The background consists of a large, abstract, grayscale image with curved, overlapping shapes that resemble a lens or a sphere, creating a sense of depth and focus.

**How Open Source
and Commercial Software
Compare:
MySQL 4.0.16**

**A Quantitative Analysis of Database
Implementations in Commercial
Software and in MySQL 4.0.16**

Fourth in a series of Open Source and
Commercial Software Comparisons

Introduction

Proponents of Open Source software have long claimed that their code is of higher quality than the equivalent commercial software. Opponents of Open Source argue just the opposite: that Open Source software is inherently unreliable. Until now, there is little independent, objective data available to support either view.

Reasoning's automated inspection service makes possible, for the first time, independent, meaningful comparisons of different software projects. Using its proprietary automated software inspection process, Reasoning identifies critical crash-causing, security related defects at the source code level.

Reasoning has embarked on a series of inspection projects of Open Source code, for the purpose of determining whether applications built in Open Source are of higher quality and reliability than commercial applications. Because they represent the most appropriate comparison with Reasoning's knowledge base of code inspections, selection criteria for our Open Source projects are based on the following:

1. An active Open Source community, which shows that there are stakeholders in the development effort
2. Code in development, which allows us to see the state of code earlier in the lifecycle
3. Stable code base (i.e. 2.x or 3.x release), to avoid the vagaries of a "starter project"
4. Usage within the software industry, to ensure that there is pressure from the customer community to deliver quality software

On Feb 11, 2003 Reasoning published the first study, in which the Linux TCP/IP stack was compared to commercially developed TCP/IP stacks. This comparison showed that an active, mature Open Source project may have fewer defects than a similar commercial project.

On July 1, 2003 Reasoning published the results from its second Open Source inspection, this time of the Apache http server version 2.1. The Apache code inspected was a pre-release, less mature version that enabled us to compare the results to less mature commercial code we have inspected. In this comparison we found that Open Source and commercial software start at a very similar defect density.

On July 30, 2003 Reasoning published the results from its third Open Source study, this time inspecting the mature Tomcat 4.1.24 application server code. This was our first Java-based Open Source inspection and we found that Tomcat showed a defect density similar to proprietary code at a similar point in the development lifecycle. (Readers note: because Reasoning inspects for different defect classes in Java as compared to C/C++, we keep separate databases of our inspection results and metrics. All figures in this paper relate to our C/C++ inspections).

These reports generated numerous requests for more information about Open Source development and how it compares to commercial development. In response to those inquiries, Reasoning invited developers to vote for the next Open Source code inspection project. Not surprisingly, the winner of this was the MySQL Open Source database, the most popular Open Source database available today and one that fits within the criteria detailed above.

MySQL also brings a unique set of parameters not seen in our earlier work. Unlike Linux and Apache, MySQL is backed by a for-profit, commercial entity. With a dual licensing structure, the MySQL database is available at no cost under the GPL Open Source license and also available under a warranted commercial license. MySQL (the company) produces revenues by selling commercial software licenses, professional services, and customer support.

The specific version Reasoning inspected is MySQL version 4.0.16, which represents production-level code. MySQL 4.0.x has been a production-level product since March 2003.

**What Is
Open Source
and Why
Might It Be
Better?**

Commercial software development typically follows a model where the commercial software vendors distribute their products in the form of executable or object code, and require that their customers acquire a license to use those executables within certain guidelines. These customers do not acquire a license to change the source code, and therefore cannot extend the functionality of the executables. In rare cases, and only by specific arrangements with the vendor or an authorized contractor of the vendor, can customers gain access to the source code.

In this commercial model, customers rely on the vendor to make modifications, updates, and extensions to the source code. Customers will report defects and submit feature requests to the commercial vendor in order for these modifications, updates, and extensions to be developed and implemented by the commercial vendor. In this model, the commercial vendor owns and prioritizes the implementation of defect fixes and feature enhancements.

The Open Source development model has brought a radically different development, licensing and distribution model to the market. Two distinct characteristics that set Open Source apart from traditional proprietary models are that:

- The source code of Open Source programs is accessible to users in such a way that they can make changes or extensions to that code
- Changes and extensions are freely redistributable

These characteristics allow for source code to be worked on simultaneously by an unlimited number of geographically dispersed developers, without the need for those developers to be employed by the same software vendor.

This Open Source model encourages several activities that are not common in the development of commercial code, including:

- Many users don't just report bugs, as they would do with commercial software, but actually track them down to their root causes and fix them
- Many developers are reviewing each other's code, if only because it is important to understand the code before it can be changed or extended. It has long been known that this peer reviewing is an effective way to find defects
- In the typical Open Source model, programmers organize themselves around a project based on their contributions. The most effective programmers write the most crucial code, review the contributions of others, and decide which of these contributions make it into the next release

For these reasons, Open Source enthusiasts claim that the Open Source model produces better quality software than commercial software development.

Automated Software Inspection

Software inspection – the process of examining source code to identify defects – is a standard practice in development organizations and is widely recognized as the best way to find defects. Inspection is hardware-independent, does not require a “runable” application or a suite of test cases, and does not affect code size or execution speed. The majority of code inspections are performed manually. Theoretically, the greatest number of defects can be uncovered when a developer reads through the code line by line. However, this process is slow, painstaking, and fraught with inconsistency – and does not scale to handle the growing number of multimillion-line applications.

As a code base grows, the cost of a complete manual inspection becomes prohibitive, and the volume of code is intimidating to developers. As a result, manual inspections are only performed on subsets of the source code.

Reasoning's automated software inspection (ASI) services provide many of the benefits of a manual code review in significantly less time, and at a dramatically lower cost, than manual inspection. Because it is delivered as an outsourced service, in-house development resources are not diverted from current development projects. The Reasoning services identify defects that can cause application crashes, data corruption, or security vulnerabilities, and provides actionable reports for the development team. The results of automated code inspection are reports that:

- Make defect removal fast and simple by identifying the location and describing the circumstances under which the defects will occur
- Identify the parts of the code with the greatest risk, enabling the development organization to focus QA and testing resources where they are most needed
- Compare code quality with a benchmark

MySQL Inspection Results

Reasoning inspected MySQL version 4.0.16. We chose this version because it was the latest production release at the time of the study. The inspection was conducted using the exact same process Reasoning uses for customer projects.

The MySQL inspection consisted of 517 source files and 235,667 lines of source code, not including user include files, header files, blank lines, and comments.

The defect classes Reasoning inspected for in MySQL include:

- Memory leaks - refers to the loss of available memory space that occurs when dynamic data is no longer used, but is never de-allocated. Each time the leak occurs, the application drains the available memory pool. On some systems, memory may be allocated from a global pool, in which case the loss of available memory may affect the entire system, not just the application that caused it. Eventually, the performance degradation may lead to a fatal out-of-memory condition.
- NULL pointer dereferences (NPD's) - occur when there is an attempt to access data at an invalid address. This defect class also reports situations where a NULL pointer is used in an assignment – even though the assignment is not a defect by itself, a subsequent dereference will cause a program exception. The defect is reported at the assignment, because this is the earliest point in the code where the problem could be identified.
- Bad deallocations - use of an inappropriate memory release operation for deallocating memory, or to the deallocation of memory that was never explicitly allocated. Depending on the compiler and the specific operation, the impact of this defect may range from no effect at all, to unexpected results, a memory leak, memory corruption, or a program exception.
- Out of bounds array access's - an array index expression is not within the upper and lower bounds of the array. This defect can cause data corruption, or can lead to a program exception.
- Uninitialized variables - local and dynamic variables are not explicitly initialized prior to use. Usage of uninitialized variables can cause unpredictable results in the program, because the value of the variable is essentially random.

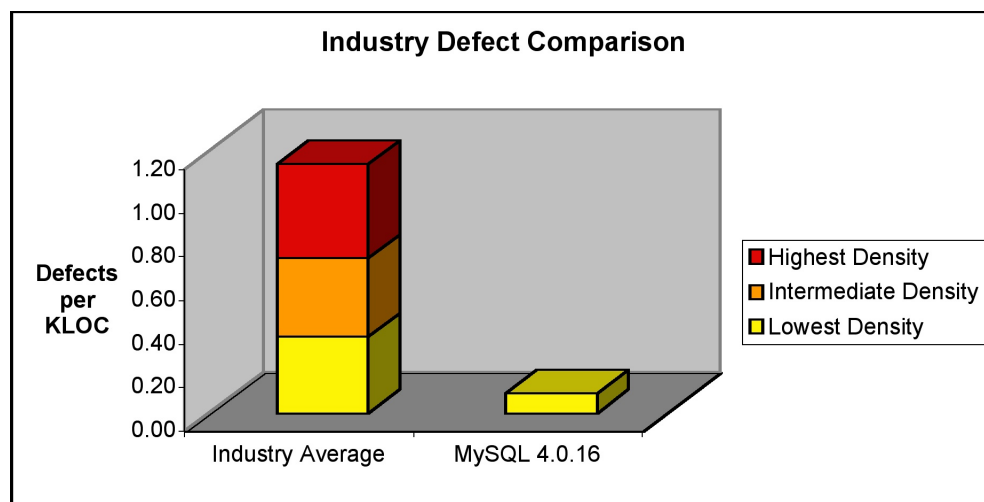
Reasoning found 21 defects, resulting in a defect density of 0.09 defects per KLOC. The following table details, per defect class, how many defects were found in the MySQL inspection.

Inspection Class	Defect Instances	Files Affected
Memory Leak <i>Reference to allocated memory is lost</i>	3	3
NULL Pointer Dereference <i>Expression dereferences a NULL pointer</i>	15	13
Bad Deallocation <i>Deallocation is inappropriate for type of data</i>	0	0
Out of Bounds Array Access <i>Expression accesses a value beyond the array</i>	0	0
Uninitialized Variable <i>Variable is not initialized prior to use</i>	3	3
Total Defect Instances	21	--

Comparison with the Code Quality of Commercial Software

In the reports to our customers, Reasoning provides a benchmark against our most recent 200 projects, totaling over 35 million lines of code. The average defect density of these projects is 0.57 defects per KLOC. In this sampling 33 percent of the projects had a defect density below 0.36 defects per KLOC (yellow), 33 percent had a defect density between 0.36 and 0.71 defects per KLOC (orange), and the remaining 33 percent had a defect density above 0.71 defects per KLOC (red).

As the following chart shows, at a defect density of 0.09 defects per KLOC, MySQL 4.0.16 falls in the category of the lowest density inspections.



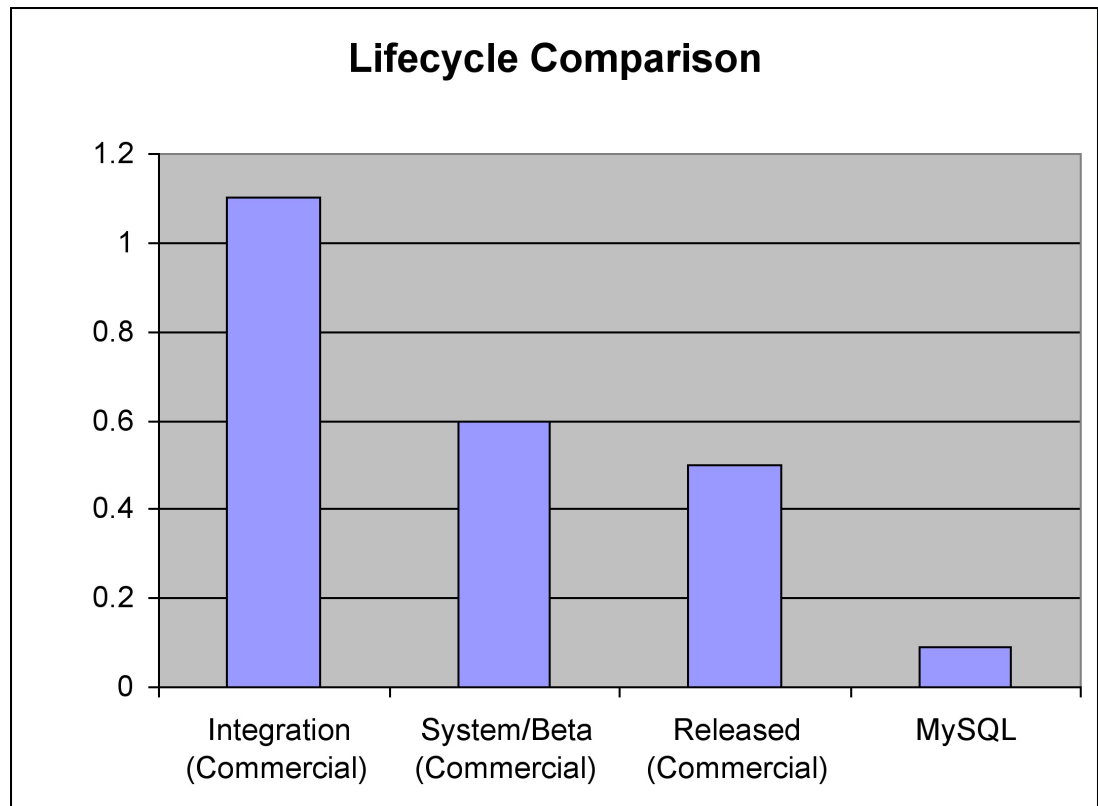
Reasoning’s commercial inspections include many database products by several different vendors, including enterprise and embedded databases. We have compared the code quality of MySQL to those products.

The overall defect density of the proprietary database code was 0.58 defects per KLOC, very close to the Reasoning average of 0.57 defects per KLOC calculated from our most recent 200 inspections.

The maturity of the commercial database components ranged between one and five years in the field. We did not find any correlation between their maturity and their defect density.

We *did* find a relation between the defect density and the phase in the lifecycle in which the commercial code was submitted. Released code had a defect density of 0.50 defects per KLOC, code that was in system or beta test had a density of 0.60, and code that was still in integration had a density of 1.1 defects per KLOC.

At a defect density of 0.09 defects per KLOC, the version of MySQL we inspected has a defect density that is about six times lower than the average of comparable proprietary projects.



For reasons of client confidentiality, we cannot disclose further information about these commercial projects.

Feedback from the MySQL Team

An important final factor is the percentage of the defects that are fixed by the client organization. Across all Reasoning's inspections, our customers have fixed an average of 85 percent of the defects reported.

Of the 21 defects that Reasoning reported, the MySQL team:

- Immediately resolved 13
- Based on their strong domain knowledge, the team determined that eight would not manifest themselves in the field. This consisted of six NPD's, one uninitialized variable, and one memory leak

One enthusiastic team member responded:

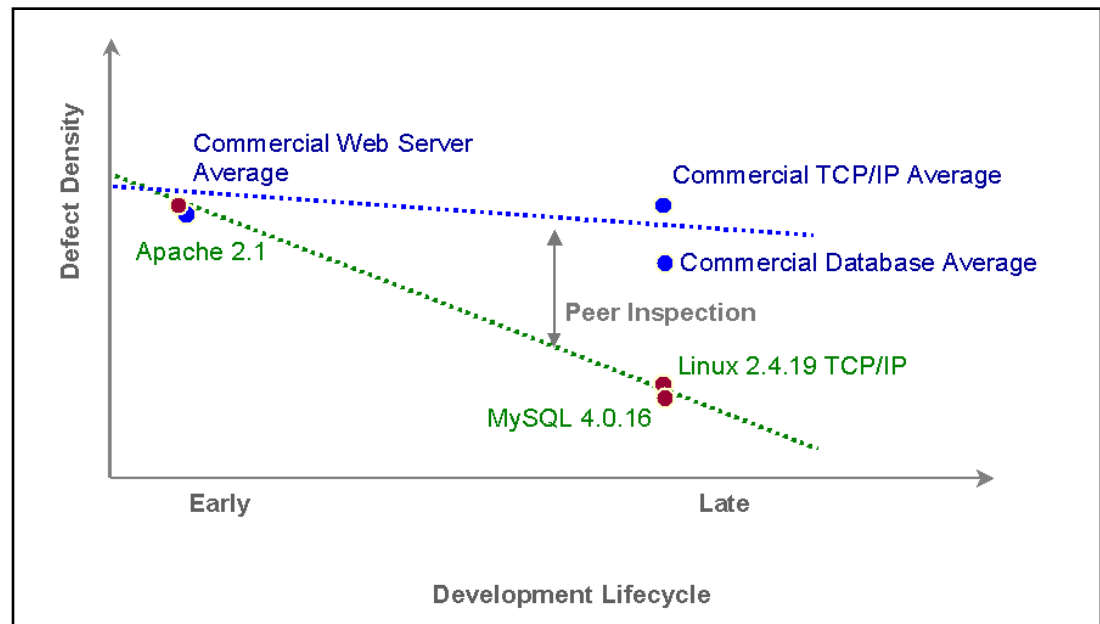
"...the report is great! Frankly speaking, I was very surprised to see how good this automated analysis was - I was somewhat skeptical, as I played with other "code analysis" tools before. This one was completely different! Deep analysis, and good reasoning."

Conclusions

From the previous Open Source inspections, Reasoning determined that the Linux TCP/IP stack showed a lower number of defects than the majority of the commercial TCP/IP stacks inspected. This was a comparison of very mature software and, on average, showed Open Source to be superior. With the Apache inspection, we gained an approximation of the initial defect density achieved by Open Source and determined that early in the development cycle, Open Source and commercial software have a very similar quality.

This MySQL inspection provides an important data point validating the earlier findings with the Linux TCP/IP study, and possibly puts an exclamation point on the value of peer review. Although more data will be needed to prove out a conclusion, when the Open Source and commercial data points are plotted, we can make an approximation of the change in defect density over time.

The figure below shows this initial approximation. Given the limited data, Reasoning sees Open Source as being faster, on average, than commercial efforts at removing defects from software. This is not as expected, since commercial software companies often invest considerably in testing tools and time in order to meet reliability requirements demanded by their customers and the marketplace.

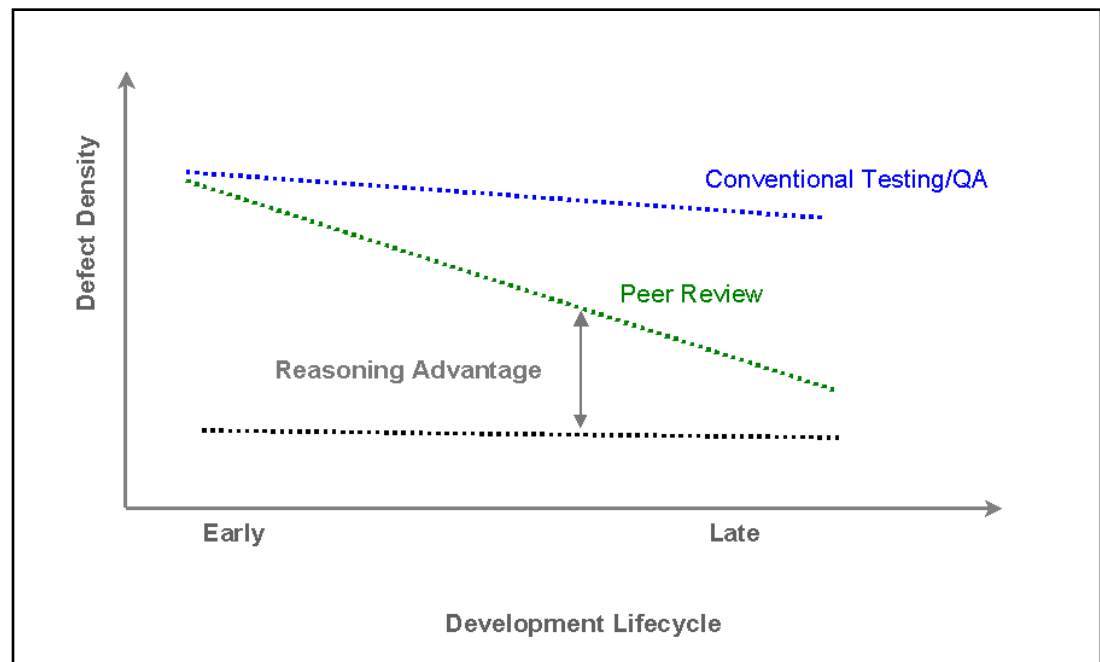


It should be noted that Open Source can end up with fewer defects. Because the evidence shows that both development environments are likely to start with a similar number of defects, the core of the difference must be after development starts. In that time period, the main difference is the number of developers actually looking at the code. Commercial software companies are more likely to have sophisticated tools; however, they are unlikely to have achieved the same level of peer review that Open Source code can achieve, which is critical in finding defects.

This is certainly not a new concept. Software engineering leaders such as Michael Fagan and Capers Jones have repeatedly pointed out the advantages of peer review and software inspection. The advantage of the inspection process – to identify and remove defects – is likely to be heightened in Open Source projects, since the reviewers are often quite independent from the contributors. This provides the autonomous nature that Fagan’s review guidelines are aimed at. While it may be difficult within a company, it is a natural consequence of Open Source.

It has been clearly demonstrated that incorporating inspection into the software development process results in significant reductions in defect rates.

Automated inspection services like Reasoning's provide the best of both worlds to commercial customers. They enable companies to continue building new features within their deadlines, while still achieving the quality benefits of peer reviews and code inspections.



By identifying defects not found in conventional testing and by peer review, ASI provides companies willing to invest in automated software inspection services with the ability to not only close the gap, but to create an advantage.

Considerations As earlier stated, Reasoning has chosen Open Source projects to best compare against commercial development. Clearly, the data presented here cannot be extrapolated to all Open Source efforts, or to all commercial efforts. Other Open Source projects may or may not show the same improvement in defect densities as they mature. Key drivers in this are:

- The number of active developers reviewing the project
- Maturity level since the last major release (i.e. later dot releases)
- Strong customer use

However, when selecting an Open Source (or any third-party) product, we recommend a thorough analysis of its reliability.

About Reasoning

Reasoning is a leading provider of automated software inspection services that help development organizations reduce the time and cost involved in finding software defects. The company's business is focused on organizations that develop Java, C, and C++ applications.



For more information, contact:

Reasoning, LLC

PO Box 478

Menlo Park, CA 94026-0478

650 324-2510 (phone)

415 762-1992 (fax)

Email: reasoninginfo@reasoning.com