On Code Reviews

Reviewee: Preparing Code Reviews

1. **Review your own code** before seeking a review
   - is it ready to be reviewed by someone else?
2. Use **automation tools** to solve mundane tasks before reviews
   - Linting, Formatting, static code analysis, “test coverage”
3. Ensure **coding standards** established in the team are followed
   - Naming conventions, commenting, functional vs. OOP
4. Make sure you provide only a **small part** to review
   - Avoid overly large PRs/MRs
   - Allow to review in **incremental steps**?
5. Set clear **goals** of what to achieve with the code review (for the reviewer)
   - E.g. Improve code quality, find bugs, ensure coding best practices, learn about the work and the domain

Reviewer: During the Review

1. **ALWAYS**: Be constructive, friendly, and avoid personal criticisms
2. Identify **potential issues** (bugs, vulnerabilities, etc.)
3. **Prioritize readability & maintainability**
   - Is the code easy to understand? – if the reviewer says no, it isn't.
   - Focus on improving names, simplify complex logic, code smells
   - Outside-in mindset: test understandable? Interface understandable? Implementation understandable?
4. Verify "test coverage"
   - Are the tests readable (business behavior) and adequate for the task to solve?
   - Are both happy paths and edge cases covered?
   - Can tests be used as documentation (test names, GWT/AAA, etc.)?
5. Avoid commenting on code that was not touched by the reviewee

Fostering Collaboration and Appreciation

1. **Always**: encourage healthy dialogue and discussions
2. If possible, do reviews together or schedule a follow up meeting to go through comments
   - Builds trust and understanding – find common ground
   - Work together in pairs on the same task – review becomes simpler
3. Express **gratitude** and **acknowledge work/time invested**
4. **Recognize good work**
   - publicly acknowledge high-quality code
   - Provide positive reinforcement
5. **Lead by Example**
   - Reviewer: always think about how you would react if you got a similar comment. Provide as much context to understand your comment as unambiguously as possible.
   - Reviewee: be open to and actively seek feedback to your code. Don’t be overly defensive. It’s usually not personal.
   - If a discussion leads nowhere – one party (usually reviewer) should accept it.
   - Never become dogmatic! Every decision has its pros and cons.

Avoiding Conflicts

1. **LEAVE YOUR EGO AT THE DOOR – ALWAYS**
   - Approach reviews as **Learning Opportunities**
   1. Pair / Team Programming makes ego problems go away
2. Leave **clear** and **respectful comments**
   - Never criticise the developer personally!
3. **Balance** a thorough review against nitpicking
   1. **Prioritize** – you can never do everything!
   2. Focus on the **goal** of the review
4. Handle **pushback gracefully**
   - Don’t engage in lengthy discussions spanning multiple threads
   - Resort to personal reviews quickly should it get out of hand

Addressing Found Issues

1. Reviewer & Reviewee: prioritize what to fix
2. Reviewee: fix it timely
3. Reviewee: request a new review
4. Reviewer: review new changes timely & thoroughly, avoid glancing over!
5. In general: seek to come to a conclusion soon
   - Try to close changes / pull requests ASAP

Questions to Ask Yourself During a Code Review

1. Do you approach the process with a positive attitude?
2. Are you nonjudgmental with colleagues?
3. Are you choosy and rejecting reviews only because of nitpicking minutia in the reviewed code?
4. Is competition out of the door?
5. Is the process tedious?
6. Are you making the defects more about the programmer rather than the code?
7. Are you rewarding success?
8. Are you valuing teaching more than critiquing?
9. Are commits gigantic scattered pieces of code that need to be reviewed?
10. Are you using code reviews as a way of improving communication within the team?
11. Are you using code reviews as a learning experience?
12. Are the objectives of the code review clearly defined?
13. Are all asked changes properly prioritized and addressed?
14. Is the review followed by a plan to put in action to fix defects?
15. Are you fully committed as a reviewer?
16. Are you biased by the perception of yourself (e.g., too young or too senior to perform a review)?
17. Are you biased by the perception of others?
18. Are you glancing over code?
19. Do you rebuild and test the code?
20. Do you check that the code is accompanied by additional documentation and tests?
21. If code is unclear, do you ask for clarification?
22. If you asked for a change, do you carefully review again the newer code to ensure important defects have been addressed?

https://learning.oreilly.com/library/view/implementing-effective-code/9781484261620/html/485619_1_En_11_Chapter.xhtml