



Open-source Versus Commercial Software: A Quantitative Comparison

**Rix Groenboom
Reasoning NL BV**

rix.groenboom@reasoning.com



Agenda

KEEP A CLOSE EYE ON YOUR SOFTWARE 

- ❖ **About Reasoning**
- ❖ **The Study**
- ❖ **Inspection Results**
- ❖ **Analysis**
- ❖ **Conclusions**
- ❖ **New results**
- ❖ **Input for Discussion**



About Reasoning

KEEP A CLOSE EYE ON YOUR SOFTWARE 



- ❖ Reasoning provides an automated inspection service for organizations that develop software
- ❖ Enables building better software in less time and at lower cost
- ❖ Support C, C++, and Java
- ❖ Have inspected over 1B LOC



The Study

KEEP A CLOSE EYE ON YOUR SOFTWARE 

First, some background:

- ❖ **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

KEEP A CLOSE EYE ON YOUR SOFTWARE 

Why do the study ?

- ❖ **Our customers wanted to understand the real differences between Open-Source software and Commercial software**
 - Reasoning is uniquely positioned to provide this information

When was the study performed ?

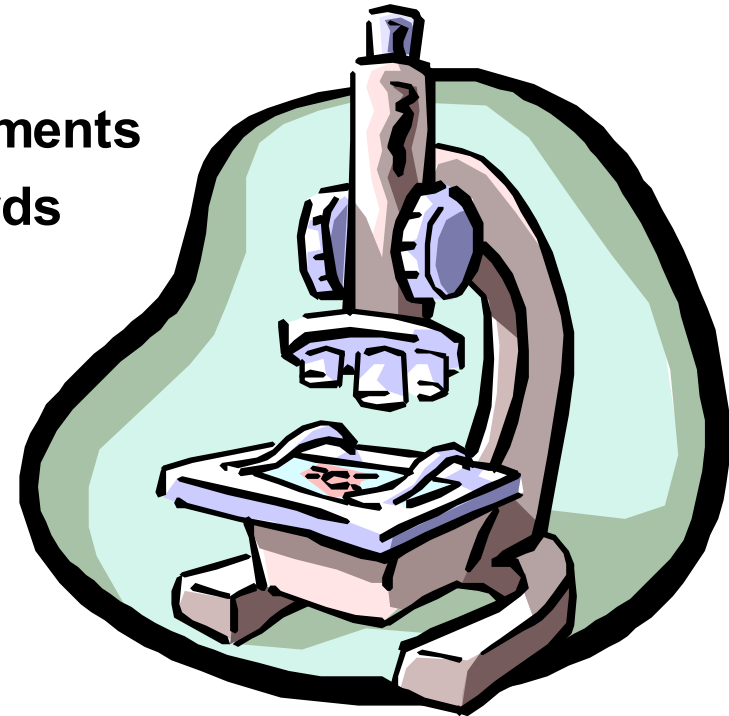
- ❖ **Open-source inspection = December 2002**
- ❖ **Commercial inspections = Throughout 2002**



Used Software Inspection

KEEP A CLOSE EYE ON YOUR SOFTWARE 

- ❖ **AKA Peer Review**
- ❖ **Implicit in Extreme Programming**
- ❖ **Examination of source code to:**
 - Detect defects
 - Trace code to requirements
 - Check coding standards



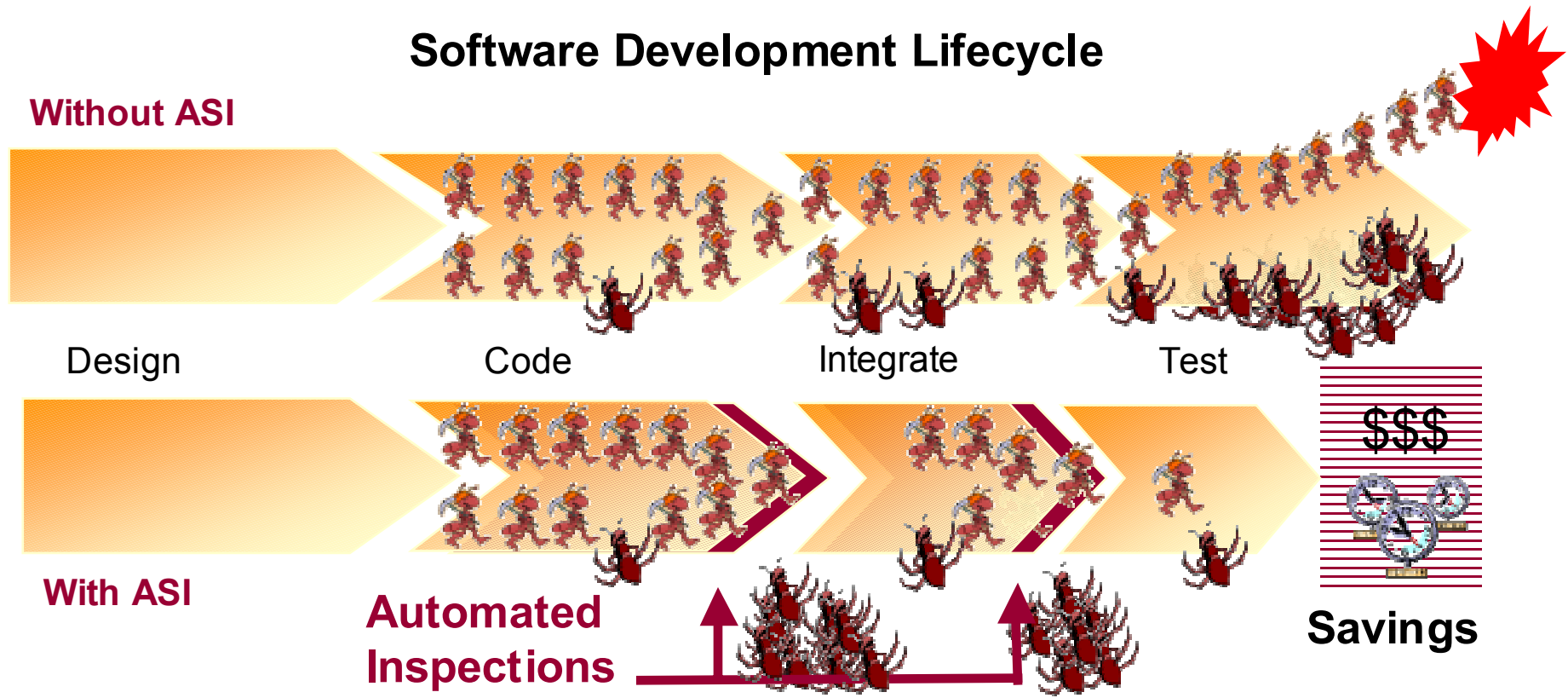


Value of Inspections

KEEP A CLOSE EYE ON YOUR SOFTWARE 

Increased Reliability, Reduced Cost and Time-to-Market

Software Development Lifecycle





Types of Defects Searched for

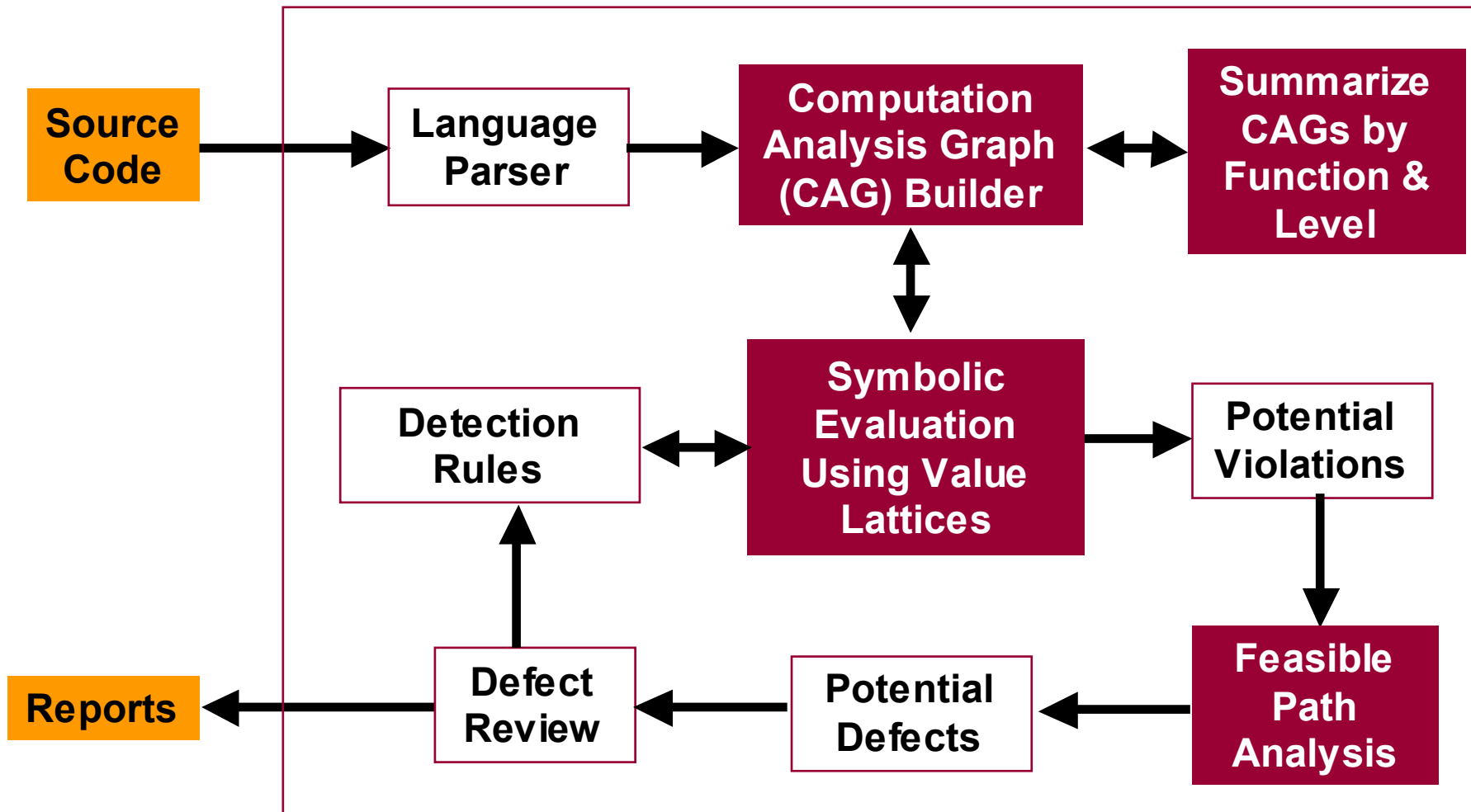
KEEP A CLOSE EYE ON YOUR SOFTWARE 

- ❖ **NULL Pointer Dereference**
- ❖ **Out of Bounds Array Access**
- ❖ **Memory Leak**
- ❖ **Uninitialized Variable**
- ❖ **Bad Deallocation**

Using an Automated Software Inspection (ASI) methodology.



Architecture





Project Scope

KEEP A CLOSE EYE ON YOUR SOFTWARE 

- ❖ **Open Source implementation of TCP/IP in version 2.4.19 of Linux Kernel**





- ❖ **Five commercial implementations of TCP/IP in commercial, general purpose operating systems and telecommunications equipment**

- ❖ **Why TCP/IP ?**
 - **Well-defined set of published requirements**
 - **Implementations have been in existence for several years**
 - **Publicly available conformance tests**



Can You Spot the Defect?

KEEP A CLOSE EYE ON YOUR SOFTWARE 

```
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)  If this is true  
    {  
         .. this memory leaks  
        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)  .. and this true  
        return -EINVAL; 
```



Actual Report

DEFECT CLASS: Memory Leak

KEEP A CLOSE EYE ON YOUR SOFTWARE 

LOCATION: src/linux-2.4.19/net/socket.c : 750

DESCRIPTION Local variable **fna**, declared on line **735**, is assigned a pointer to a block of memory allocated by **kmalloc** on line **741**. No other pointer refers to this memory block, so it is inaccessible (still allocated, but unreachable) once **fna** goes out of scope after line **750**.

PRECONDITIONS The conditional expression **(on)** on line **739** evaluates to **true** AND
The conditional expression **(fna==NULL)** on line **742** evaluates to **false** AND
The conditional expression **((sk=sock->sk) == NULL)** on line **749** evaluates to **true**.

CODE FRAGMENT

```
733 static int sock_fasync(int fd, struct file *filp, int on)
734 {
735     struct fasync_struct *fa, *fna=NULL, **prev;
736     struct socket *sock;
737     struct sock *sk;
738
739     if (on)
740     {
741         fna=(struct fasync_struct *) kmalloc(sizeof(struct fasync_struct
GFP_KERNEL);
742         if (fna==NULL)
743             return -ENOMEM;
744     }
745
746     sock = socki_lookup(filp->f_dentry->d_inode);
747
748     if ((sk=sock->sk) == NULL)
749         return -EINVAL;
750
751     lock_sock(sk);
752
753     prev=&(sock->fasync_list);
754
755     for (fa=*prev; fa!=NULL; prev=&fa->fa_next, fa=*prev)
756         if (fa->fa_file==filp)
757             break;
758
759     if(on)
760
```



Defect Overview

KEEP A CLOSE EYE ON YOUR SOFTWARE 

	Total Defects in 5 Commercial Implementations	Total Defects in Linux Kernel
Memory Leak	43	1
Null Pointer Dereference	128	3
Bad Deallocation	0	0
Out of Bounds Array Access	9	3
Uninitialized Variable	132	1
Totals	312	8



Feedback Linux Developer

KEEP A CLOSE EYE ON YOUR SOFTWARE 

- ❖ **On OOB access:**
 - Nope, not wrong, the table is indexed off-by-one. If you were right, rnetlink would simply not work.
- ❖ **On NPD:**
 - In the cases where SKB is NULL, opt is never NULL, check the two callers.
- ❖ **On ML:**
 - Fixed in the subsequent release.



Feedback Commercial Developer

KEEP A CLOSE EYE ON YOUR SOFTWARE 

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.



Fix Rates

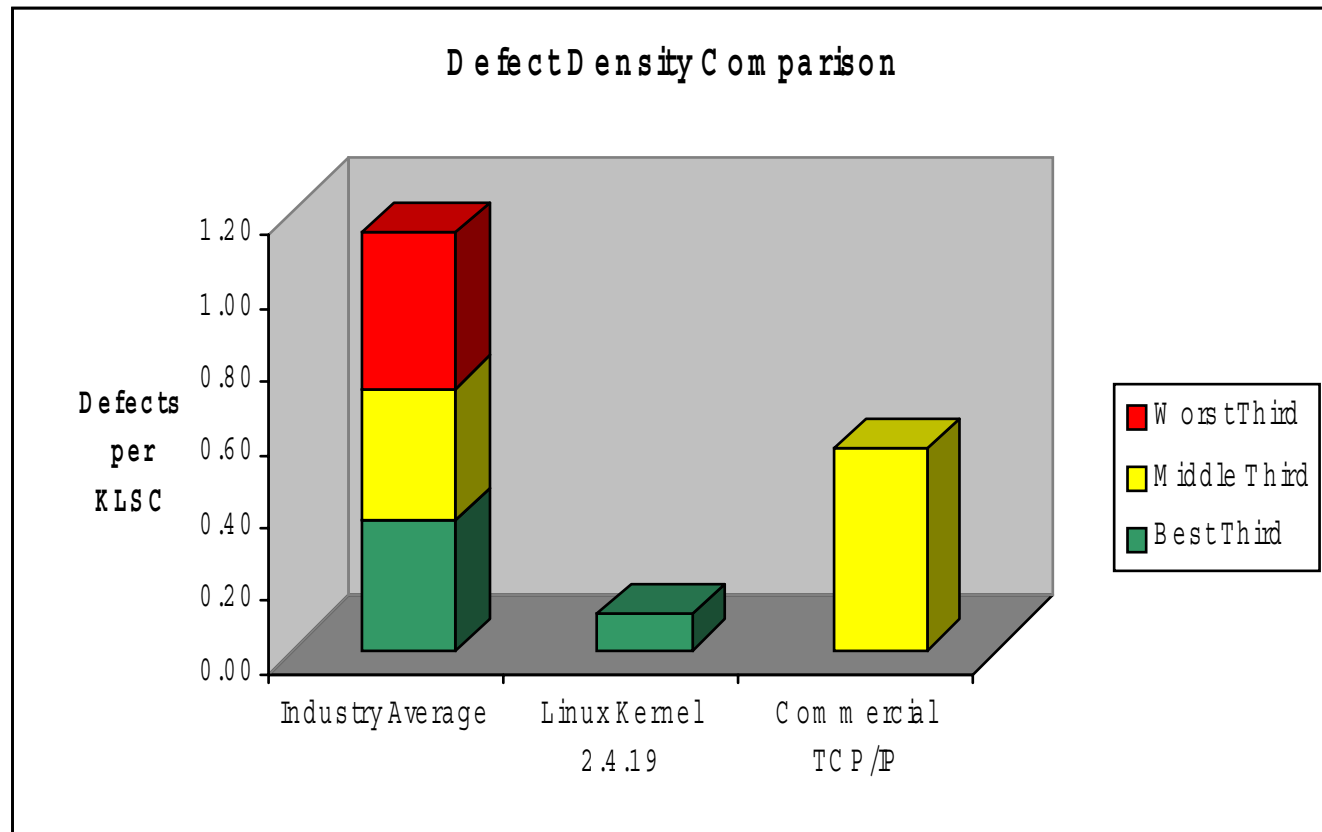
KEEP A CLOSE EYE ON YOUR SOFTWARE 

	Reported	Repaired	%
Commercial Implementations	312	235	75.3
Linux Kernel 2.4.19	8	1	12.5



Metrics Comparison

KEEP A CLOSE EYE ON YOUR SOFTWARE 

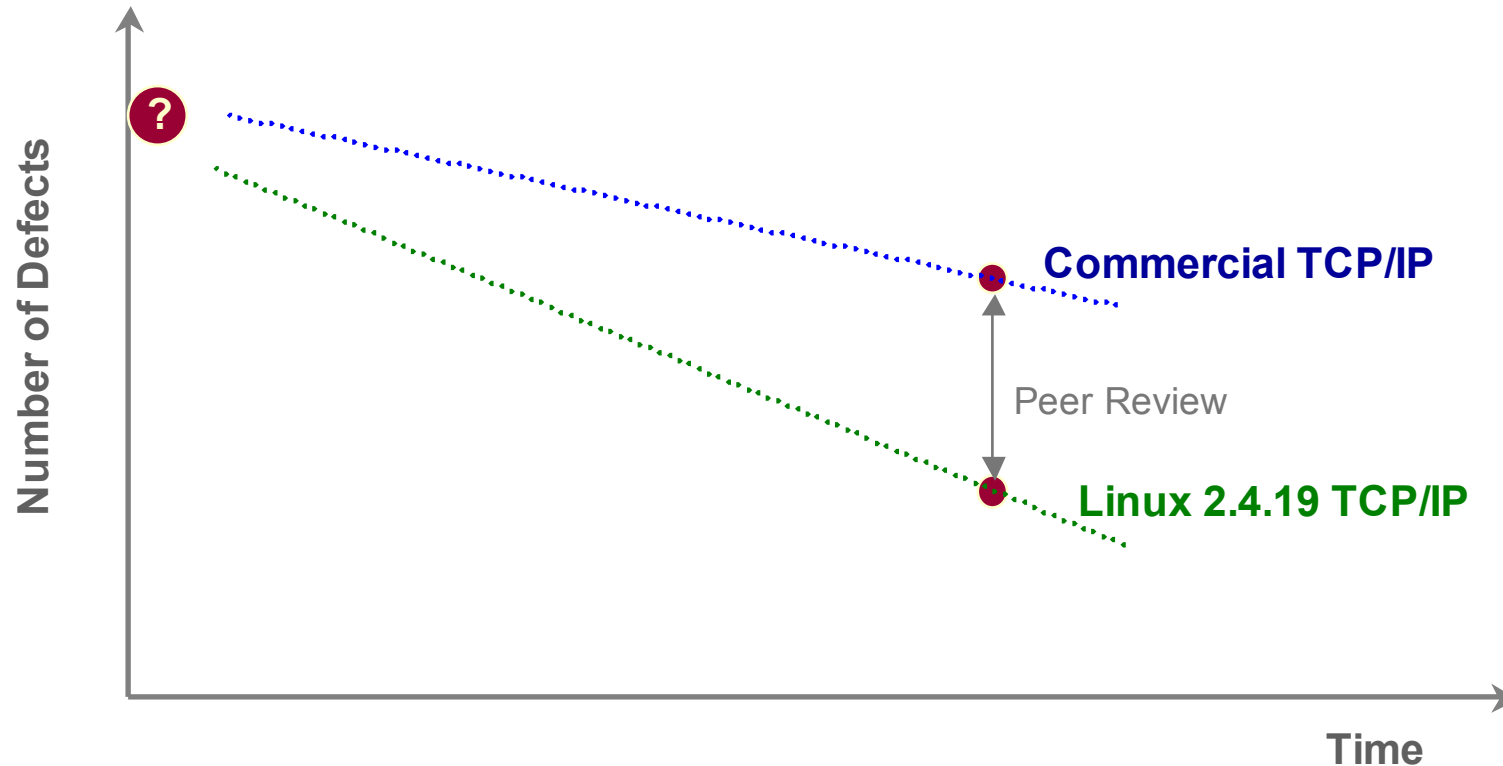


- **Linux stack: 0.10**
- **Commercial stack: 0.55**



Preliminary Hypothesis

KEEP A CLOSE EYE ON YOUR SOFTWARE 



❖ More Research Needed

- Initial defect densities of Open Source vs. Commercial?
- Defect removal rates of Open Source projects



Conclusions

KEEP A CLOSE EYE ON YOUR SOFTWARE 

- ❖ **Open-source is not inherently worse**
- ❖ **More research is required**
- ❖ **Code inspections still find critical defects in extremely well tested software**



New results

KEEP A CLOSE EYE ON YOUR SOFTWARE 

❖ Apache test

- Httpd 2.1-dev (development version 01/31/03)
- Will determine, track, and report on defect density through lifecycle
- Results on www.reasoning.com

❖ Java implementation

- Tomcat Jakarta
- Results on www.reasoning.com (soon)

❖ What else would you like to see?

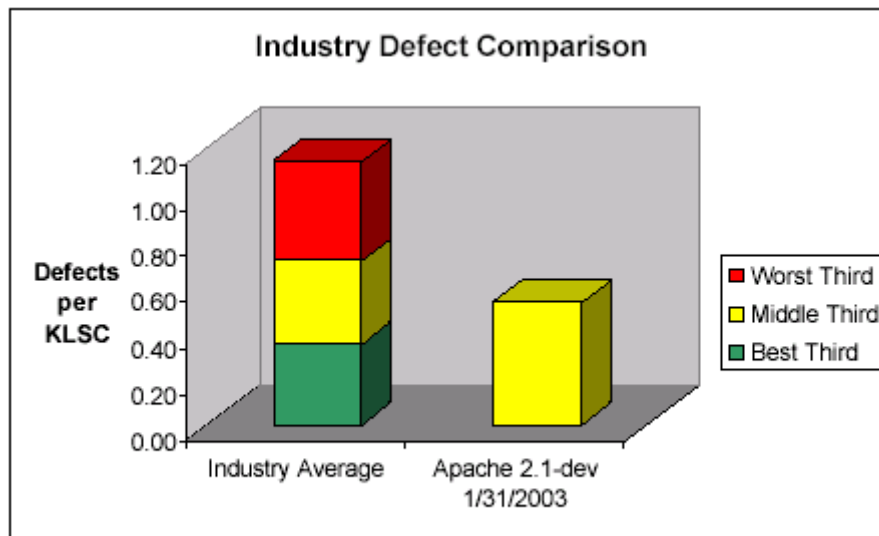


New results

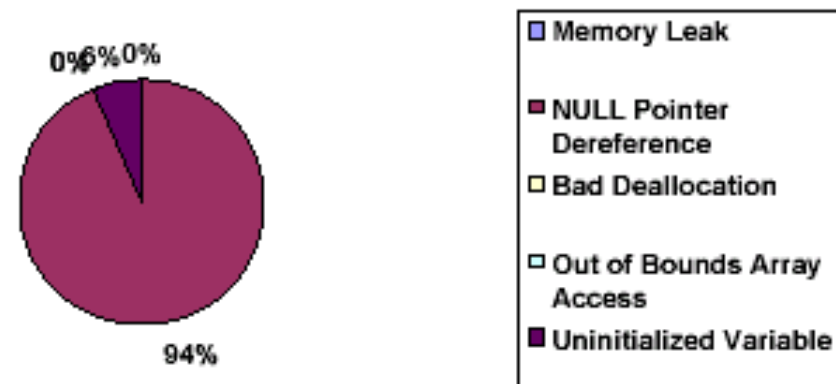
KEEP A CLOSE EYE ON YOUR SOFTWARE

❖ Apache implementation

- Httpd 2.1-dev (development version from 01/31/03)
- 31 Defects in 59 KLOC; density 0.53
- Normal defect density; non typical defect distribution



Defect Distribution





New results

KEEP A CLOSE EYE ON YOUR SOFTWARE 

❖ **Java inspections:**

- **Out of Bounds Array Access**
- **NULL pointer dereferences**
- **String Comparison**

❖ **Jakarta Tomcat 4.1.24:**

- **11 Defects in 71 KLOC; density 0.15 (Java average commercial software is 0.16).**
- **Wait for feedback from the community**
- **Next inspection will include resource leaks**



Towards discussion

KEEP A CLOSE EYE ON YOUR SOFTWARE 

- ❖ **OK, nice “hobby horse” for Reasoning and others for research on test efficiency:**
 - See SPIDER News for a Stanford study on Linux
 - Les Hatton has compared Linux and CMM level
- ❖ **Ownership:**
 - Who finds the bugs ?
 - Who fixes the bugs ?
- ❖ **“Open” source:**
 - Will commercial end-user will actually make changes to the source ?
 - Who will actually review the source (e.g. for quality of the algorithms / security) ?



Discussion !!!

KEEP A CLOSE EYE ON YOUR SOFTWARE 

