
THE LOST
ART OF
CODE
REVIEWS

The CodeMash "Law of Two Feet"

If a session just isn't doing it for you, don't feel obligated to stay.

PDF of slides


CODE REVIEW BISTRO - DAILY SPECIALS
1. APPETIZER: Why do we perform code reviews?
2. SOUP DU JOUR: What a code review is not
3. SALAD: Preparing for a code review
4. MAIN COURSE: Code review basics
5. SPICY DISH: Reviewing code for security
6. CHEF'S SPECIAL: Reviewing REST APIs
7. FROM THE CELLAR: Reviewing database code
8. FAST FOOD: Reviewing code for performance
9. SIDE DISH: Reviewing unit tests
10. DESSERT: The human side of code reviews
11. COFFEE & FUTURE: Reviewing AI-written code

Kelly Morrison, Ph.D. Computer Engineering

The University of Georgia

TEAM Daugherty Est. 1985

CNN

THE HOME DEPOT

COLORBUS

CGI

AUBURN UNIVERSITY

Turner

DIGITAL output MAGAZINE

PATHFIRE Enabling Digital Media

Edward Jones MAKING SENSE OF INVESTING

THE CLIMATE CORPORATION

NASA Marshall Space Flight Center

Performance MATTERS

BAYER

Poll

How many of you do code reviews?

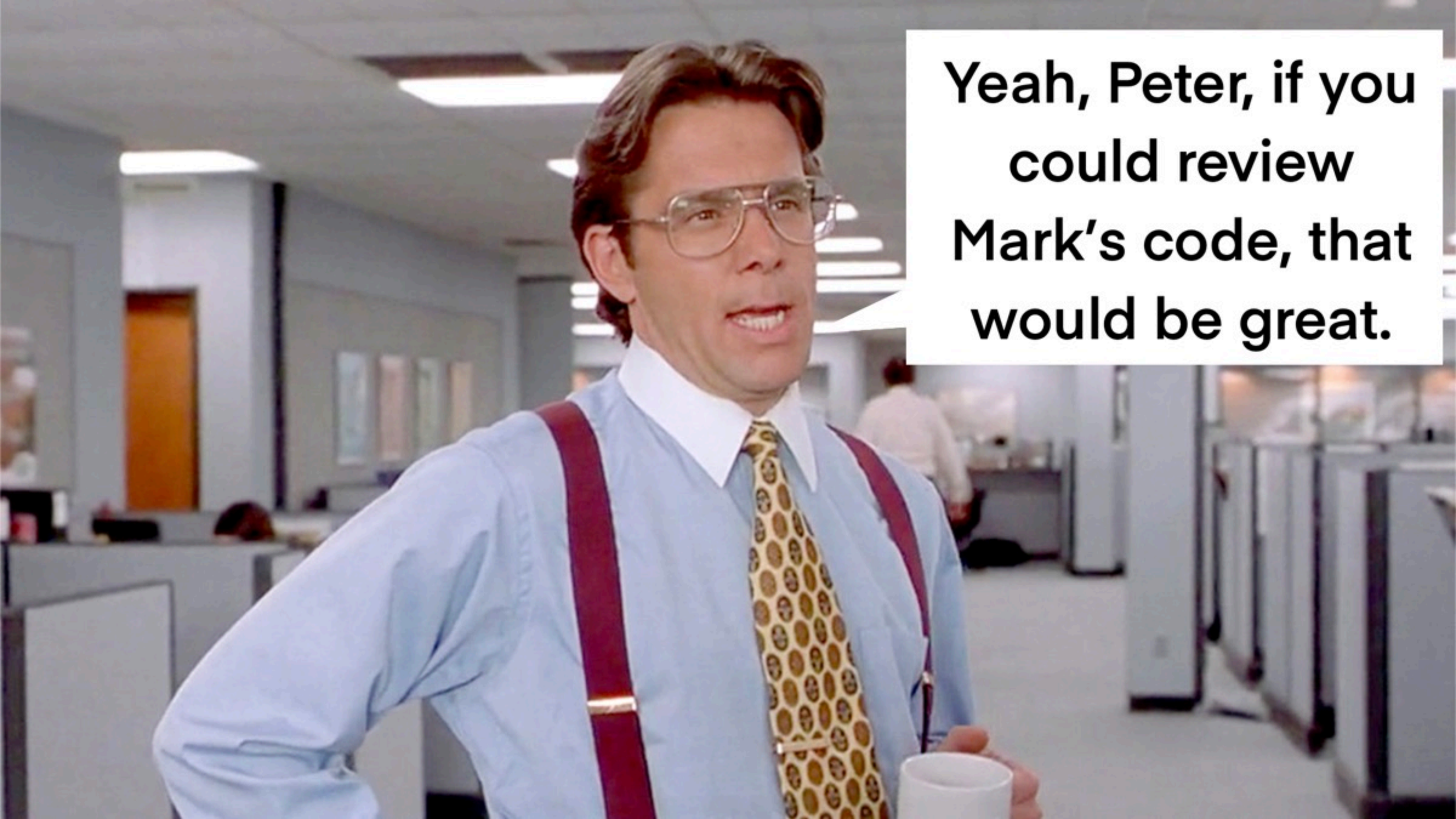
Poll

Were you ever ***taught*** to do a code review?

Poll

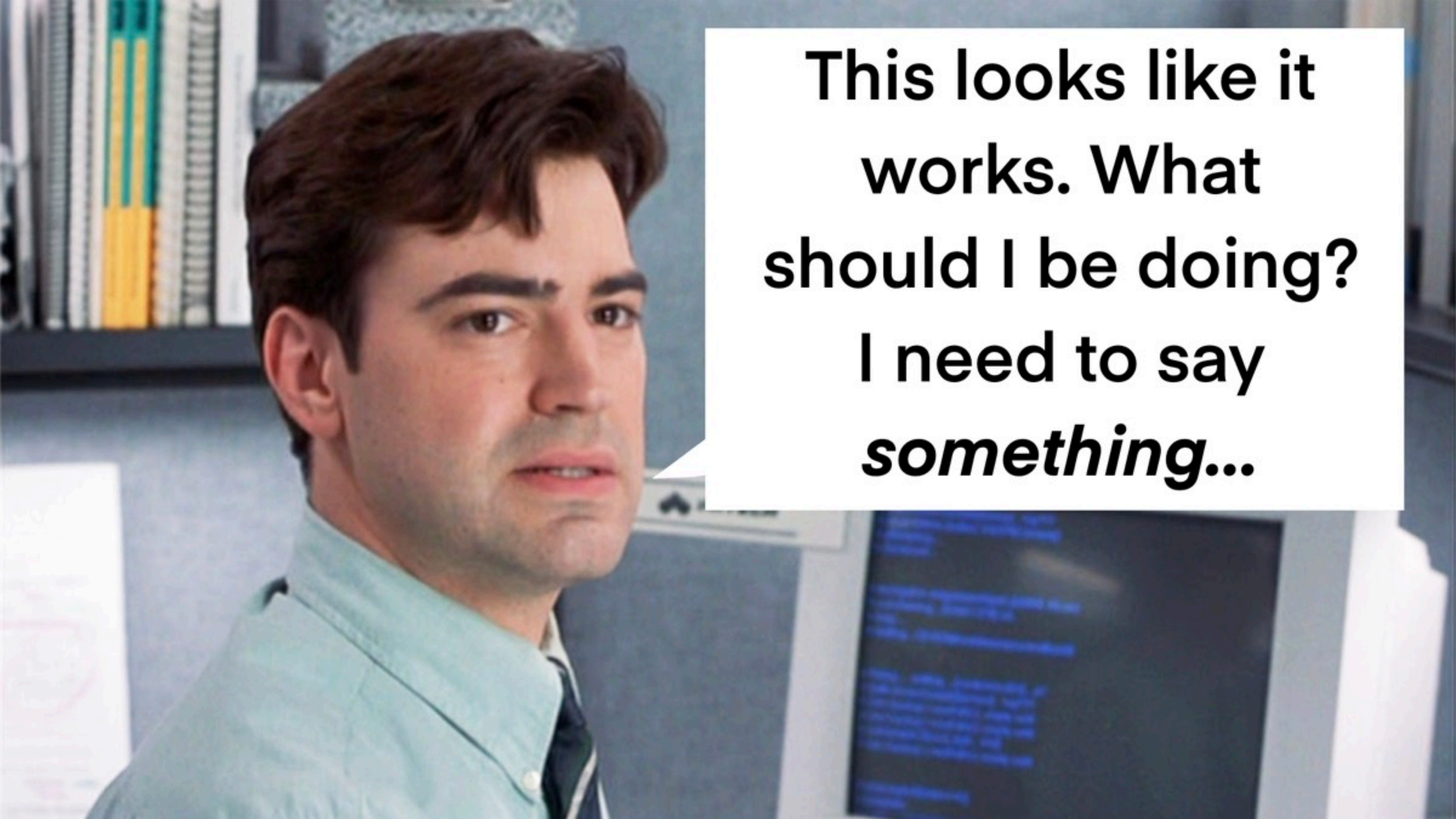
Does your company have a standard for code reviews?

Typical scenario


Yeah, Peter, if you could review Mark's code, that would be great.


Sure

```
for (Order order : orders) {
    if (order.isActive()) {
        for (LineItem item : order.getItems()) {
            if (item.getQuantity() > 0) {
                Product p = catalog.get(item.getProductId());
                if (p != null && p.isSellable()) {
                    int total = item.getQuantity() * p.getUnitPrice();
                    if (total > 1000) {
                        counts.merge(p.getCategory(), 1, Integer::sum);
                    } else if (order.isPriority()) {
                        counts.merge("PRIORITY_SMALL", 1, Integer::sum);
                    }
                }
            }
        }
    }
}
```


This looks like it works. What should I be doing? I need to say *something*...

```
for (Order order : orders) {
    if (order.isActive()) {
        for (LineItem item : order.getItems()) {
            if (item.getQuantity() > 0) {
                Product p = catalog.get(item.getProductId());
                if (p != null && p.isSellable()) {
                    int total = item.getQuantity() * p.getUnitPrice();
                    if (total > 1000) {
                        counts.merge(p.getCategory(), 1, Integer::sum);
                    } else if (order.isPriority()) {
                        counts.merge("PRIORITY_SMALL", 1, Integer::sum);
                    }
                }
            }
        }
    }
}
```

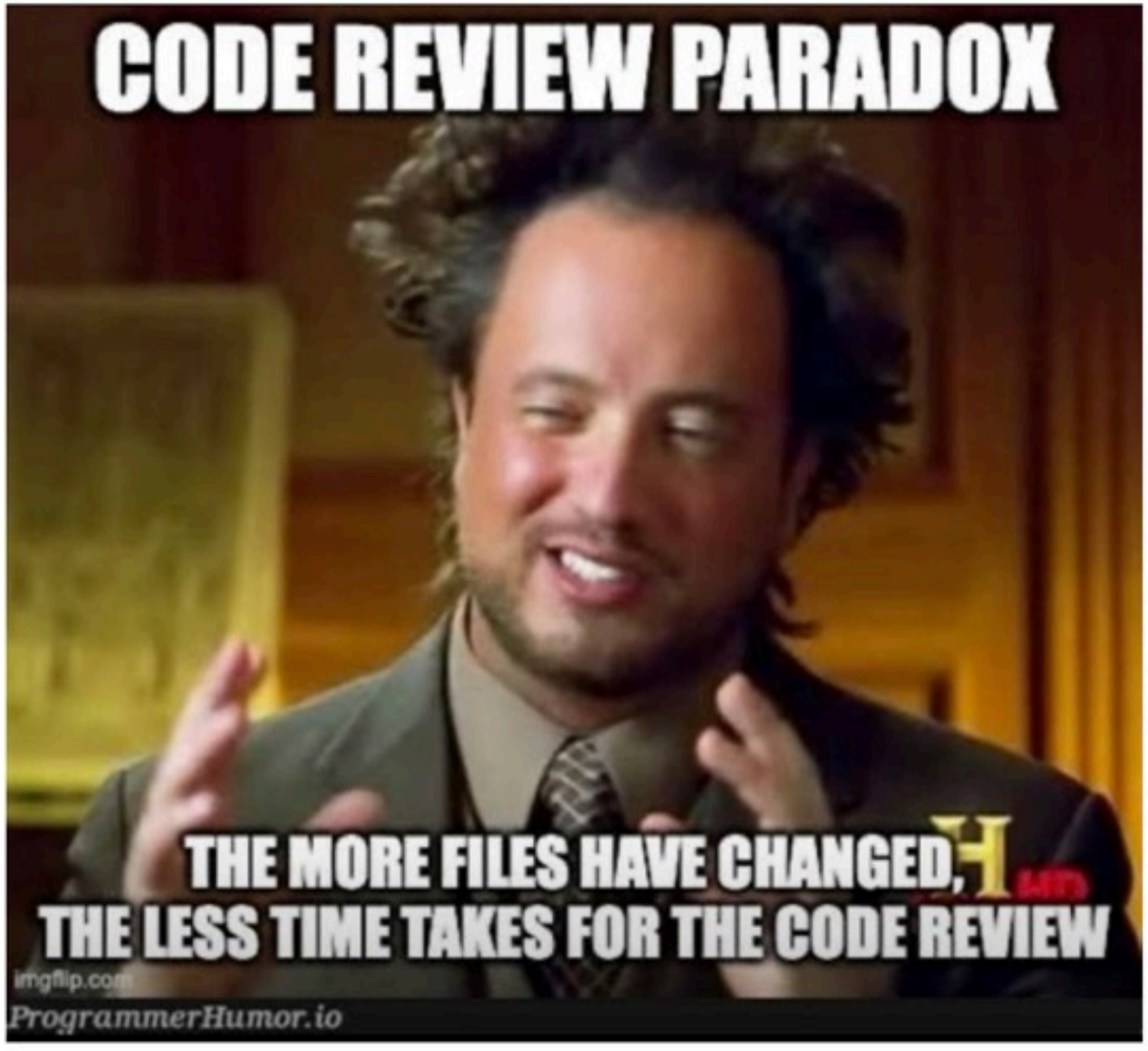
Peter

Maybe change p to prod?


Done


"Code Review"
Me

Another scenario



I Am Devloper
@iamdevloper

10 lines of code = 10 issues.

500 lines of code = "looks fine."

Code reviews.

11:58 AM · Nov 5, 2013 · Tweetbot for iOS

8,294 Retweets **168** Quote Tweets **6,773** Likes


"Code Review"
Me

These are not
“code reviews.”

They are simply
“going through
the motions.”

Why do we perform code reviews?

Stability / correctness / security

Long term maintainability

Consistency

Knowledge transfer

Stability / Correctness / Security

November 18, 2025
Cloudflare Outage

Affected:

X, ChatGPT, Spotify, Canva, email, project management software, CRM software, and more

```rust
/// Fetch edge features based on `input` struct into [`Feat
pub fn fetch_features(
    &mut self,
    input: &dyn BotsInput,
    features: &mut Features,
) -> Result<(), (ErrorFlags, i32)> {
    // update features checksum (lower 32 bits) and copy ed
    features.checksum &= 0xFFFF_FFFF_0000_0000;
    features.checksum |= u64::from(self.config.checksum);
    let (feature_values, _) = features
        .append_with_names(&self.config.feature_names)
        .unwrap();
```

January 2026

Claude Code token highjacking

Ed Andersen

@edandersen · **Follow**

So the reason Claude Code was so popular is because people were hijacking the desktop app's auth token and using it to call the API from other tools for 1/20th the cost of regular API calls?

lollllllll

8:15 AM · Jan 9, 2026

6K **Reply** **Copy link**

Long term maintainability

```python
def f(x, y):
    if x == 1:
        return y * 1.0
    elif x == 2:
        return y * 0.9
    elif x == 3:
        return y * 0.8
    else:
        return y
```

```python
from enum import Enum

class CustomerType(Enum):
    REGULAR = "regular"
    MEMBER = "member"
    VIP = "vip"

DISCOUNT_RATE = {
    CustomerType.REGULAR: 0.00,
    CustomerType.MEMBER: 0.10,
    CustomerType.VIP: 0.20,
}

def calculate_price(customer_type, base_price):
    discount = DISCOUNT_RATE.get(customer_type, 0.0)
    return base_price * (1 - discount)
```

Consistency

```
// Logging style 1 (print)
def process(x):
    print(f"Processing {x}")
    return x * 2
```

```
// Logging style 2 (logger)
import logging

logging.basicConfig(level=logging.INFO)

def process(x):
    logging.info("Processing %s", x)
    return x * 2
```

```
// Logging style 3 (with context)
import logging
from contextvars import ContextVar

request_id = ContextVar("request_id")

def process(x):
    logging.info("id=%s x=%s", request_id.get(), x)
    return x * 2
```

Knowledge Transfer

```python
def allocate(capacity, requests):
    used = 0
    result = []
    for r in requests:
        if used + r <= capacity:
            used += r
            result.append(r)
    return result
```

```python
def allocate(capacity, requests):
    """
    Allocate capacity using a greedy, first-come-first-served strategy.

    Business rules:
    - Requests are evaluated in the order received.
    - Requests are either fully accepted or fully rejected (no partials).
    - This is intentional: fairness > optimal utilization.

    Example:
        capacity = 10
        requests = [4, 7, 3]
        result = [4, 3]

    Why not optimize?
    - A knapsack solution would increase utilization,
      but violates fairness guarantees promised to customers.
    """
    used = 0
    result = []

    for r in requests:
        if used + r <= capacity:
            used += r
            result.append(r)

    return result
```

What A Code Review Is *Not*

NOT… a personal judgement

NOT… a “design by committee” exercise

NOT… a replacement for testing

NOT… a place to re-litigate agreed-upon standards

Preparing for a Code Review

Pull the code and run the unit tests

```
# Check out the PR branch
$ git checkout pradeep-pr
Branch 'pradeep-pr' set up to track remote branch 'pradeep-pr' from 'origin'.
Switched to a new branch 'pradeep-pr'

# (Optional) Create and activate a virtual environment
$ python -m venv .venv
$ source .venv/bin/activate

# Install dependencies
$ pip install -r requirements.txt

# Run pytest
$ pytest
============================= test session starts ==============================
platform linux -- Python 3.11.6, pytest-7.4.4
collected 42 items

tests/test_api.py ..............
tests/test_models.py ...........
tests/test_utils.py ............

============================== 42 passed in 1.23s ==============================
```

Run linters and static code analysis on the new or changed code in the branch

Examples:

SonarQube

Ruff

SpotBugs

PMD

Read the JIRA ticket

Story: Add GET `/users/{id}` REST endpoint

Description

Add a REST API endpoint to retrieve a user by ID. The endpoint must be secured using OAuth2 authentication and return a JSON representation of the user. OpenAPI annotations should be included so the endpoint appears correctly in the Swagger UI.

Requirements

- Endpoint: `GET /users/{id}`
- Authentication: OAuth2 (Bearer token)
- Retrieves a user by ID from the repository
- Returns a JSON response
- Returns appropriate HTTP status codes
- Includes OpenAPI annotations for Swagger documentation

Acceptance Criteria

- Endpoint requires a valid OAuth2 access token
- When a user exists:
  - Returns `200 OK`
  - Returns a JSON object representing the user
- When a user does not exist:
  - Returns `404 Not Found`
- When the request is unauthorized:
  - Returns `401 Unauthorized`
- OpenAPI metadata includes:
  - Endpoint summary and description
  - Path parameter documentation (`id`)
  - Response schemas and status codes
  - Security scheme definition for OAuth2

Review the code, not the coder

```
def allocate(capacity, requests):
    used = 0
    result = []

    for r in requests:
        if used + r <= capacity:
            used += r
            result.append(r)

    return result
```

Code Review Basics

Does it solve the right problem?

Is it consistent with our project standards?

Does it handle errors gracefully?

Does it introduce unneeded complexity?

Does it break backward compatibility?

Is it documented?

Does it have tests?

Does it solve the right problem?

Problem: Get a list of active users

```
public String formatUserNames(List<User> users) {
    StringBuilder sb = new StringBuilder();

    for (User u : users) {
        sb.append(u.getFirstName())
          .append(" ")
          .append(u.getLastName())
          .append(", ");
    }

    return sb.toString();
}
```

```java
public String formatUserNames(List<User> users) {
    return users.stream()
            .filter(User::isActive)
            .map(u -> u.getFirstName() + " " + u.getLastName())
            .collect(Collectors.joining(", "));
}
```

Does it follow project standards?

```java
// Example 1: does not follow
//   - class naming conventions
//   - function naming conventions
//   - returning a magic value if not found
//   - SQL injection (more on this later)
public class userMgr {

    public int getusr(String id) {
        if (id == null) return -1;
        return Db.q("select * from usr where id='" + id + "'");
    }
}

// Example 2: follows project standards
public class UserManager {

    public Optional<User> getUserById(String userId) {
        Objects.requireNonNull(userId, "userId");

        return userRepository.findById(userId);
    }
}
```

Does it handle errors gracefully?

```java
// Bad: blows up, no helpful error msg,
//    no way to recover
public User loadUser(String id) {
    try {
        return userRepository.find(id);
    } catch (Exception e) {
        throw new RuntimeException("Error");
    }
}

// Better: catches error, logs a helpful message,
//    recovers using a Java Optional
public Optional<User> loadUser(String id) {
    try {
        return Optional.ofNullable(userRepository.find(id));
    } catch (DatabaseException e) {
        log.error("Failed to load user {}", id, e);
        return Optional.empty();
    }
}
```

Does it introduce unneeded complexity?

```java
// Bad: looks like resume-driven code
public boolean isAdmin(User user) {
    return Optional.ofNullable(user)
            .map(User::getRoles)
            .filter(r -> r.stream().anyMatch("ADMIN"::equals))
            .isPresent();
}

// Better: simple, easy to understand
public boolean isAdmin(User user) {
    if (user == null) {
        return false;
    }
    return user.getRoles().contains("ADMIN");
}
```

Does it introduce breaking changes?

```
// Original service
public class UserService {

    public User getUser(String id) {
        return repository.findById(id);
    }
}

// Breaking change: different function signature
public class UserService {

    public Optional<User> getUser(String id) {
        return Optional.ofNullable(repository.findById(id));
    }
}
```

Better

```java
// Better introduce new method until the code
//     can wean off the old method
public class UserService {

    @Deprecated
    public User getUser(String id) {
        return repository.findById(id);
    }

    public Optional<User> getUserOptional(String id) {
        return Optional.ofNullable(getUser(id));
    }
}
```

Is it documented?

```python
def allocate(capacity, requests):
    used = 0
    result = []
    for r in requests:
        if used + r <= capacity:
            used += r
            result.append(r)
    return result
```

```python
def allocate(capacity, requests):
    """
    Allocate capacity using a greedy, first-come-first-served strategy.

    Business rules:
    - Requests are evaluated in the order received.
    - Requests are either fully accepted or fully rejected (no partials).
    - This is intentional: fairness > optimal utilization.

    Example:
        capacity = 10
        requests = [4, 7, 3]
        result = [4, 3]

    Why not optimize?
    - A knapsack solution would increase utilization,
      but violates fairness guarantees promised to customers.
    """
    used = 0
    result = []

    for r in requests:
        if used + r <= capacity:
            used += r
            result.append(r)

    return result
```

Security Code Review

Input validation

Authentication & authorization

Secrets handling

Logging sensitive data

Error messages leaking information

Dependency vulnerabilities

Secure defaults

Does it validate inputs?

```java
// Bad: will take anything for a user ID
public User getUser(String userId) {
    // No validation - assumes userId is always valid
    return userRepository.findById(userId);
}


// Better: validates that the user ID looks valid
public User getUser(String userId) {
    if (userId == null || !userId.matches("[a-zA-Z0-9_-]+")) {
        throw new IllegalArgumentException("Invalid user id");
    }
    return userRepository.findById(userId);
}
```

Does user have authorization?

```java
// Bad: blindly returns data without authorization
public User getUserData(String userId) {
    // Assumes caller is authorized - no checks!
    return userRepository.findById(userId);
}

// Better: checks to see if user can perform this operation
public User getUserData(String userId, User caller) {
    if (!caller.isAuthenticated()) {
        throw new SecurityException("User must be logged in");
    }

    if (!caller.hasRole("ADMIN") && !caller.getId().equals(userId)) {
        throw new SecurityException("Access denied");
    }

    return userRepository.findById(userId);
}
```

Are secrets handled properly?

```java
// Bad: hardcoded password!
public class EmailSender {
    public void sendEmail(String to, String body) {
        String password = "SuperSecret123"; // Hard-coded!
        SmtpClient client = new SmtpClient("smtp.example.com", "user", password);
        client.send(to, body);
    }
}

// Better: password kept externally
public class EmailSender {
    private final String password = System.getenv("SMTP_PASSWORD");

    public void sendEmail(String to, String body) {
        if (password == null) {
            throw new IllegalStateException("SMTP password not configured");
        }
        SmtpClient client = new SmtpClient("smtp.example.com", "user", password);
        client.send(to, body);
    }
}
```

Does it log sensitive info?

```
// Bad: logs PII (Personally Identifiable Information)
//    - email address, social security number
//    - Logging PII could violate GDPR, HIPAA, etc.
public void processPayment(User user, double amount) {
    LOG.info("Processing payment for user: " + user.getEmail() +
                        " social security: " + user.getSSN() +
                        " amount: $" + amount);
    paymentService.charge(user, amount);
}

//Better: does not log PII
public void processPayment(User user, double amount) {
    // Good: avoid PII, log context only
    LOG.info("Processing payment for user id: {} amount: {}", user.getId(), amount);
    paymentService.charge(user, amount);
}
```

Do error messages leak info?

```java
// Bad - tells a hacker whether a user exists
public User login(String username, String password) {
    try {
        return authService.authenticate(username, password);
    } catch (UserNotFoundException e) {
        throw new RuntimeException("User " + username + " does not exist");
    } catch (InvalidPasswordException e) {
        throw new RuntimeException("Invalid password for user " + username);
    }
}

// Better: does not leak info about existence of a user
public User login(String username, String password) {
    try {
        return authService.authenticate(username, password);
    } catch (AuthenticationException e) {
        throw new RuntimeException("Invalid username or password");
    }
}
```

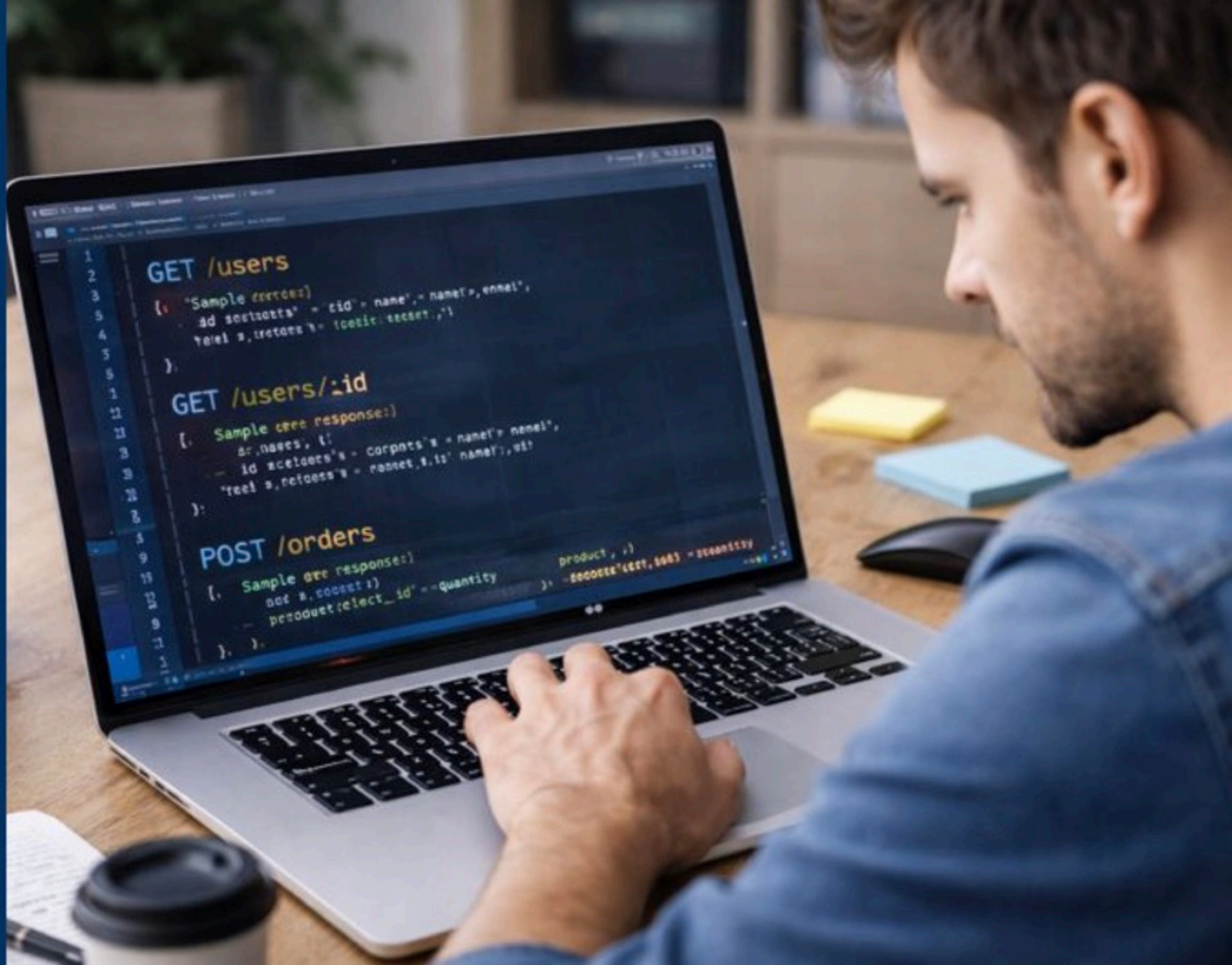
Do any dependencies have vulnerabilities?

```
// Maven POM snippet
// Bad: uses a version with a known vulnerability
<dependency>
    <groupId>commons-collections</groupId>
    <artifactId>commons-collections</artifactId>
    <version>3.2.1</version> <!-- Known vulnerability: CVE-2015-6420 -->
</dependency>

// Better: uses a safe version
// Use OWASP Dependency-Check, Snyk, GitHub Dependabot, Sonatype, etc.
<dependency>
    <groupId>org.apache.commons</groupId>
    <artifactId>commons-collections4</artifactId>
    <version>4.4</version> <!-- Safe version -->
</dependency>
```

Does it use secure defaults?

```java
// Bad: creates a file readable by anyone by default
public class FileUploader {

    public void uploadFile(File file) throws IOException {
        // Default: all files are publicly readable
        Files.copy(file.toPath(),
                   Paths.get("/uploads", file.getName()),
                   StandardCopyOption.REPLACE_EXISTING);
    }
}

// Better: sets default permissions that are tightly restrictive
public class FileUploader {

    public void uploadFile(File file) throws IOException {
        Path target = Paths.get("/uploads", file.getName());
        Files.copy(file.toPath(), target, StandardCopyOption.REPLACE_EXISTING);

        // Set restrictive permissions
        Set<PosixFilePermission> perms = PosixFilePermissions.fromString("rw-------");
        Files.setPosixFilePermissions(target, perms);
    }
}
```

REST API Code Review



API Design Consistency

Versioning Strategy

Correct HTTP verbs

Error Handling

Status Codes

Pagination & Filtering

Rate Limiting

API Design Consistency

```
// Bad - does not follow REST standard
GET /getUsers
POST /user/create
GET /UserDetails?id=123

// Better - follows REST standard
GET  /users
POST /users
GET  /users/{id}
```

Versioning Strategy

```
// Bad - no versioning strategy
GET /users

// Better - versioned API.
//    - Can support multiple API versions
//      at the same time.
GET /api/v1/users
```

Correct HTTP Verbs

```
// Bad - confused, incorrect usage
GET /users/create
GET /users/delete/123
POST /users/123

// Better - proper usage
POST   /users          → create user
GET    /users/{id}     → retrieve user
PUT    /users/{id}     → replace user
PATCH  /users/{id}     → partial update
DELETE /users/{id}     → delete user
```

Error Handling

```
// Bad - Leaks internal details
//     - Does not return proper REST error
throw new RuntimeException("NullPointerException in UserService.java:42");


// Better - Does not expose details about implementation.
//        - Uses standard REST status code
return ResponseEntity
    .status(HttpStatus.BAD_REQUEST)
    .body(new ErrorResponse("INVALID_INPUT", "User ID is required"));
```

Status Codes

```
// Bad - "ok" should not return an error
return ResponseEntity.ok("User not found");


// Better - uses correct status for "not found"
return ResponseEntity
    .status(HttpStatus.NOT_FOUND)
    .build();
```

Pagination And Filtering

```
// Bad - no pagination. What if 10 million users?
GET /users

// Better - allows pagination
GET /users?page=1&size=20&status=ACTIVE
```

Rate Limiting

```
// Bad - no rate limiting
@GetMapping("/login")
public Token login(@RequestBody LoginRequest req) {
    return authService.authenticate(req);
}

// Better - uses rate limiting
@GetMapping("/login")
@RateLimited(requests = 5, perSeconds = 60)
public Token login(@RequestBody LoginRequest req) {
    return authService.authenticate(req);
}
```

Database Code Review

Indexing

Nullable vs Non-Nullable

Defaults

N+1 Queries

Transactions

Migrations

Indexing

```
// Bad - full table scan on unindexed column
SELECT * FROM orders WHERE customer_id = ?;

// Better - creating an index will improve
//          performance of that query
CREATE INDEX idx_orders_customer_id
ON orders(customer_id);
```

Nullable vs Non-Nullable

```
// Bad - forces app to handle nulls
email VARCHAR(255) NULL

// Better - requires an email
email VARCHAR(255) NOT NULL
```

Defaults

```
// Bad - no defaults
created_at TIMESTAMP

// Better - uses a default
created_at TIMESTAMP NOT NULL
DEFAULT CURRENT_TIMESTAMP
```

N+1 Queries

```
// Bad - one query to get the orders
//       plus one query for each order to get items
//       Total: N+1 queries
List<Order> orders = findOrders();
for (Order o : orders) {
    findOrderItems(o.getId());
}

// Better - just one query, not N+1 queries
SELECT o.*, i.*
FROM orders o
JOIN items i ON i.order_id = o.id;
```

Transactions

```java
// Bad - can leave db in a bad state if creditAccount fails
public void transferFunds(long fromId, long toId, BigDecimal amount) {
    debitAccount(fromId, amount);   // succeeds
    creditAccount(toId, amount);    // throws exception
}


// Better - if an error happens, everything is rolled back
@Transactional
public void transferFunds(long fromId, long toId, BigDecimal amount) {
    debitAccount(fromId, amount);
    creditAccount(toId, amount); // if this fails, rollback occurs
}
```

Migrations

```
# Bad - breaking change for queries using 'name'
ALTER TABLE users RENAME COLUMN name TO full_name;


# Better - backward compatible migration
ALTER TABLE users ADD COLUMN full_name VARCHAR(255);
# -- populate
# -- later remove old column
```

Performance Reviews

Time Complexity

Memory Usage

I/O and Network Calls

Caching

Blocking vs Async

Time Complexity

```java
// Bad - nested loop over large collections
for (User u : users) {
    for (Order o : orders) {
        if (o.getUserId().equals(u.getId())) {
            process(o);
        }
    }
}


// Better - one pass to collect items, one pass to compare
Map<String, List<Order>> ordersByUser = orders.stream()
    .collect(groupingBy(Order::getUserId));

for (User u : users) {
    ordersByUser.getOrDefault(u.getId(), List.of())
                .forEach(this::process);
}
```

Memory Usage

```
// Bad - uses much memory
List<Record> records = repository.findAll();

records.forEach(this::process);

// Better - stream reduces memory usage
repository.streamAll()
          .forEach(this::process);
```

I/O and Network Calls

```
// Bad - one service call for each user
for (String id : ids) {
    userService.fetchUser(id); // HTTP call
}

// Better - fetch all users in one call
userService.fetchUsers(ids);
```

Caching

```
// Bad - service call each time config is needed
public Config getConfig() {
    return configService.fetchFromRemote();
}


// Better - config is fetched once and cached
private Config cached;

public Config getConfig() {
    if (cached == null) {
        cached = configService.fetchFromRemote();
    }
    return cached;
}
```

Blocking vs Async

```java
// Bad - blocking request
@GetMapping("/report")
public Report getReport() {
    return reportService.generate(); // long-running
}

// Better - asynchronous execution
@GetMapping("/report")
public CompletableFuture<Report> getReport() {
    return CompletableFuture.supplyAsync(reportService::generate);
}
```

Unit Test Reviews

Behavior, Not Implementation

Clear Test Names

Deterministic Tests

Edge Cases

Good Error Messages

Brittle Mocks

Behavior, Not Implementation

```java
// Bad - test looks at the implementation
@Test
void savesUserCallsRepositoryOnce() {
    service.createUser(user);
    verify(repo, times(1)).save(any());
}

// Better - test checks the behavior
@Test
void createsUserSuccessfully() {
    service.createUser(user);
    assertTrue(repo.existsById(user.getId()));
}
```

Clear Test Names

```
// Bad - doesn't really say what the test does
@Test
void testUser() { }


// Better - says what the test checks
@Test
void createUser_fails_whenEmailIsMissing() { }
```

Deterministic Tests

```java
// Bad - test depends on what time it is!
@Test
void expiresAfterFiveMinutes() {
    assertTrue(token.isExpired());
}

// Better - use a testing clock
@Test
void expiresAfterFiveMinutes() {
    Clock fixedClock = Clock.fixed(now.plus(5, MINUTES), UTC);
    assertTrue(token.isExpired(fixedClock));
}
```

Edge Cases

```java
// Bad - only test for happy path
@Test
void parsesValidInput() {
    assertEquals(10, parser.parse("10"));
}

// Better - test for edge cases like null
@Test
void parse_throwsException_onNullInput() {
    assertThrows(IllegalArgumentException.class,
                 () -> parser.parse(null));
}
```

Good Error Messages

```
// Bad - poor failure message
assertTrue(result);


// Better - meaningful error message
assertTrue(result, "Expected user to be active after login");
```

Brittle Mocks

```
// Bad - test is too dependent on object structure
when(repo.find(id)).thenReturn(user);
when(user.getProfile().getRole()).thenReturn("ADMIN");


// Better - test just returns an object created for
//          testing purposes
User user = TestUsers.admin();
when(repo.find(id)).thenReturn(user);
```

The Human Side Of Code Reviews


Be Kind. Offer some compliments, not just criticism.

Be specific.

Bad:

"This code is crap."
"This doesn't match our standards."
"This is wrong. Fix it."

Good:

"This REST call should return a 201, not 200."
"Use the logger in our logging package."
"This should be checking for active users only."

Don’t go crazy insisting on your way for minor things.

Use the “Southern” criticism style for emotionally sensitive coworkers.

Wrap a criticism inside two compliments.

compliment
  criticism
compliment

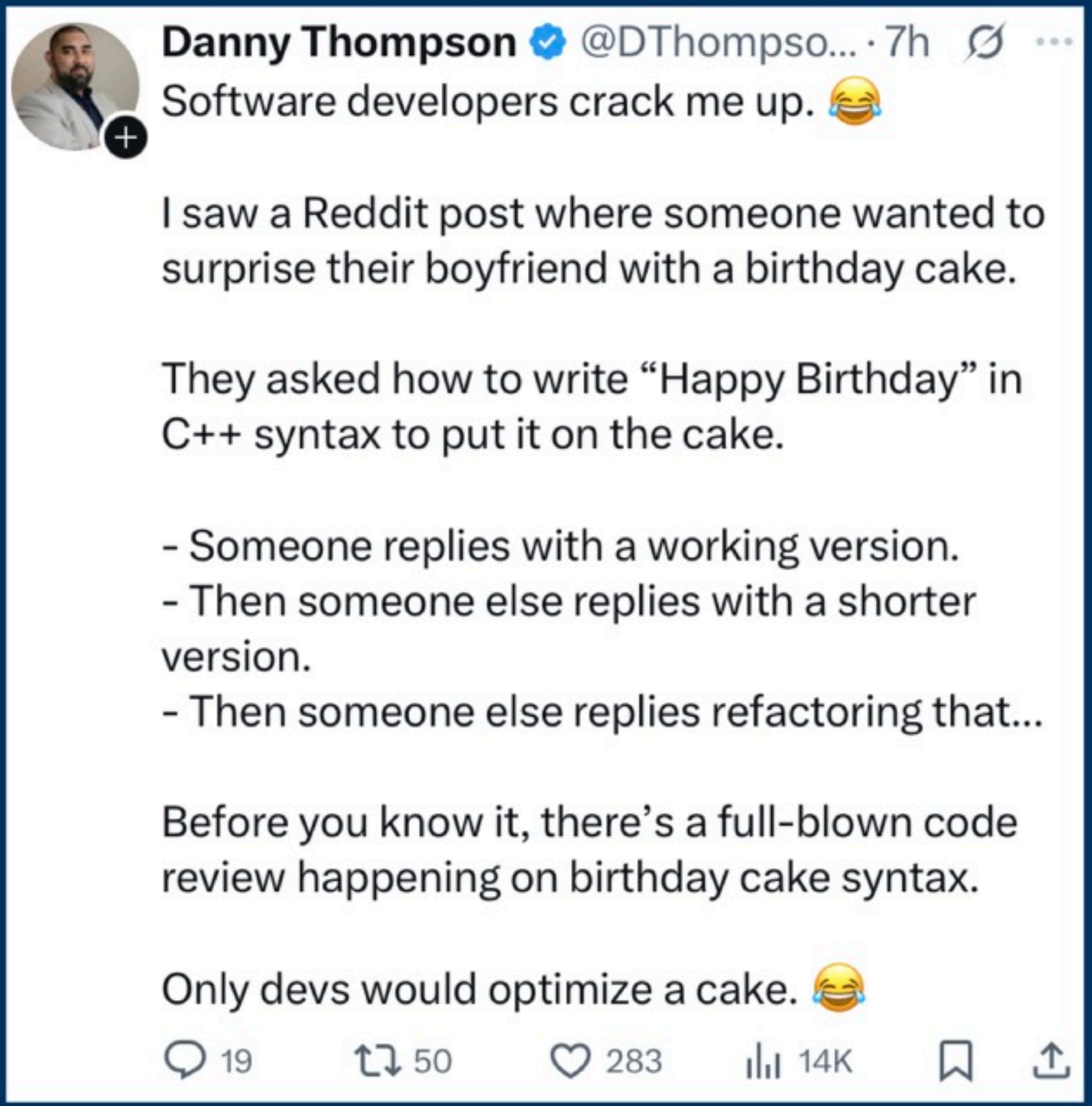
Example:

This is a good implementation of a REST API endpoint - you specified the right return status code, added the OpenAPI calls for a good Swagger page, and returned a proper result. One thing that’s missing, though, is OAuth2 security on this endpoint. But that’s a minor thing that you can fix easily. Everything else looks great!

Instead of:

“You forgot to do OAuth2 security on this endpoint.”

Don’t get hung up on reviewing things that don’t matter.



Danny Thompson @DThompso... · 7h

Software developers crack me up. 😂

I saw a Reddit post where someone wanted to surprise their boyfriend with a birthday cake.

They asked how to write “Happy Birthday” in C++ syntax to put it on the cake.

- Someone replies with a working version.
- Then someone else replies with a shorter version.
- Then someone else replies refactoring that...

Before you know it, there’s a full-blown code review happening on birthday cake syntax.

Only devs would optimize a cake. 😂

19 50 283 14K

Reviewing AI Code

AI doesn't always get things right

Mike Coutermarsh 

@mscccc

X.com

Learned this morning that my ai coded app for tracking my body weight, macros and step count has been storing all it's data in sqlite without a year.

So it has stopped working when the year changed.

10:29 AM · 1/1/26 · **128K** Views

Confident but incorrect logic

Outdated or insecure patterns

Hallucinated APIs

Overengineering

License and attribution risks

Incorrect Logic

```java
// Bad: AI generates incorrect code
// AI-generated comment: "Handles leap years correctly"
boolean isLeapYear(int year) {
    return year % 4 == 0;
}

// Better: the real way to calculate a leap year
boolean isLeapYear(int year) {
    return (year % 4 == 0 && year % 100 != 0) || year % 400 == 0;
}
```

Outdated Patterns

```java
// Bad - MD5 is insecure for passwords
MessageDigest md = MessageDigest.getInstance("MD5");
byte[] hash = md.digest(password.getBytes());

// Better - modern approach
PasswordEncoder encoder = new BCryptPasswordEncoder();
String hash = encoder.encode(password);
```

Hallucinated Code

```
// Bad - method sounds plausible, but doesn't exit
HttpClient client = HttpClient.newClient();
client.enableAutoRetry();

// Better - uses a real, existing method
HttpClient client = HttpClient.newHttpClient();
```

Overengineered

```
// Bad - unnecessary work
UserFetchStrategy strategy =
    UserFetchStrategyFactory.create(env);

User user = strategy.fetch(id);

// Better - simple and to the point
User user = userRepository.findById(id);
```

Licenses

```java
// Bad - copied code without license or attribution
// Source: "Found online"
public String encode(String input) {
    return Base64.getEncoder().encodeToString(input.getBytes());
}

// Better - lists source and license
// Uses Java standard library (Apache 2.0 compatible)
String encoded = Base64.getEncoder()
                       .encodeToString(input.getBytes());
```

KELLY MORRISON WILL RETURN IN AVENGERS: DOOMSDAY

The Lost Art Of Code Reviews

Slides on GitHub

LinkedIn

CGI

kellyivymorrison@gmail.com