C++ now

2025

# Mastering the Code Review Process

*Boosting Code Quality in your Organization*

Peter Muldoon

# Mastering the Code Review Process

## *Boosting Code Quality in your organization*

Bloomberg

Engineering

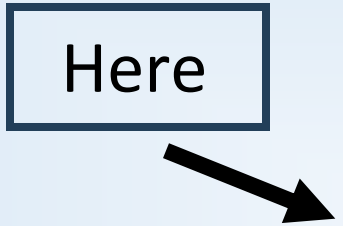
**C++Now 2025**
**May 01, 2025**

**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

# 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 process
- Improving code review culture

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

# Where will we *not* be going?

- The *feelings/emotions* part of technical feedback - while important - is only tangentially touched upon
- 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

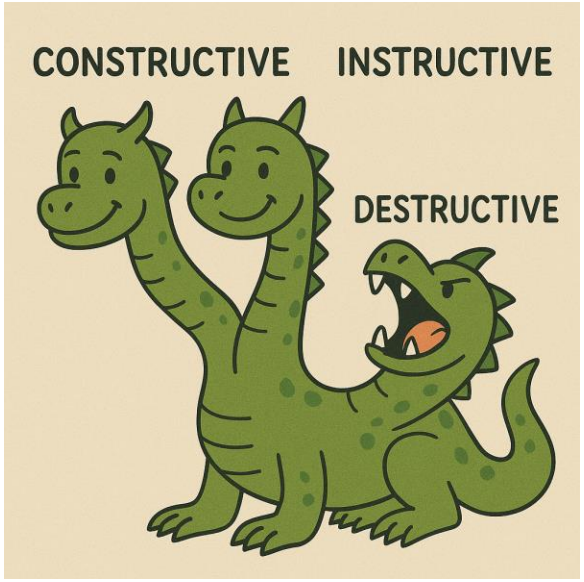
A **critique** is a careful judgment in which you give your opinion about the good and bad parts of something – Merriam-Webster

***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

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

How would your rank the following developer skills?

1. Solution Design
2. Write Code
3. Review Code
4. Drink Coffee


Image generated by ChatGPT/DALL-E

# Perspective

State of Code Report 2020 by SmartBear

**Number 1 way to code quality?**

***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.
Results also indicated that ***Unit Testing*** is the second most important at 20% of responses
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*:


## Standards too rigid

- Doesn’t apply here
- Better done than perfect
- Perfect Is the enemy of good

## Standards too Lax

- “For now”
- Focus on quick wins/short time horizon

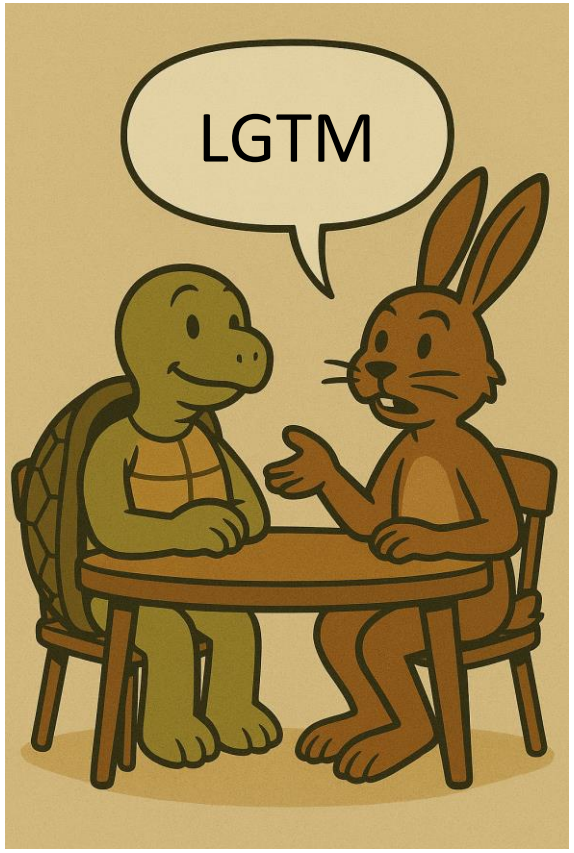


Image generated by ChatGPT/DALL-E

# Code Review Tensions

**Code reviews can have mutually exclusive directives:**

Contending viewpoints on *Timing*:




## Code reviews rushed

- Don’t have time for proper review but LGTM
- Normally your stuff is ok, soooo……
- Afraid to challenge or admit I’m lost

## Code reviews tardy

- Prioritize writing code over reviewing
- Your PRs are unpleasant !
- nitpicking



Image generated by ChatGPT/DALL-E

# Code Review Tensions

**Code reviews can have mutually exclusive directives:**

Contending viewpoints on *scope*:

## Minimalist

- Address one thing and one thing only
- Short PR
- Missed opportunities

## Expansive

- Leave the code in a better state than you found it
- Refactor as needed
- Just one more thing

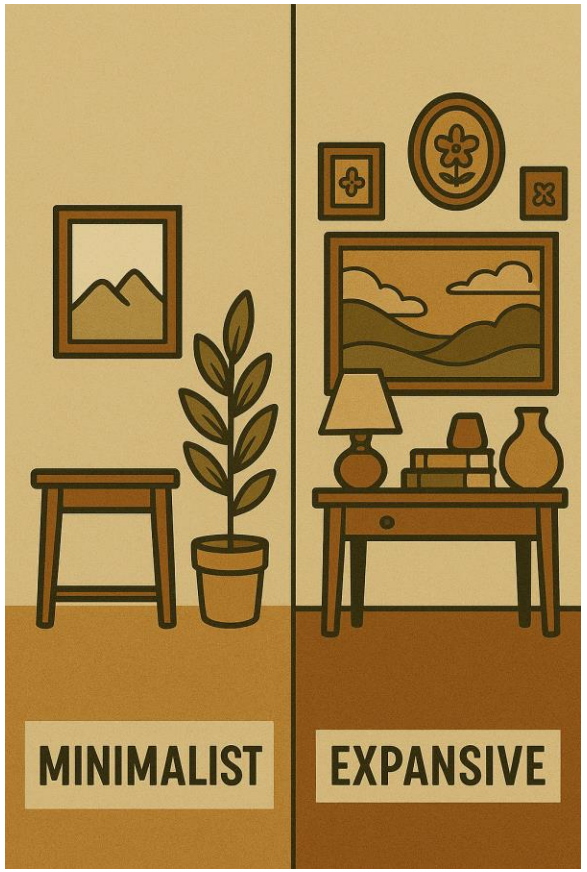



Image generated by ChatGPT/DALL-E

# Code Review Tensions

**Code reviews can have mutually exclusive directives:**

Contending viewpoints on *new feature use*:

Conservative

- Keep everything consistent

Modern

- Use every new feature you can cram in




Image generated by ChatGPT/DALL-E

# Code Review Tensions

**Code reviews can have mutually exclusive directives:**

Contending viewpoints on :

- Standards
- Timing
- Scope
- New feature use

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

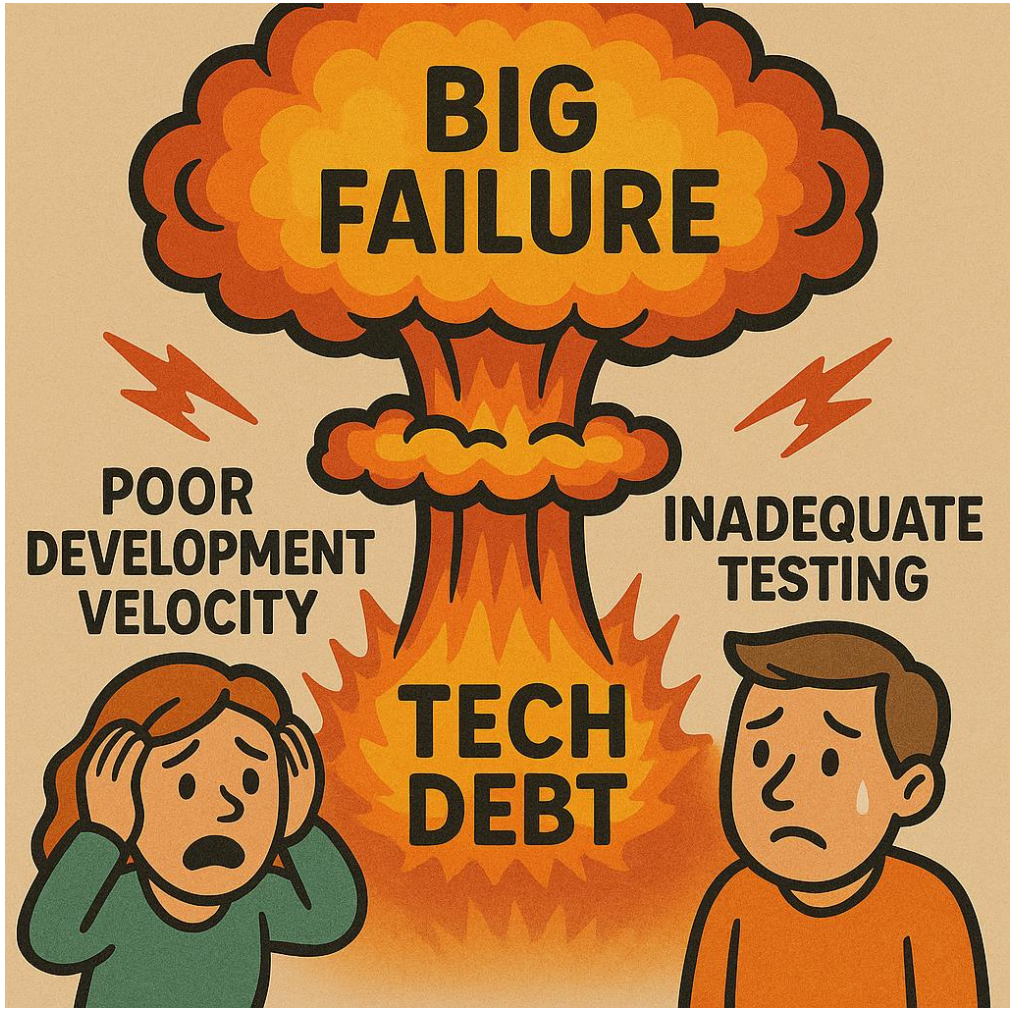



Image generated by ChatGPT/DALL-E

# Poor Code Reviews

## What are the consequences?

- Review burnout
  - Small number of people doing most of the reviewing
  - Massive reviews
- Missed learning opportunities / Team Friction
  - Senior engineers dismissive
  - No explanation of reasoning
  - Junior engineers not included


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

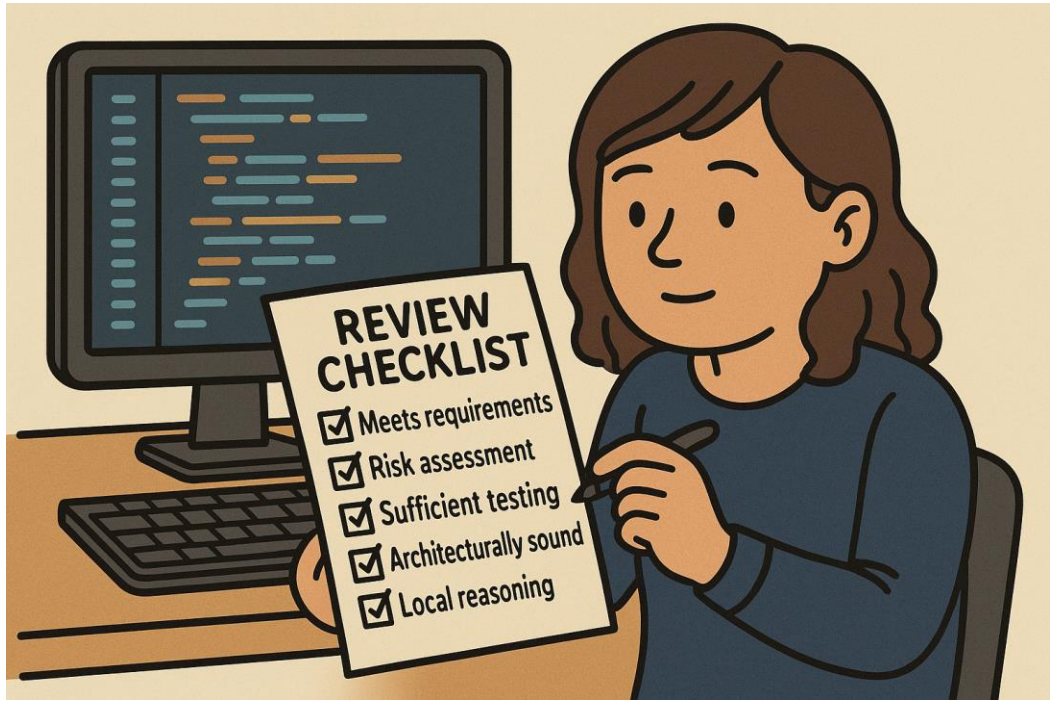





Image generated by ChatGPT/DALL-E

# Code Review Preparation

## Before you ask for a review

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 diff
- Why – the reason or background for the change
  - *Probably not* deducible from 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

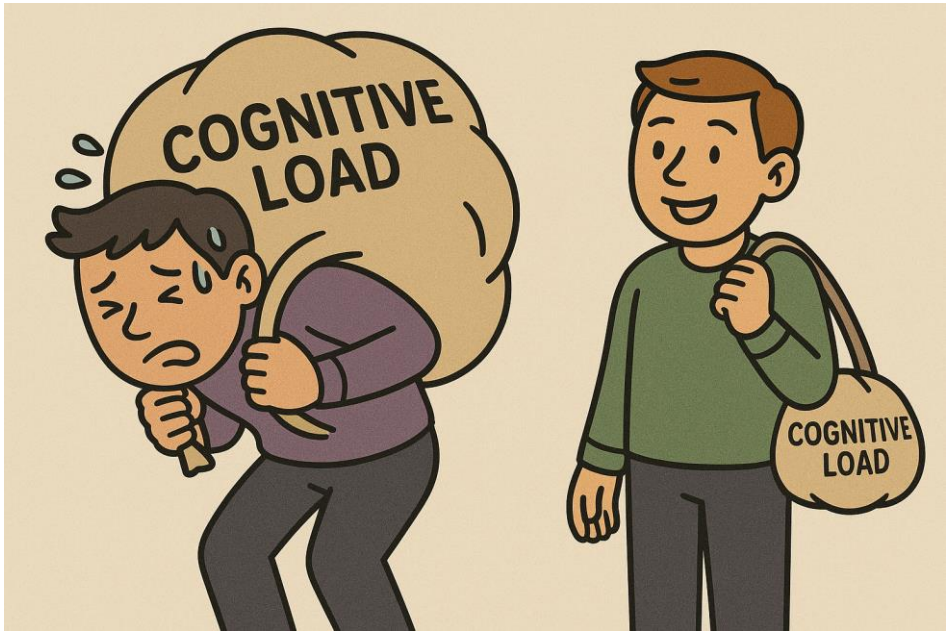





Image generated by ChatGPT/DALL-E

# Code (P)Reviews

**Draft PRs:**

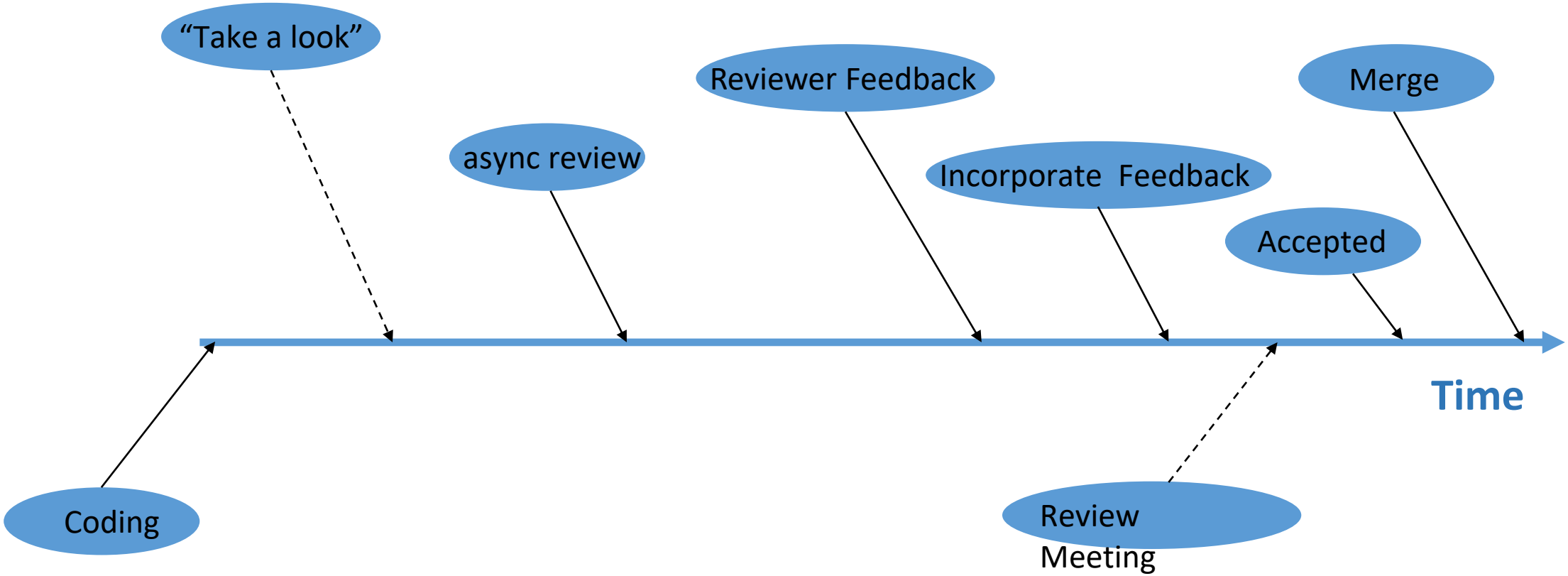
Usually for POCs, aims to discover

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

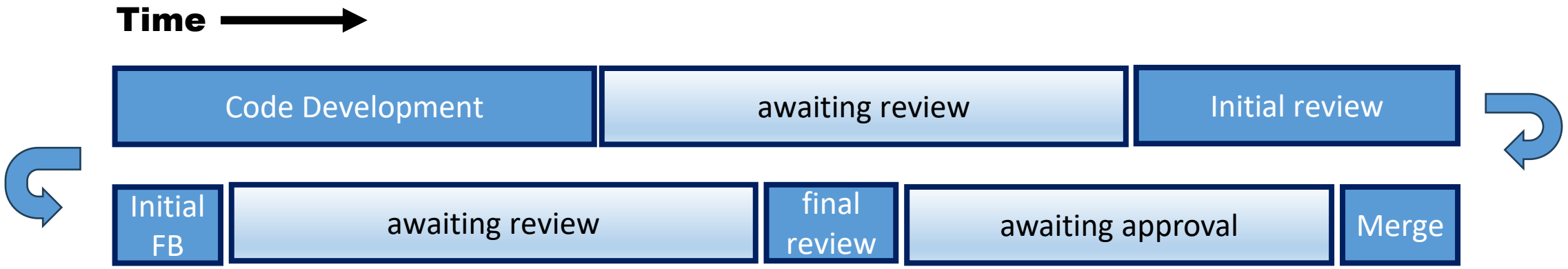



Image generated by ChatGPT/DALL-E

# Lifecycle of a Code Review



# 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 worse the codebase where you are working, the smaller the change should be
  - Smaller meaning a single aspect
- Spacing/indentation/renaming should mostly be left to their own PR
  - Ideally automated using tooling

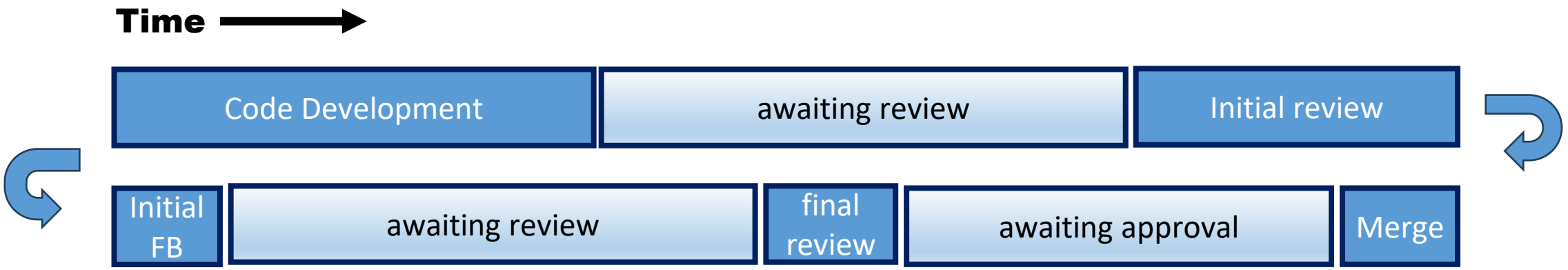
*Note: Poorly understood changes have more risk due to poorer feedback

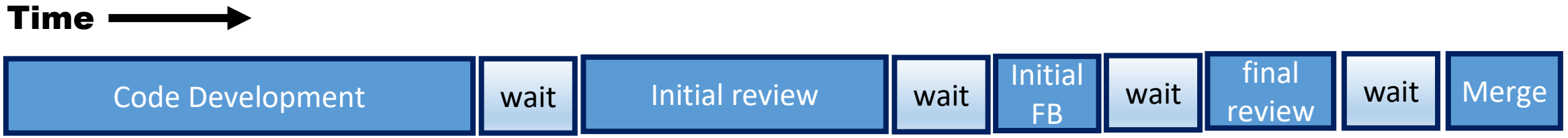
# Code Review Timing

**Timing is everything:**

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

# Code Review Prioritization



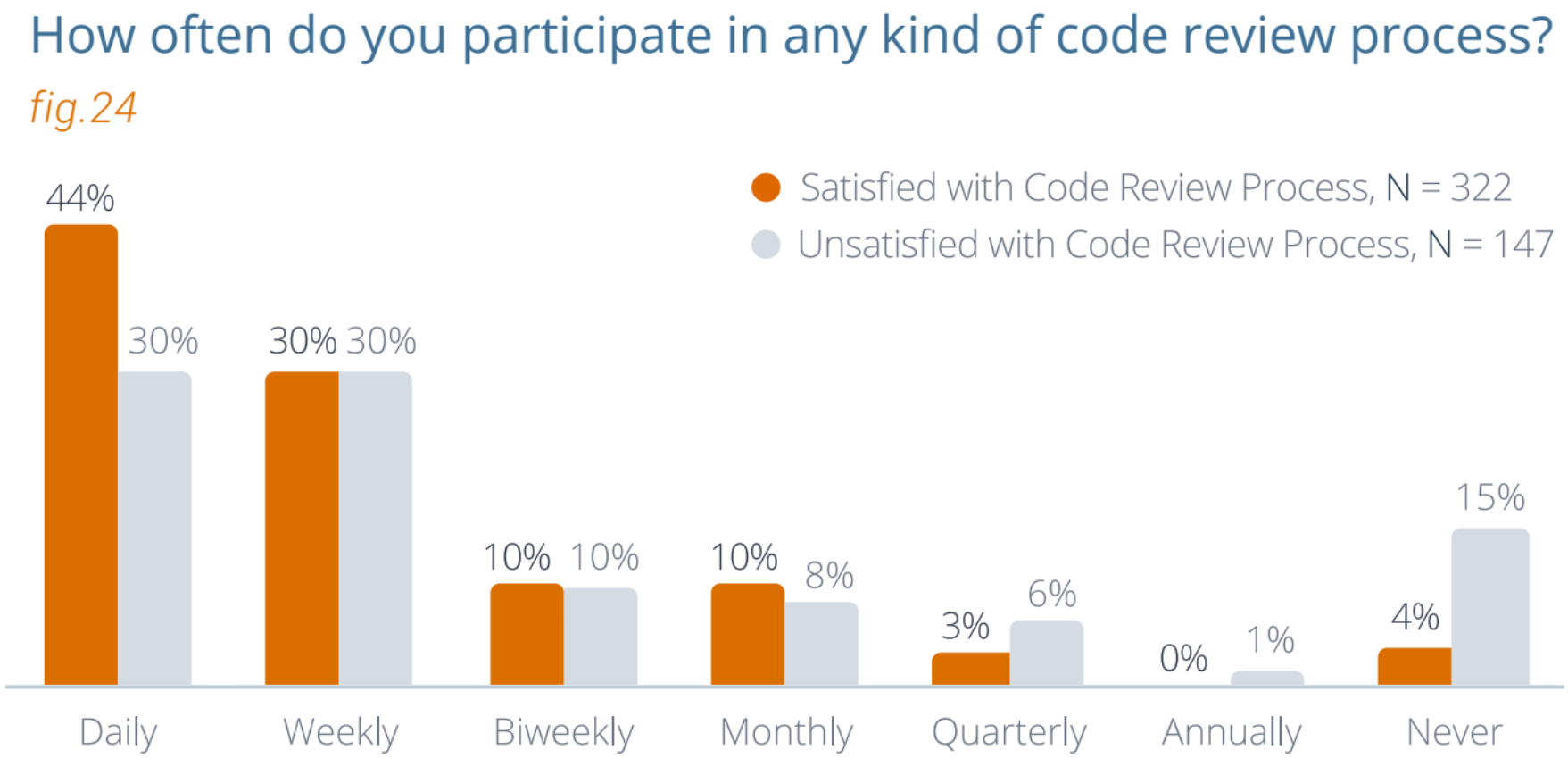
# Code Review Timing

**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 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



Image from smartbear code review 2020

# 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
- 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* and 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


Lines per PR

- Smaller PRs are easier and faster to review
- Keep PRs under 400–500 lines where possible



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

## Reviews per person

- Fair and Balanced workload across the team
- Identify bottlenecks
- Avoid burnout and promote shared ownership




Image generated by ChatGPT/DALL-E

# Code Review Metrics

**How to measure code review process *effectiveness*?**

Outstanding PRs

- How many

Avg/Median age per merged PR

- How long in the tooth
- Auto close of aged PRs

Number of PRs merged per sprint

- Probably dangerous

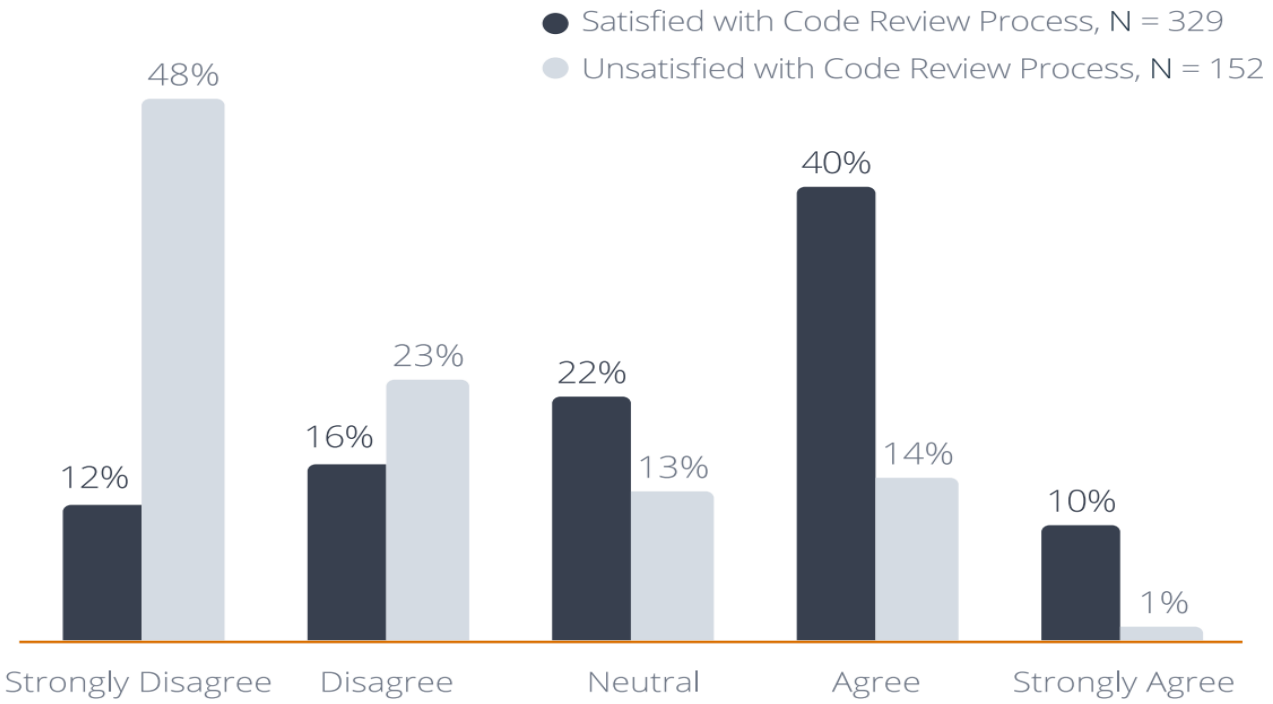

*Beware of the system being gamed/process being corrupted*



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

# 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

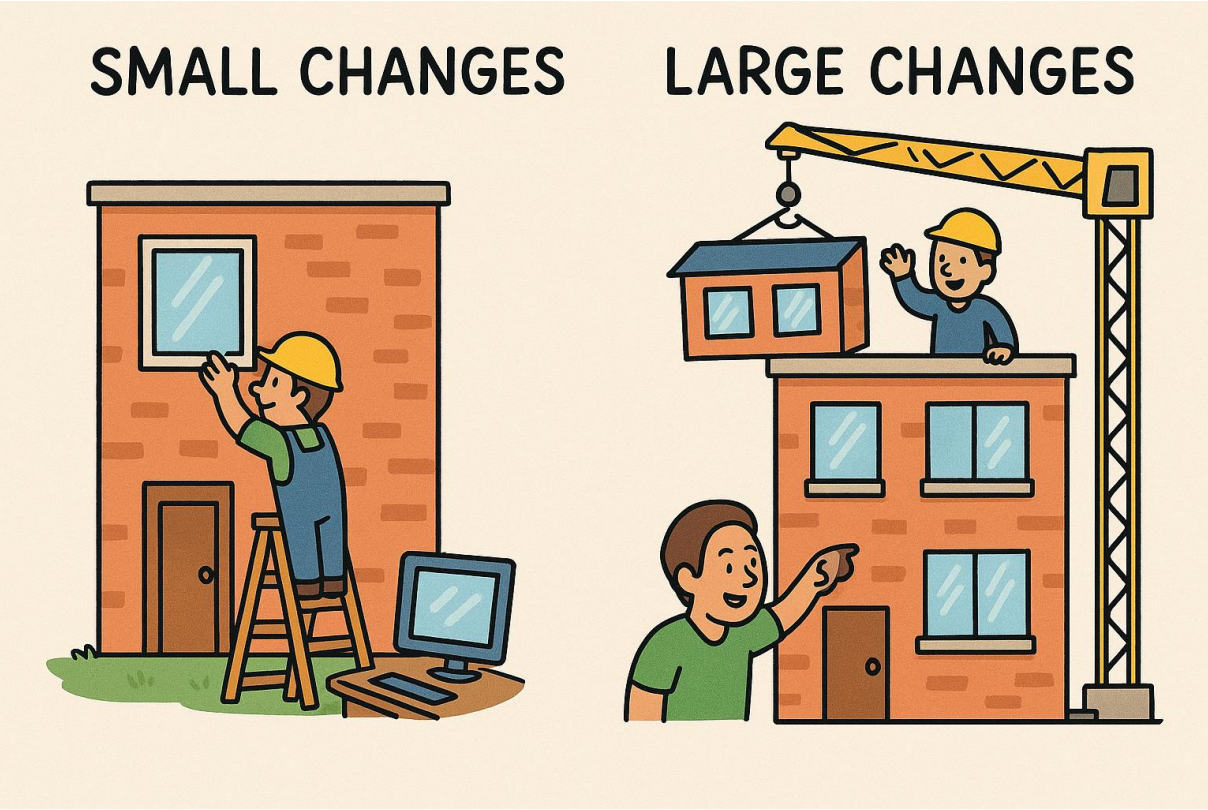




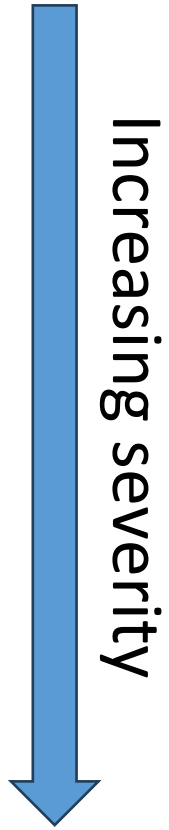

Image generated by ChatGPT/DALL-E

# Code Review Basics

**Code reviews – Composition aka the who's who**

- Peers
- TLs
- Historical/Domain expert
- Architectural expert

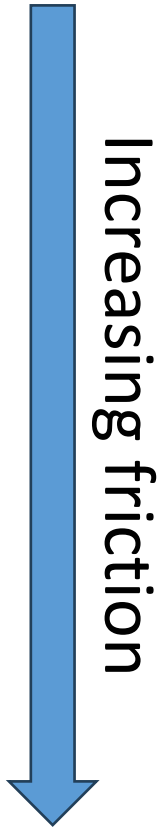
* More than one reviewer/approver



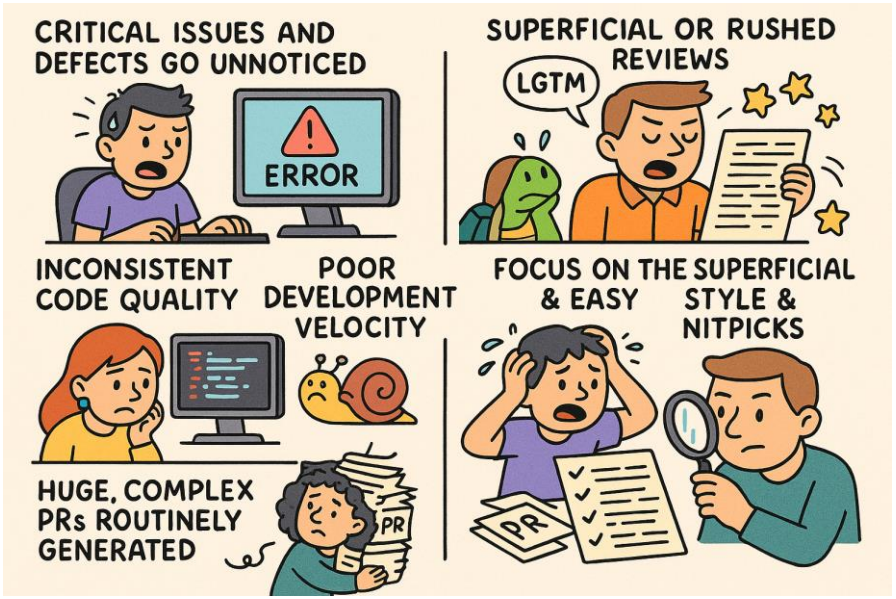
# Code Review Basics

## Stages of a code review

- Informal chats
  - Call a peer over
- Offline PR review
  - Standard async written review
- 2 person meet-up
  - Review and reviewer work things out
- General Code review or tie-breaker meeting
  - Know when this is needed
    - Excessive back and forth on a PR
  - Personality conflict
  - Old vs New thinking



# Code Review Culture




**How to teach/reinforce good practice?**

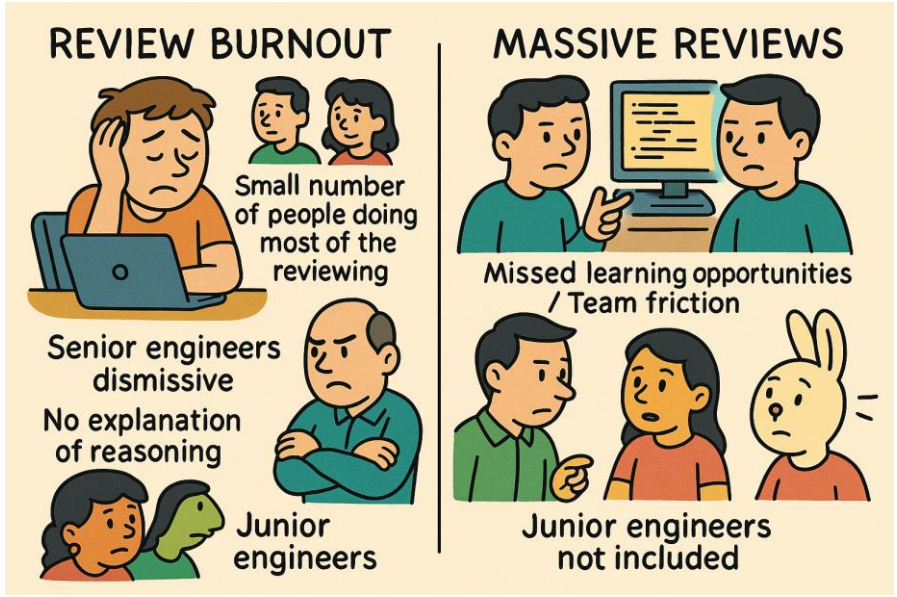





Image generated by ChatGPT/DALL-E

# 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 specific and kind
  - ❑ 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 or rushed reviews
- Poor feedback
- Large complicated PRs
- Focus is on style not substance
- Death by a thousand nitpicks
- Reviewer/Reviewee fatigue

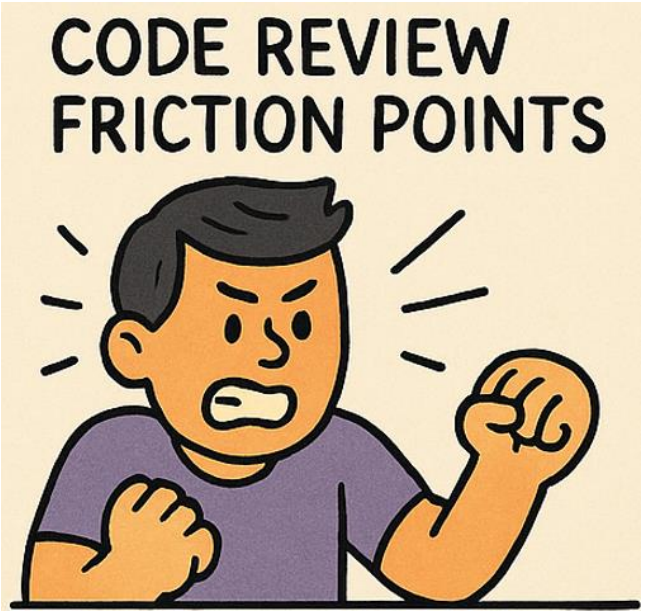



Image generated by ChatGPT/DALL-E

# 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

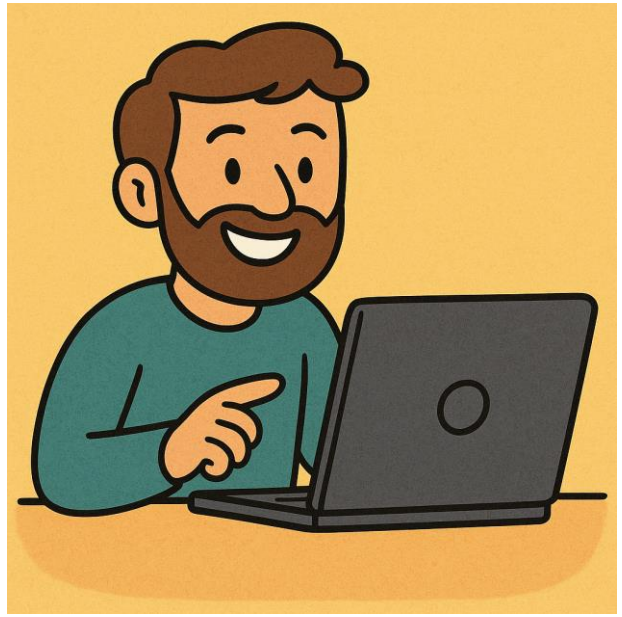




Image generated by ChatGPT/DALL-E

# Common pitfalls

**Argument progression against change**

- Logical rationale, Open mind
  - Present case and reach a compromise
- More extreme corner cases
  - Justify current stance on all regular cases by pointing to extreme cases
- Change focus to negative cost of Change
  - Too hard/time consuming
  - Disruptive
  - Theoretically non-performant
- Disengagement
  - We will come back sometime later after considering it

# 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

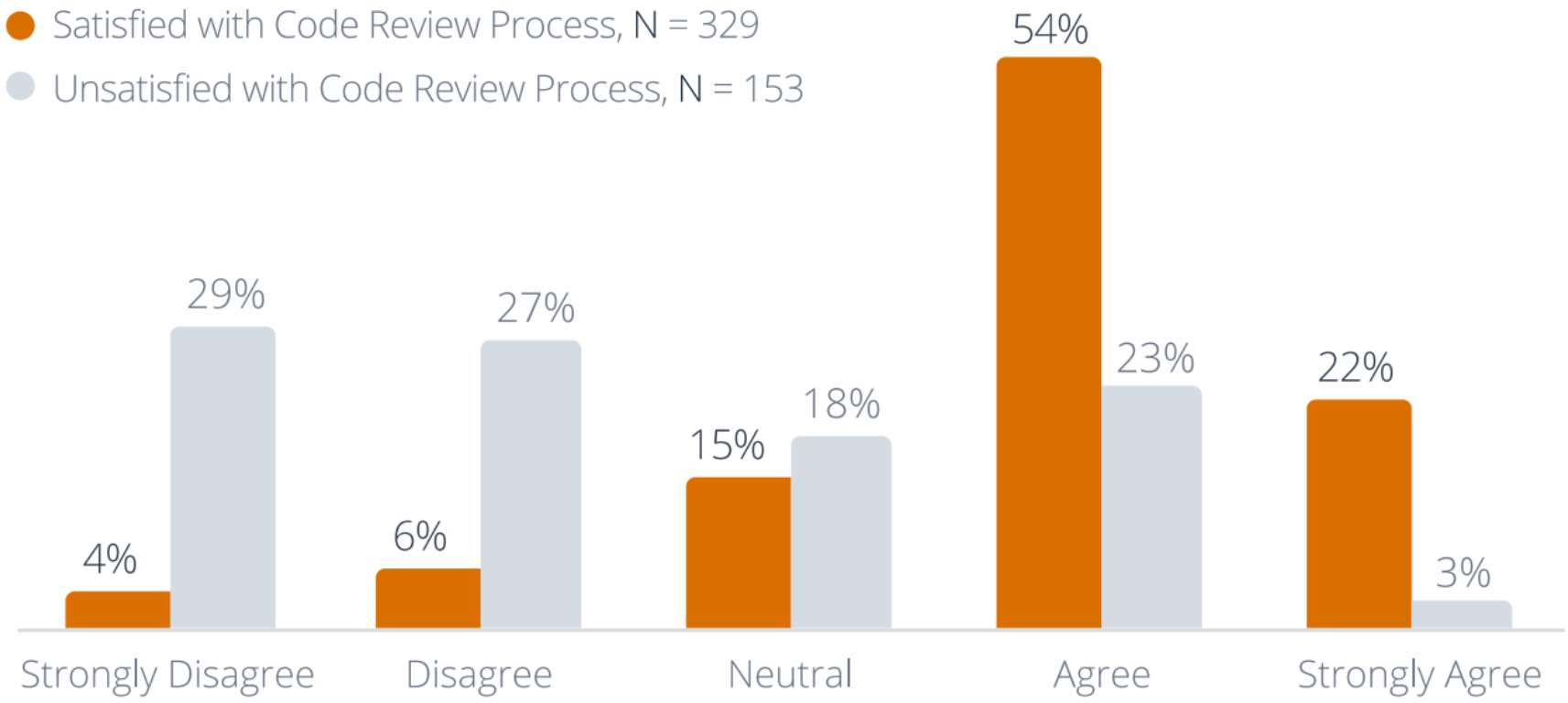
- Focused on the general codebase
- General best practices/patterns
- Comprehensive and detailed explanations and examples
- Populated with all examples of everything

Example: Sample unit testing using GTest framework

# Code Review Checklist

## My team has guidelines for how reviews should be performed.

*fig.26*



# 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: including pregame checks
- Good Metrics: Track in some fashion – at least initially
- Ensure constructive feedback: Positive respectful tone
- Periodic Group Code Review: Walk through sample PRs
- Normalize “mistakes” as learning opportunities
- Iterate and evolve




Image generated by ChatGPT/DALL-E

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