or sink based on its technical merits, and feedback from real developers is a fast and direct validation channel. On the other hand, the decentralized nature of open source development means there is no clearly defined decision maker. There is no one who can declare by executive fiat that his engineering team will henceforth use my tools on all their millions of lines of code. Small deployments are easy to arrange, but large ones are difficult. Researchers working outside a commercial environment should keep these trade-offs in mind as they consider using open source as a proving grounds for their work.

Further Reading on Cooperative Bug Isolation

[1] Ben Liblit. The Cooperative Bug Isolation Project. `<http://www.cs.wisc.edu/cbi/>`.

[2] Ben Liblit, Alex Aiken, Alice X. Zheng, and Michael I. Jordan. Bug isolation via remote program sampling. In Jr. James B. Fenwick and Cindy Norris, editors, *Proceedings of the ACM SIGPLAN 2003 Conference on Programming Language Design and Implementation (PLDI-03)*, volume 38, 5 of *ACM SIGPLAN Notices*, pages 141–154. ACM Press, 2003.

[3] Ben Liblit, Alex Aiken, Alice X. Zheng, and Michael I. Jordan. Sampling user executions for bug isolation. In Alessandro Orso and Adam Porter, editors, *RAMSS '03: 1st International Workshop on Remote Analysis and Measurement of Software Systems*, pages 5–8, Portland, Oregon, May 9 2003.

[4] Ben Liblit, Mayur Naik, Alice X. Zheng, Alex Aiken, and Michael I. Jordan. Public deployment of cooperative bug isolation. In Alessandro Orso and Adam Porter, editors, *Proceedings of the Second International Workshop on Remote Analysis and Measurement of Software Systems (RAMSS '04)*, pages 57–62, Edinburgh, Scotland, May 24 2004.

[5] Ben Liblit, Mayur Naik, Alice X. Zheng, Alex Aiken, and Michael I. Jordan. Scalable statistical bug isolation. In *Proceedings of the ACM SIGPLAN 2005 Conference on Programming Language Design and Implementation*, Chicago, Illinois, June 11–17 2005.

[6] Benjamin Robert Liblit. *Cooperative Bug Isolation*. PhD thesis, University of California, Berkeley, December 2004.

[7] Alice X. Zheng, Michael I. Jordan, Ben Liblit, and Alex Aiken. Statistical debugging of sampled programs. In Sebastian Thrun, Lawrence Saul, and Bernhard Schölkopf, editors, *Advances in Neural Information Processing Systems 16*. MIT Press, Cambridge, MA, 2004.