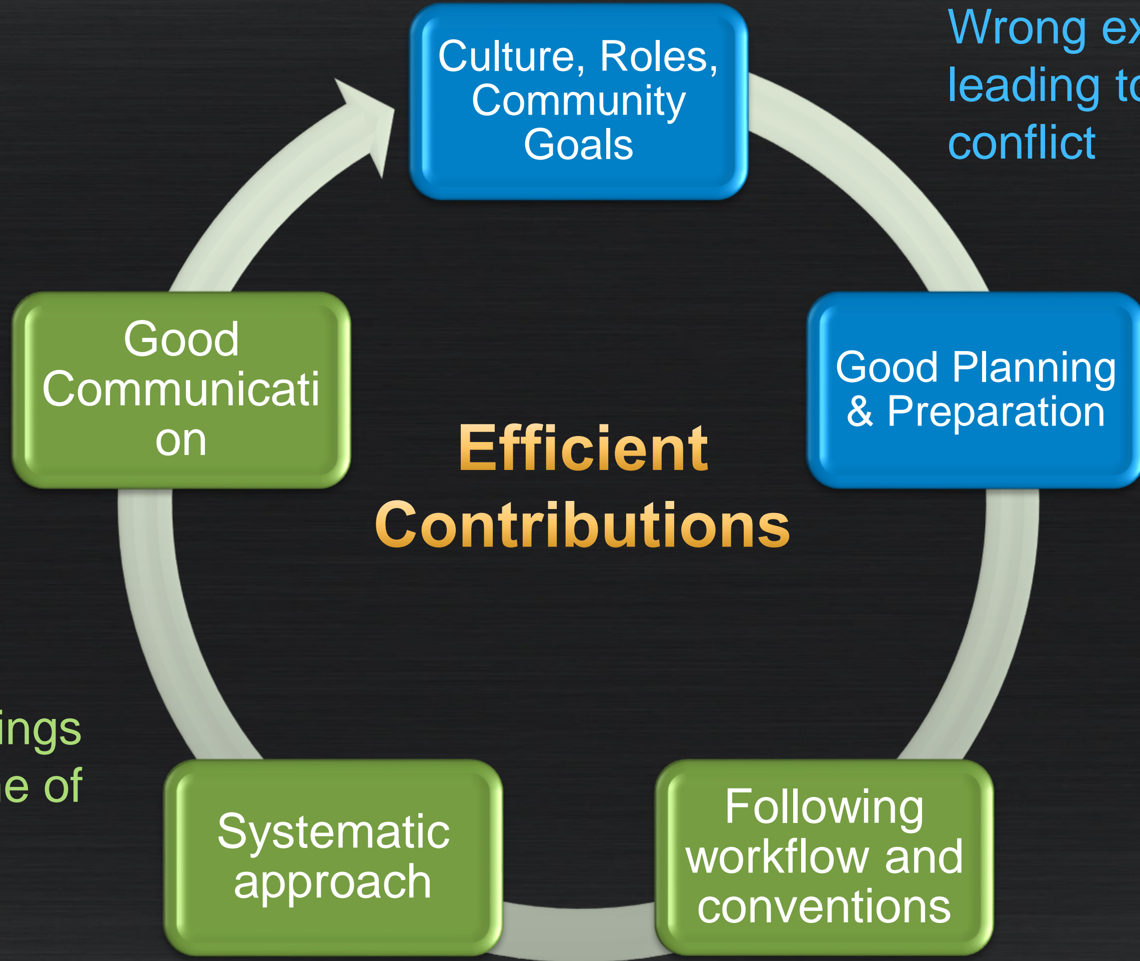
Xen Project Contributor Training
Part 1 : Setting the Scene

Lars Kurth
Community Manager, Xen Project
Chairman, Xen Project Advisory Board
Director, Open Source Business Office, Citrix

lars_kurth

Wrong expectations leading to frustration and conflict



Unnecessary iterations, misunderstandings and wasted time of contributor and reviewers time

Outline for Part 1 – Setting the Scene

Goal: Effective Contributions

Factors that impact Effectiveness

Motivators of Community Stakeholders

Common factors for disagreement when trying to contribute

The case for Code Reviews: Find bugs early & often

Factors impacting Review Duration

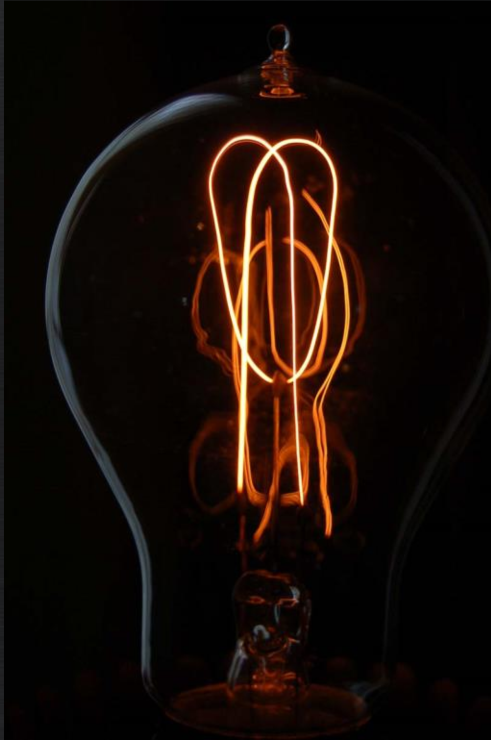
Jumbled Reviews and Phasing

Code Reviews: Theory

Relevant Conventions and Processes in the Xen Project

Systematic approach to acting on Feedback

Communication is key : Avoid Misunderstandings



Vinovyn @ Flickr

Goal:

Enable you to Work Efficiently with the Xen Project Developer Community

Quiz:

Which factors impact the length of time it takes your patch to be up-streamed?

ROLES: Contributor Motivation

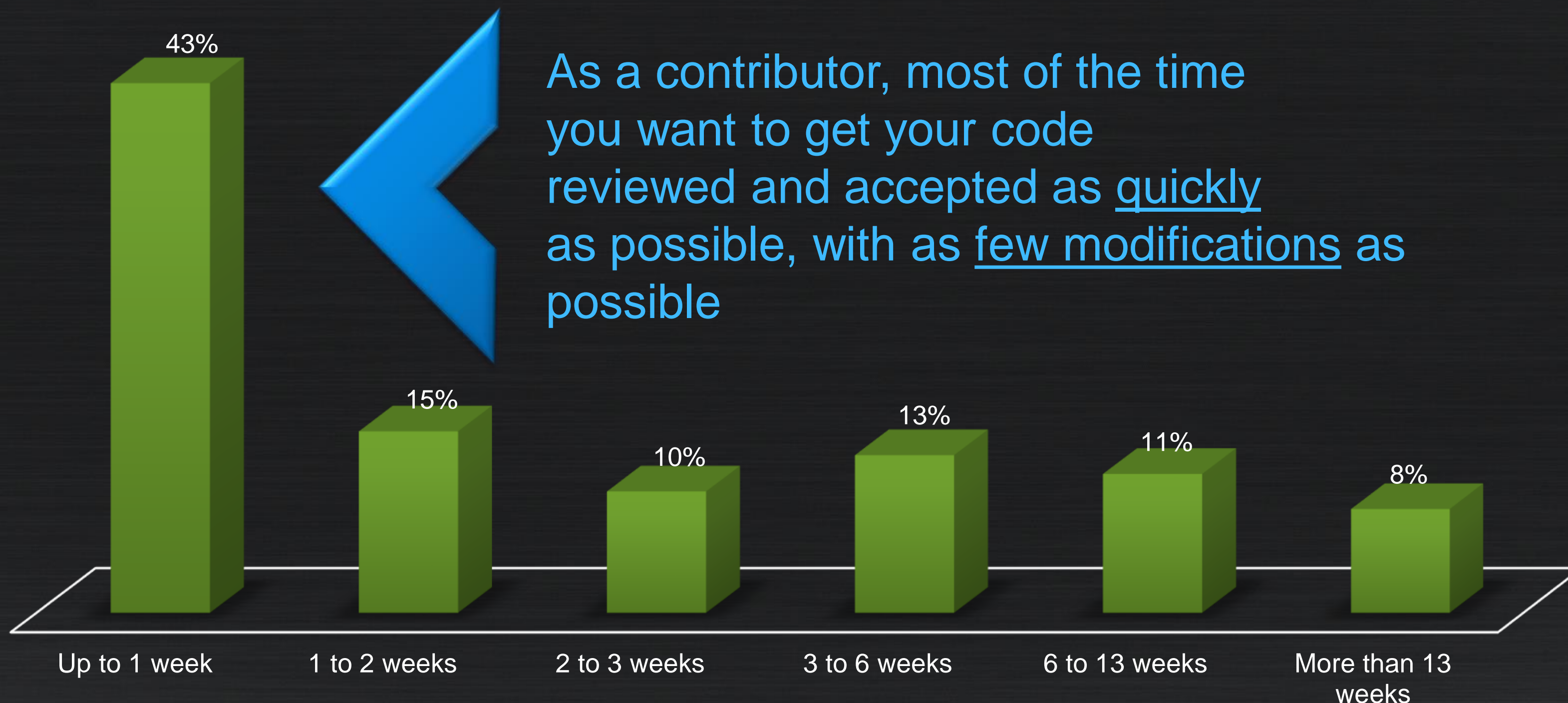
- Get your code into the code line
- Get it in as quickly as possible, with as little re-work as possible
- Or at least, make the process of contributing predictable
- You may be under pressure from a (product) manager

Enabling feature or API which you want to be widely used

Feature or API, that you are using in your product or service. In other words you don't care much if someone else uses it

Research purpose

Hypervisor Code Review times



ROLES: Community “Gatekeepers”

Reviewers, Maintainers, Committers, Project Lead

- System Properties: Code readability, understanding what goes in, maintainability, quality, performance, scalability, …
- Practical Issues:
  - Is this patch one I have to look at?
  - Reviewer / maintainer of the patch series
  - Archaeologist: years down the line – why is the code as it is?
- Workload and personal:
  - Wants to avoid un-necessary workload
  - Day-job: aka other commitments
  - Has a personal communication style
  - Reputation within the community

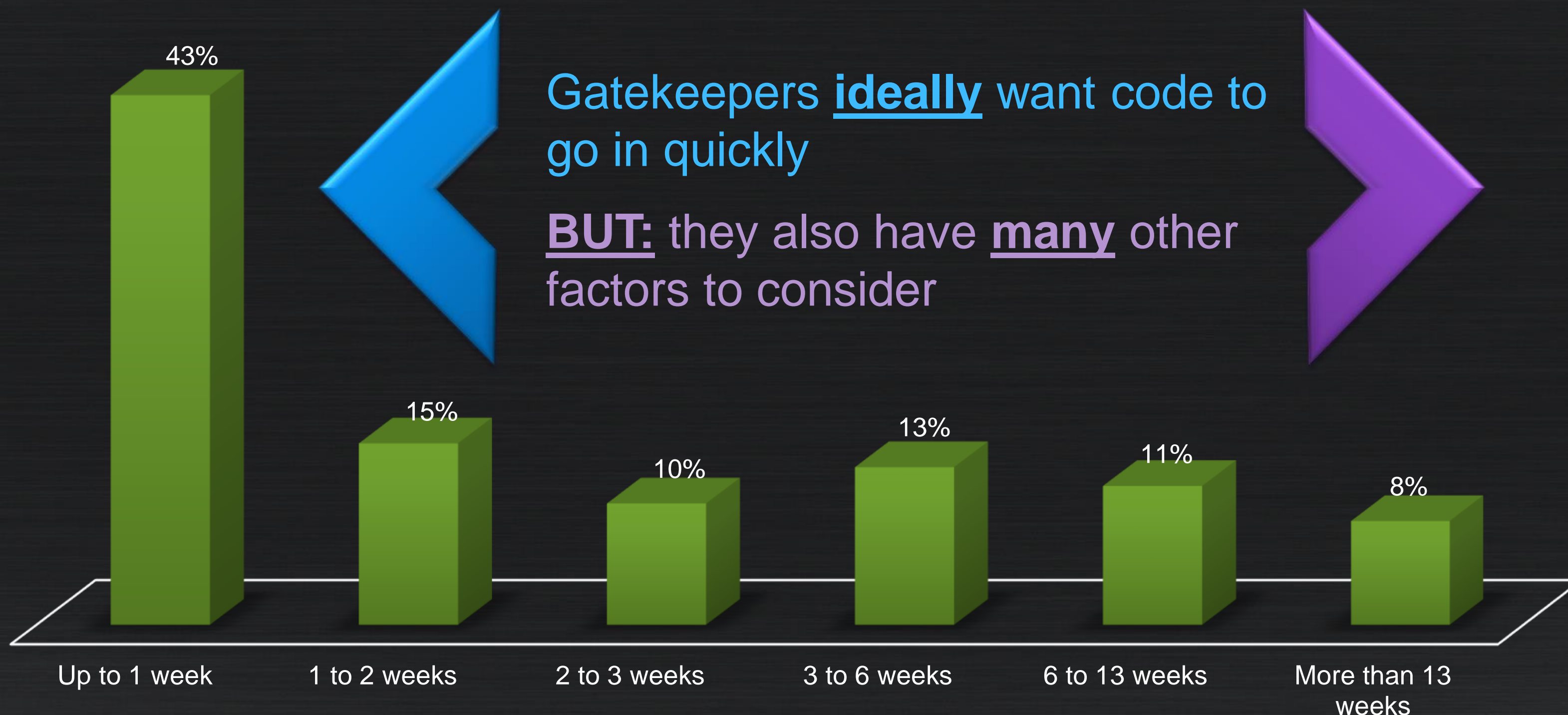
Problem

30% Community Growth p.a.

Contributors competing for review time from stretched maintainer / reviewer base

Average review time up from 28 to 32 days in 6 months

Hypervisor Code Review times



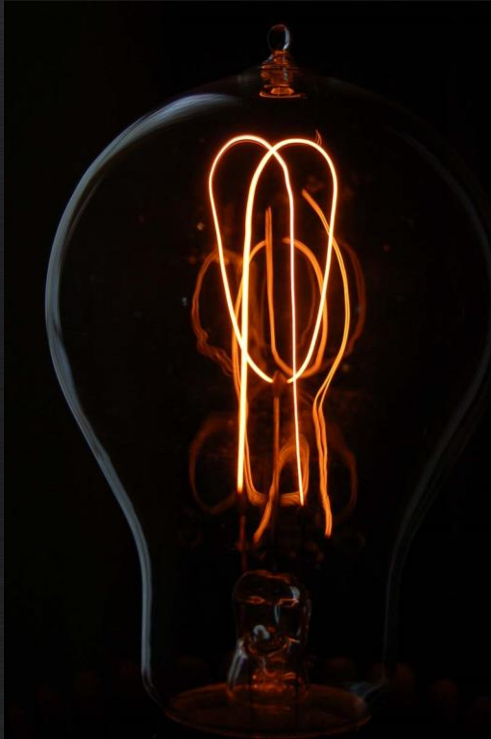
Potential for Disagreement

Contributor **Gatekeeper**

Different mindset
Different expectations
Bad communication
Misunderstandings

Usually common interest
Process, convention & tools

Awareness, Mindset / Empathy, Planning,
Good Communication, Trust / Respect, etc.

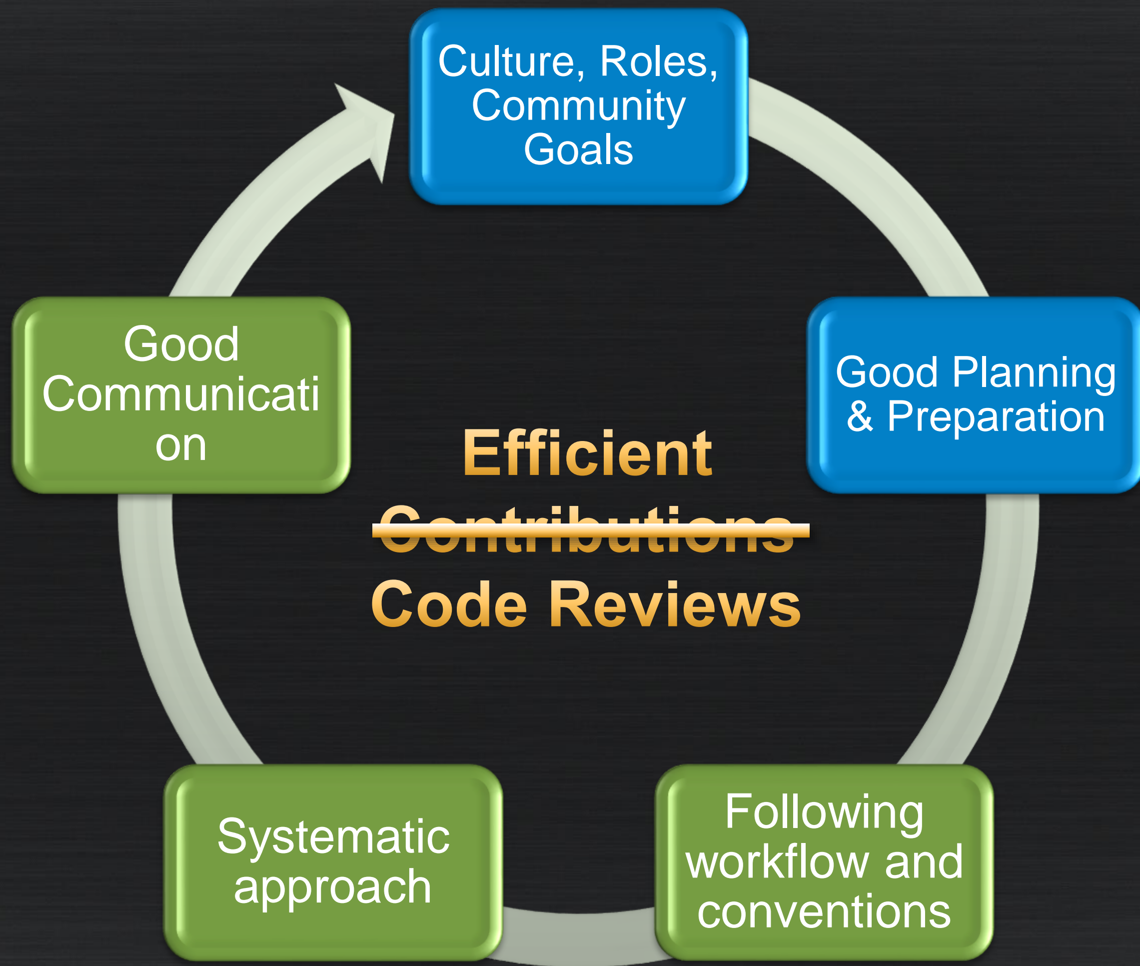


Vinovyn @ Flickr

Observation:

The tension identified is **not** specific to OSS development, but is a property of Code Review

Aka the tension between submitter and gatekeeper (reviewer)


Culture, Roles, Community Goals
Good Planning & Preparation
Following workflow and conventions
Systematic approach
Good Communication
Efficient
Contributions
Code Reviews

Interlude:

The case for Peer Code Review

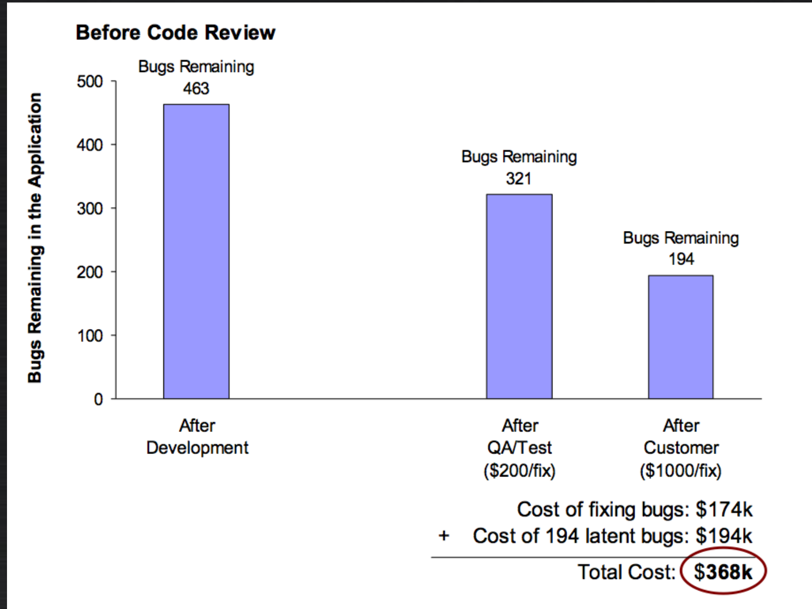
Find Bugs Early and Often

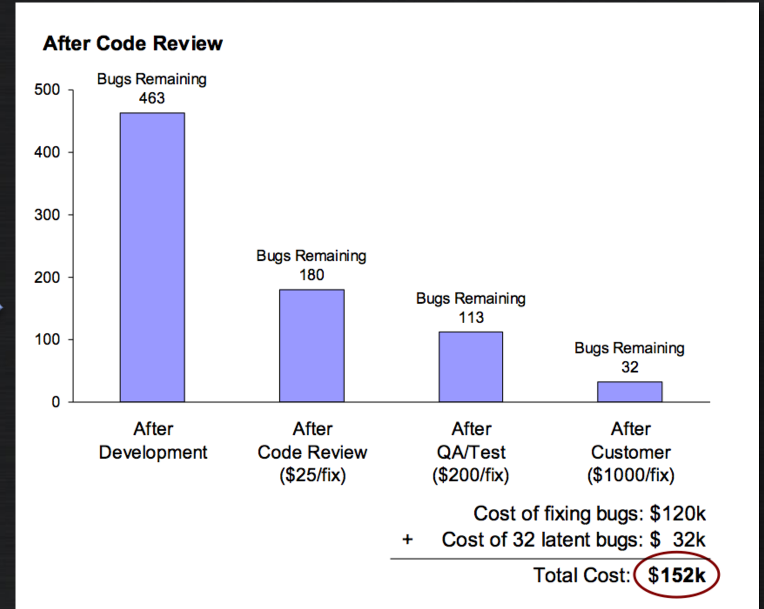
One of our customers set out to test exactly how much money the company would have saved had they used peer review in a certain three-month, 10,000-line project with 10 developers. They tracked how many bugs were found by QA and customers in the subsequent six months. Then they went back and had another group of developers peer-review the code in question.

Using metrics from previous releases of this project they knew the average cost of fixing a defect at each phase of development, so they were able to measure directly how much money they would have saved.

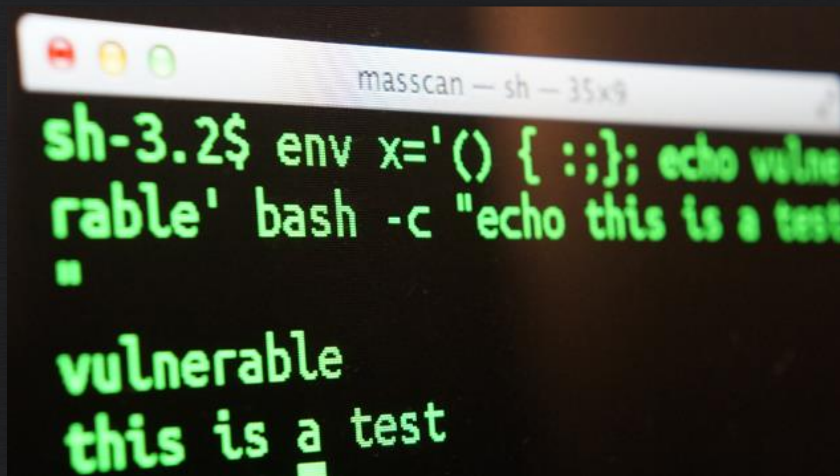
http://smartbear.com/smartbear/media/pdfs/best-kept-secrets-of-peer-code-review.pdf

Saving $150K = $15 per LOC





$1 Billion Dollar Bugs



snoopsmouse @ Flickr

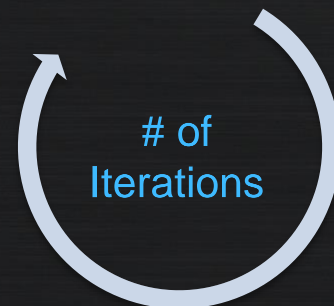
Linking back to Open Source & Xen Project

- Projects don't normally have a QA team
  ➔ Bugs discovered later ➔ even more expensive to fix
- Customer (user) discovered bugs are usually found in derivatives
  ➔ time-lag and thus cost to fix is even more expensive
- Bugs in FOSS projects are often not fixed
- Bugs and bad quality can damage the reputation of a project
- And by extension they can damage the business interests and reputation of contributors to that project (including your own)
- Asking maintainers to take your patch in without good review =
  Asking others to fix bugs and carry significant cost for you in future

Culture, Roles, Community Goals
Good Planning & Preparation
Following workflow and conventions
Systematic approach
Good Communication
Efficient Contributions

Factors that impact Review Duration

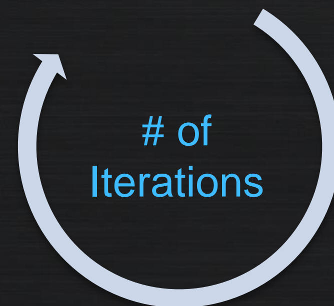
- Type of feature
  - Useful to many stake-holders or just to a single vendor?
  - Is the use-case explained or understood?
  - Do reviewers have all the information they need to be able to review?
- Complexity and Modularity
  - How many files and lines per patch?
  - How many components (hypervisor, qemu, toolstack, APIs, …)?
  - Does the structure of the patch/patch series help a review?
  - Do you need a design?
- Readability
  - Is it easy to infer the design from the patch?
  - Do you follow Coding standards?
  - Are complex code snippets explained within comments in the patch?



Each question may require a re-submit

Factors that impact Review Duration

- Code Quality
- Test failures, Coverity Scan, …
  - Will Coverity Scan throw up issues?
  - Do you need new Test Cases?
  - Should you include tests upfront?
- Time and Experience
  - Delays may require rebasing the patch!
  - How responsive are you to reviewer comments?
  - How responsive is the reviewer? He/she may have a queue of requests!
  - Use past submission experience to estimate # of iterations
  - Your standing in the community (your track record)
- Other factors
  - Some patches may require documentation (e.g. API docs)

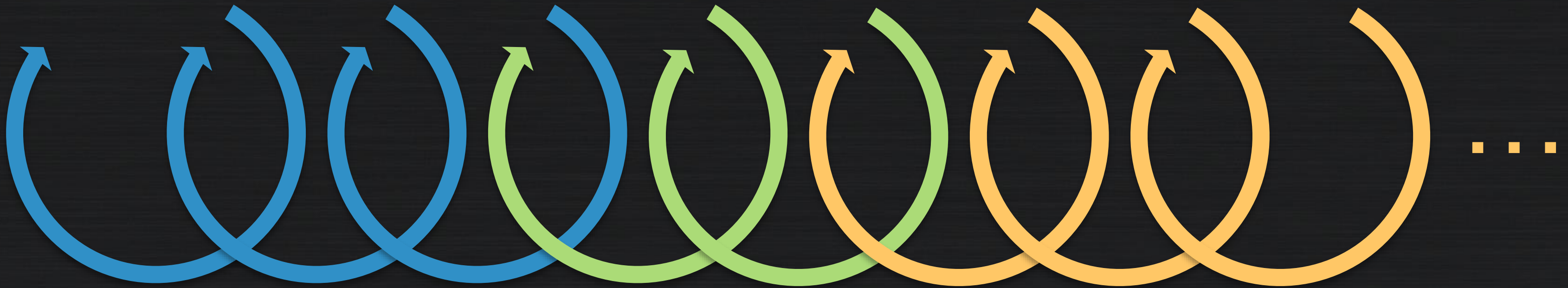


Mission Creep
(e.g. additional requirements to minimize risk)

Elapsed time per iteration adding up

Avoiding a "Jumbled" Review

Later it becomes clear that
there is an issue with the use-case,
design, architecture or assumptions



Reviewer takes pity on contributor
Gives some feedback (e.g. coding style, …)
May ask some questions about the use-case
and/or the design

Significant re-work
Extra effort for contributor and reviewer
At this stage both may be somewhat annoyed
(and we will get communication issues)

Quiz:

Why do “jumbled” reviews happen?

Causes for Jumbled Reviews

- Missing Information
- Wrongly set expectations due to misunderstandings
- The reviewer giving too detailed information before agreeing that he is happy with the use-case, architecture, design – and thus setting wrong expectations
- Another reviewer getting involved later down the road
  - There is also then potential for disagreement

Can good planning help?

A rough Planning Framework

Rationale
(more if controversial)

Use Case

Context
(Additional Information)

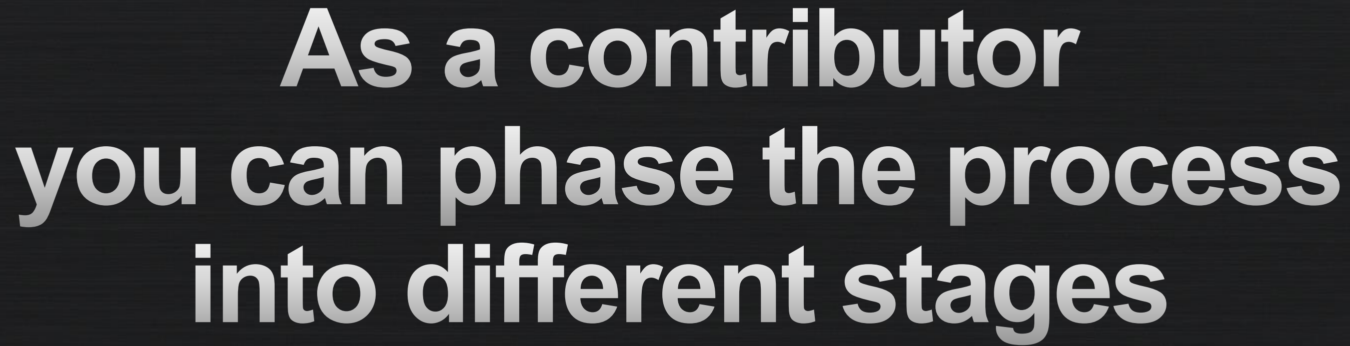
Design
(if complex)

Assumptions
(that you made)

Code &
Code Review

Systematic approach to acting on Feedback

Dealing with test issues


As a contributor
you can phase the process
into different stages

Techniques to Phase

{MORE LATER}

- The Xen Project does not have a design requirement, but …
  - Designs are welcome, when it makes sense
  - When unsure, whether a design helps, ask: Outline the use-case, problem and approach you are planning to take.
  - Design discussions labeled “Design” + some version number + some text
- Requests For Comments (RFCs)
  - For use-cases, prototypes, proof of concepts, etc.
  - Ask reviewers specific questions about, use-case, architecture, design, etc. & look at specific issues you want feedback on
- Timing
  - Design or related questions best at beginning of release cycle
  - Make sure you understand and engage with the Release and Roadmap Process
- Communication
  - Prompt reviewers: *Do you agree with Y (e.g. the design), given X (e.g. that I got some detailed feedback on the code, but also some design related questions)?*
  - The community is open to meetings in some cases (e.g. IRC meetings, calls, etc.) : high velocity communication can be more effective than mail.
    - BUT: it only works, if the key stake-holders agree to attend.
    - AND: document agreements / disagreements / open questions post the meeting by posting a summary to the list, such that there is a record

Interlude:

Interesting Facts about Code Reviews

In May of 2006 Cisco Systems performed a 10 month study of code reviews encompassing 2500 reviews of 3.2 million lines of code written by 50 developers.

This is the largest case study ever done on what's known as a "lightweight" code review process.

Some interesting Conclusions

- Reviewers become ineffective when reviewing code for more than an hour at a time ➔ Thus, a patch should be reviewable in < 1 hour
- Reviewers are most effective at reviewing small amounts of code.
  - Anything below 200 lines produces a high rate of defects, several times the average ➔ Thus, a patch should ideally be < 200 LOC and not larger than 400
  - After that the results trail off considerably; no review larger than 250 lines produced more than 37 defects per 1000 lines of code
- Reviews with author preparation (annotations explaining changes) have <u>significantly</u> smaller defect densities compared to reviews without ➔ Incidentally that helps the reviewer also

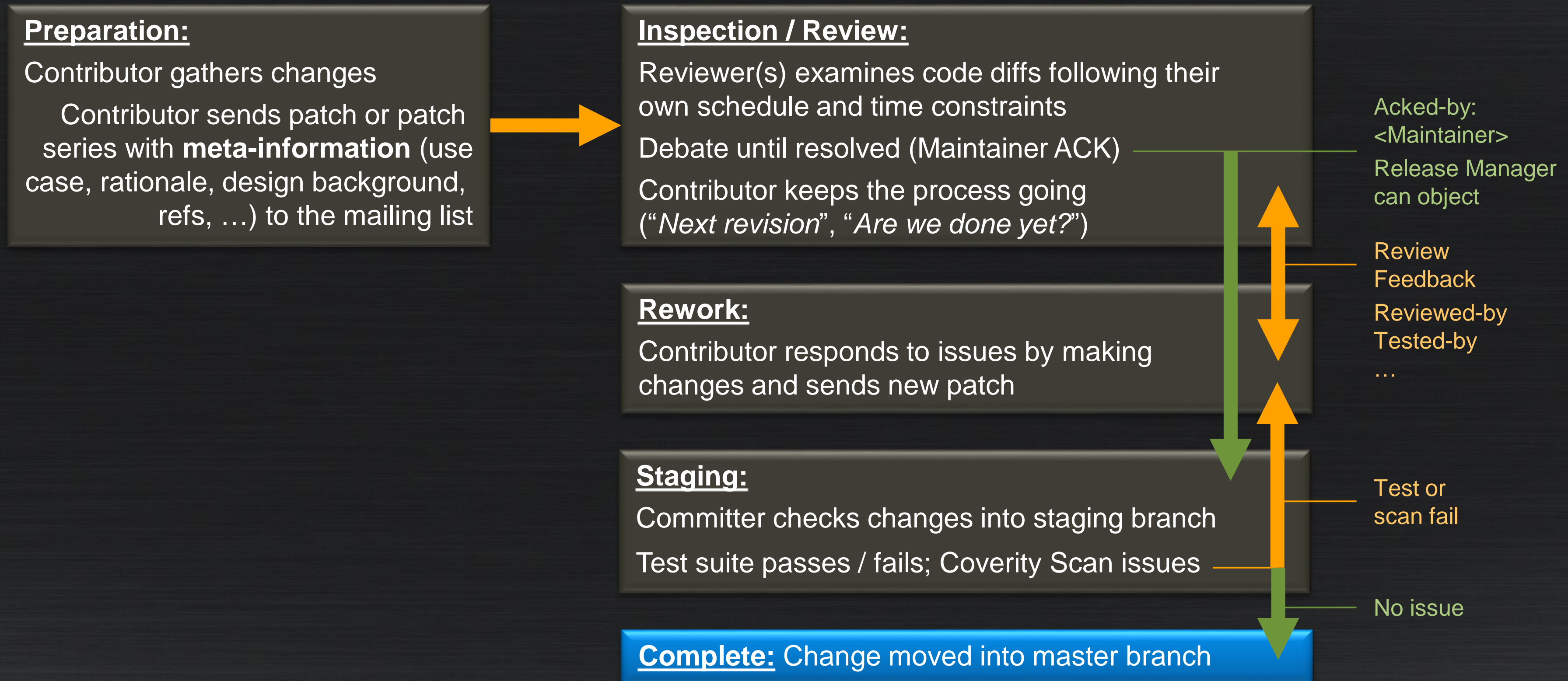
Social Effect of Peer Reviews

- The "Ego Effect": Developers whose code is being reviewed immediately develop code with fewer defects in them
- Systematic Personal Growth: Developers who systematically address issues raised and make notes of classes of issues found, learn from their mistakes and from feedback and become better developers through self-awareness
- Hurt Feelings: Taking criticism (in particular in public) isn't easy. The point of code review is to find issues. Hurt feelings are in most cases, the consequence of miscommunication and/or misunderstandings and not intentional → Which is why we will look at communication techniques later

Culture, Roles, Community Goals
Good Planning & Preparation
Following workflow and conventions
Systematic approach
Good Communicati on
Efficient Contributions

Workflow:
Mailing list based
(enable people in different time-zones to collaborate)

E-mail based review Process



Xen Project: Relevant Processes

From Xen Project Governance

- Principles: Openness, Transparency, Meritocracy
- Roles: Maintainers, Committers, Project Lead
- Conflict Resolution : Refereeing
- Contribution Guidelines: Developer Certificate of Origin
- Security Vulnerability Policy (relevant for Coverity Scan)

Xen Project: Relevant Conventions

Documented and regularly used undocumented conventions
(however changes to these are made only in line with governance)

- Patch contribution workflow
- Sign off (Acked-by, reviewed-by, etc.)
- Coding style
- Release Manager Role and Release Process (aka different stages)
- Access to Coverity Scan
- Staging-to-master pushgate and automated testing
- Personal repos hosted by Xen Project
- Design reviews (informal)
- Hackathons, Developer meetings, Ad-hoc meetings to resolve issues (informal)

Culture, Roles, Community Goals
Good Planning & Preparation
Following workflow and conventions
Systematic approach
Good Communicati
on
Efficient
Contributions

Two major causes of extra Iterations in Code Reviews

Some feedback (a specific issue or a class of issues) is not addressed

(At least one extra iteration with extra elapsed time)

Code which had positive feedback is changed

(May require re-review of already reviewed parts of a patch or patch series)
(May invalidate previously agreed sets of changes and reviews)

Particular challenges in email reviews

- Feedback comes in hierarchical threads
- Not always from one person
- Feedback is not received in a linear list over time

You need a “system” to systematically address issues

- Needs to fit your working style and personal preference
  - Otherwise you will get tired of it and won’t use it
- Usually, it comes down to having one master list of issues somewhere
  - Otherwise you will “loose” or not act on bits of feedback

Exercise:

How do you keep track of feedback from an email based code review?

Thomas Galvez @ Flickr

Culture, Roles, Community Goals
Good Planning & Preparation
Following workflow and conventions
Systematic approach
Good Communicati on
Efficient Contributions

Group Exercise:

You have 30 seconds to write down terms you associate with “scalability”

Thomas Galvez @ Flickr

"Blinking Words" (an MIT term)

Blinking Words are words or phrases that take on many possible interpretations, and where definitions blink between different meanings depending upon who hears it.

Note that the reason for the exercise is to show, how people with similar background can interpret terminology that is commonly used in their field very differently.

Communication: Adversarial vs. Collaborative

Adversarial Style: Two ideas enter, one idea leaves

Collaborative Style: Participants build off of each others' ideas, working together to create something new

Observation:

- Education systems across the world have often a bias towards adversarial communication
- The goal for code reviews (and patch reviews) is inherently collaborative
- BUT: often become Adversarial

Techniques to minimize Misunderstandings

Or Techniques to Debug a Conversation

- High Quality Explanation
- High Quality Inquiry
- The Left Hand Column (what was said and what you were thinking)
- The Ladder (a cognitive process on how humans draw conclusions)

More Later

Culture, Roles, Community Goals
Good Planning & Preparation
Following workflow and conventions
Systematic approach
Good Communicati on
Efficient Contributions

What makes a Smooth Contribution?

Awareness of Culture and others' Perspective

Good Planning and Preparation

Following the Process and Conventions

Systematic approach to acting on Feedback

Good Communicatio
n

More Later

Part 2: Xen Project Processes, Conventions and Governance

Part 3: Communication – or avoiding misunderstandings