BAD THINGS

Ad hominem

This ought to go without saying: review the code and not the coder. Comments about a person will only make it harder for that person to apply critiques about their code.

When making negative comments, refer to the patch or the branch rather than you. For example, You've introduced a bug in `get_message` becomes this patch introduces a bug in `get_message`.

Unclear Outcomes

If *all* you say is, this patch introduces a bug in `get_message`, then you have failed as a reviewer. The goal is to improve the code, not to provide a series of puzzles for the author.

The author should be able to look at a review and be able to tell how to address each point and also when they have addressed all points. Reviews with unclear outcomes turn into open-ended discussions about the patch, which sometimes become focused on making the reviewer happy, rather than improving the code.

Confusing personal preference with objective worth

This is a problem in all spheres of review. Film critics, literary editors and acadamic reviewers all do it. What I like is not necessarily the same as what's good, although part of becoming a better programmer is having your preferences align better with reality.What I dislike is perhaps even less likely to be the same as what's bad.

When reviewing a perfectly acceptable patch that solves a problem using imperative-style programming, *do not* criticize it simply because it isn't in a more functional style. Doing otherwise makes reviews a game of guess what the reviewer likes rather thanwrite good code.

Reviewers can avoid this trap by phrasing review comments as questions, Did you consider using a more functional style?, Why aren't you using regular expressions to solve this problem? etc.

See also Specific feedback is good feedback.

While you're there...

When reviewing code, it is tempting to ask the author to fix other problems that are in the same area. Without care, this can quickly become an exercise in making this area of the code perfect, when previously both reviewer and author were concerned about merely improving the code.

The solution here is to strongly value incremental improvements, see Better is better, perfect is impossible.

Filibustering

Filibustering is an American political term for indefinitely prolonging debate on a bill in order to prevent that bill from being enacted. It is sometimes used in Free Software projects to prevent patches from going in.

The effects of filibustering can sometimes happen unintentionally. A reviewer rejects a patch for not having unit tests, the author asks how do I unit test this code? and the reviewer never quite gets around to responding.

Apart from being simply an unpleasant process, the author has work left hanging, which it harder for them to submit more patches.

See Unclear outcomes and Fast feedback is good feedback.

GOOD THINGS

Specific feedback is good feedback

Pinpoint the exact line of code that could be improved. Say what is wrong with it. Suggest one way to make it better. Make sure the author can tell when it will be good enough.

Fast feedback is good feedback

Once authors submit patches for review, there's nothing more they can do on those patches. They have to wait until the reviewer has replied before they can proceed. Reply to reviews quickly to keep the author's attention focused.

Better is better, perfect is impossible

No patch will be perfect, and no patch will fix all problems in existing code. Don't aim for perfection, aim for incremental improvements.

Be thankful

Someone has just tried to improve your product. They've put thought, effort and creativity into helping you, and now they have put their work up for critique: thank them.

Divmod have a policy of always saying one good thing in each code review. There is always something nice to say, even if it's just I'm glad someone is looking at this part of the code.

Ask questions

Reviewers can't go wrong with asking questions. In the worst case, the author will get an obvious answer. In the best case, both reviewer and author learn something new.