Code Reviews
From Bottleneck to Superpower

Dr. Michaela Greiler
Consultant and Trainer, Researcher
www.awesecodereviews.com

Code reviews correlate with a reduction of defects.
Fagan, Kemerer and Paulk, Tanaka et al.

Unreviewed code is 2X more likely to introduce bugs.
Bavota and Russo, Thongtanunam et al.

80% of code reviews lead to code improvements at Goggle.
Bacchelli and Sadowski.

Code reviews increase knowledge sharing.
Rigby, 2013
Pain Points

waiting
interruption
large reviews
low feedback quality
time pressure
difficult review
context switch
performance pressure
low/no rewards
feeling attacked
bullying
conflicts

Socio-technical engineering practice
Organizational Skills

The Secret Sauce for Successful Code Reviews

- BETTER COMMUNICATION
- REVIEWERS’ SUCCESS
- PURPOSE DRIVEN PROCESSES
Better Communication

What & How
Not all code review feedback is equal!

Source: Characteristics of useful code reviews: an empirical study at Microsoft, Bosu, Greiler, Bird
Code Review Feedback

Best:
- Functional defects, missing corner cases or validation, Api usage, best practices

OK:
- Documentation, coding style & conventions, spelling mistakes

No-go:
- Alternatives without benefits, existing tech debt and refactoring, planning and future work

Reading modes

Ad Hoc: no predefined way to do the code reviews

Checklist-based: review based on questions, instructions

Scenario-based: scenario is “data type consistency”

Test-driven code reviews: analyze by starting with the test code
### Implementation
- Does this code change do what it is supposed to do?
- Can this solution be simplified?
- Does this change add unwanted compile-time or run-time dependencies?
- Was a framework, API, library, service used that should not be used?
- Was a framework, API, library, service not used that could improve the solution?
- Is the code at the right abstraction level?
- Is the code modular enough?
- Would you have solved the problem in a different way that is substantially better in terms of the code's maintainability, readability, performance, security?
- Does similar functionality already exist in the codebase? If so, why isn't this functionality reused?
- Are there any best practices, design patterns or language-specific patterns that could substantially improve this code?
- Does this code follow Object-Oriented Analysis and Design Principles, like the Single Responsibility Principle, Open-closed Principle, Liskov Substitution Principle, Interface Segregation Principle, Dependency Injection?

### Logic Errors and Bugs
- Can you think of any use case in which the code does not behave as intended?

### Dependencies
- If this change requires updates outside of the code, like updating the documentation, configuration, readme files, was this done?
- Might this change have any ramifications for other parts of the system, backward compatibility?

### Security and Data Privacy
- Does this code open the software for security vulnerabilities?
- Are authorization and authentication handled in the right way?
- Is sensitive data like user data, credit card information securely handled and stored?
- Is the right encryption used?
- Does this code change reveal secret information like keys, passwords, or user names?
- If code deals with user input, does it do input sanitization and cross-site scripting? SQL injection?
- Does this code change improve libraries choices?

### Readability
- Write

### Ethics and Morality
- Does this change make use of user data in a way that might raise privacy concerns?
- Does the change exploit behavioral patterns or human weaknesses?
- Might the code, or what it enables, lead to mental and physical harm for (some) users?
- If the code adds or alters ways in which people interact with each other, are appropriate measures in place to prevent/limit/report harassment or abuse?
- Does this change lead to an exclusion of a certain group of people or users?
- Does this code change introduce unjust impact on people, particularly those related to sensitive characteristics such as race, ethnicity, gender, nationality, income, sexual orientation, ability, and political or religious belief?
- Does this code change introduce any algorithm, AI or machine learning bias?

Source: https://github.com/mgreiler/code-review-checklist
Not all code review feedback is equal

High quality code adheres to coherent coding styles and has no spelling mistakes.

Do not comment on my coding style and spelling mistakes.

Perspective
Written communication especially tricky

This variable should have the name ‘userId’.

This variable should have the name ‘userId’.

https://www.michaelagreiler.com/respectful-constructive-code-review-feedback/
The Secret Sauce for Successful Code Reviews

31
I need more information to understand the code and give valuable feedback.

For the change I worked on, there is no need for a lengthy review description. The context and code are clear enough.

Expert Blindspot

“Experts are frequently so familiar with their subject that they can no longer imagine what it’s like to not understand the world that way...”
10 lines of code = 10 issues.

500 lines of code = "looks fine."

Code reviews.

Defect Density vs. LOC

200-400 LOC are the maximum a developer can effectively process

Source: Best Kept Secret of Code Reviews a study of Cisco's code review
The Secret Sauce for Successful Code Reviews

BETTER COMMUNICATION

REVIEWER’S SUCCESS

PURPOSE DRIVEN PROCESSES
How should your code review process look like?

- When is a code change ready for review?
- Who should be on the code review?
- When should the review take place?
- What should reviewers look for?
- When can a code change be approved?

**Goals and Policy**

- Finding Defects: Reviewer Selection, Approval Criteria
- Improvements: Code Review Characteristic, Timing
- Awareness: Focus of the Code Review
- Mentor and Teach: Code Review Behavior
- Tracing & Tracking: Communication Channels, Code Review Tools, Automation
Main Pain Points

- Slow Turn-Around Time
- Low Review Quality

The Code Review Quadrant

- **Value Reviews**: thorough reviews impacting development speed
- **Power Reviews**: thorough reviews delivered in a timely manner
- **Blocking Reviews**: sloppy reviews impacting development speed
- **Omissible Reviews**: sloppy reviews delivered in a timely manner
Is this possible?

Or does this happen?
Most complaints about the code review process are resolved by making the process faster.
No one-size fits all strategy

Development teams face unique resource, time, and scope constraints that influence the choice of workflow and practices used.
Align your Process to your Goals

**Goals and Policy**

- Finding Defects
- Improvements
- Awareness
- Mentor And Teach
- Tracing & Tracking

- Reviewer Selection
- Approval Criteria
- Code Review Characteristic
- Timing
- Focus of the Code Review
- Code Review Behavior
- Communication Channels
- Code Review Tools
- Automation

---

**Proven Best Practices from Microsoft**

- **Self-Review**: Read through the changes carefully before submitting the code review
- **Small Reviews**: Aim for small, incremental changes
- **Coherence**: Cluster related changes
- **Context**: Describe the purpose and motivation of the change
- **Training**: Set-up for success through training and education

---

52
Proven Best Practices from Microsoft

- **Tests**: Run tests before submitting a code review
- **Automation**: Automate what can be automated
- **Tools**: Right code review tools and CI/CD integration
- **Skip/Indicate**: Skip/Indicate unnecessary reviews
- **Crystal Clear**: Crystal clear guidelines and policies reflecting your goals

---

Summary
BETTER COMMUNICATION

FOCUS ON REVIEWER’S SUCCESS

PURPOSE DRIVEN PROCESSES

michaelagreiler.com
t@greiler
Get Dr. Greiler’s Code Review E-Book including Code Review Checklists

bit.ly/CRE-Book

Questions & Discussion

@mgreiler

michaelagreiler.com

hi@michaelagreiler.com