Software Risk Management: Code Review

17-313 Fall 2023

Foundations of Software Engineering

https://cmu-313.github.io

Andrew Begel and Rohan Padhye

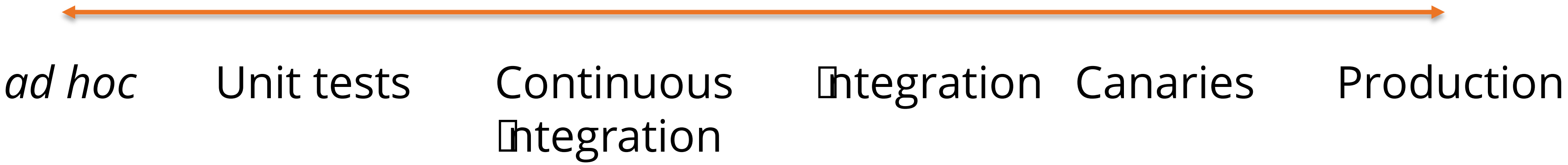
Administrivia

- Mid-term exam next week (Oct 10) in class
- Recitation this week: midterm review (**come prepared**!)
  - https://cmu-313.github.io/recitations/reci6-midterm-review/
  - Work through problems on the previous midterms – many students found this helpful.
  - Any questions on the previous midterm questions – bring them to recitation to discuss as a class.
- Fill in Team Assessment Survey by Friday 3:00pm
- Final Presentations (P5):
  Tuesday December 12th, 5:30 pm - 8:30pm, Room TBD

Ways to Test and Validate Your Code

- Static Validation
  - Stare at the code
- Dynamic Validation
  - Run the source code

Dynamic Validation



ad hoc

Unit tests

Continuous Integration

Integration

Canaries

Production

Static Validation

- Style guides
- Compiler warnings and errors
- Static analysis
  - FindBugs
  - clang-tidy
  - Pylons Webtest
- Code review

Style Guide

- List of environment-specific preferred practices
- Could include:
  - Libraries / idioms to use
  - Formatting

Style Guide Examples

- https://www.python.org/dev/peps/pep-0008/
- https://github.com/airbnb/javascript
- https://subversion.apache.org/docs/community-guide/conventions.html
- https://google.github.io/styleguide/cppguide.html
- https://google.github.io/styleguide/pyguide.html
- Linux kernel style guide

Who writes these style guides?

👀

Who writes these style guides?

(*ad hoc* 🍑🗣️) Self-proclaimed code protectors

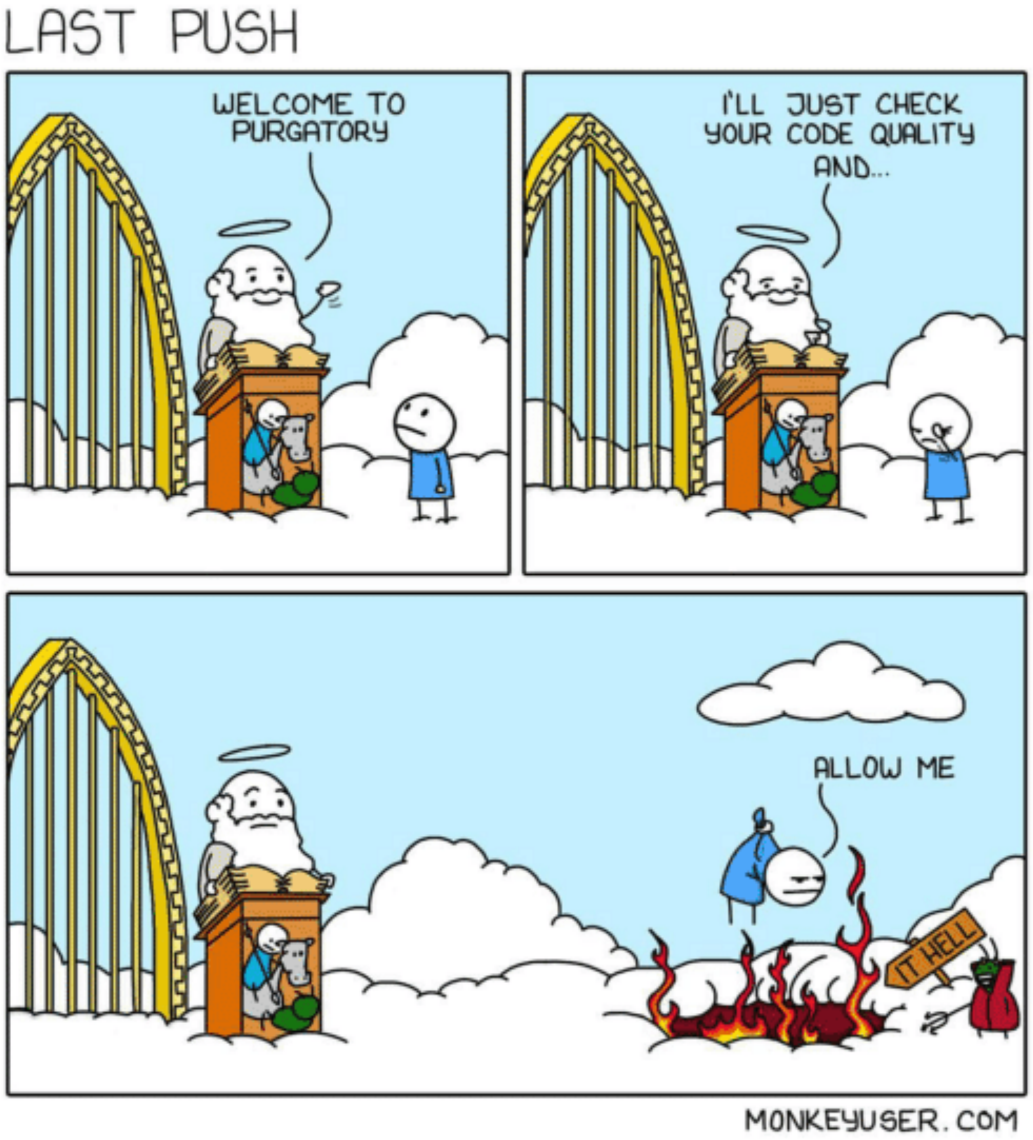
(*wisdom*) Team veteran developers

(*copy-paste)* Google search for blog posts by experts

(*empirical study*) Evidence-based analysis of code styles that correlate with bugs

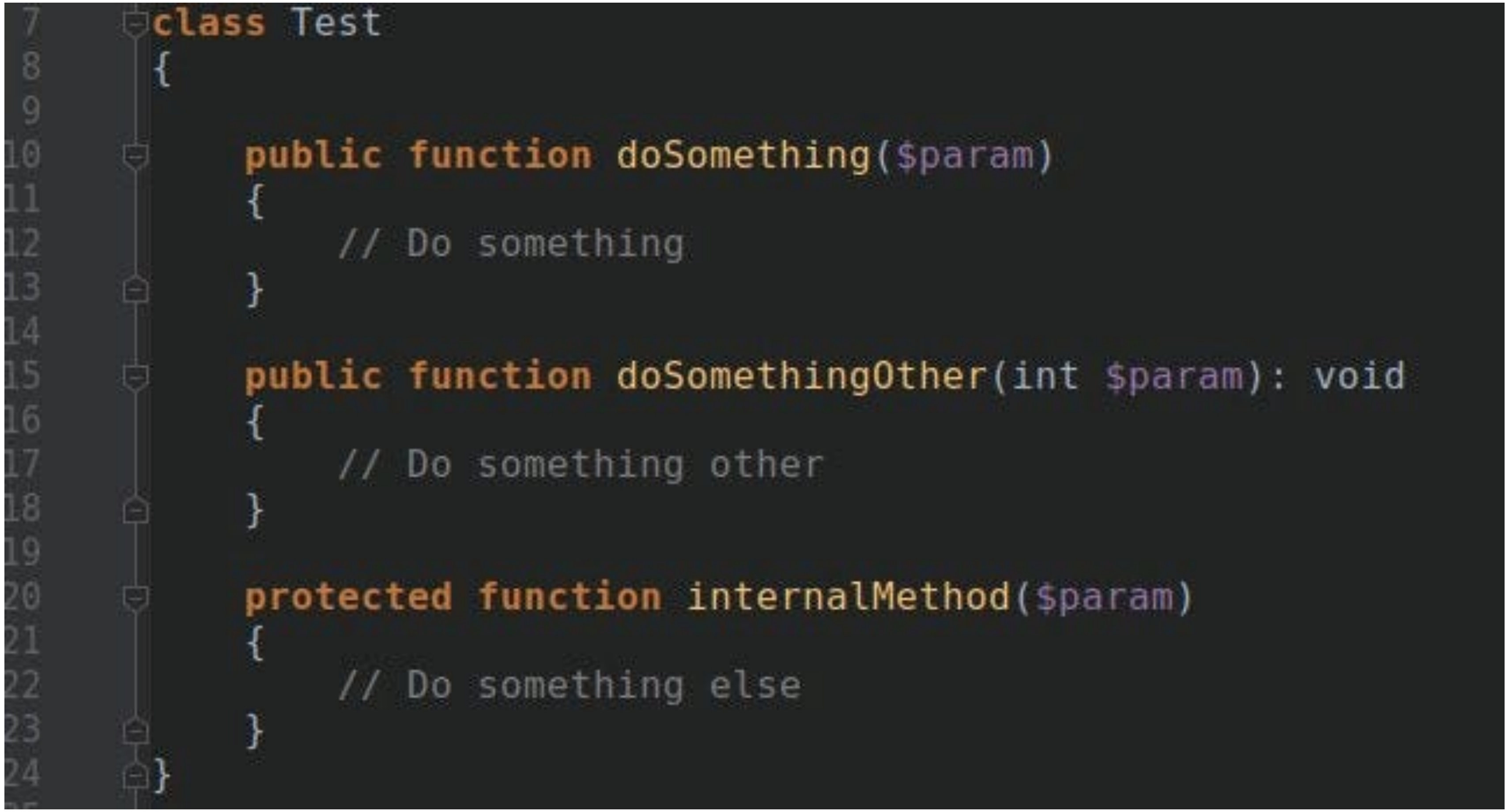
Code Review

- Does this code do what it claims?
  - Are there any programming bugs?
- Why are we making this change?
  - Are there any design bugs?


LAST PUSH
WELCOME TO PURGATORY
I'LL JUST CHECK YOUR CODE QUALITY AND...
ALLOW ME
IT HELL
MONKEYUSER.COM

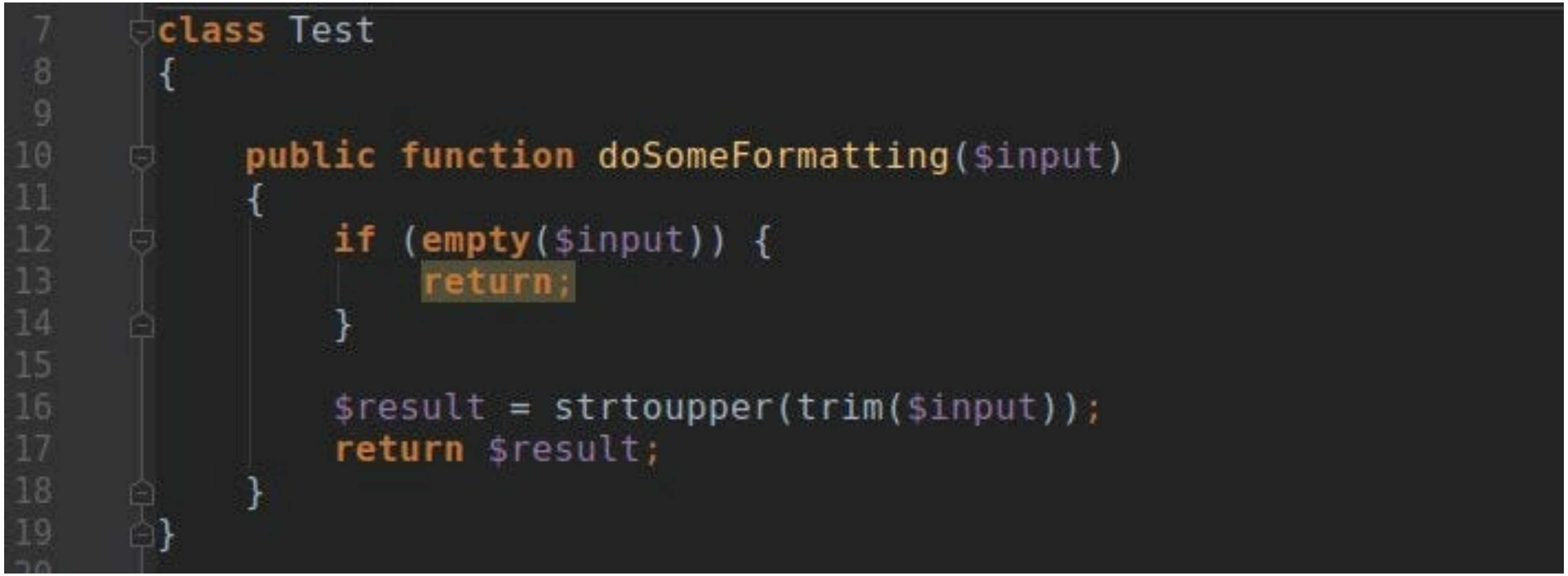
last push

Fishy Code Example #1



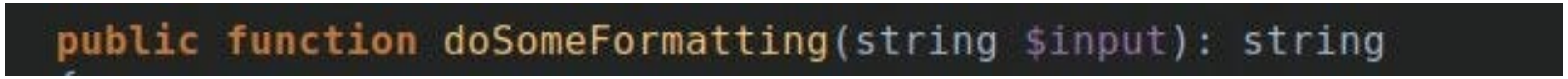
```php
class Test
{

    public function doSomething($param)
    {
        // Do something
    }

    public function doSomethingOther(int $param): void
    {
        // Do something other
    }

    protected function internalMethod($param)
    {
        // Do something else
    }
}
```

Fishy Code Example #2a



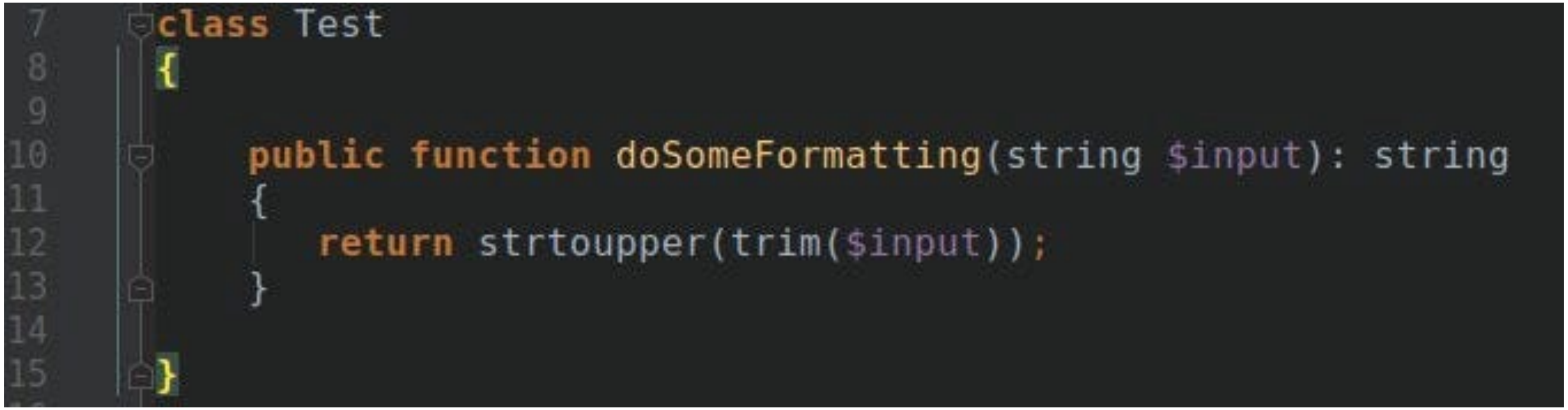
```
class Test
{

    public function doSomeFormatting($input)
    {
        if (empty($input)) {
            return;
        }

        $result = strtoupper(trim($input));
        return $result;
    }
}
```

Fishy Code Example #2b



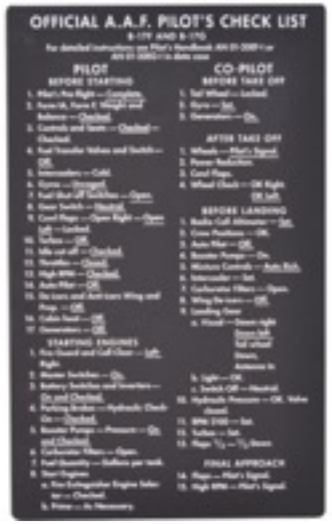
```
public function doSomeFormatting(string $input): string
```

Fishy Code Example #2c



```
class Test
{

    public function doSomeFormatting(string $input): string
    {
        return strtoupper(trim($input));
    }

}
```

Checklists help manage complex processes



OFFICIAL A.A.F. PILOT'S CHECK LIST

B-17F AND B-17G

For detailed instructions see Pilot's Handbook AN 01-20EF-1 or AN 01-20EG-1 [illegible]

PILOT

BEFORE STARTING

1. Pilot's Preflight — Complete.
2. Form 1A, Form F, Weight and Balance — Checked.
3. Controls and Seats — Checked — Checked.
4. Fuel Transfer Valves and Switch — Off.
5. Intercoolers — Cold.
6. Gyros — Uncaged.
7. Fuel Shut-off Switches — Open.
8. Gear Switch — Neutral.
9. Cowl Flaps — Open Right — Open Left — Locked.
10. Turbos — Off.
11. Idle cut-off — Checked.
12. Throttles — Closed.
13. High RPM — Checked.
14. Auto Pilot — Off.
15. De-icers and Anti-icers, Wing and Prop. — Off.
16. Cabin Heat — Off.
17. Generators — Off.

STARTING ENGINES

1. Fire Guard and Call Clear — Left, Right.
2. Master Switches — On.
3. Battery Switches and Inverters — On and Checked.
4. Parking Brakes — Hydraulic Check — On — Checked.
5. Booster Pumps — Pressure — On and Checked.
6. Carburetor Filters — Open.
7. Fuel Quantity — Gallons per tank.
8. Start Engines:
   a. Fire Extinguisher Engine Selector — Checked.
   b. Prime — As Necessary.

CO-PILOT

BEFORE TAKE OFF

1. Tail Wheel — Locked.
2. Gyro — Set.
3. Generators — On.

AFTER TAKE OFF

1. Wheels — Pilot's Signal.
2. Power Reduction.
3. Cowl Flaps.
4. Wheel Check — OK Right, OK Left.

BEFORE LANDING

1. Radio Call, Altimeter — Set.
2. Crew Positions — OK.
3. Auto Pilot — Off.
4. Booster Pumps — On.
5. Mixture Controls — Auto-Rich.
6. Intercooler — Set.
7. Carburetor Filters — Open.
8. Wing De-icers — Off.
9. Landing Gear
   a. Visual — Down right, Down left, Tail wheel Down, Antenna in.
   b. Light — OK.
   c. Switch Off — Neutral.
10. Hydraulic Pressure — OK. Valve closed.
11. RPM 2100 — Set.
12. Turbos — Set.
13. Flaps 1/3 — 1/3 Down.

FINAL APPROACH

14. Flaps — Pilot's Signal.
15. High RPM — Pilot's Signal.

The Checklist: https://www.newyorker.com/magazine/2007/12/10/the-checklist

Activity: Create your own checklist

- □n pairs, think about dumb mistakes your “friend” made the last time they were coding.
  - Write your names on a piece of paper.
  - Write down two checklist items that would have caught those errors.
- Divide into teams: left and right sides of the classroom.
- Shout your ideas to Prof Begel, who will write them on the chalkboard.
  - Which team had the most unique/good entries in their list?
- By 5pm, upload a picture of your paper to Gradescope: October 5 Activity.

Sample Low-Level Coding Checklist
(not complete)

- General
  - Are all changes relevant?
  - Do the classes and methods fulfill their purpose?
  - Are the messages and texts for the user correct?
- Classes
  - Are all assignments of attributes correct?
  - Are the classes implemented correctly?
- Arguments
  - Are the correct arguments used in all method calls?
- Recursion
  - Does recursion terminate properly?
- Methods
  - Do methods always return a valid value?
  - Do methods check parameters for validity (if needed)?
  - Are all parameters used?
  - Do methods have parameter and return types declared? Variables
  - Are all variables, counters, and accumulators initialized properly and, if necessary, re-initialized every time they are used?
  - Are all declared variables being used?
- If-Then Statements
  - Do the if-else statements fit the intended purpose?
  - Are all edge cases handled?
- Loops
  - Do the loops end under all possible conditions?
  - Are the break and continue statements used properly?
- Errors
  - Are exceptions handled correctly?
- Final Check
  - Are all changes consistent with one another?

Formal Inspections

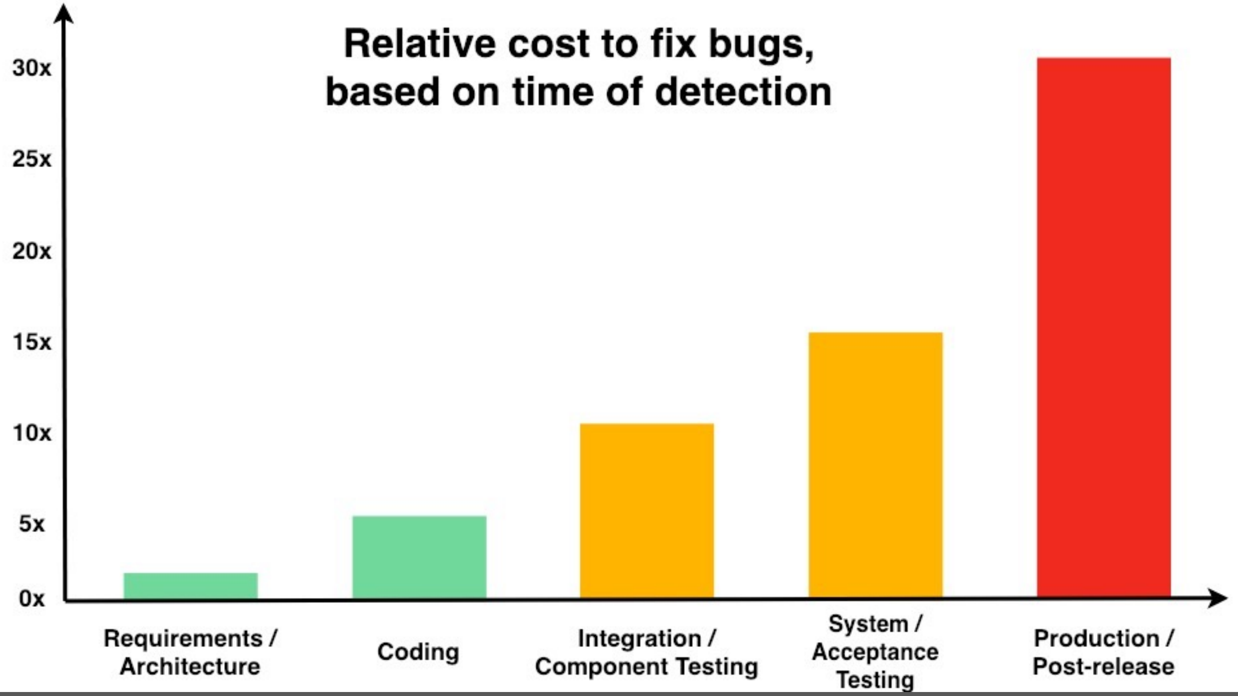
- Idea popularized in 70s at IBM
- Broadly adopted in 80s, much research
    - Sometimes replaced component testing
- Group of developers meets to formally review code or other artifacts
- Most effective approach to find bugs
    - Typically, 60-90% of bugs found with inspections
- Expensive and labor-intensive

Inspection Team and Roles

- Typically, 4-5 people (min 3)
- Author
- Inspector(s)
  - Find faults and broader issues
- Reader
  - Presents the code or document at inspection meeting
- Scribe
  - Records results
- Moderator
  - Manages process, facilitates, reports

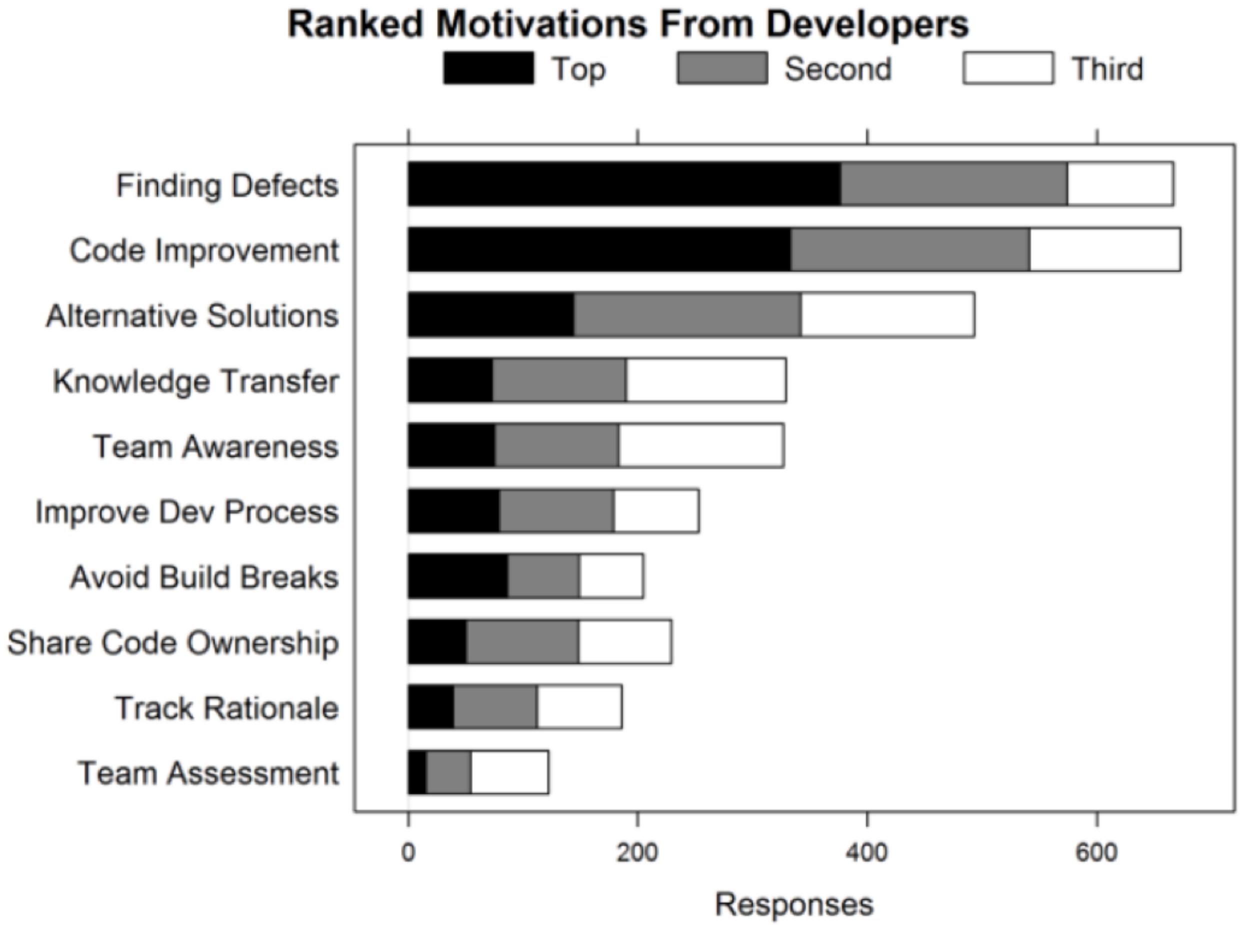
Motivation

- Linus’s Law: “Given enough eyeballs, all bugs are shallow.”
  - - The Cathedral and the Bazaar, Eric Raymond



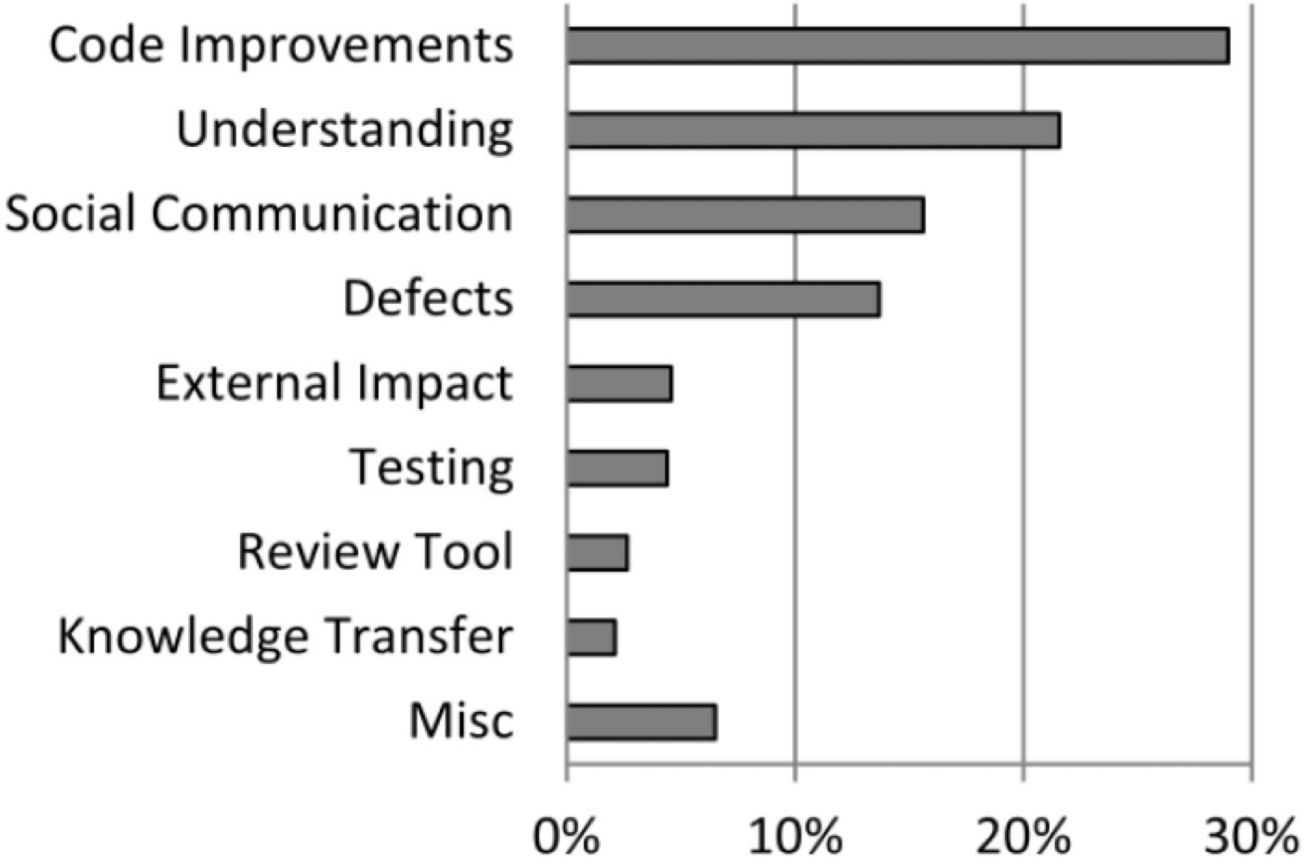
Expectations and Outcomes

Code Review at Microsoft



Bacchelli, Alberto and Christian Bird. "Expectations, outcomes, and challenges of modern code review." Proceedings of the 2013 International Conference on Software Engineering. IEEE Press, 2013.

Outcomes (Analyzing Reviews)



Mismatch of Expectations and Outcomes

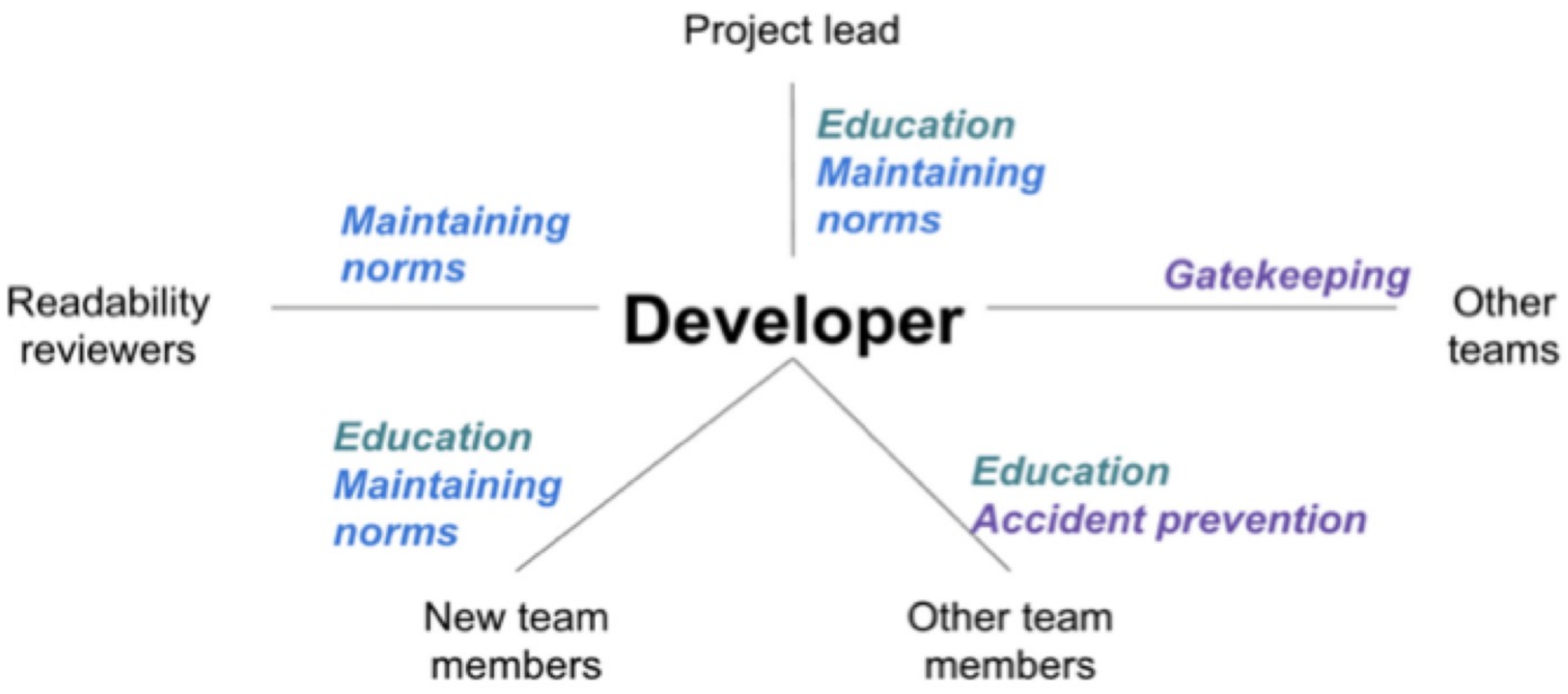
- Low quality of code reviews
  - Reviewers look for easy errors, as formatting issues
  - Miss serious errors
- Understanding is the main challenge
  - Understanding the reason for a change
  - Understanding the code and its context
  - Feedback channels to ask questions often needed
- No quality assurance on the outcome

Code Review at Google

- Introduced to “force developers to write code that other developers could understand”
- Three benefits:
  - checking the consistency of style and design
  - ensuring adequate tests
  - improving security by making sure no single developer could commit arbitrary code without oversight

Caitlin Sadowski, Emma Söderberg, Luke Church, Michal Sipko, and Alberto Bacchelli. 2018. Modern Code Review: A Case Study at Google. International Conference on Software Engineering

Reviewing Relationships



Project lead

Education
Maintaining norms

Readability reviewers

Maintaining norms

Developer

Gatekeeping

Other teams

Education
Maintaining norms

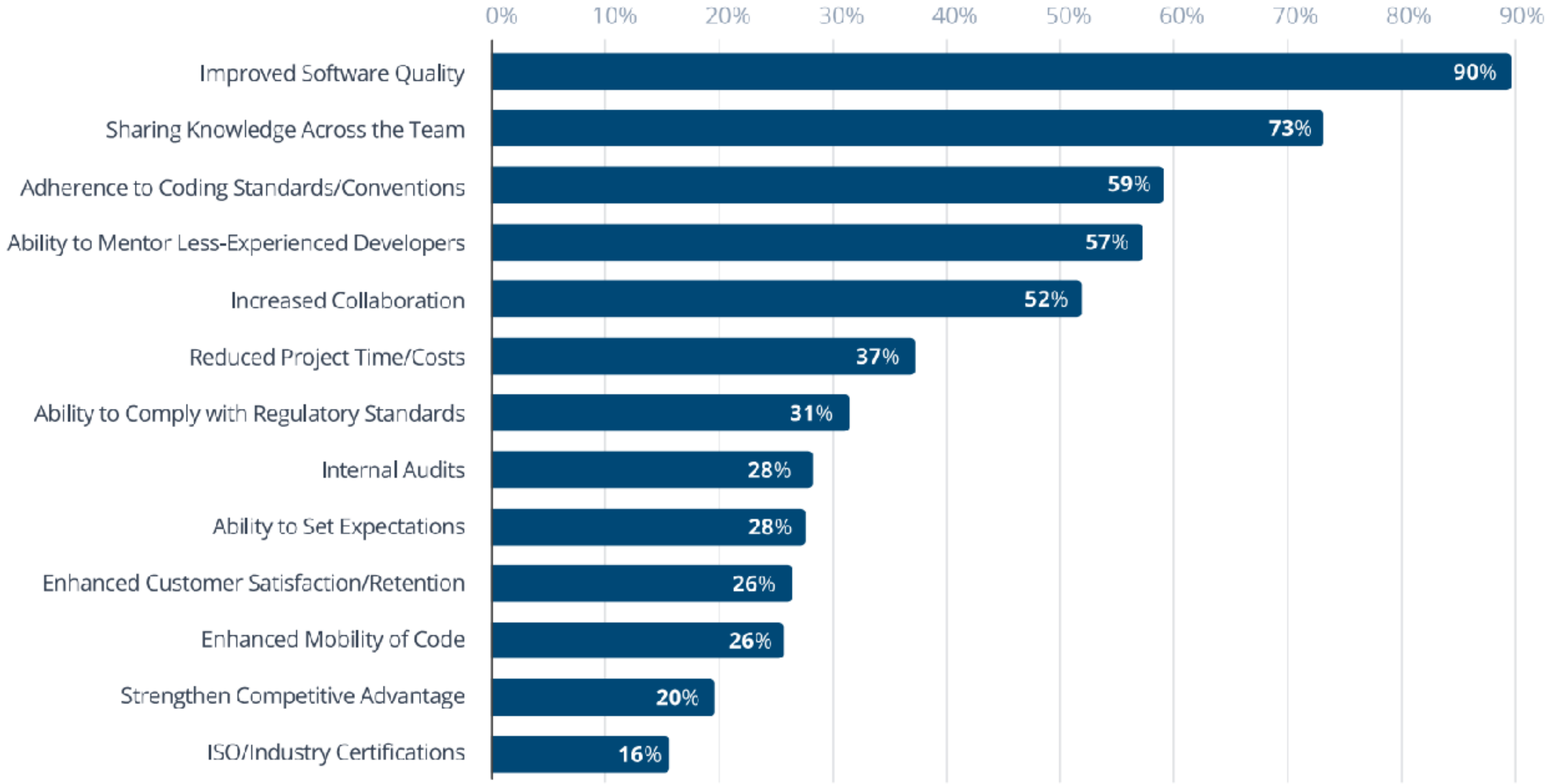
New team members

Education
Accident prevention

Other team members

The State of Code Review survey

What do you believe are the most important benefits of code review?



n = 1129

Code Review

- Start with the “big ideas”
- Automate the little things
- Focus on understanding
- Remember a person wrote the code
- Don’t overwhelm the person with feedback

Don’t forget that coders are people with feelings

- A coder’s self-worth is in their artifacts
- C□can avoid embarrassment
- □dentify defects, not alternatives; do not criticize coder
  - “*you* didn’t initialize variable a” -> “□don’t see where variable a is initialized”
- Avoid defending code; avoid discussions of solutions/alternatives
- Reviewers should not “show off” that they are better/smarter
- Avoid style discussions if there are no guidelines
- The coder gets to decide how to resolve fault