INTEL
OpenSource
TECHNOLOGY CENTER

Patch Review

Daniel Vetter, v1.3

Goals
Technical Review
Process
Q&A

Goal: Reviewed-by tag

- **Formal statement: “Reviewer’s statement of oversight”**
  - See Documentation/SubmittingPatches
  - Shouldn’t be dropped onto patches lightly
  - Forging Reviewed-by: tags is not ok
  - Includes legal disclaimer
- **Perfect review is impossible**
  - Every reviewer has strengths and weakness
  - Often details are personal choice, but hard to assess that
  - Hence the name of an r-b tag is the important part

Reviewer's statement of oversight

By offering my Reviewed-by: tag, I state that:

a) I have carried out a technical review of this patch to evaluate its appropriateness and readiness for inclusion into the mainline kernel.

b) Any problems, concerns, or questions relating to the patch have been communicated back to the submitter. I am satisfied with the submitter's response to my comments.

c) While there may be things that could be improved with this submission, I believe that it is, at this time, (1) a worthwhile modification to the kernel, and (2) free of known issues which would argue against its inclusion.

d) While I have reviewed the patch and believe it to be sound, I do not (unless explicitly stated elsewhere) make any warranties or guarantees that it will achieve its stated purpose or function properly in any given situation.

Goal: Why Review at all?

- **Catch technical issues with the patch**
  - We don't have full regression test coverage and tests can't catch everthing – review is a proven process to augment testing
  - Not just "code correct", but the entire package (design, tests, documentation/comments, …)
  - We have a unified driver (across platforms and products) and so crucially rely on a clean codebase for long-term success and review to get it
- **Knowledge osmosis in the virtual&globally distributed team**
  - Longterm team members mentoring new people
  - Product/Focused teams sharing their special knowledge
  - ...
  - Unified diffs are our whiteboard replacement

Technical Review: Code Correctness

- **Corner cases in C programming**
  - Userspace argument checking and other security critical issues
  - Overflows, integer math, other C programming curveballs
  - Asserts and trying not to blow up too badly if things go wrong
  - Error handling and cleanup paths
- **Behaviour review**
  - We don’t autogenerate headers, so everything needs to be cross-checked with docs
  - Modeset sequences, setup sequences and any other ordering constraints
  - Make sure invariants the hardware requires are asserted – ordering of operations is one of the most bug-prone areas in our code

Technical Review: Design

- **Code Design and interfaces**
  - Check that new interfaces work across all platforms we have – we have a unified driver
  - Review all driver-internal interfaces, e.g. Rusty's C API safety levels: http://sweng.the-davies.net/Home/rustys-api-design-manifesto
  - Goes from +10 "impossible to get wrong" to -10 "impossible to get right", and the kernel has/had lots of examples for -10 level interfaces, including drm/i915
- **Naming and consistency**
  - Follow established infrastructure and patterns
  - Reuse existing infrastructure and functions in the driver/kernel instead of rolling your own
  - Lots of people are not native English speakers on our team, so suggest good names

Technical Review: Design (cont'd)

- **Architecture review**
  - Userspace ABI cast in stone essentially forever, can never break it again (http://blog.ffwll.ch/2013/11/botching-up-ioctls.html)
  - Do the patches achieve their goal without undue complexity and without cutting corners?
  - Does it fit into existing code and not reinvents new wheels?
- **Kernel programming issues**
  - Mostly for code correctness, but the root-cause for issues is usually bad design
  - Locking, execution contexts (atomic context, interrupt context, …), memory allocation points and recursion issues, stack usage, …
- **Optimize code for readability**
  - It'll be written once, but read every time someone needs to debug/change related code

Technical Review: Testing

- **Testing is an entire training itself**
  - Overview: http://blog.ffwll.ch/2013/11/testing-requirements-for-drmi915.html
  - Presentation: https://fosdem.org/2014/schedule/event/gfx_driver_testing/ (slides+video)
  - i-g-t documentation: http://people.freedesktop.org/~danvet/igt/
- **Summary**
  - Aim for orthogonal validation to augment review
  - Almost exclusively black-box testing due to eternal ABI guarantees
  - Catching regressions
  - Achieved with a combination of userspace tests (i-g-t, mesa+piglit) and in-kernel self-checks (lockdep, modeset state checker, …)
  - Focus testing on areas where mistakes are expensive (e.g. userspace ABI) or code is known to be fragile (bug-driven testing)

Technical Review: Boring Details

- **Documentation**
  - Kerneldoc and docbook for new interfaces, e.g. i915_cmd_parser.c
  - Code comments and commit message: Document why, not what/how – the code should explain the what/how itself already
  - Reference all relevant resources in the commit message (Bspec chapter, previous commits that break something/are relevant, relative performance measurements, …).
- **Coding style**
  - scripts/checkpatch.pl
- **Patch metadata**
  - s-o-b line, bugzilla/jira links, regression references, mailing-list references

Process

- **Fast turn-around is key**
  - Otherwise author drops the ball and context-switching causes needless delays
  - 1-2 days for small fixes (especially regressions), < 1 week for bigger patch series
- **Review won't happen magically**
  - Patch author needs to push the patch through the process
  - Tracked on the review board
  - Please follow the escalation BKM to avoid overloading key people
- **Review is open to everyone**
  - Quick comments and insights are always welcome, no need for a review AR
  - People who contributed big features are expected to keep a tap on other patches touching that code – subscribe to mailing lists and read them!

Process (cont'd)

- **Comment on what you've reviewed**
  - It's about communication!
  - E.g. tricky cases that puzzled you, opportunities for cleanup in the future, …
  - Especially important for new reviewers to assess their strengths
  - Even more important if you see gaps where you're unsure about correctness
- **Resending revised patches**
  - In-reply-to to individual patches for small polish to keep the discussion together
  - Full resend for major changes or if the discussion gets too messy
- **Proper patch splitting**
  - Optimize big feature work for the review by properly splitting patches – the order code was written in is rarely the best way to read it
  - Optimize patches for the future engineer who needs to debug/read your code

Summary and Q&A

- **Review is a major tool for knowledge sharing**
  - Actually communicate, don’t just slap r-b tags onto patches
- **Review isn’t just “code correct”**
  - Design, interfaces & naming things
  - Test coverage in i-g-t and self-tests/asserts
  - Documentation, comments and commit message
  - checkpatch.pl