Mastering the Code Review Process

PETER MULDOON

Mastering the Code Review Process

Balancing Quality, Speed and Sustainability in your codebase

Engineering

Bloomberg

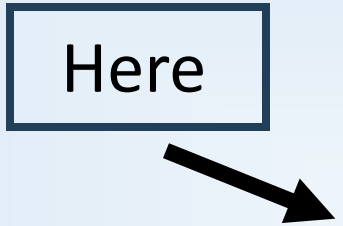
CppCon 2025
Sep 17, 2025

Pete Muldoon
Senior Engineering Lead

TechAtBloomberg.com

Questions

#include <slide_numbers>



Who Am I?

- Started using C++ professionally in 1991
- Professional Career
  - Systems Analyst & Architect
  - 21 years as a consultant
  - Bloomberg Ticker Plant Engineering Lead
- Talks focus on practical Software Engineering
  - Based in the real world
  - Demonstrate applied principles
  - Take something away and be able to use it

CppCon 2025

Talks :

- Back to Basics : Code Review
- API Structure and Technique : Learnings from Code Review
- Seamless Static Analysis with Cppcheck: From IDE to CI and Code Review
- Mastering the Code Review Process

Where will we be going?

- Talk will be about having an effective *Technical Code Review* process
- Consequences of poor technical code reviews
- The fundamental elements of the effective Code Review
- Tracking/Measuring your progress
- Improving code review culture

Talk is rooted in ***real-world*** enterprise environments

Where will we *not* be going?

- The *formatting* of text/commenting style is completely unaddressed
  - Style guides should cover this and preferably enforced via automated tooling

Terminology

What is a Critique?

A **critique** is a formal analysis and evaluation of your own or someone else's work

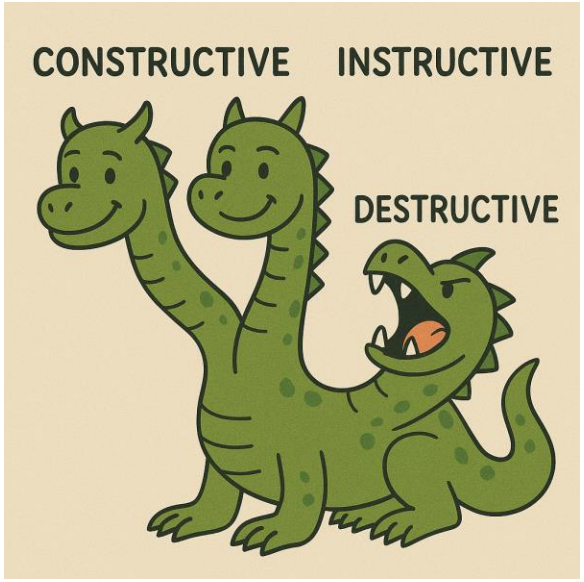
A **critique** is a careful judgment in which you give your opinion about the good and bad parts of something – Merriam-Webster

Criticism seeks to tear a person down and pass judgement, while a *Critique* seeks to help & motivate improvement.

Terminology

There are three main flavors of critique: constructive, instructive and destructive.

- Constructive builds up, identifies a problem and offers solutions
- Instructive avails of the opportunity to add to what someone's knowledge base
- Destructive tears down and is generally unhelpful



Bloomberg

Image generated by ChatGPT/DALL-E

Perspective

What are the core objectives of a code review?

The core goals of code reviews is to ensure that code changes are correct, maintainable, and fit for purpose—while supporting fast feature delivery and maintaining the long-term health of the codebase

Perspective

What are the less obvious objectives of a code review?

- Promote knowledge sharing
- Promote shared ownership
- Spread the workload

Perspective

State of Code Review Report 2020 by SmartBear

Number 1 way to code quality

1. ***Code Review*** : Over the last 3 years – and somewhat expected – our respondents have told us that the ***number one way a company can improve code quality*** is through ***Code Review***. This year, ***24% of our respondents*** indicated this.
2. Results also indicated that ***Unit Testing*** is the second most important at 20% of responses
3. followed by ***Continuous Integration and Continuous Testing***

Perspective

What place in (practical) software engineering do Code Reviews sit?

Rationale:

- Read more code than you write
- Last line of defense for your codebase
  - Key to ensuring long term sustainability of the codebase
- It can be a force multiplier for good practice
  - Opportunity to disseminate/collaborate
  - Opportunity for learning *

Unlikely to be replaced by AI

Image generated by ChatGPT/DALL-E

Code Review Tensions

Code reviews can have mutually exclusive directives:

Contending viewpoints on :

- Standards
  - Rigid vs Lax
- Timing
  - Rushed vs Tardy
- Scope
  - Minimalist vs Expansive
- New feature use
  - Conservatively consistent vs newly Modern

All can be taken to an extreme so a balance must be found

Image generated by ChatGPT/DALL-E

Poor Code Reviews

What are the consequences?

- Critical issues and defects go unnoticed
  - Superficial or rushed reviews
- Inconsistent code quality
  - Differing or no standards
- Poor development velocity
  - PRs sit idle
  - Huge complex PRs routinely generated
  - Spotty test coverage
- Bike shedding
  - Focus on the superficial & easy
  - Style & nitpicks

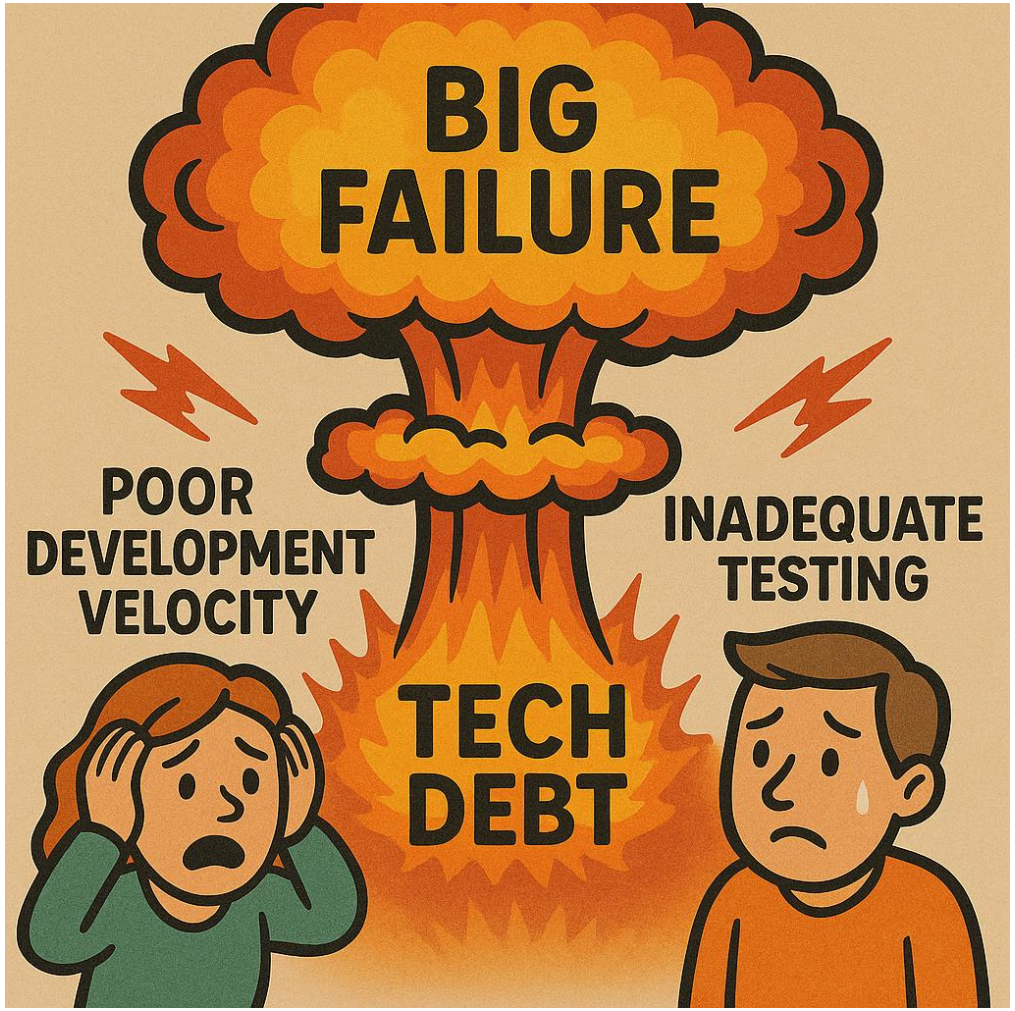


Image generated by ChatGPT/DALL-E

Code Review Preparation

Before you ask for a review

Check list:

- Static Code analysis / Compiler warnings
  - Taken care of
- Passes all unit/integration tests
  - New tests have been added
- Self review
  - Catch silly mistakes
  - Use AI

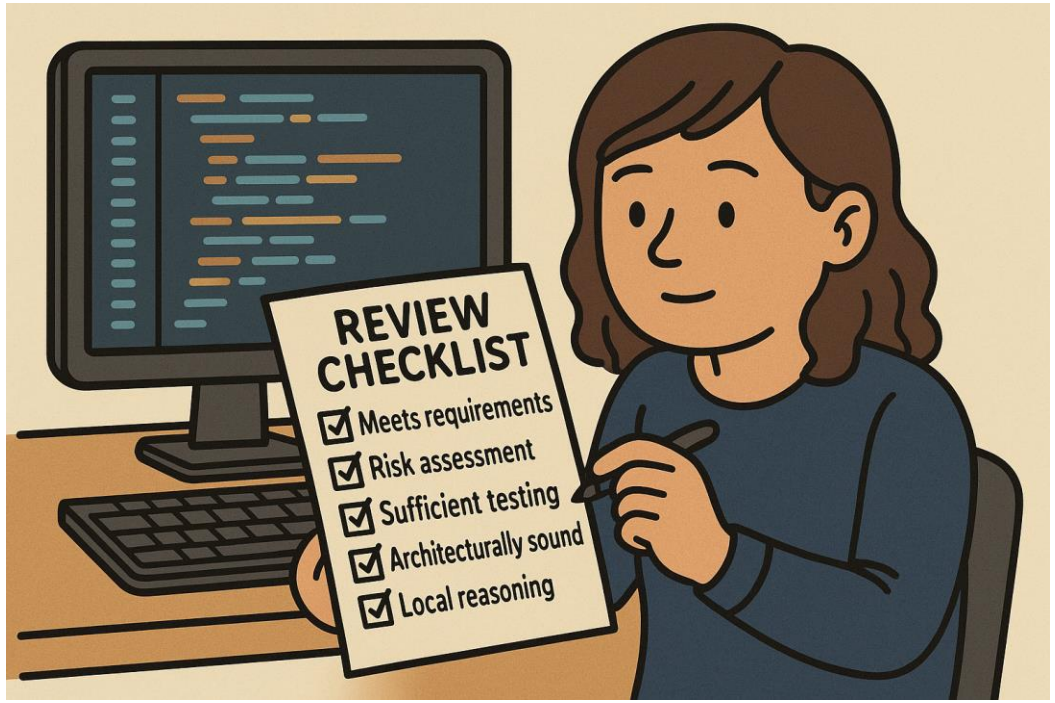


Image generated by ChatGPT/DALL-E

Code Review Preparation

First things first

Write a proper ***title*** and ***description*** describing the change and detailing the overall motives and any useful background information and requirements.

Focus on:

- What – The change you are effecting
  - *May* be deducible from the diff
- Why – the reason or background for the change
  - *Probably not* deducible from the diff
- How you are verifying/testing the change?

Code Review Preparation

Take it easy on your reviewer(s)

- Do a self review first
  - Reviewers should not be catching obvious / careless errors
  - Fix things that don’t look right
- Reduce cognitive load
  - Limit the size/scope
  - Don’t sneak in unrelated fixes
  - Expressive code and comments
  - Highlight where key changes are located

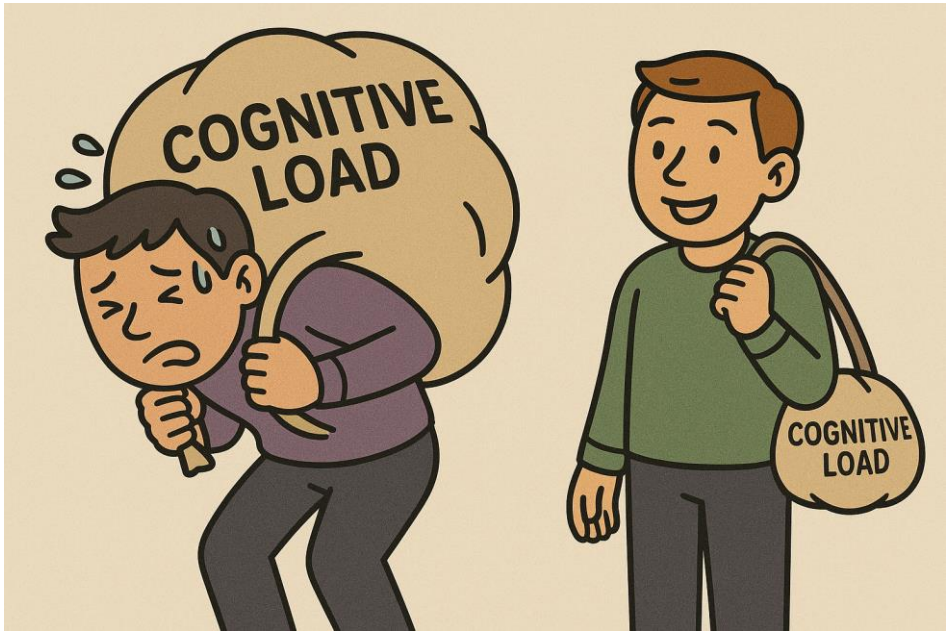


Image generated by ChatGPT/DALL-E

Code (P)Reviews

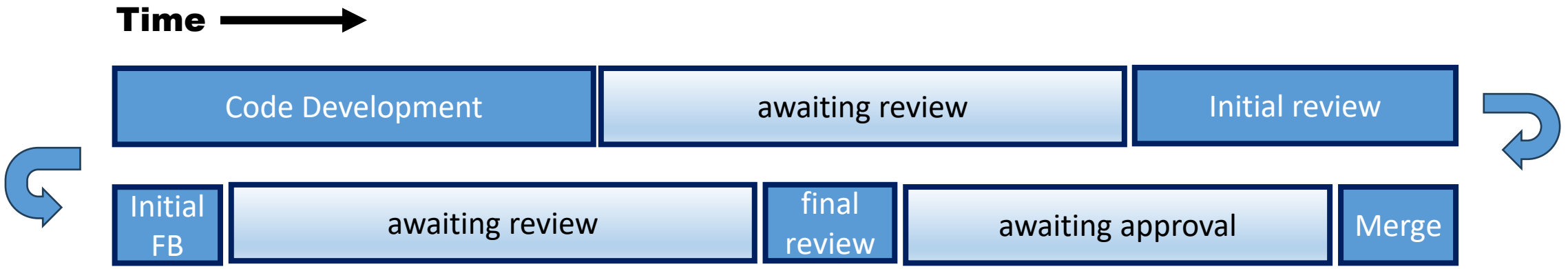
Draft PRs:

Usually for POCs, aims to discover

- Early feedback, am I on the right track?
- Identify fundamental flaws in early design
- Interface scrutiny
- Unknown behavior – to you – that's unaccounted for
- Does ***not*** get merged

Image generated by ChatGPT/DALL-E

Code Review Timeline



Code Review Sizing

Size matters:

- Large mega change takes a long time to move through the review process
  - Hard to spot potential problems
  - Worried about approving
  - Fine for simple repetitive change to many files
- High cognitive load make for unpleasant reviewing
  - Takes a long time
  - Hard to propose alternatives
- The more unstructured the codebase, the smaller the change set should be
  - Smaller meaning a single aspect
- Spacing/indentation/renaming should mostly be left to their own PR
  - Ideally automated using tooling

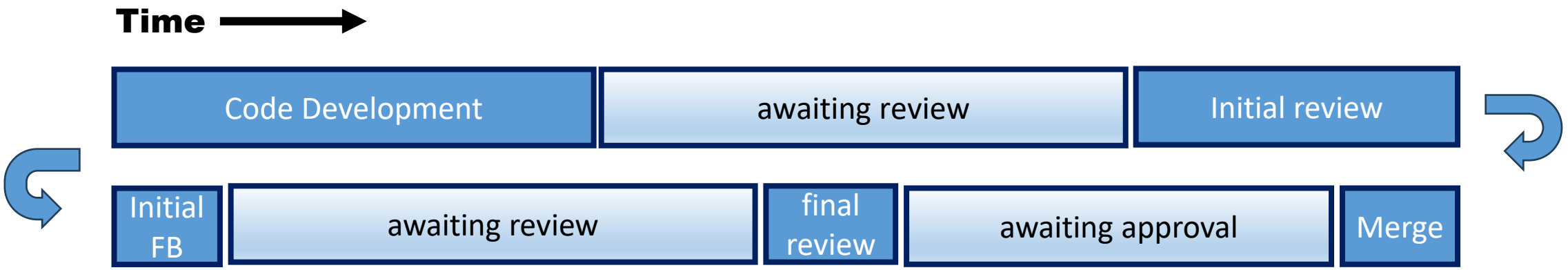
*Note: Poorly understood changes have more risk due to ***poorer*** feedback

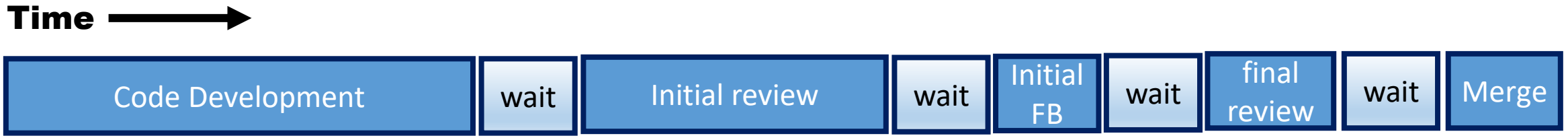
Code Review Timing

Timing is everything:

- No change is useful until it shows up working everywhere in PROD

Code Review Prioritization





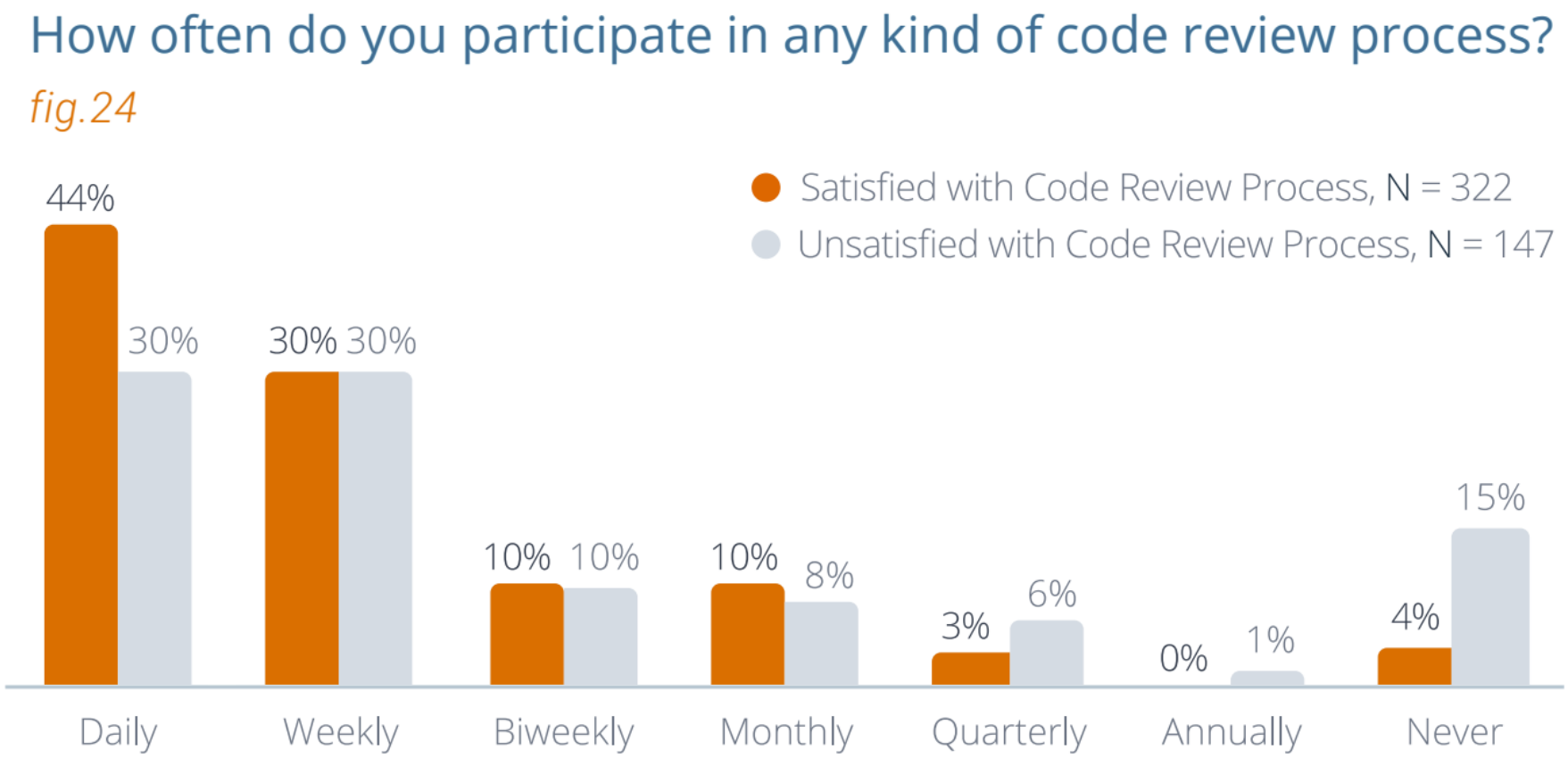
Code Review Timing

Timing is everything:

- No change is useful until it shows up working everywhere in PROD
- Rushing to write a piece of code then lollygagging on the review makes no sense
- Blocks progress, so completing reviews should be a high priority
- Retention fades with time
  - 2 weeks later, I will have to reload everything both PR and FB into my brain
- Reviewers should have a standard response time to begin a review (24 hours)
  - If you can't do it in that time, you are publicly reminded/dropped
- Review feedback response should be similarly constrained(24 hours)

Once an initial PR is sent out, The developer is responsible to push the code review to completion aka the merge in a timely manner

Code Review Timing



Satisfied teams perform daily code reviews

Feedback Categories

There are 3 classes of feedback each with approval outcomes:

- *Approval Blocker: Must* be fixed ⛔
  - Identify outright mistakes
  - badly structured solutions
- Approval Delay: *May* be fixed 🛠️
  - Alternative patterns/implementations
  - Persuasion to your viewpoint but likely not an outright blocker
- Approval Given: *Take it or Leave it* 🔨
  - Personal nitpicks/preferences
  - Sharing a personal viewpoint
- Question ❓
  - Unsure of the intent/why
- Great Job ✅

Denote clearly in your Feedback which category it is in

Code Reviews

Code Review Do's

- Standardize the basics with a guide/checklist
  - Variable quality is a bane to an enterprise codebase
  - Don't review with the premise does this code look like all the other code
- Drive the perception of the Technical Critique of Code
  - As helpful collaboration and not as a punishment/blame
  - Train the skill
- Education on features, paradigms and their use
  - C++/Xxx is evolving rapidly
  - Keys to keeping current of features
    - Conferences / Conference videos
    - Training / Workshops
    - Book/Video clubs

Code Reviews

Code Review Do's

Review the <***expletive***> ***tests***, check for

- Unit/Integration tests
- Happy/unhappy path testing
- Corner cases/Boundary conditions
- Magic numbers
- Poor test naming
- Tests act as usage examples

Image generated by ChatGPT/DALL-E

Code Review don’ts

Have bad pre-conceptions:

Does this code look like all the other code around it?

- Keeps bad habits alive
- Prevents use of new features or evolved thinking
- Legacy codebases suffer greatly from this inertia

Although same *look* and *feel* is *important* (many style guides focus on the formatting)

Consistency can be the enemy of progress, rigidly adhering to a past specific method or approach, while seemingly consistent, can stifle innovation and prevent genuine advancement

Image generated by ChatGPT/DALL-E

Code Review don’ts

Have bad pre-conceptions:

Must use the new everything everywhere?

- Everything looks like a nail
- No moderation

Change for changes sake is a path to nowhere

Image generated by ChatGPT/DALL-E

Code Review Metrics

How to measure code review process *effectiveness*?

Tests per PR

- Are changes being verified with testing
- Unit/Integration Tests should always be included
- Heatmap in an organization where testing is slipping

PRs Created vs Merged

- How many month to month
- Is there a backlog developing?

Image generated by ChatGPT/DALL-E

Code Review Metrics

How to measure code review process *effectiveness*?

Time to Review

- What is the time delay for first reviews
- Set the expectation: 24 hours is a good start

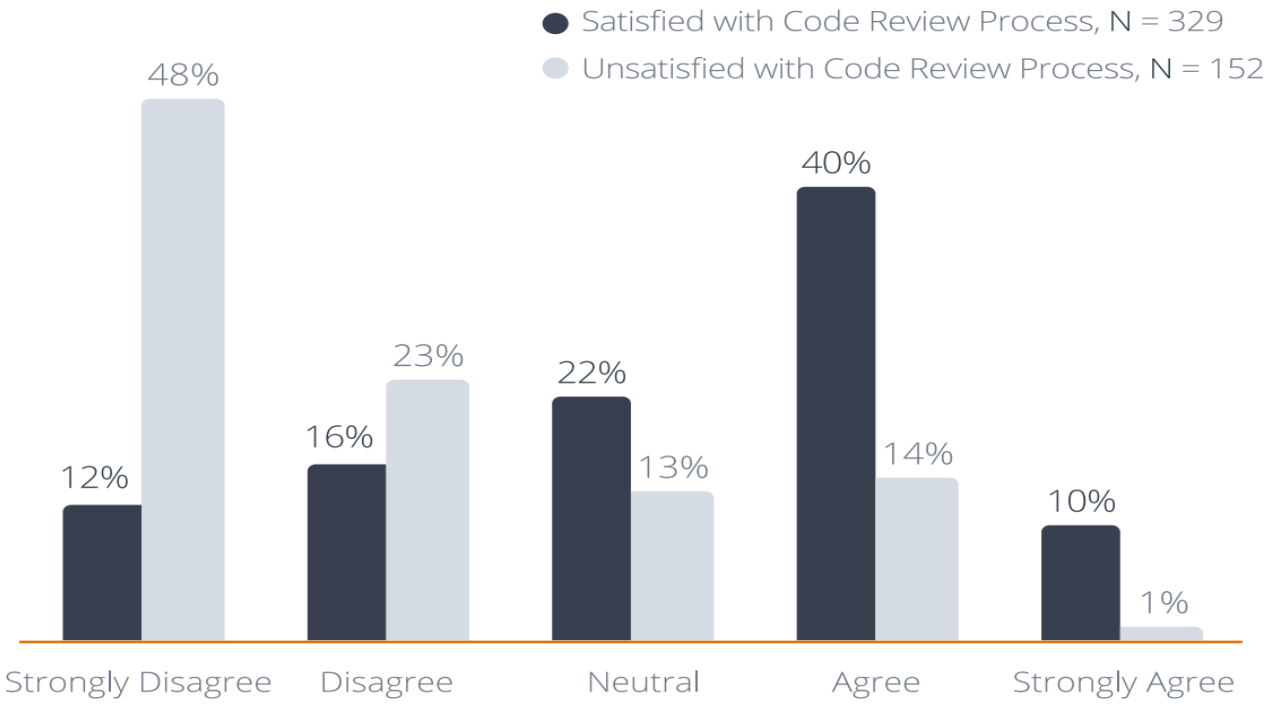
Time to merge

- Total time from PR creation to merging
- Track velocity of reviews

Image generated by ChatGPT/DALL-E

Code Review Metrics

My team regularly pulls reports and metrics on our code review process. *fig.27*



Teams that report on their process are more than ***3X as likely to be satisfied*** with their code reviews

Image from smartbear code review 2020

Change Severity

Change Impact: what is the likely blast radius of the changes

- Functional Change
  - Small / well defined
  - Single right answer
  - Lower risk
- Architectural Change
  - Larger considerations/risk
  - New Interfaces
  - Code restructure/re-engineering
  - Adheres to ADR

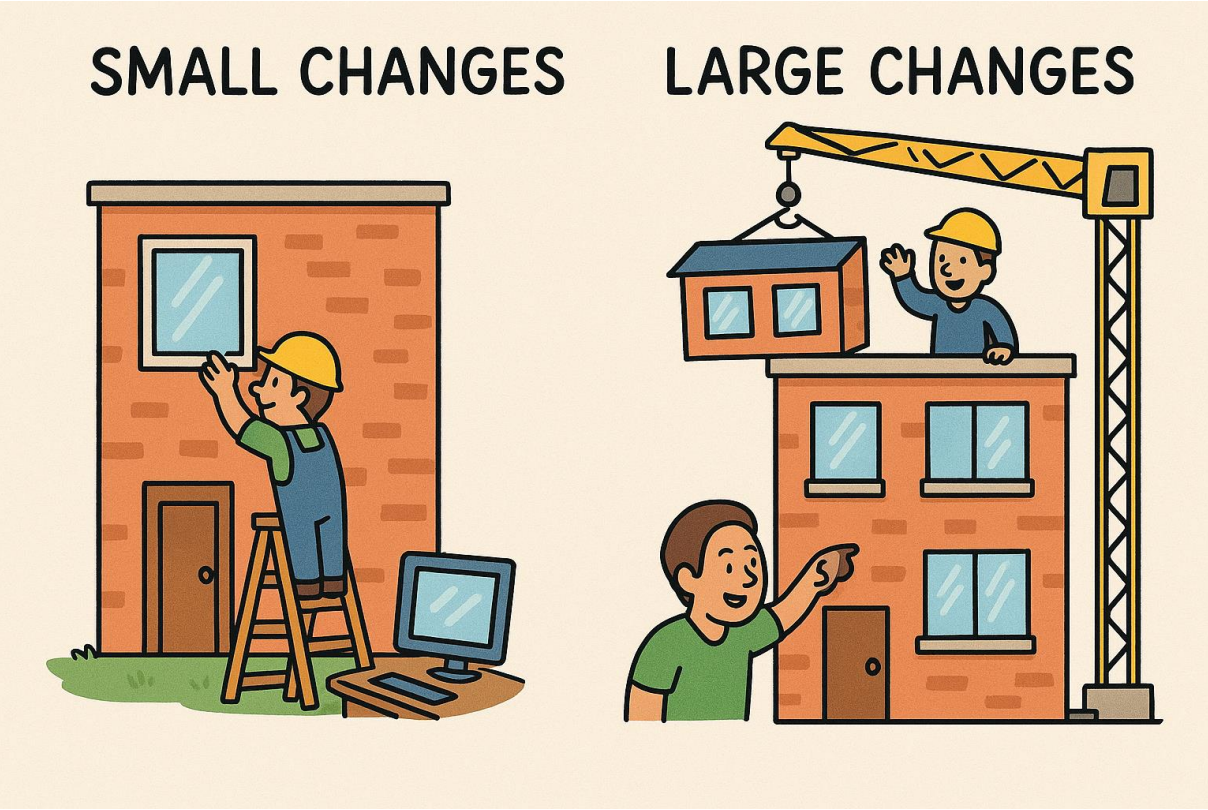


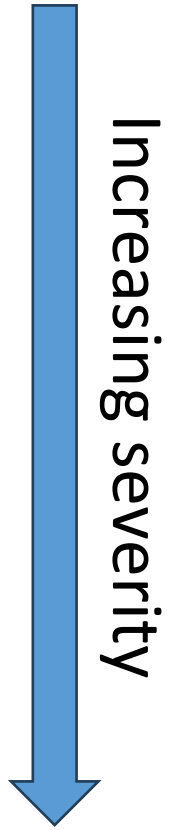
Image generated by ChatGPT/DALL-E

Code Review Basics

Code reviews – Composition aka the who's who

- Peers
- TL
- Historical/Domain expert
- Architectural expert

* More than one reviewer/approver



Code Review LifeCycle

Before the PR is Created

1.Design Review (for significant changes): For major architectural changes or new features, it's helpful to align with the team on the approach. Consider documenting this in an **Architectural Decision Record (ADR)** for future reference.

2.Informal Help (Pairing/Team Discussions): For smaller changes, seek early feedback through pair programming or team discussions. This can help identify issues before diving into the full implementation.

PR Creation

1.Draft PR (Optional): Submit a draft PR to get early feedback, especially for larger or more complex changes. This allows reviewers to help identify significant issues before the final submission.

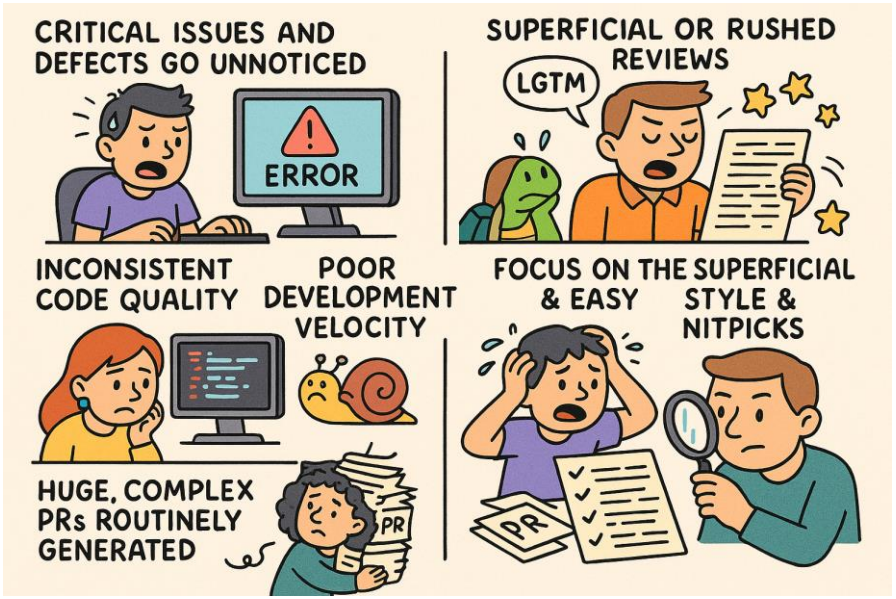
2.Main PR Submission: Submit the main PR for formal review. Make sure it has undergone a self-review using the checklist before requesting feedback.

After the PR is Submitted

1.PR Review Meeting (Optional): If there are contentious issues or complex design decisions, a review meeting can be useful for real-time discussions.

Merge: Once the PR is approved and all feedback has been addressed, the code can be merged into the main branch

Code Review Culture



How to teach/reinforce good practice?

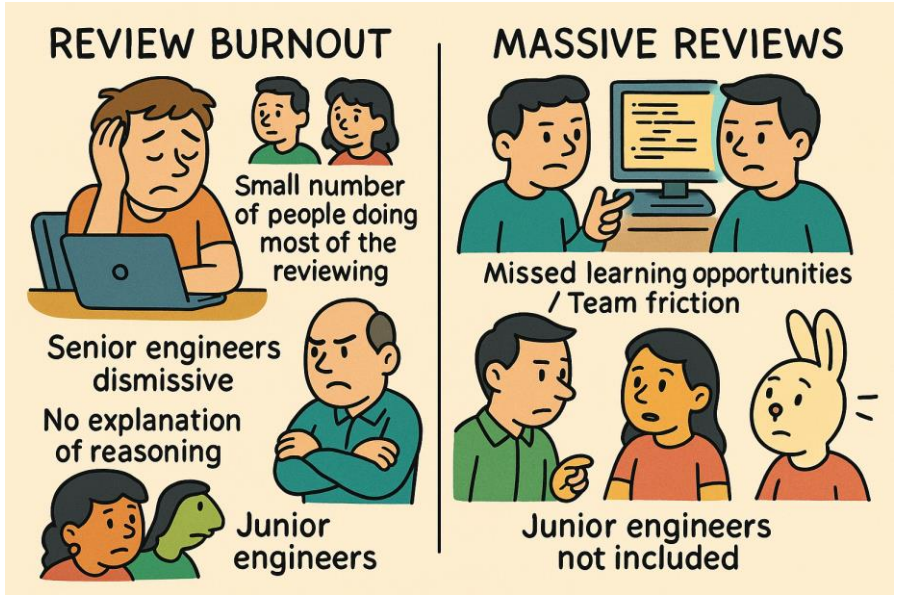


Image generated by ChatGPT/DALL-E

Code Review Culture

How to teach/reinforce good practice?

- Code Review Checklist
  - ❑Standardize review criteria
  - ❑Remove basic contention
  - ❑Add common mistakes seen in PRs
- Code review coaching/pairing
  - ❑Reinforce the basics
  - ❑Create champions
- Team review walkthroughs
  - ❑Guidance where points were missed
  - ❑Alternative perspectives
  - ❑Group buy-in

Code Review Culture

How to teach/reinforce good practice?

- Gauging progress
  - ❑Metrics
  - ❑Spot checking PRs
  - ❑Drift or Evolution?
- Full autonomy and independent evolution
  - ❑Feedback ideas into general review criteria

How to give Constructive Feedback

- **Do:**
  - ❑ Be courteous
  - ❑ Be specific
  - ❑ Ask questions
  - ❑ Explain 'why'
  - ❑ Praise good code
- **Don’t:**
  - ❑ Use a harsh/Judgmental tone
  - ❑ Show a lack of respect
  - ❑ Be vague, confusing or rude
  - ❑ Review the developer / make it personal

D.B.A.D

Image generated by ChatGPT/DALL-E

Common pitfalls

Code review friction points

- Slow (but ultimately rushed) reviews
- Poor feedback
- Large complicated PRs
- Focus is on style not substance
  - Death by a thousand nitpicks
- Reviewer/Reviewee fatigue

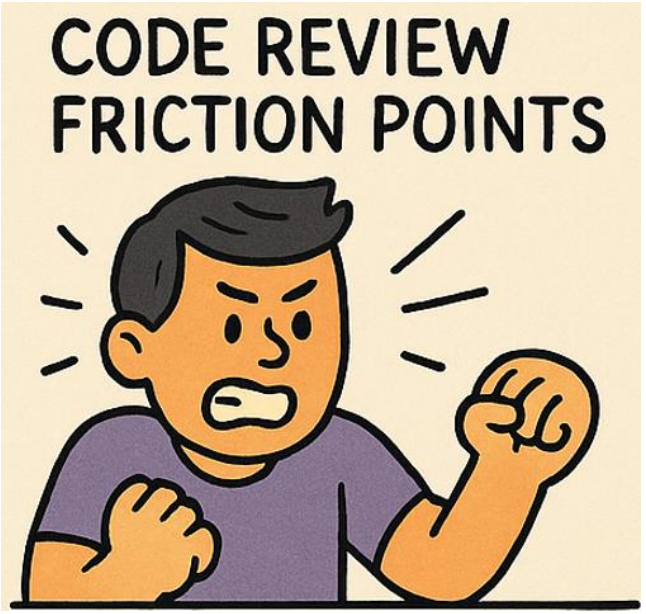


Image generated by ChatGPT/DALL-E

Common pitfalls

Reducing Code review friction points

- Set shared expectations
- Use a Checklist
- Specific & Respectful language
- All team members participate in review process
- Celebrate good PRs and helpful reviews

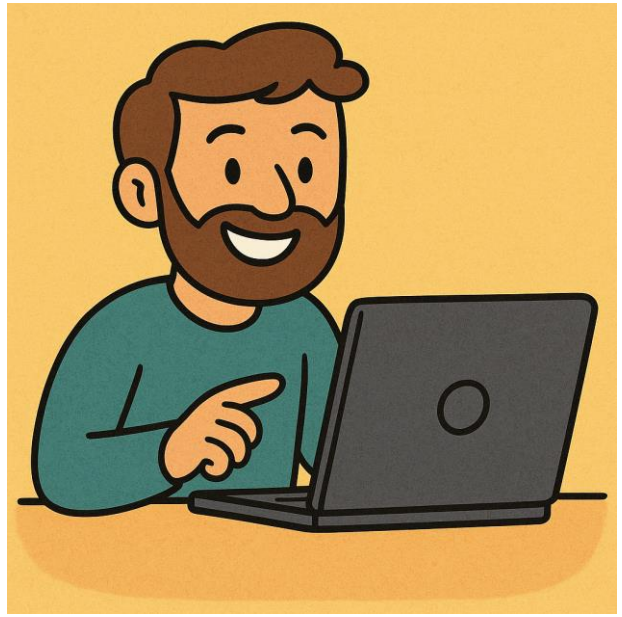


Image generated by ChatGPT/DALL-E

Basic Checklist

What would a Basic Code Review checklist look like?

Something like this: link

General Code Review Checklist

- Meets requirements with no premature optimization/unnecessary complication
- Code review sizing/cognitive load considerations
- Identify the importance/risk of the changes
- Are the changes self-documenting or unduly surprising
- Do the changes adhere to local reasoning
- View the surrounding code for context
- Has sufficient testing been added
- Is functionality being cobbled together
- ~~Does this code look like all the code around it~~

C++ Code Review Checklist

- Look for basic const / override correctness
- Look for magic numbers / strings
- Prefer functions to return an std::optional instead of a bool and an in-out parameter
- Function parameters slimmed down to smallest type
- Return multiple values from a function via a struct
- Ensure related parameters are grouped together in a class or struct
- Initialize structs using designated initializers at call sites
- Do not mix exceptions with return values
- Leverage the use of constexpr, consteval, constinit
- Proper use of ASSERT_*, EXPECT_* in testing

Code Review Checklist vs Coding Standard

Code Review Checklist

- Focused on specific codebase changes
- Accomplishes specific PR goal(s)
- Concise actionable items
- Populated with common mistakes to look for

Example: Ensure adequate Testing including corner cases

Coding Standard

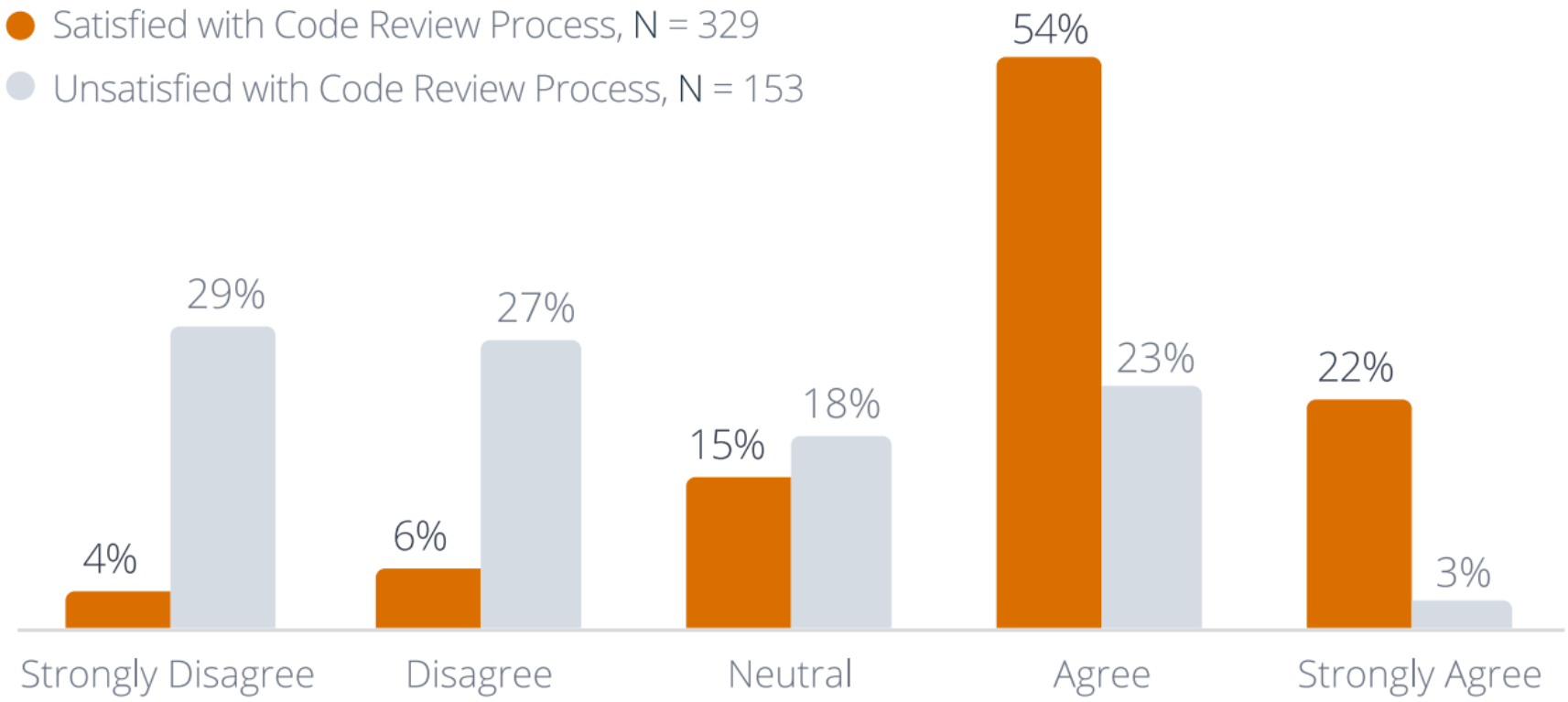
- Focused on the general codebase
- General best practices/patterns
- Comprehensive and detailed explanations and examples
- Populated with all examples of everything

Example: Sample unit testing using GTest framework

Code Review Checklist

My team has guidelines for how reviews should be performed.

fig.26



Teams with guidelines are 2***X as likely to be satisfied*** with their code reviews

Image from smartbear code review 2020

Code Reviews & AI

How to leverage AI?

Why review when you can effect the change pre-review ?

What is AI good at ?

- Simple transformations
- Locally reasonable
- Can be applied with low risk
- Long lists of rules

* Especially for legacy codebases

Code Reviews & AI

Code Review / Clean up

1. For overridden functions, remove the virtual keyword and replace with override
2. Add const to member functions and function parameters where appropriate
3. Add noexcept on simple member functions that only access class member variables
4. Replace any unscoped enums with enum class
5. Replace typedefs with using
6. Replace for/while loops over collections with ranged for
7. In for loops, use structured bindings when accessing tuples/pairs
8. Unused variables removed from function bodies
9. Inline simple member functions from cpp files
10. Empty class constructors/destructors removed or where virtual set to default
11. Remove unused header includes
12. Apply use of C++17-style nested namespace

Code Reviews & AI

Rule 5: Modern C++ - Using Declarations

****MANDATORY CHECK**** - Replace typedef with using for better readability and template support.

// WRONG - Old-style typedef

typedef std::map<std::string, int> StringIntMap; // ❌ BAD

// CORRECT - Modern using declarations

using StringIntMap = std::map<std::string, int>; // ✅ GOOD

Code Reviews & AI

Rule 3: Noexcept Correctness

noexcept Safety Rules

- ✅ **SAFE**: Only accesses member vars (get/set/compare/arithmetic)
- ✅ **SAFE**: Calls only functions marked `noexcept`
- ⚠️ **UNSAFE**: Calls functions without `noexcept`
- ⚠️ **UNSAFE**: Calls functions with unknown exception behavior
- ➢ ⚡ Violating `noexcept` → `std::terminate` — check **all** calls!

When to add `noexcept`

- return member_; // getter
- member_ = value; // setter
- return member_ == value; // compare
- return member_ + 1; // arithmetic
- return obj.fn(); // if fn() is noexcept

Code Reviews & AI

```
class Example {
public:
   // ✅ SAFE - Only accesses member variables
   bool isActive() const noexcept { return active_; }
   int getId() const noexcept { return id_; }
   void setId(int id) noexcept { id_ = id; }

   // ✅ SAFE - Calls noexcept method on member object
   bool isValid() const noexcept { return member_.isValidNoexcept(); }  // member_.isValidNoexcept() is noexcept

   // ❌ UNSAFE - Calls non-noexcept method (or unknown exception behavior)
   bool processData() const noexcept { return member_.process(); }  // member_.process() not declared noexcept
  …
private:
   bool active_;
   int id_;
   SomeClass member_;
};
```

Code Reviews & AI

```
    deferred to future                                    deferred to future
59  -     virtual Status processLockedHeader(         55  +     Status processLockedHeader(
60              const Tickers::Ticks::Handle& tick,     56              const Tickers::Ticks::Handle& tick,
61  -           LazyObject &obj)                        57  +           LazyObject &obj) override;
      BSLS_KEYWORD_OVERRIDE;
```

```
58  -     typedef SideOutput::Side Side;              54  +     using Side = SideOutput::Side;
59  -     typedef SideOutput::SidePair SidePair;      55  +     using SidePair = SideOutput::SidePair;
60  -     typedef SideCollectors::UpdateRules         56  +     using UpdateRules =
        UpdateRules;                                          SideCollectors::UpdateRules;
```

```
205 -         PendingList::iterator it = d_pendingCommits.begin();   217 +         for (auto& header : d_pendingCommits) {
206 -         while (it != d_pendingCommits.end()) {                 218 +             header->commit();
207 -             (*it)->commit();
208 -             ++it;
209           }                                                      219           }
```

Code Reviews & AI

Code Review / Clean up

1. For overridden functions remove the virtual keyword and replace with override
2. Add const to member functions and function parameters where appropriate
3. Add noexcept on simple member functions that only access class member variables
4. Replace any unscoped enums with enum class
5. Replace typedefs with using
6. Replace for/while loops over collections with ranged for
7. In for loops, use structured bindings when accessing tuples/pairs
8. Unused variables removed from function bodies
9. Inline simple member functions from cpp files
10. Empty class constructors/destructors removed or where virtual set to default
11. Remove Unused header includes
12. Apply use of C++17 style nested namespace

Perspective

What are the core objectives of a code review?

The core goals of code reviews is to ensure that code changes are ***correct***, ***maintainable***, and ***fit for purpose***—while supporting ***fast feature delivery*** and maintaining the ***long-term health*** of the codebase

But it should also promote ***knowledge sharing***, ***collaboration*** and a sense of ***shared ownership***

Code Review Culture

How to establish a foundation for a strong code review culture

- Code review checklist: Standardize the basics
- Encourage Self reviews: Catch obvious errors
- Good Metrics: Track in some fashion – at least initially
- Ensure constructive feedback: Positive constructive respectful tone
- Periodic Group Code Review: Walk through a PR
- Normalize “mistakes” as learning opportunities
- Iterate and evolve

In Summary

Code Reviews are about improving code quality and shipping Product

Opportunity for coaching/onboarding junior engineers

Strive for an objective code review based on guidelines/standards

Leads to improved solutions / productivity / group dynamic

Call to Action

Code Reviews are about the *Process*

Its **not** solely about the code

Acknowledge your responsibility as a Reviewer

Fix/Improve the Process

Mastering the Code Review Process

The End

Thank you

Other Engineering talks by yours truly:

Retiring The Singleton Pattern: Concrete Suggestions on What to Use Instead
Redesigning Legacy Systems: Keys to success
Managing External APIs in Enterprise Systems
Exceptionally Bad: The Story on the Misuse of Exceptions and How to Do Better
(**Exceptions in C++:** Better Design Through Analysis of Real-World Usage)
Dependency Injection in C++: A Practical Guide
Software Development Completeness: Knowing when you are done and why it matters

Questions?

Contact: pmuldoon1@Bloomberg.net

Bloomberg is still hiring
https://www.bloomberg.com/careers
https://techatbloomberg.com/cplusplus