Code Review Guide

由 Markus Geiss创建, 最后修改于一月 07, 2016

What’s a code review for?

In some organizations code reviews degrade to a rubber stamp. Or a proving ground. But this is not the true purpose of a code review.

A code review should:

- educate the reviewer about changes in the code, and programming concepts familiar to the reviewee,
- educate the reviewee about programming concepts familiar to the reviewer, and
- keep quality high
- ...of the code change, and **not** all the code that the change touches.

For the reviewee, it is an opportunity to step back and look at the bigger picture of his change. It is an opportunity to pat himself on the back for finishing his task. It is also an opportunity to learn from the reviewer. And it is an opportunity to do a final round of polishing on his work.

For the reviewer it is an opportunity to learn not only about the code, but about the reviewee. Which kinds of quality does he value? What skills does he bring to the table?

What kinds of problems can occur with code reviews?

By nature a code review is an asymmetrical exchange. The code reviewee offers his work product, the reviewer offers a third and fourth eyeball. This is the main source of problems in a code review. The reviewer, out of respect for what the reviewee has done, may hesitate to point out real problems in the code. The reviewee, out of pride in his accomplishment, may find it difficult to accept legitimate criticism. Or worse, the reviewer, out of envy that he didn’t get to make the change, may complain about features of the code that are matters of taste. The reviewee, out of insecurity, may allow himself to be pressured into making changes he doesn’t believe in.

What should I be checking in a code review?

This list of questions should help both the reviewer and the reviewee keep a code review on track. You’ll notice that at no point do we state what the answer to a question should be. That is not because we don’t have an opinion, and most of the time that will be obvious. However, it is not our code review; it’s yours. The code reviewer and the code reviewee will decide together what the correct answers to these questions are. Just make sure you’re not avoiding any of these questions because you’re afraid you’ll fight over the right answer.

Correctness

- Does it make sense to fix this bug or add this feature?
- Does the code change solve the problem it is intended to solve?
- Does the code change degrade the performance of the code?
- Does the code change cause any undesirable program behavior?
- Does the code compile?
- Is it free of warnings?

Security

- Is the code free of code injection vulnerabilities?
- Does the code adequately protect sensitive data such as passwords, user names, and financial information?

Style

- Does the code follow the style guidelines the community has agreed on?
- Do the changes preserve the code history? When another programmer looks at the code, how easy will it be to find the commit messages and tickets associated with a particular line of code?

Comprehensibility

- Do I understand the code?
- Would an undergraduate CS student understand the code?
- Are code artifacts such as classes, functions, and variables given informative names?

- Is the documentation helpful? Does it contain any necessary information which cannot be derived directly from the code?
- Are JavaDocs of public functions and classes provided where they are appropriate?
- Do the JavaDocs cover error cases and boundary conditions?
- Do the changes have code-external documentation where appropriate?
- Does any user-facing documentation need adjustment?

Error handling

- Does this code handle potential error sources from the code it calls?
- Will it respond well if a null is returned?
- Will it respond appropriately if an unchecked exception is thrown?
- Are checked exceptions handled thoughtfully?
- Does the code perform appropriate input/precondition checks?
- Does the code signal errors in a manner useful to code or users that might call it?

Test coverage

- Does it have unit tests? Do they pass?
- What is the code coverage?
- What boundary conditions do the unit tests address?
- Does it have integration tests? Do they pass?
- What boundary conditions do the integration tests address?

Architecture

- What is the kind of code cohesion within the classes and packages touched?
- Does the commit change the code cohesion?
- What is the kind of code coupling within the classes and packages touched?
- Does the commit change the code coupling?

Deployment

- Do the changes effect deployment of the system?
- Are the deployment changes documented in a place that the effected users can find?
- Do the changes demand no more configuration from users than that information which the software absolutely cannot determine automatically?
- Are configuration defaults reasonable?

Sources

- https://en.wikipedia.org/wiki/Cohesion_%28computer_science%29
- https://en.wikipedia.org/wiki/Coupling_%28computer_programming%29
- http://www.infoq.com/news/2008/03/code-review-antipatterns

无标签