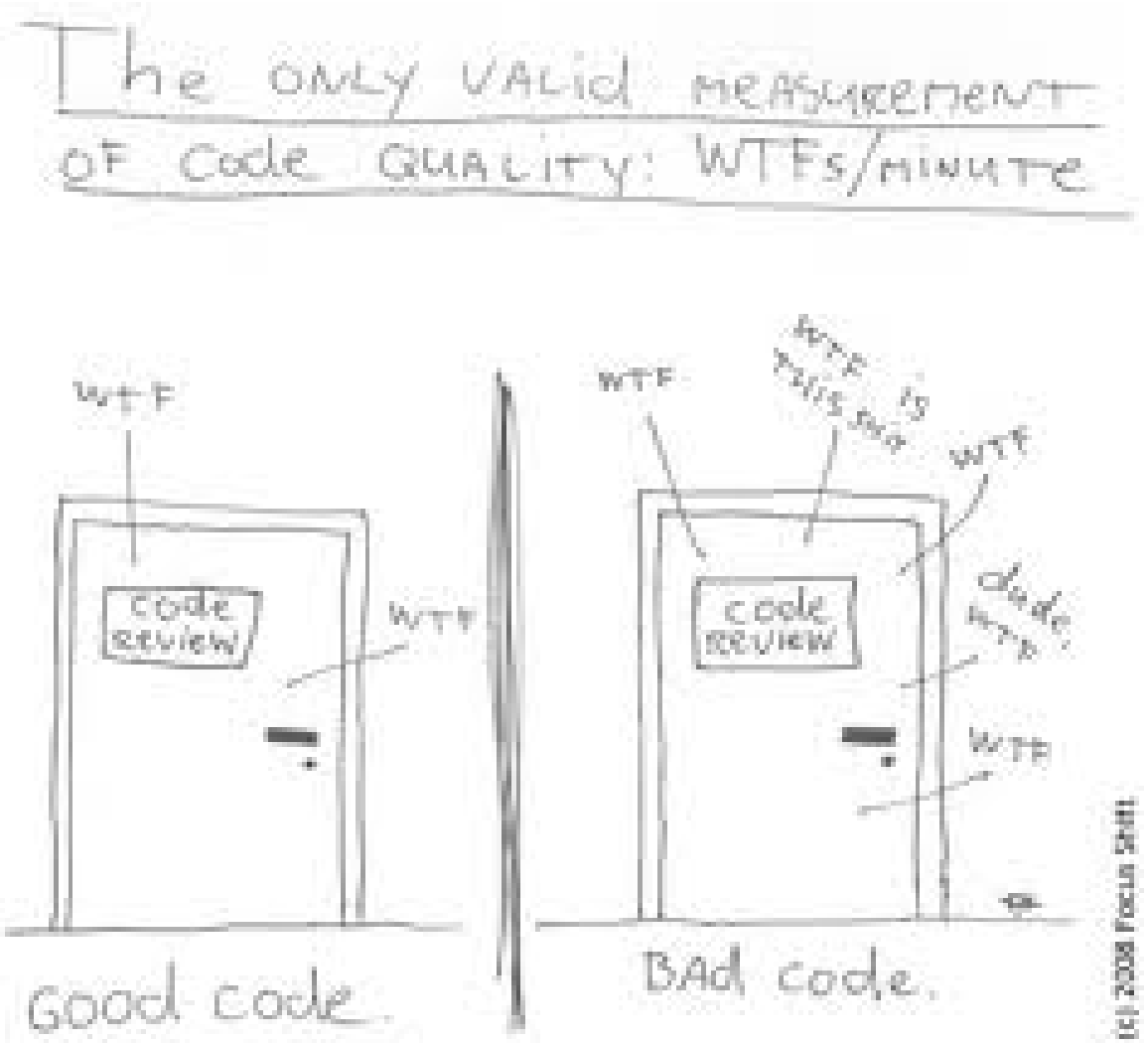
Broadcast Based Peer Review on Open Source Software Projects

Peter C. Rigby and Margaret-Anne Storey
University of Victoria
ICSE 2011

WTFs/minute


The ONLY VALID MEASUREMENT
OF Code QUALITY: WTFs/minute
WTF
code review
WTF
Good code.
WTF
WTF
code review
WTF
WTF
BAd code.

Why do peer review?

“The human eye has an almost infinite capacity for not seeing what it does not want to see [...] Programmers, if left to their own devices will ignore the most glaring errors in the output -- errors that anyone else can see in an instant.”

Weinberg

Background: OSS Peer Review

```
diff --git a/arch/mips/kernel/s
index 383aeb9..32a2561 100644
--- a/arch/mips/kernel/smp.c
+++ b/arch/mips/kernel/smp.c
@@ -193,6 +193,22 @@ void __dev
 */
```

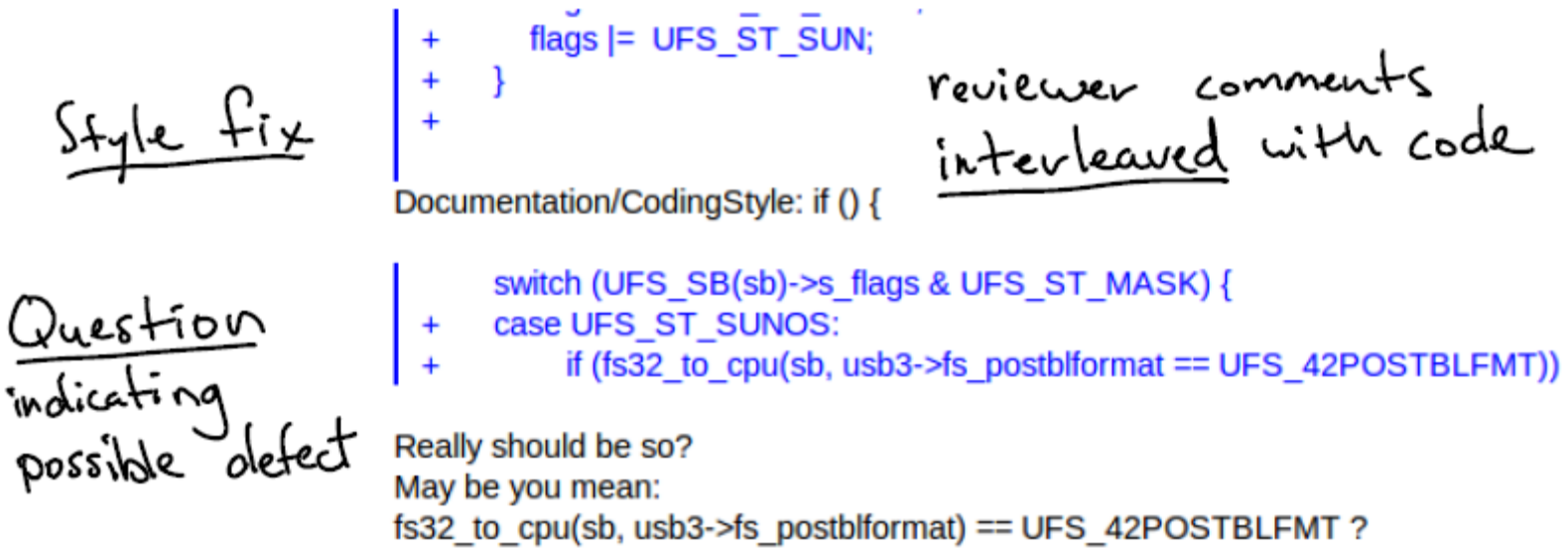
[PATCH 1/1] Clocksource: Move the Hyper-V staging linux | x

K. Y. Srinivasan	Move the Hyper-V clocksource dri
Thomas Gleixner	Please do not use hard coded co
john stultz	The mult/shift assignments can be dropp
Christoph Hellwig	> +#include <linux/version.h> Th
KY Srinivasan	> -----Original Message----- > From: j
KY Srinivasan	> devel@linuxdriverproject.org; johns
KY Srinivasan	> devel@linuxdriverproject.org; tglx@
Thomas Gleixner	to KY, gregkh, linux-l show details

What are the underlying behaviours and mechanisms that facilitate broadcast based review?

Method and Data

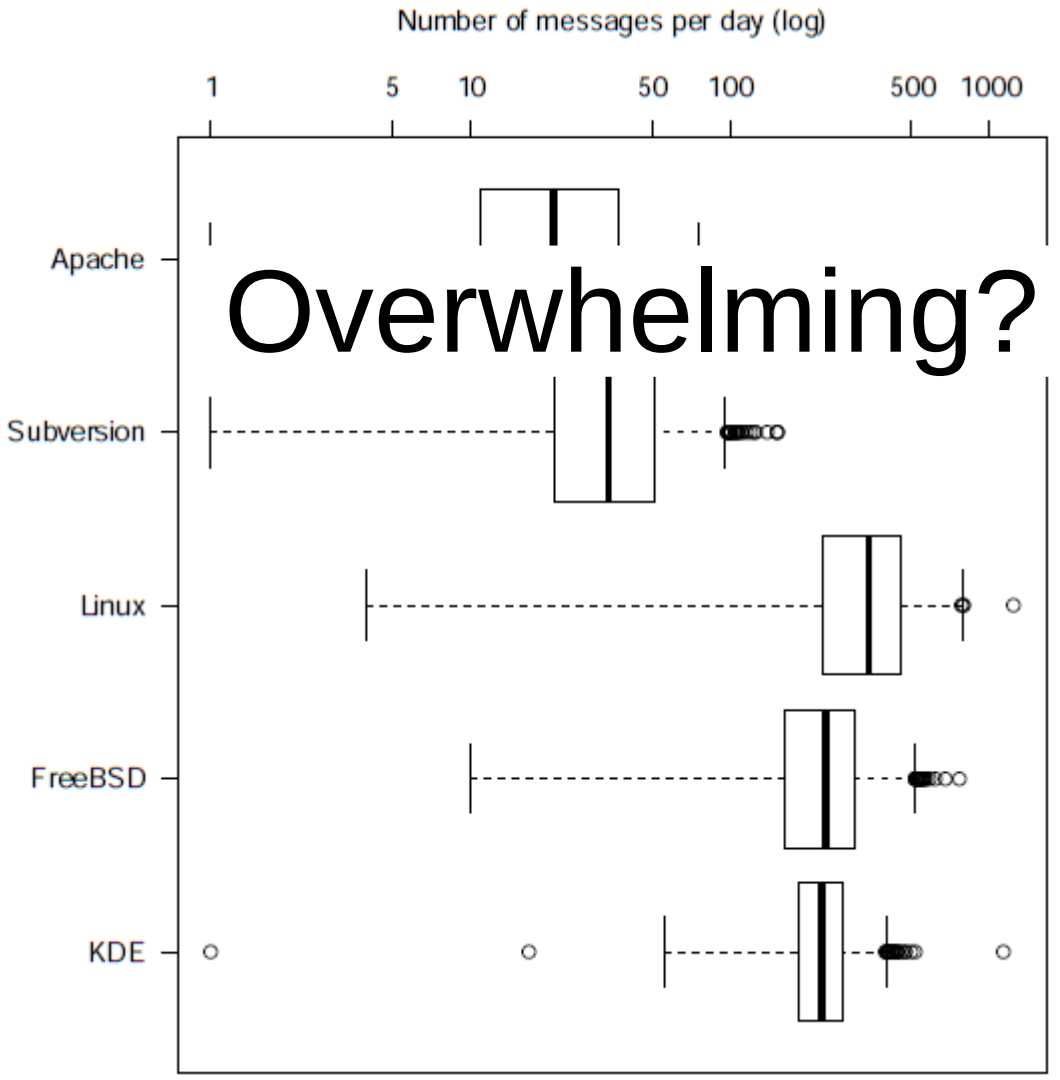
- Grounded Theory
  - Coded 460 review threads
  - Interviewed 9 core developers (top reviewers)



We examine

- Scalability issues involved in broadcasting reviews in large projects,
- The techniques used to find patches for review,
  - Patch structure and norms,
- Review outcomes, stakeholders, interactions,
- The effect of too many opinions during a review, and
- The impact of ignored reviews

Mailing list traffic



Overwhelming?

A Day in Life for Greg KH

- **Total messages: 2067**
- 844 messages from mailing lists he skims
  - e.g., 595 lkml, 127 git
- 237 messages from 18 mailing lists that he read everything for
  - e.g., 51 meego-dev, 42 opensuse mailing lists
- 35 messages from kernel review requests
- 90 messages that are responses to reviews
- **2 personal emails**

Finding and Refinding

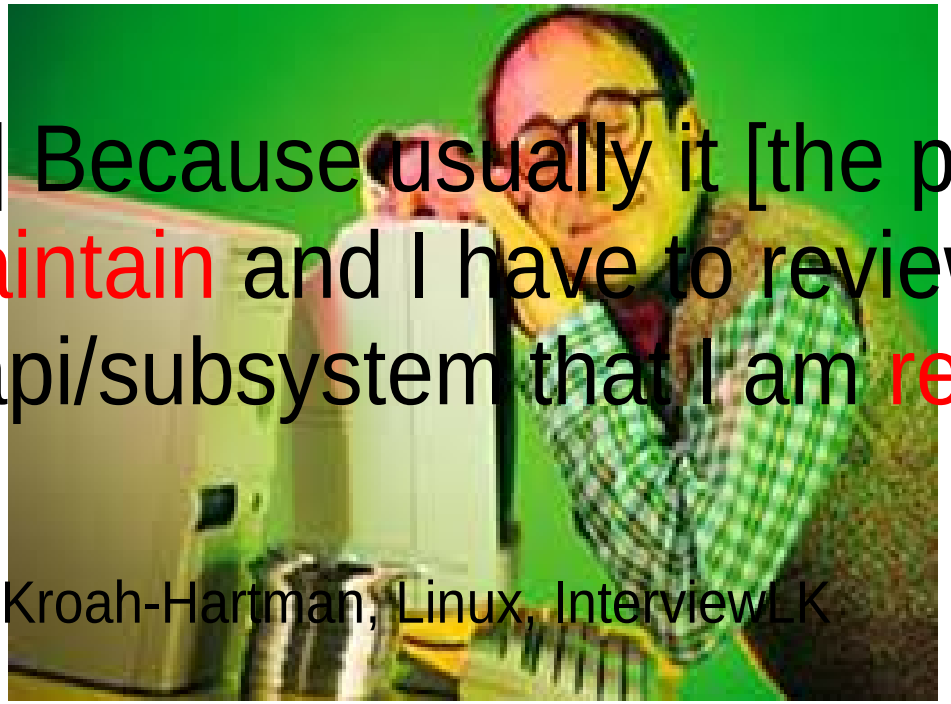
How do experts decide which contributions to review in a broadcast of general development discussion?

Reviewer Motivation

“Mostly [I review because of] interest or experience in the subject area [of the patch].”

Anonymous, FreeBSD, InterviewF2

“[I review] Because usually it [the patch] is for code I maintain and I have to review it. Or it uses an api/subsystem that I am responsible for.”



Kroah-Hartman, Linux, InterviewLK

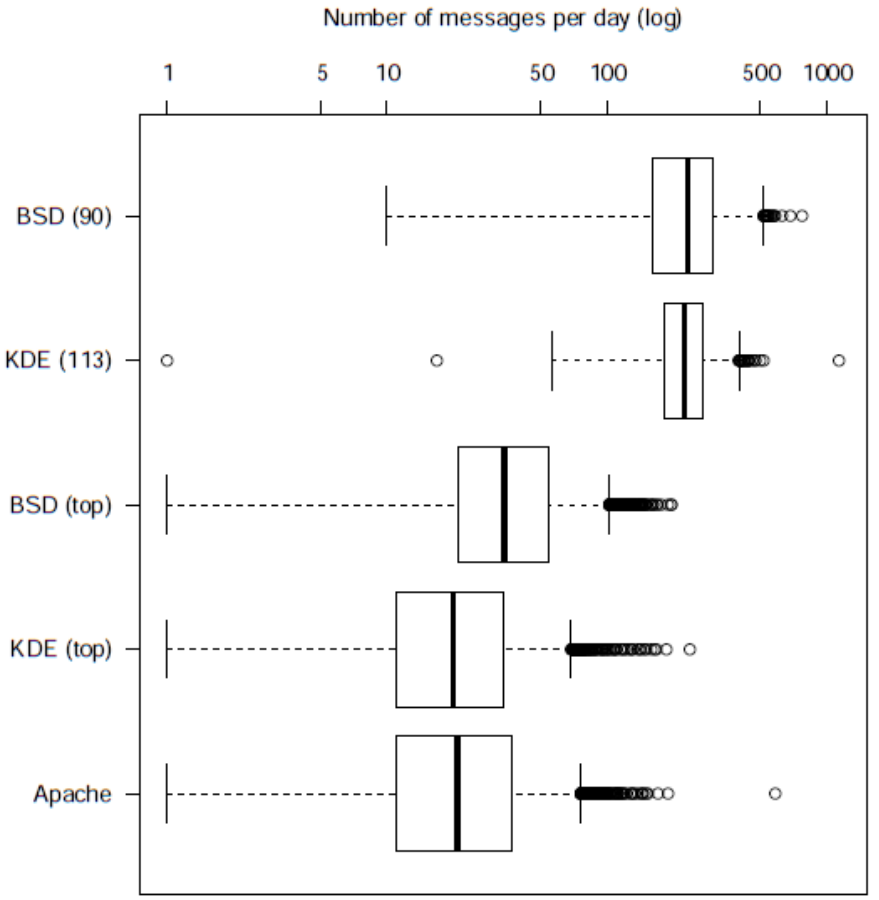
Scanning and Filtering

“Because of those filters, no, I do not feel overwhelmed [by the Linux Kernel Mailing List (LKML)]. On subsystem-specific mailing lists (like the linux-usb or linux-pci mailing lists), I read all emails.”

Kroah-Hartman, Linux, Interview, LK

Multiple, Topic Specific Lists

- Multiple lists reduce traffic, but isolation?
- Few messages crosscut lists (5%)
- Many individuals post to multiple lists (25%)
- Subscribed to 5 and 30 lists
- Developers Bridge Lists



We examine

- Scalability issues involved in broadcasting reviews in large projects,
- The techniques used to find patches for review,
  - Patch structure and norms,
- Review outcomes, stakeholders, interactions,
- The effect of too many opinions during a review, and
- The impact of ignored reviews

Structure: Progressive detail

```
    oom: oom_kill_process: fix the child_points logic

    oom_kill_process() starts with victim_points == 0.  This means that
    (most likely) any child has more points and can be killed erroneously.

    Also, "children has a different mm" doesn't match the reality, we should
    check child->mm != t->mm.  This check is not exactly correct if t->mm ==
    NULL but this doesn't really matter, oom_kill_task() will kill them
    anyway.

    Note: "Kill all processes sharing p->mm" in oom_kill_task() is wrong
    too.

    Signed-off-by: Oleg Nesterov <oleg@redhat.com>
    Signed-off-by: Linus Torvalds <torvalds@linux-foundation.org>

diff --git a/mm/oom_kill.c b/mm/oom_kill.c
index 7dcca55..b19c78e 100644
--- a/mm/oom_kill.c
+++ b/mm/oom_kill.c
```

"A good and descriptive subject line draws immediate attention."

Structure: Progressive detail

```
    oom: oom_kill_process: fix the child_points logic

    oom_kill_process() starts with victim_points == 0.  This means that
    (most likely) any child has more points and can be killed erroneously.

    Also, "children has a different mm" doesn't match the reality, we should
    check child->mm != t->mm.  This check is not exactly correct if t->mm ==
    NULL but this doesn't really matter, oom_kill_task() will kill them
    anyway.

    Note: "Kill all processes sharing p->mm" in oom_kill_task() is wrong
    too.

    Signed-off-by: Oleg Nesterov <oleg@redhat.com>
    Signed-off-by: Linus Torvalds <torvalds@linux-foundation.org>

diff --git a/mm/oom_kill.c b/mm/oom_kill.c
index 7dcca55..b19c78e 100644
--- a/mm/oom_kill.c
+++ b/mm/oom_kill.c
```

Change log gives conceptual understanding

Structure: Progressive detail

```
    oom: oom_kill_process: fix the child_points logic

    oom_kill_process() starts with victim_points == 0.  This means that
    (most likely) any child has more points and can be killed erroneously.

    Also, "children has a different mm" doesn't match the reality, we should
    check child->mm != t->mm.  This check is not exactly correct if t->mm ==
    NULL but this doesn't really matter, oom_kill_task() will kill them
    anyway.

    Note: "Kill all processes sharing p->mm" in oom_kill_task() is wrong
    too.

    Signed-off-by: Oleg Nesterov <oleg@redhat.com>
    Signed-off-by: Linus Torvalds <torvalds@linux-foundation.org>

diff --git a/mm/oom_kill.c b/mm/oom_kill.c
index 7dcca55..b19c78e 100644
--- a/mm/oom_kill.c
+++ b/mm/oom_kill.c
```

The diff shows the file changes

Interleaved, Relevant History

Original diff

```
--- a/kernel/kexec.c
+++ b/kernel/kexec.c
@@ -1433,6 +1433,7 @@ module_init(crash_save_vmcoreinfo_init)
 int kernel_kexec(void)
 {
        int error = 0;
+       int locked;

        if (xchg(&kexec_lock, 1))
                return -EBUSY;
@@ -1498,7 +1499,8 @@ int kernel_kexec(void)
 #endif

  Unlock:
-       xchg(&kexec_lock, 0);
+       locked = xchg(&kexec_lock, 0);
+       BUG_ON(!locked);

        return error;
 }
```

Snipped content

```
On Wed, 13 Aug 2008, Huang Ying wrote:
>
> -     xchg(&kexec_lock, 0);
> +     locked = xchg(&kexec_lock, 0);
> +     BUG_ON(!locked);

Why do you want to do this at all?

And why do you implement your locks with xchg() in the
first place? That's total and utter crap.

Hint: we have _real_ locking primitives in the kernel.

                        Linus
```

Interleaved, Relevant History

Snipped content

```
On Wed, 13 Aug 2008, Huang Ying wrote:
>
> -     xchg(&kexec_lock, 0);
> +     locked = xchg(&kexec_lock, 0);
> +     BUG_ON(!locked);

Why do you want to do this at all?

And why do you implement your locks with xchg() in the
first place? That's total and utter crap.

Hint: we have  real  locking primitives in the kernel.

                        Linus
```

Author responds

```
Linus Torvalds <torvalds@linux-foundation.org> writes:

> On Wed, 13 Aug 2008, Huang Ying wrote:
>>
>> -     xchg(&kexec_lock, 0);
>> +     locked = xchg(&kexec_lock, 0);
>> +     BUG_ON(!locked);
>
> Why do you want to do this at all?
>
> And why do you implement your locks with xchg() in the first place? That's
> total and utter crap.
>
> Hint: we have _real_ locking primitives in the kernel.

This part certainly.

The way the code should work, and the way it has in the past is:
image = xchg(&kexec_image, NULL)
if (!image)
   return -EINVAL;
```

Refinding: Recipient Building

- Private email
  - Number of recipients doesn't change
- Mailing list
  - Number of recipients increases with the number of people who respond
  - Reviewers respond, refinding would be inefficient
  - Reply indicates interest, automatically added to Cc list, so
  - Message arrives in inbox as well as list

We examine

- Scalability issues involved in broadcasting reviews in large projects,
- The techniques used to find patches for review,
  - Patch structure and norms,
- Review outcomes, stakeholders, interactions,
- The effect of too many opinions during a review, and
- The impact of ignored reviews

Review Discussions

- Purely technical discussion
  - “Does the patch do what it claims to do and are there any flaws in the implementation?”, SW
- Scope, politics, and necessity of change arguments
  - Feature freeze
  - Misplaced or overly specialize fix
  - No significant improvement over existing code

Too many trivial opinions

- Parkinson's Law of triviality
- Lack of core group participation
- Exacerbated by large community?
- Impact of outsiders?

Outsider Involvement

	Apache	Subversion	FreeBSD	KDE
Percentage of reviews:				
Outsiders involved	30%	23%	13%	18%
Outsiders majority	5%	3%	2%	4%
"Outsiders' influence" measure correlated with:				
Time for review	−.17	−.24	−.32	−.20
Total messages	−.53	−.59	−.60	−.38
Core-dev messages	−.76	−.77	−.77	−.63

Too Few Reviewers

"Lack of time can postpone the actual reviewing, but not really have an effect on whether to review or the quality of it."

Faure, KDE, Interview, KF

We examined

- Scalability issues involved in broadcasting reviews in large projects,
- The techniques used to find patches for review,
  - Patch structure and norms,
- Review outcomes, stakeholders, interactions,
- The effect of too many opinions during a review, and
- The impact of ignored reviews

Takeaways

pcr@uvic.ca

- Invested co-developers review because of interest and obligation
  - Simple tools, techniques, and social norms to find reviews
- Postpone review rather than rush
  - Ignore incompetent, biased authors
- Asynchronous reviews facilitate group discussion and roles that accord with varied skills
- Politicized pointless discussion happen infrequently
- Scaling through multiple lists and explicit requests

Threats to Credibility

- Fit: send results to interviewees
  - 5 interviewees responded, all positive
- Triangulation: multiple methods and data
  - Archival and interview data
  - Grounded theory and quantification of parameters
- Generalizability
  - Successful, mature OSS projects
  - 4 of 5 infrastructure
- Publicly available data

Theory: Parameters of Review

1. Early, frequent reviews
2. of small, independent, complete contributions
3. that, despite being asynchronously broadcast to a large group of stakeholders, are reviewed by a small group of self-selected experts,
4. resulting in an efficient and effective peer review technique.