coverity

Measuring software quality

A Study of Open Source Software

Ben Chelf
Chief Technology Officer
Coverity, Inc.

Executive Summary

As part of a Department of Homeland Security (DHS) federally-funded analysis, Coverity established a new baseline for security and quality in open source software based on sophisticated scans of 17.5 million lines of source code using the latest research from Stanford University's Computer Science department. The LAMP stack — popular open source packages Linux, Apache, MySQL, and Perl/PHP/Python — showed significantly better software security and quality above the baseline with 0.290 defects per thousand lines of code compared to an average of 0.434 for 32 open source software projects analyzed.

The analysis is the first public result of a grant from the Department of Homeland Security to improve the security and quality of software. The three-year grant, called the "Vulnerability Discovery and Remediation Open Source Hardening Project," includes research on the latest source code analysis techniques developed by Coverity and Stanford scientists. The analysis identified 40 of the most critical security vulnerabilities and defect types found in software. The analysis is one of the first-ever comprehensive attempts to establish a technique and framework for measuring the quality and security of software.

Measuring software quality – a new approach

In the past decade, the open source model of software development has gained tremendous visibility and validation though popular projects like Linux, Apache, and MySQL. This new model, based on the "many eyes" approach, has led to fast evolving, easy to configure software that is being used in production environments by countless commercial enterprises. However, how exactly (if at all) do consumers of open source measure the quality and security of any piece of software to determine if it is a good fit for their stack?

Few would disagree that many eyes reviewing code is a very good way to reduce the number of defects. However, no effective yardstick has been available to measure how good the quality really is. In this study, we propose a new technique and framework to measure the quality of software. This technique leverages technology that automatically analyzes 100% of the paths through a given code base, thus allowing a consistent examination of every possible outcome when running the resulting software. Using this new approach to measuring quality, we aim to give visibility into how various open source projects compare to each other and suggest a new way to make software better.

By comparing the LAMP stack with our new quality baseline, we see that it is demonstrably better than most other open source projects reviewed in this report.

The cost of software defects

Bugs are a fact of life for software development organizations of all sizes. Unfortunately, some level of defects has also become the expected norm for a software purchase, whether for a small business' payroll system, a medium business' inventory controls system, or a large business' enterprise resource planning (ERP) deployment. However, in a 2002 study, the National Institute of Standards and Technology (NIST) estimated that software defects cost the U.S. economy upwards of $60 billion a year. NIST also found that detecting these defects earlier and with more diagnostic accuracy could result in as much as $22 billion in annual savings. The hard truth is that software defects affect both open source and commercial software and are very costly to all users and producers of software.

Discovering defects automatically

In recent years, ground-breaking research from the Computer Systems Laboratory at Stanford University has made it possible to quickly and automatically analyze tens of millions of lines of code to look for defects that can cause run-time crashes, performance degradation, incorrect program behavior, and even exploitable security vulnerabilities. At Coverity, we commercialized this new approach to improving software quality and security and made it available to enterprises needing to vastly improve their testing efforts.

Now, for the first time, we apply our technology on a wide range of open source projects in aggregate and assess the results. By identifying defects and making them available to the open source community, we expect the quality and security of open source software to accelerate through the addition of a billion "automated" eyes. Our hope is that the results of this study over open source software can also set a higher bar of quality and security for developers of proprietary software.

Measuring quality automatically

No metric is perfect. This report does not propose the results of source code analysis as an absolute measure of quality, but rather as a new and effective way to assess code quality directly in terms of the number of software defects. No automated analysis can detect all of the bugs in a piece of software. However, many program level defects fall into the range of bugs that we can detect, making our results not only a good measure of the overall quality, but also a standard and repeatable metric with which to compare two code bases. Furthermore, the advances made recently in terms of scalability, low false positive rate, and ease of integration allow us, for the first time, to plug in dozens of open source packages to be analyzed with little human intervention required.

Rather than using metrics such as cyclomatic complexity to indirectly tell us the quality of code, we rely on actionable, easy to verify defect cases that pinpoint the root cause and exact path to a software problem. Compare the two approaches here:

Cyclomatic complexity framework
(1) "Function 'foo' has too many paths through it."

Coverity framework
(2) "Function 'foo' has a memory leak on line 73 that is the result of an allocation on line 34 and the following path decisions on lines 38, 54, and 65 …"

Our belief is that a metric based on the latter is much more valuable in measuring source code quality. Today, many open source packages rely on our static source code analysis as a key indicator of reliability and security. For example, MySQL, PostgreSQL, and Berkeley DB have certified versions of their software that contain zero Coverity defects.

Analysis Details

Speed and accuracy of analysis

In this report, we analyzed 32 different open source packages. Most took between minutes and a few hours to analyze, with the total analysis time for all 32 projects requiring 27 hours on a 3.2 gHz Linux box. Also, our checks typically report less than 20% false positives.

Required set up

Very little manual work was required to integrate our source code analysis technology to analyze these open source packages. Once we had the code, all that was required for most was the typical './configure; make' command.

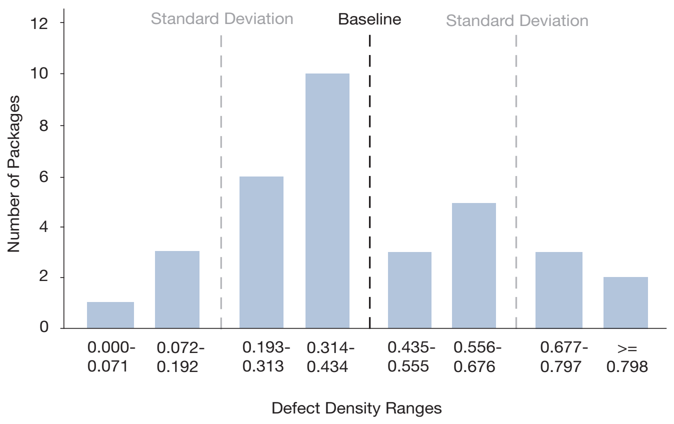
MySQL, PostgreSQL, and Berkeley DB have certified versions of their software that contain zero Coverity defects.

Results

The average defect density for the 32 open source packages that we analyzed was 0.434 defects per thousand lines of code. The standard deviation for this set of results was 0.243. Table 1 (on the following page) shows the raw data including lines of code analyzed, number of defects found, analysis time, and defect density (number of errors per thousand lines of code). Graph 1 (below) shows the distribution of defect density based on ranges that represent 1/2 of a standard deviation. Graph 2 (below) shows a comparison of the LAMP stack with the baseline derived from the analysis of the 32 open source packages. The average defect density for LAMP was 0.290 and all but one of the LAMP packages had a better than average defect density.

Graph 1

Distribution of defect densities in open source packages



Graph 2

How does LAMP stack up?

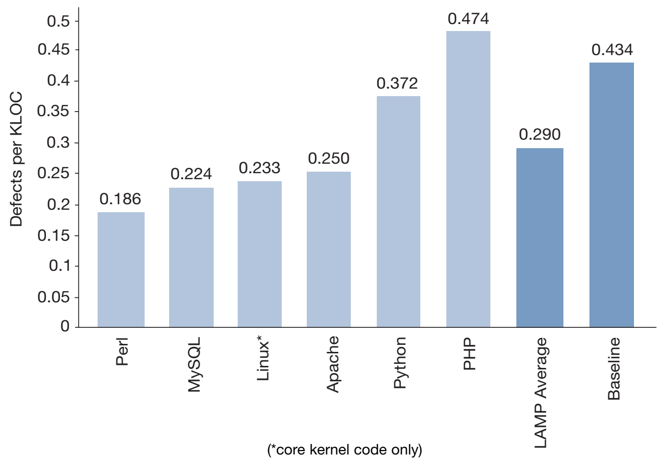


Table 1

Code Base	Lines of Code	Number of Errors	Analysis Time (min.)	Defect Density
Amanda	87,332	108	8	1.237
Apache	127,839	32	10	0.250
Ethereal	1,157,801	143	108	0.124
Firebird	239,701	163	13	0.680
Firefox	303,908	108	24	0.355
FreeBSD	1,582,166	635	257	0.401
Gaim	320,930	113	18	0.352
Gcc	692,980	140	65	0.202
Gnome	1,954,504	896	172	0.458
Icecast	37,047	12	1	0.324
Inetutils	71,892	29	4	0.403
Linux*	3,171,631	1062	254	0.335
Mplayer	484,554	284	38	0.586
MySQL	607,639	136	68	0.224
NetSNMP	173,138	148	16	0.855
OpenLDAP	254,004	158	20	0.622
OpenSSL	194,751	66	19	0.339
OpenVPN	69,610	7	4	0.101
Perl	479,759	89	25	0.186
PHP	430,817	204	36	0.474
PostgreSQL	815,562	295	38	0.362
ProFTPD	89,834	26	4	0.289
Python	258,272	96	16	0.372
Samba	310,592	216	34	0.695
Snort	82,919	48	4	0.579
SQLite	60,727	31	6	0.510
Squid	134,690	53	8	0.393
TCL	120,538	69	11	0.572
WxWidgets	303,283	73	39	0.241
X	2,353,980	1681	224	0.714
Xine	576,526	347	35	0.602
XMMS	116,788	6	4	0.051

(*including all drivers)

Conclusions

Given the observed distribution of defects over the 32 packages analyzed, we conclude that using the results of source code analysis is an effective metric in determining software quality. Furthermore, by comparing the LAMP stack with our new quality baseline, we see that it is demonstrably better than most other open source projects reviewed in this report.

While our data suggests that size of project in lines of code is not an indicator of quality, we hypothesize that the following factors determine the defect density of software as measured in our study and should be considered when assessing the quality of software:

- The ratio of developers to size of code
- The percentage of possible execution environments that are tested pre-release
- The number of users utilizing the software regularly

The benefit of our new approach in measuring quality is that it provides a measurement that is an objective combination of the above factors. Though it may be difficult or impossible to determine exactly how much of the code is tested, how many people are using it, or even how many developers are working on it at any given time, we have shown that overall code quality can still be measured.

Future work

Producing reports about defects in code doesn't actually make the software better. As such, we are releasing the defects discovered in this study to anyone who is an active maintainer of the projects analyzed. While we realize that this creates somewhat of a Heisenberg Principle problem (i.e., we observe the quality of the baseline and as a result it changes), we hope to answer the following questions as the work from this project continues:

- What types of defects are considered the most critical?
- What factors lead to a defect being considered critical? (e.g., type of defect, location in code, perceived impact, etc.)
- What factors can be used to predict the measured defect density of the code?
- How does the quality and security of development branches compare to that of stable branches?
- How can we combine defects found via other means as part of this metric?

Answering these questions will help us further understand how to address the problem of defects in code in our attempt to make the world's software better.

If you are interested in learning more, go to: http://scan.coverity.com

As our analysis is continually run over open source projects, this site provides data on the latest analysis results of the code bases reviewed in this study.

By identifying defects and making them available to the open source community, we expect the quality and security of open source software to accelerate through the addition of a billion "automated" eyes.

About the technology used in this study

Coverity Prevent™ is an advanced static software analysis tool designed to make software more reliable and secure. It relies on a combination of dataflow analysis, abstraction, and highly efficient search algorithms that can detect over 40 categories of crash-causing defects while achieving 100% path coverage. Types of defects detected include memory leaks, buffer overruns, illegal pointer accesses, use after frees, concurrency errors and security vulnerabilities. Coverity Prevent™ also efficiently detects hard-to-see bugs that span functions and modules. Most importantly, no changes to the code or build are required and the analysis is fast, scaling linearly with the code size.

coverity

Coverity Inc. Headquarters
185 Berry St. Suite 3600
San Francisco, CA 94107
(800) 873-8193