Is Open Source Software Better than Commercial Software?

OSQA/ASQ
Ottawa, June 24, 2004

Jasper Kamperman, PhD
Director of Product Management
Reasoning Inc.

jasper.kamperman@reasoning.com

Agenda

- The Debate
- The Study
- The Method
- The Results
- Analysis
- Conclusions
- About Reasoning

The Debate

- Proponents of Open Source software have long claimed their code is of higher quality
  - Power of peer review
  - Root cause analysis on site enables easier fix
- Commercial software vendors have long claimed their code is of higher quality
  - Market focused
  - Defined processes for development and testing

The Study – why?

- Reasoning is uniquely positioned:
  - Inspection data on over 100 MLOC of commercial code
  - Independent validation
- Our customers wanted to understand the real differences between Open Source software and Commercial software
- Illustrates the value of Software Inspection with real-world examples – unlike commercial source, we can disclose defects we found in Open Source

The Study – scope

- Linux TCP/IP ?
  - Well-defined set of published requirements
  - TCP/IP implementations have been in existence over 20 years
  - Version 2.4 had been fielded for over a year
- Apache ?
  - Wide market acceptance
  - Stable code base
  - Version 2.1-dev is pre-release
- MySQL ?
  - Wide market acceptance
  - Commercially-backed
  - Production-level code
- Tomcat ?
  - Active Open Source community
  - Java based

REASONING

The Study – timing

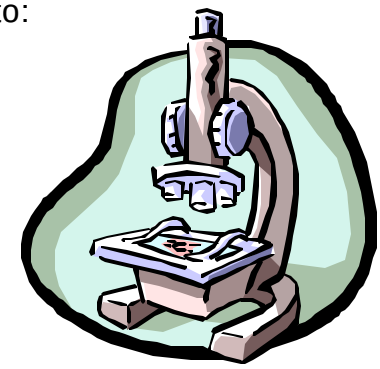
When was the study performed?

- Linux TCP/IP v2.4.19 = December 2002
- Apache v2.1 = April 2003
- MySQL 4.0.16
- Tomcat v4.1.24 = May 2003

REASONING

The Method: Software Inspection

- AKA Peer Review
- Implicit in Extreme Programming
- Examination of source code to:
  - Detect crash-causing defects
  - Trace code to requirements
  - Check coding standards
- Formal design and code inspections average about 65% in defect removal efficiency. Most forms of testing are less than 30% efficient*

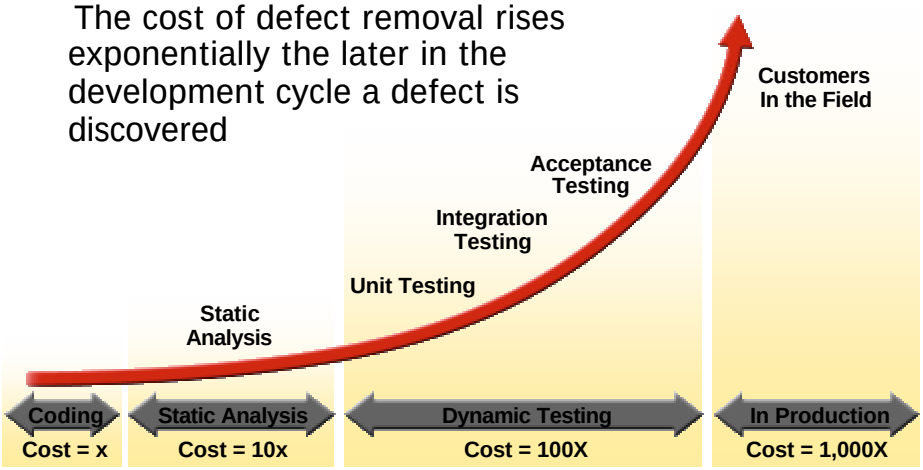


*Source: Capers Jones, ***Software Quality: Analysis and Guidelines for Success***

REASONING

Inspection Reduces Cost Dramatically

The cost of defect removal rises exponentially the later in the development cycle a defect is discovered



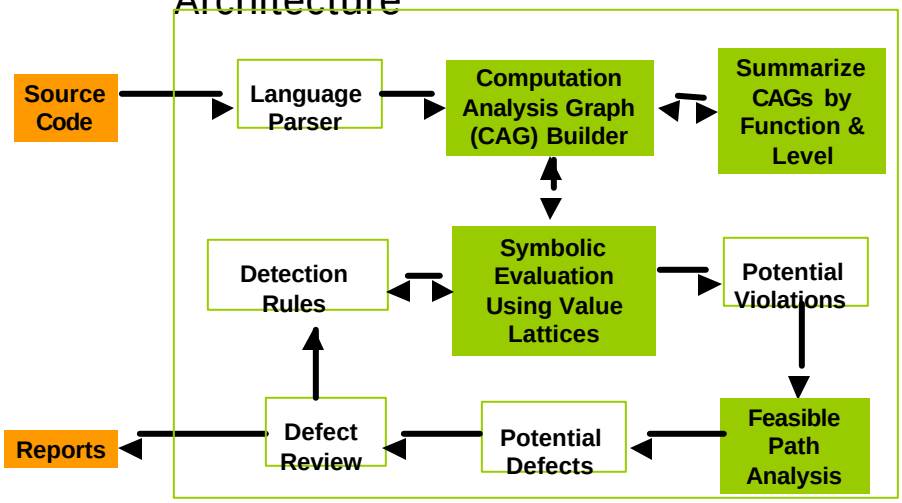
REASONING

Types of Defects

- We look for crash-causing defects, not conformance to requirements or coding Standards
- Java
  - Resource Leaks
    - File and Socket Descriptors
    - Database Connections, Statements and Result Sets
  - Null Pointer Dereference
  - Out of Bounds Array Access
  - String Comparison
- C/C++
  - NULL Pointer Dereference
  - Out of Bounds Array Access
  - Memory Leak
  - Uninitialized Variable
  - Bad Deallocation
  - Dangling Pointer

ASI Technology

Architecture



Can You Spot the Defect?

```
static int sock_fasync(int fd, struct file *filp, int on)
  {
  struct fasync_struct *fa, *fna=NULL, **prev;
  struct socket *sock;
  struct sock *sk;

  if (on)
  {
      fna=(struct fasync_struct *)
          kmalloc(sizeof(struct
  fasync_struct),GFP_KERNEL);
      if(fna==NULL) return -ENOMEM;
  }
  sock = socki_lookup(filp->f_dentry->d_inode);
  if ((sk=sock->sk) == NULL)
      return -EINVAL;
```

If this is true

.. this memory leaks

.. and this true

Actual Defect Page (MySQL)

DEFECT CLASS: Memory Leak **DEFECT ID** 2

LOCATION: mysql-4.0.16\readline\history.c : 287

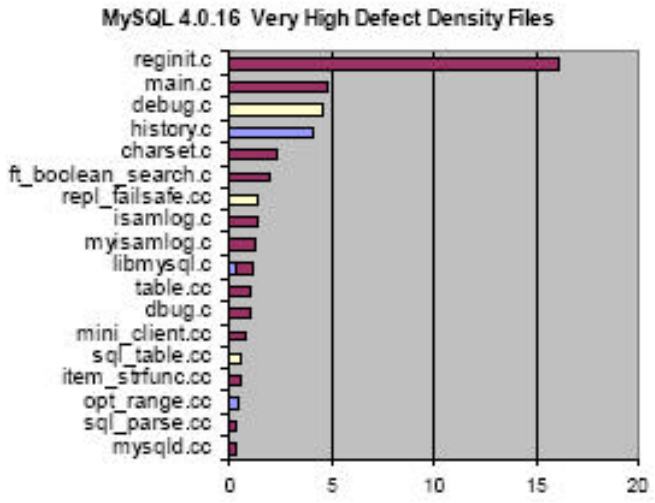
DESCRIPTION Local variable **temp**, declared on line **283**, is assigned a pointer to a block of memory allocated by **xmalloc()** on line **283**. No other pointer refers to this memory block, so it is inaccessible (still allocated, but unreachable) once **temp** goes out of scope after line **287**.

PRECONDITIONS The conditional expression (**which >= history_length**) on line **286** evaluates to **true**.

CODE FRAGMENT

```
277  HIST_ENTRY *
278  replace_history_entry (which, line, data)
279       int which;
280       char *line;
281       histdata_t data;
282  {
283    HIST_ENTRY *temp = (HIST_ENTRY *)xmalloc (sizeof (HIST_ENTRY));
284    HIST_ENTRY *old_value;
285
286    if (which >= history_length)
287      return ((HIST_ENTRY *)NULL);
288
289    old_value = the_history[which];
290
291    temp->line = savestring (line);
292    temp->data = data;
293    the_history[which] = temp;
294
295    return (old_value);
296  }
```

Metrics Report Identifies Clusters



Feedback Linux TCP/IP Developer

- On Out of Bounds Array Access:
  Nope, not wrong, the table is indexed off-by-one. If you were right, rtnetlink would simply not work.
- On Null Pointer Dereference:
  In the cases where SKB is NULL, opt is never NULL, check the two callers.

Feedback Commercial Developer

> For the Linux community to just shrug these off with "well the kernel works so it must be ok" doesn't really cut it. I think the NULL dereference checks should be added, and definitely the out of bounds array checking [..] For example, the out of bounds array reference could start causing a problem by just rearranging the order variables are declared.

What happened to the Code?

- The one memory leak was fixed between 2.4.19 (the subject of the study) and 2.4.20, before publication of the study.
- Two out of three out-of-bounds array accesses were fixed between 2.4.20 and 2.4.21, after publication of the study.

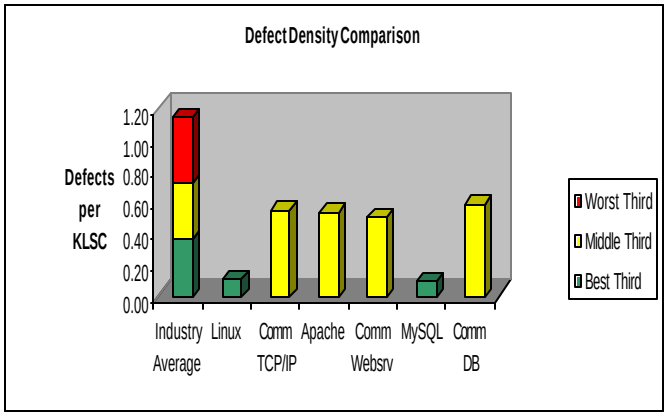
Feedback of the MySQL team

- **Of the 21 defects that Reasoning reported:**
  - **MySQL Immediately resolved 13**
  - **Based on domain knowledge, MySQL determined that 8 would not manifest themselves in the field**
    - **Six Null Pointer Dereferences**
    - **One Uninitialized Variable**
    - **One Memory Leak**
- **One enthusiastic team member:**
  - **"…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."**

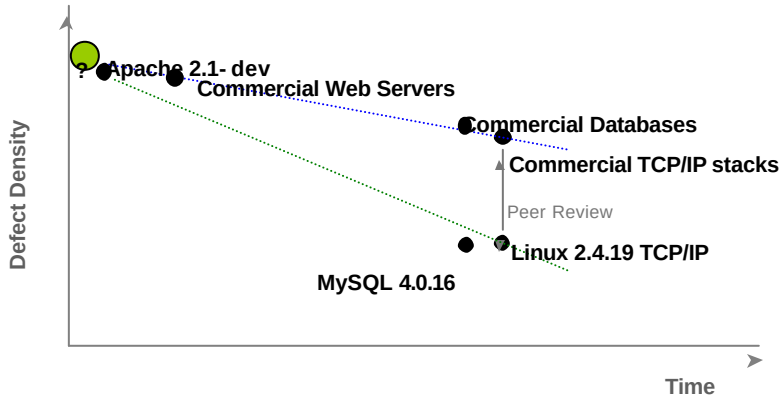
Overview of All Inspections

	MySQL	Linux	Apache	Tomcat
Memory/Resource Leak	3	1	0	5
Null Pointer Dereference	15	3	29	8
Bad Deallocation	0	0	0	--
Out of Bounds Array Access	0	3	0	1
Uninitialized Variable	3	1	2	--
String Comparison	--	--	--	2
Totals	21	8	31	17
Defects/KLOC	0.09	0.10	0.53	0.24

Metrics Comparison – C/C++



Preliminary Hypothesis



- More Research Needed
  - Initial defect densities of Open Source vs Commercial ?
  - Defect removal rates of Open Source projects

Conclusions

- Code inspections find critical defects even in extremely well tested software
  - Commercial
  - Open Source
- Open Source is not inherently worse
  - Projects that have a large developer base seem to improve faster than similar commercial projects
- Java is not the silver bullet
- More research is required
  - Number of projects
  - Initial defect density of Open Source projects
  - Defect removal rates
  - Large number of users but not developers

What's next?

- Next Open Source projects?
  - OpenOffice
  - OpenLDAP
  - Complete LAMP with one of the three Ps
    - Perl
    - PHP
    - Python
  - FreeBSD
  - PostGres
- What else would you like to see?
- Register to download one of our white papers and receive updates as we inspect new projects

About Reasoning

- Reasoning provides automated inspection service for software development organizations
  - Uncovers security vulnerabilities and crash-causing defects
  - Reduces development costs
  - Improves time to market
  - Yields better software quality
- Supports Java, C, and C++
- Have inspected over 1 Billion LOC
- www.reasoning.com
- Email: jasper.kamperman@reasoning.com

