Code Review Guidelines

BEST PRACTICES FOR CODE REVIEW

A2SV
AFRICA TO SILICON VALLEY

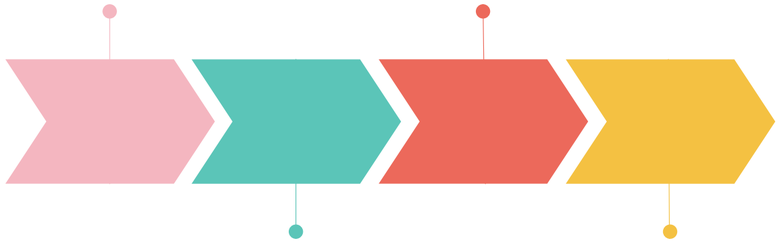
AGENDA

- The Standard of Code Review
- Guideline for PR creators
- What to Look For In a Code Review
- Navigating a PR in Review
- How to write code review comments
- Scenarios/FAQ

THE STANDARD OF CODE REVIEW

A2SV
AFRICA TO SILICON VALLEY

- The primary purpose of code review is to make sure that the codebase stays consistent, maintainable, the overall health of code base is improving over time.
- Key points of code review
  - **Trade-offs**: Balancing progress and code quality.
  - **Reviewer's Role**: Ensure changes improve the codebase.
  - **"Better, Not Perfect"**: Prioritize continuous improvement.
  - **Ownership**: Reviewer ensures code consistency.
  - **Approval Criteria**: Changes should improve system health.



THE STANDARD OF CODE REVIEW

AFRICA TO SILICON VALLEY

- **Principles of Code Review**
  - Technical Facts over Opinions: Base feedback on solid technical data.
  - Style Guide Authority: Follow style guide for consistency.
  - Consistency with Codebase: Maintain consistency to promote system health.
- **Conflict Resolution**
  - Consensus Building: Resolve conflicts through discussion.
  - Face-to-Face Meetings: Consider direct discussions for complex issues.
  - Escalation Path: Involve broader team or technical lead if needed.

GUIDELINE FOR PR CREATORS

- Small PRs
  - The right size for a PR is one self-contained change.
  - The PR makes a minimal change that addresses just one thing.
  - The PR should include related test code. (new tests or updated tests).
  - Everything the reviewer needs to understand about the PR (except future development) is in the PR, the PR’s description, the existing codebase, or a PR they’ve already reviewed.
  - The system will continue to work well for its users and for the developers after the PR is checked in.

GUIDELINE FOR PR CREATORS

A2SV
AFRICA TO SILICON VALLEY

- Smaller PRs are useful
  - Quicker reviews
  - Easier to merge
  - Reviewed more thoroughly
- Large PRs are okay only when
  - They involve deletions of whole files
  - When it is generated by automatic refactoring tools that are trusted
- Reviewers can reject your change outright for the sole reason of it being too large. They will thank you for your contribution but request that you somehow make it into a series of smaller changes.

WRITING GOOD PR DESCRIPTIONS

- First Line:
  - Short summary of what is being done
  - Complete sentence, written as an imperative
  - Informative enough for future code searchers
- Body:
  - Fill in the details and include any supplemental information
  - Describe the problem being solved and why this is the best approach
  - Include background information (e.g., bug numbers, benchmark results, design documents)

WRITING GOOD PR DESCRIPTIONS

- Examples:
  - Delete the FizzBuzz function and replace it with the new system
  - Deleting the FizzBuzz function and replacing it with the new system
- Focus on clarity and usefulness in understanding the PR. Examples of bad PR descriptions:
  - “Fix build.”
  - “Add patch.”
  - “Moving code from A to B.”
  - “Phase 1.”
  - “Add convenience functions.”
  - “kill weird URLs.”

WHAT TO LOOK FOR IN A CODE REVIEW

AFRICA TO SILICON VALLEY

- **Design:** Is the code well-designed and appropriate for your system?
- **Functionality:** Does the code behave as the author likely intended? Is the way the code behaves good for its users?
- **Complexity:** Could the code be made simpler? Would another developer be able to easily understand and use this code when they come across it in the future?
- **Tests:** Does the code have correct and well-designed automated tests?
- **Naming:** Did the developer choose clear names for variables, classes, methods, etc.?
- **Comments:** Are the comments clear and useful?
- **Style:** Does the code follow our style guides?
- **Documentation:** Did the developer also update relevant documentation?

PRACTICAL EXAMPLE

- **Step 1: Take a Broad View of the Change**
  - Review the PR description and overall purpose.
  - Determine if the change is sensible and has a clear description.
- Practice PR
  - https://github.com/bontu-fufa/code-review-practice/pull/1

PRACTICAL EXAMPLE

- **Step 2: Examine the Main Parts of the PR**
  - Identify the files with the most significant logical changes.
  - Focus on reviewing these major parts first to gain context.
  - Communicate any major design issues promptly.

PRACTICAL EXAMPLE

- **Step 3: Look Through the Rest of the PR in an Appropriate Sequence**
  - Establish a logical order to review the remaining files.
  - Review the remaining files in order, checking for any related changes or potential impacts.
  - Send comments to the developer highlighting the design issue and suggesting improvements.

HOW TO WRITE CODE REVIEW COMMENTS

AFRICA TO SILICON VALLEY

- In general
  - Be kind and respectful.
  - Explain your reasoning.
  - Encourage developers to simplify code.
- Label comment severity
  - **Nit:** This is a minor thing. Technically you should do it, but it won't hugely impact things.
  - **Optional (or Consider):** I think this may be a good idea, but it's not strictly required.
  - **FYI:** I don't expect you to do this in this CL, but you may find this interesting to think about for the future.

SCENARIOS/FAQ

- What if I don't understand a particular piece of code that I'm reviewing?
- What if I disagree with the author's implementation?
- How do I ensure that the code I'm reviewing is readable?
- What should I look for in terms of performance and efficiency?
- What are some common code smells to look out for?

RESOURCES

- **A2SV:**
  - Revised Review Guideline
- **Style guides**
  - C# at Google Style Guide | styleguide
  - Google TypeScript Style Guide
  - Go style guide at Google
  - Dart style guide
- **Documentation templates and style guides:**
  - 6 Best Practices to Manage Pull Request Creation and Feedback
  - Using templates to encourage useful issues and pull requests - GitHub Docs

Thank you!