

# Build Software Safely!

17-313: Foundations of Software Engineering

<https://cmu-313.github.io>

**Michael Hilton** and Chris Timperley

Fall 2025

# Learning Goals

- Learn to discuss risk in a project
- Strategize about ways to mitigate risk
- Learn to get early feedback to reduce risk
- Find ways to catch our technical errors

2

# Administrivia

- P2B Due Fri, Sept 26th @ 11:59pm
- Midterm review session Sunday October 5<sup>th</sup> 7pm
- Midterm 1 Oct 9<sup>th</sup>

# Smoking Section

- Last full row



# Risk

# Risk



# Definition: Risk

Risk is a measure of the potential inability to achieve overall program objectives within defined cost, schedule, and technical constraints.



7

# Risk is defined by two key components



**The probability (or likelihood) of failing to achieve a particular outcome**



**The consequences (or impact) of failing to achieve that outcomes**



# Internal vs. External Risk



Risks that we **can** control



Risks that we **cannot** control

# Levels of Risk Management

1. **Crisis management:** Fire fighting; address risks only after they have become problems.
2. **Fix on failure:** Detect and react to risks quickly, but only after they have occurred.
3. **Risk mitigation:** Plan ahead of time to provide resources to cover risks if they occur, but do nothing to eliminate them in the first place.
4. **Prevention:** Implement and execute a plan as part of the software project to identify risks and prevent them from becoming problems.
5. **Elimination of root causes:** Identify and eliminate factors that make it possible for risks to exist at all.

# Levels of Risk Management

1. **Crisis management:** Fire fighting; address risks only after they have become problems.
2. **Fix on failure:** Detect and react to risks quickly, but only after they have occurred.
3. **Risk mitigation:** Plan ahead of time to provide resources to cover risks if they occur, but do nothing to eliminate them in the first place.
4. **Prevention:** Implement and execute a plan as part of the software project to identify risks and prevent them from becoming problems.
5. **Elimination of root causes:** Identify and eliminate factors that make it possible for risks to exist at all.



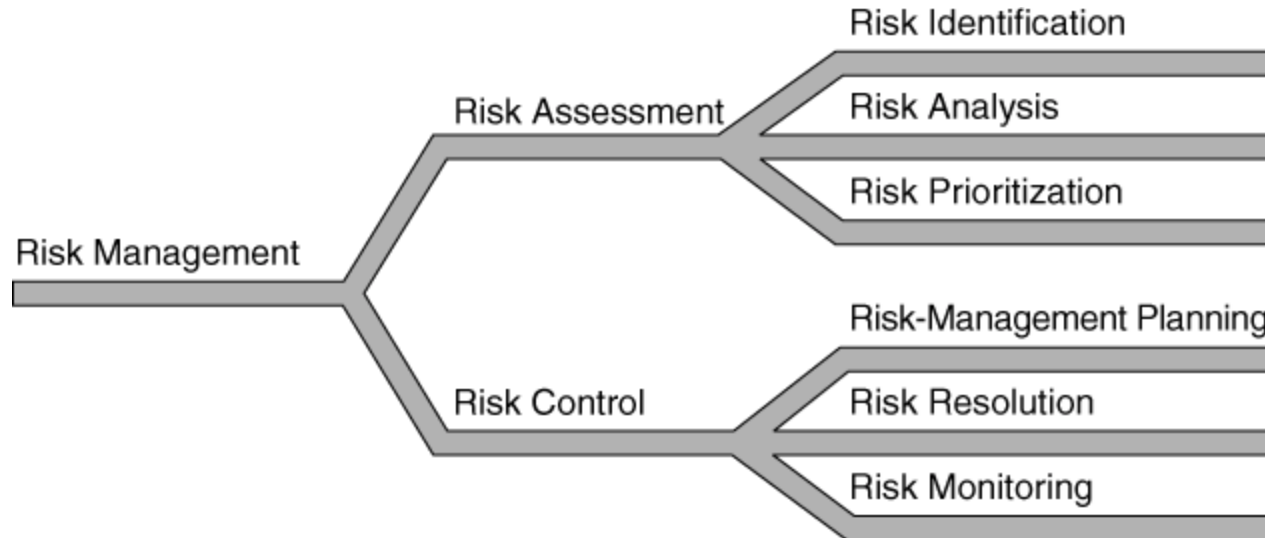
S3D

Software and Societal

Systems Department

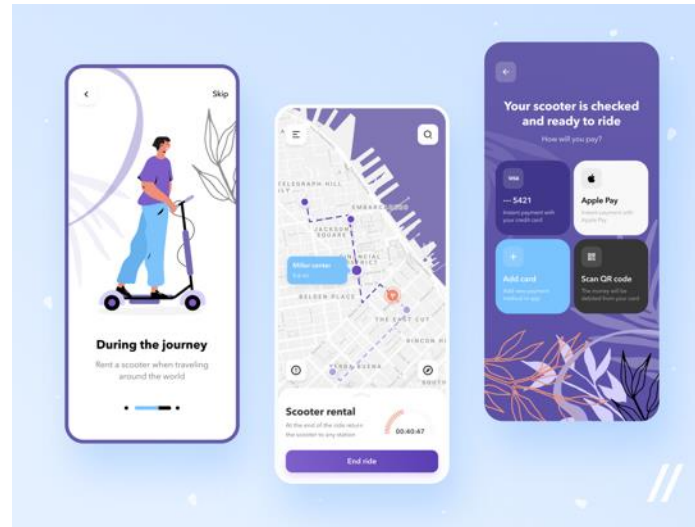
"Rapid Development: Creating Wild Software Schedules," Steve McConnell, 1996

# Risk Management



# Team Exercise: Risk Identification

- What risks exist for the scooter app?



13

# Risk assessment matrix



TABLE III. Risk assessment matrix

RISK ASSESSMENT MATRIX				
SEVERITY PROBABILITY	Catastrophic (1)	Critical (2)	Marginal (3)	Negligible (4)
Frequent (A)	High	High	Serious	Medium
Probable (B)	High	High	Serious	Medium
Occasional (C)	High	Serious	Medium	Low
Remote (D)	Serious	Medium	Medium	Low
Improbable (E)	Medium	Medium	Medium	Low
Eliminated (F)	Eliminated			

• MIL-STD-882E

<https://www.system-safety.org/Documents/MIL-STD-882E.pdf>

# Aviation failure impact categories

- **No effect** – failure has no impact on safety, aircraft operation, or crew workload
- **Minor** – failure is noticeable, causing passenger inconvenience or flight plan change
- **Major** – failure is significant, causing passenger discomfort and slight workload increase
- **Hazardous** – high workload, serious or fatal injuries
- **Catastrophic** – loss of critical function to safely fly and land

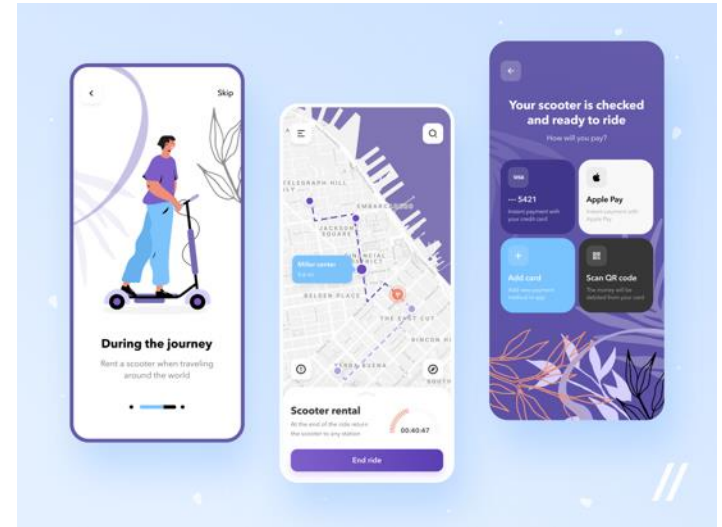
# Risk Analysis

Risk	Probability (%)	Size of Loss (weeks)	Risk Exposure (weeks)
Overly optimistic schedule	50%	5	2.5
Additional features added by marketing (specific features unknown)	35%	8	2.8
Project approval takes longer than expected	25%	4	1.0
Management-level progress reporting takes more developer time than expected	10%	1	0.1
New programming tools do not produce the promised savings	30%	5	1.5
...	...	...	...
<b>Total</b>			12



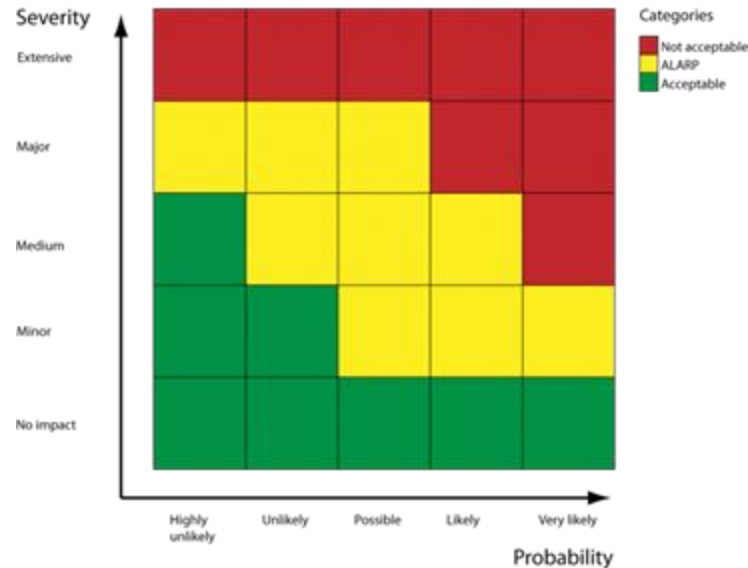
# Exercise: Risk Analysis

- What is the risk severity for your scooter app?



# Risk Prioritization

## Focus on risks with the highest exposure



18

# Risk Control

- What steps can be taken to avoid or mitigate the risk?
- Can you better understand and forecast the risk?
- Who will be responsible for monitoring and addressing the risk?
- Have risks evolved over time?
- Bake risks into your schedule
  - Don't assume that nothing will go wrong between now and the end of the semester!

19

# DECIDE Model

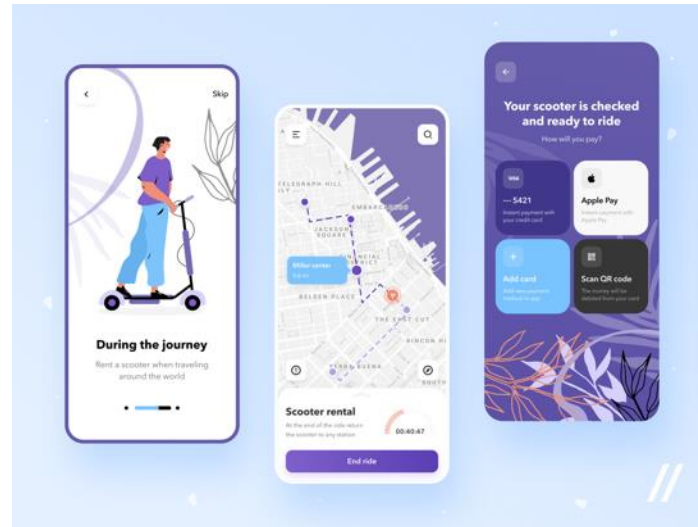


**Detect** that the action necessary  
**Estimate** the significance of the action  
**choose** a desirable outcome  
**Identify** actions needed in order to  
achieve the chosen option  
**do** the necessary action to achieve  
change  
**Evaluate** the effects of the action

[https://www.faa.gov/regulations\\_policies/handbooks\\_manuals/aviation/media/FAA-H-8083-2.pdf](https://www.faa.gov/regulations_policies/handbooks_manuals/aviation/media/FAA-H-8083-2.pdf)

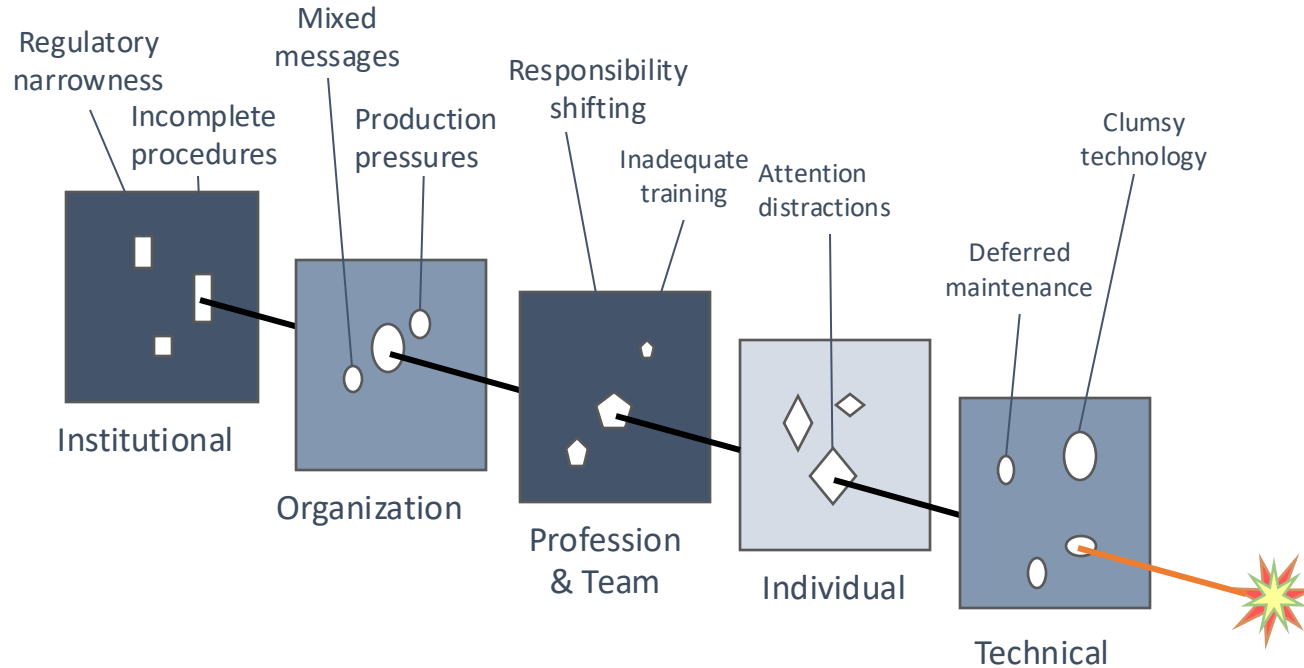
# Discussion: Risk Elimination and Mitigation

- How can you eliminate/mitigate risk for your scooter app?



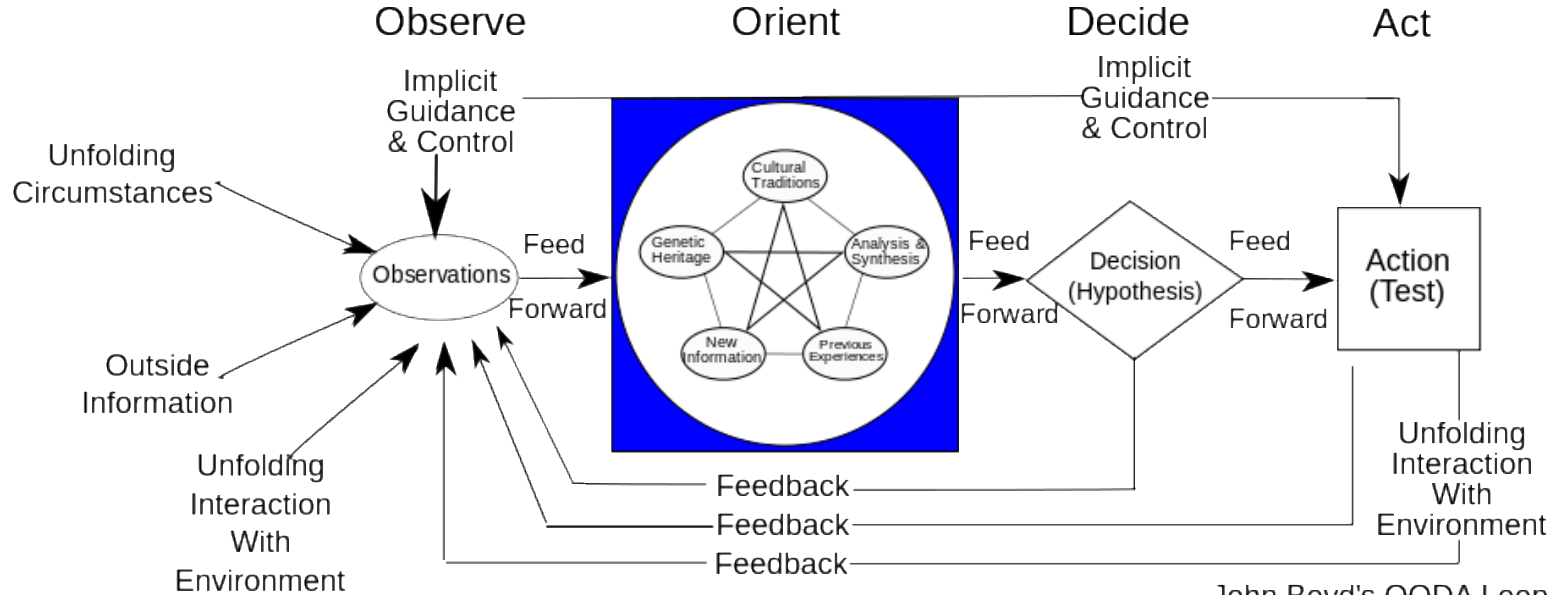
21

# The Swiss cheese model



Modified from Reason, 1990, by R.J. Cook

# OODA Loop



John Boyd's OODA Loop

By Patrick Edwin Moran - Own work, CC BY 3.0,  
<https://commons.wikimedia.org/w/index.php?curid=3904554>

# No matter what you do

- Some idiots won't follow your rules 😊





# Pre-mortems

- "unlike a typical critiquing session, in which project team members are asked what *might* go wrong, the premortem operates on the assumption that the 'patient' has died, and so asks what *did* go wrong."

Project Management

## Performing a Project Premortem

by Gary Klein

From the Magazine (September 2007)



Tweet



Post



Share



Save



Buy Copies

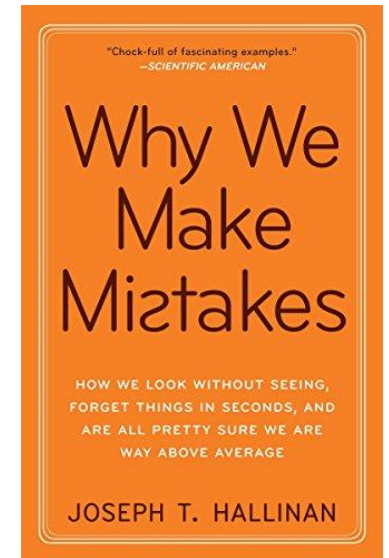


Print

**Summary.** Reprint: F0709A In a premortem, team members assume that the project they are planning has just failed—as so many do—and then generate plausible reasons for its demise. Those with reservations may speak freely at the outset, so that the project can be... [more](#)

# Why do we make mistakes?

26



# Generalization

- ...in the words of psychologist Tom Stafford, we can't find our typos because we're engaging in a high-level task in writing. **Our brains generalize simple, component parts to focus on complex tasks**, so essentially we can't catch the small details because we're focused on a large task.

27

<https://medium.com/swlh/why-we-miss-our-own-typos-96ab2f06afb7>

Boredom can give rise to errors, adverse patient events, and decreased productivity—costly and unnecessary outcomes for consumers, employees, and organizations alike. As a function of boredom, individuals may feel over-worked or under-employed, and become distracted, stressed, or disillusioned. Staff who are bored also are less likely to engage with or focus on their work.

Original Articles

## Boredom in the Workplace: Reasons, Impact, and Solutions

Michelle Cleary , PhD, RN, Jan Sayers , PhD, RN, Violeta Lopez , PhD, RN & Catherine Hungerford , PhD, RN

Pages 83-89 | Received 24 Jun 2015, Accepted 13 Aug 2015, Published online: 10 Feb 2016

Download citation

<https://doi.org/10.3109/01612840.2015.1084554>



Full Article

Figures & data

References

Citations

Metrics

Reprints & Permissions

Get access

### Abstract

Boredom in the workplace is not uncommon, and has been discussed widely in the academic literature in relation to the associated costs to individuals and organizations. Boredom can give rise to errors, adverse patient events, and decreased productivity—costly and unnecessary outcomes for consumers, employees, and organizations alike. As a function of boredom, individuals may

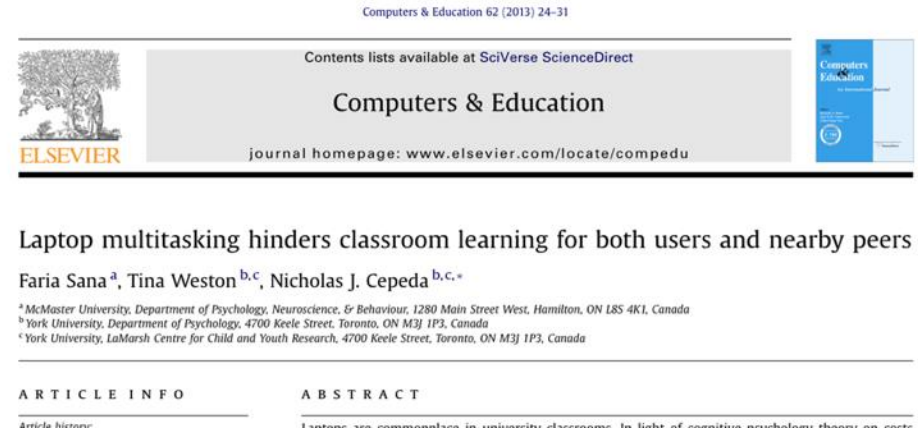
### Related rese

People also read

Boredom at work  
spillover model c  
work motivation  
boredom >

# Cognitive Load

- ...” students who switch back and forth between attending to a classroom lecture and checking e-mail, Facebook, and IMing with friends”



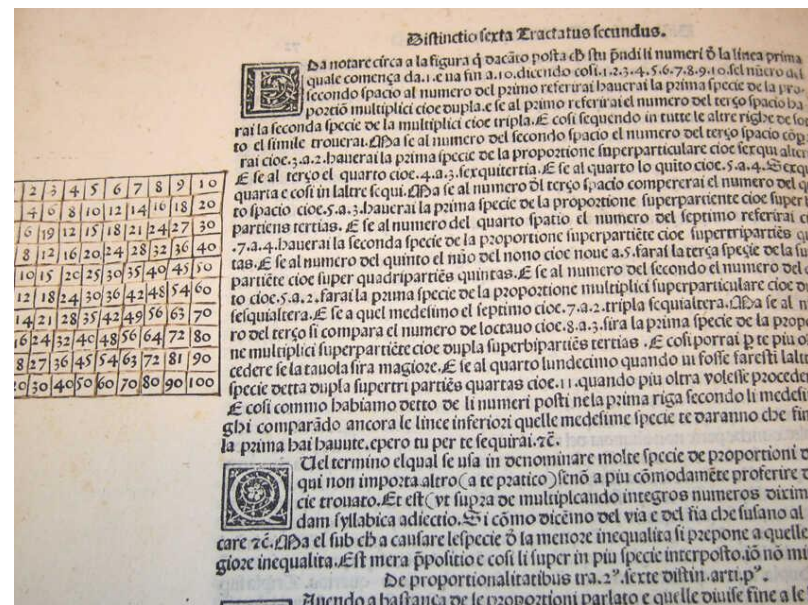
# Can we remove human error?

30

# Can we <sup>catch</sup> ~~remove~~ human error?

Can we catch human error before we ship our code?<sup>31</sup>

Can we automate tasks to prevent problems?



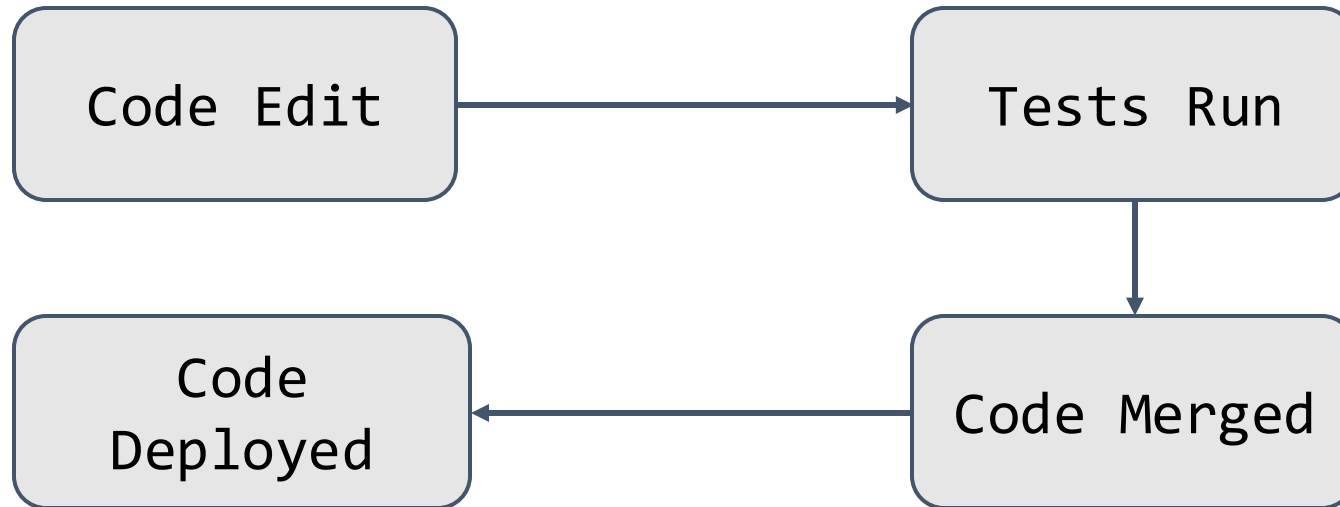


# **Approach:**

## **Automate what we can**

## **Review what we cannot**

# CI/CD Pipeline overview



# Continuous Integration:

Catch mistakes before you push your code!

35

# History of CI



(1999) Extreme Programming (XP) rule: “Integrate Often”



(2000) Martin Fowler posts “Continuous Integration” blog



(2001) First CI tool



**Jenkins** (2005) Hudson/Jenkins



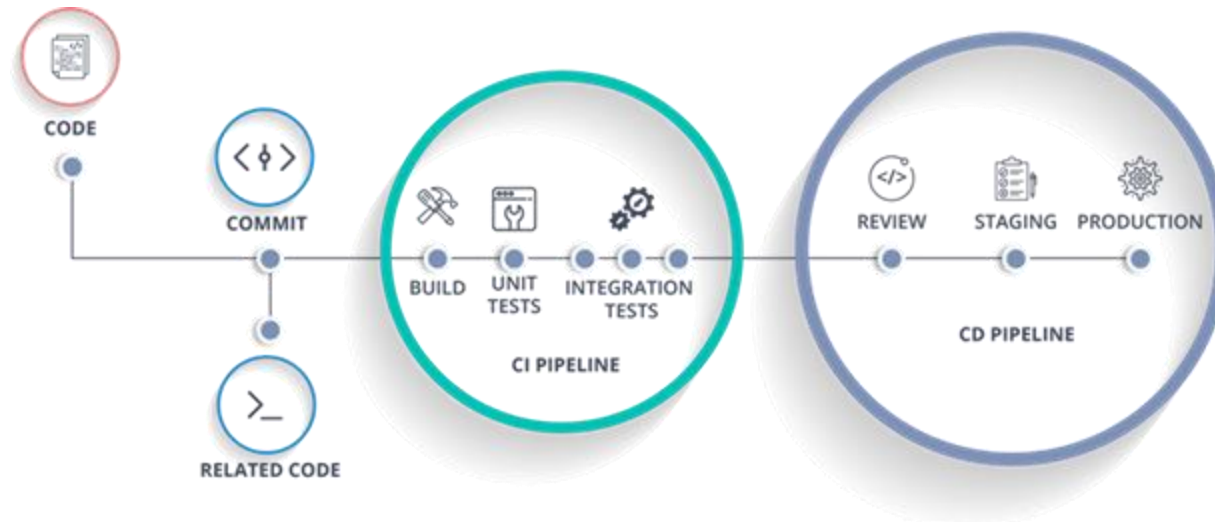
**Travis CI** (2011) Travis CI



GitHub Actions

(2019) GitHub Actions

# Example CI/CD Pipeline



# Developers say:

CI helps us catch bugs earlier

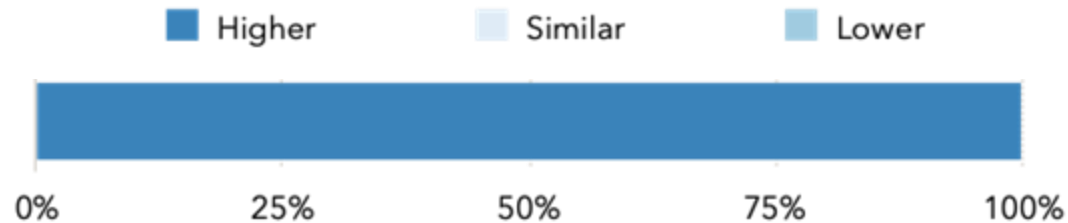
CI makes us less worried about breaking our builds

CI lets us spend less time debugging

“[CI] does have a pretty big impact on [catching bugs]. It allows us to find issues even before they get into our main repo, ... rather than letting bugs go unnoticed, for months, and letting users catch them.”

# Developers report:

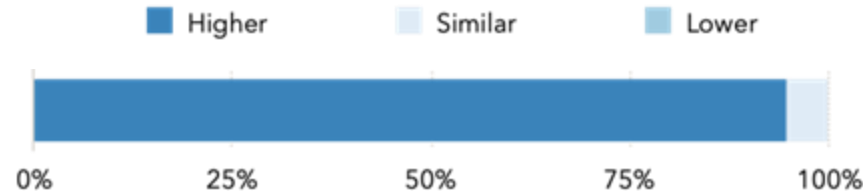
Do developers on projects with CI give (more/similar/less) value to automated tests?



# Developers report:

Do developers on projects with CI give (more/similar/less) value to automated tests?

Do projects with CI have (higher/similar/lower) test quality?



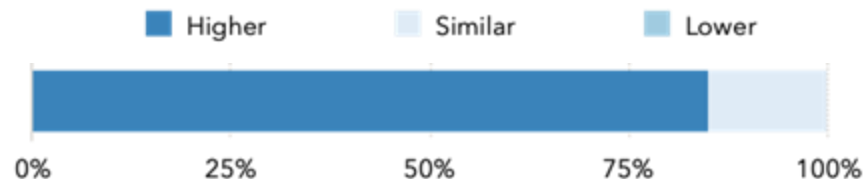


# Developers report:

Do developers on projects with CI give (more/similar/less) value to automated tests?

Do projects with CI have (higher/similar/lower) test quality?

Do projects with CI have (higher/similar/lower) code quality?



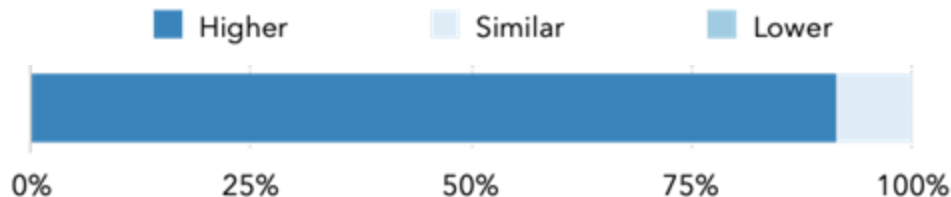
# Developers report:

Do developers on projects with CI give (more/similar/less) value to automated tests?

Do projects with CI have (higher/similar/lower) test quality?

Do projects with CI have (higher/similar/lower) code quality?

Are developers on projects with CI (more/similar/less) productive?

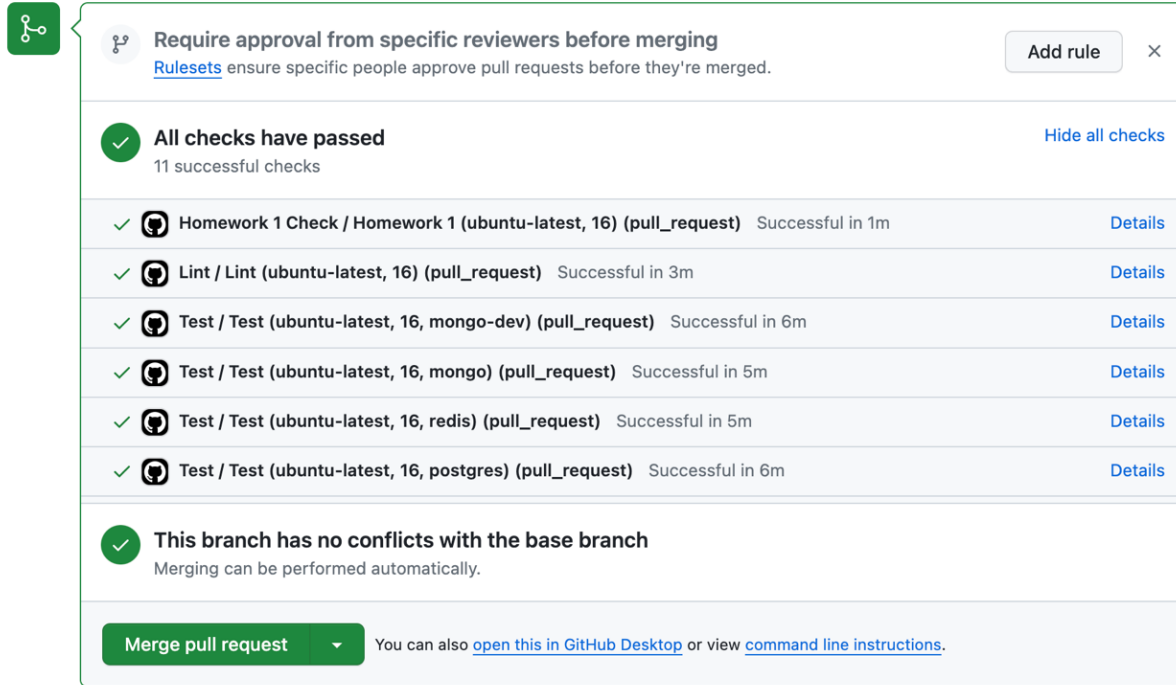


# Observation



**CI helps us catch errors  
before others see them**


45













# CI can run static and dynamic analysis





The screenshot displays a GitHub pull request interface. At the top, a green icon with a network diagram is visible. Below it, a section titled "Require approval from specific reviewers before merging" includes a link to "Rulesets" and a description: "ensure specific people approve pull requests before they're merged." To the right of this section is a button labeled "Add rule" and a close icon. Below this, a green checkmark icon is followed by the text "All checks have passed" and "11 successful checks". To the right of this text is a link labeled "Hide all checks". Below this, a list of checks is shown, each with a green checkmark icon, a GitHub logo, a description, a status, and a "Details" link. The checks are: "Homework 1 Check / Homework 1 (ubuntu-latest, 16) (pull\_request)" (Successful in 1m), "Lint / Lint (ubuntu-latest, 16) (pull\_request)" (Successful in 3m), "Test / Test (ubuntu-latest, 16, mongo-dev) (pull\_request)" (Successful in 6m), "Test / Test (ubuntu-latest, 16, mongo) (pull\_request)" (Successful in 5m), "Test / Test (ubuntu-latest, 16, redis) (pull\_request)" (Successful in 5m), and "Test / Test (ubuntu-latest, 16, postgres) (pull\_request)" (Successful in 6m). Below the list, a green checkmark icon is followed by the text "This branch has no conflicts with the base branch" and "Merging can be performed automatically." At the bottom, a green button labeled "Merge pull request" is shown, followed by the text "You can also [open this in GitHub Desktop](#) or view [command line instructions](#)."

 **Require approval from specific reviewers before merging** [Rulesets](#) ensure specific people approve pull requests before they're merged. [Add rule](#) 

 **All checks have passed** [Hide all checks](#)  
11 successful checks

	 <b>Homework 1 Check / Homework 1 (ubuntu-latest, 16) (pull_request)</b> Successful in 1m <a href="#">Details</a>
	 <b>Lint / Lint (ubuntu-latest, 16) (pull_request)</b> Successful in 3m <a href="#">Details</a>
	 <b>Test / Test (ubuntu-latest, 16, mongo-dev) (pull_request)</b> Successful in 6m <a href="#">Details</a>
	 <b>Test / Test (ubuntu-latest, 16, mongo) (pull_request)</b> Successful in 5m <a href="#">Details</a>
	 <b>Test / Test (ubuntu-latest, 16, redis) (pull_request)</b> Successful in 5m <a href="#">Details</a>
	 <b>Test / Test (ubuntu-latest, 16, postgres) (pull_request)</b> Successful in 6m <a href="#">Details</a>

 **This branch has no conflicts with the base branch**  
Merging can be performed automatically.

[Merge pull request](#)  You can also [open this in GitHub Desktop](#) or view [command line instructions](#).

# 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

**For problems we can't  
easily automate, we can  
perform code review**

# Boeing Model 299 test on October 30, 1935.

- Plane crashed because of locked elevator control surface (opposite effect of MCAS)
- 4 engines were deemed “too complex”
- Test pilots developed checklists to help them fly



# Checklists help manage complex processes



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

# Dr. Peter Pronovost

- Inspired by B-17 Story
- After checklist, ten-day line-infection rate went from eleven per cent to zero
- In 15 months, only two line infections occurred
- For one hospital, the checklist had prevented forty-three infections and eight deaths, and saved \$2M



## The Pronovost Checklist

Central venous catheters, or lines, are used for medications, blood, fluids or nutrition and can stay in for days or weeks. But bacteria can grow in the line and spread a type of infection to the bloodstream, which causes death in one out five patients who contract it. This five-step checklist for doctors and nurses to use before inserting a line can prevent infections and death.

1. **Wash hands** with soap and water or an alcohol cleanser
2. **Wear sterile clothing**—a mask, gloves, and hair covering—and cover patient with a sterile drape, except for a very small hole where the line goes in
3. **Clean patient's skin** with chlorhexidine (a type of soap) when the line is put in
4. **Avoid veins in arm and leg**, which are more likely to get infected than veins in chest
5. **Check the line for infection** each day and remove when no longer needed

Source: Dr. Peter Pronovost

<https://www.wsj.com/articles/SB10001424052748704364004576131963185893084>



# Difference between Pilot and Doctor error?

Which is Developer error more like?

# How to create a checklist?

- Start with problems we have seen before
  - “Safety regulations are written in blood”
- Justify why this is not automatable
- Not all checklist items need to be very specific
  - An item could be “does this team know we are proposing this change”

# Activity: Create a checklist

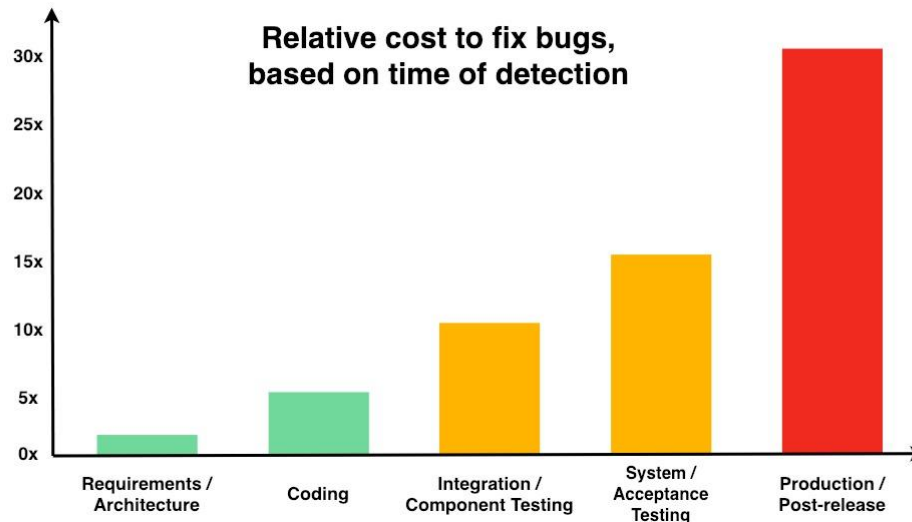
- In pairs, think about dumb mistakes your “friend” made the last time they were coding.
  - Write your names on a piece of paper.
  - Write down two checklist items that would have caught those errors.
- Divide into teams: left and right sides of the classroom.
- Which team had the most unique/good entries in their list?



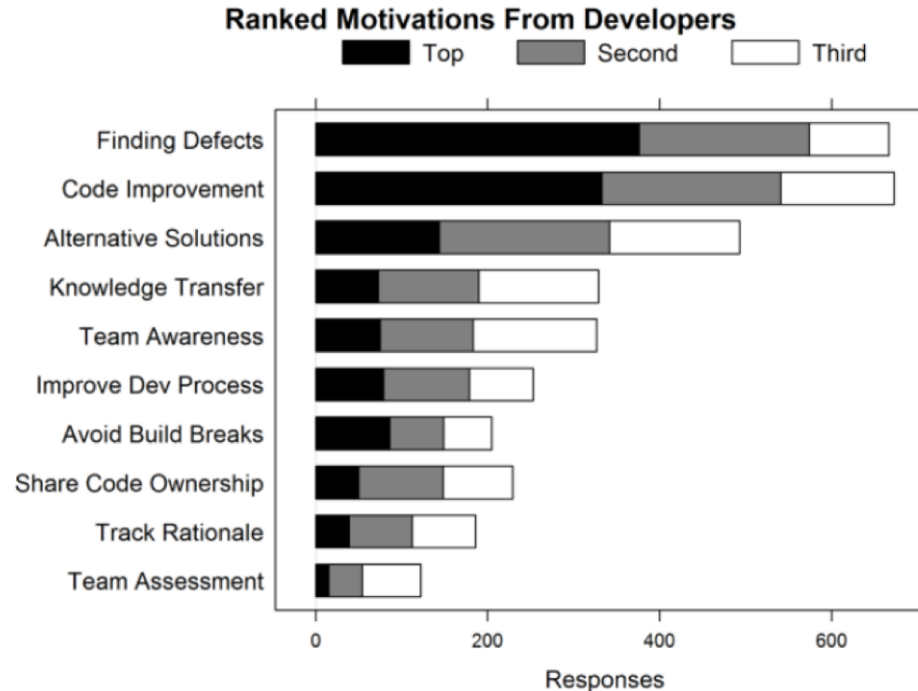
# Expectations and Outcomes for code review

# Motivation

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

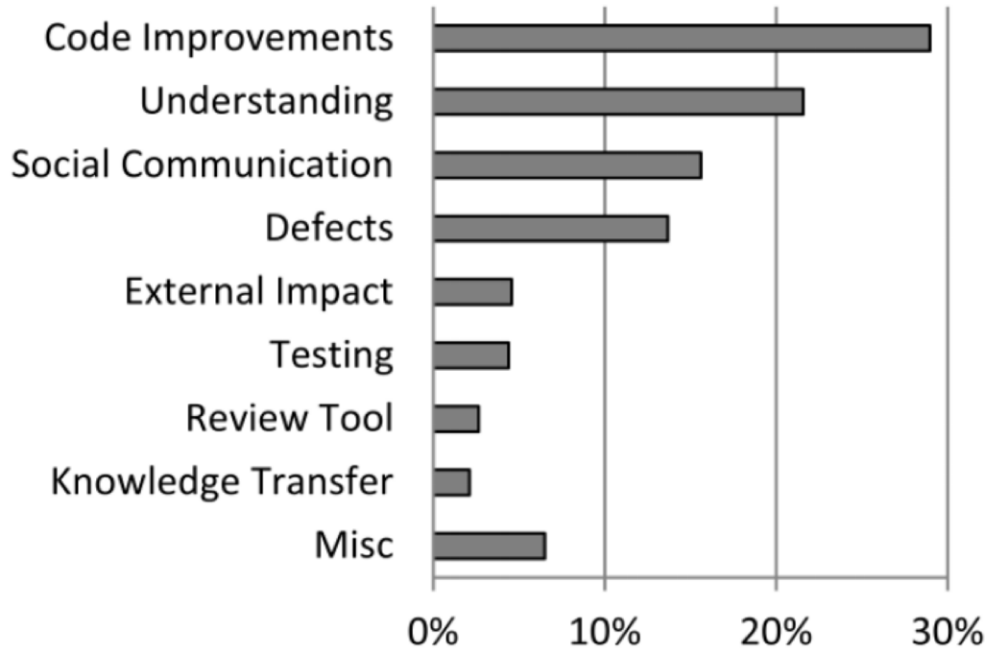


# Code Review at Microsoft



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

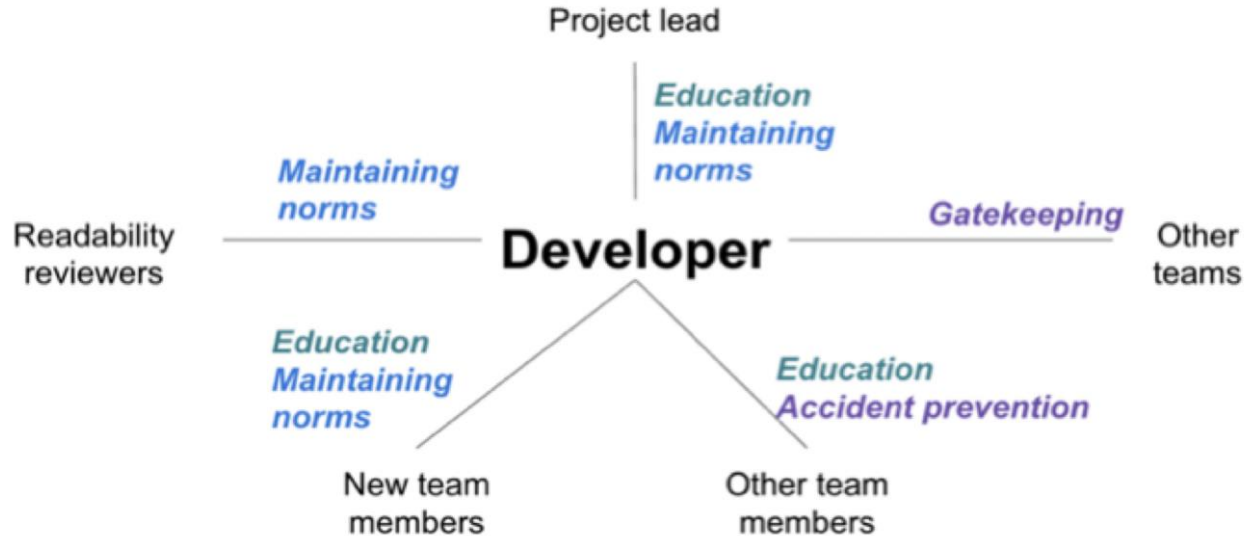
- Low quality of code reviews
  - Reviewers look for easy errors, as formatting issues
  - Miss serious errors
- Understanding is the main challenge
  - Understanding the reason for a change
  - Understanding the code and its context
  - Feedback channels to ask questions often needed
- No quality assurance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

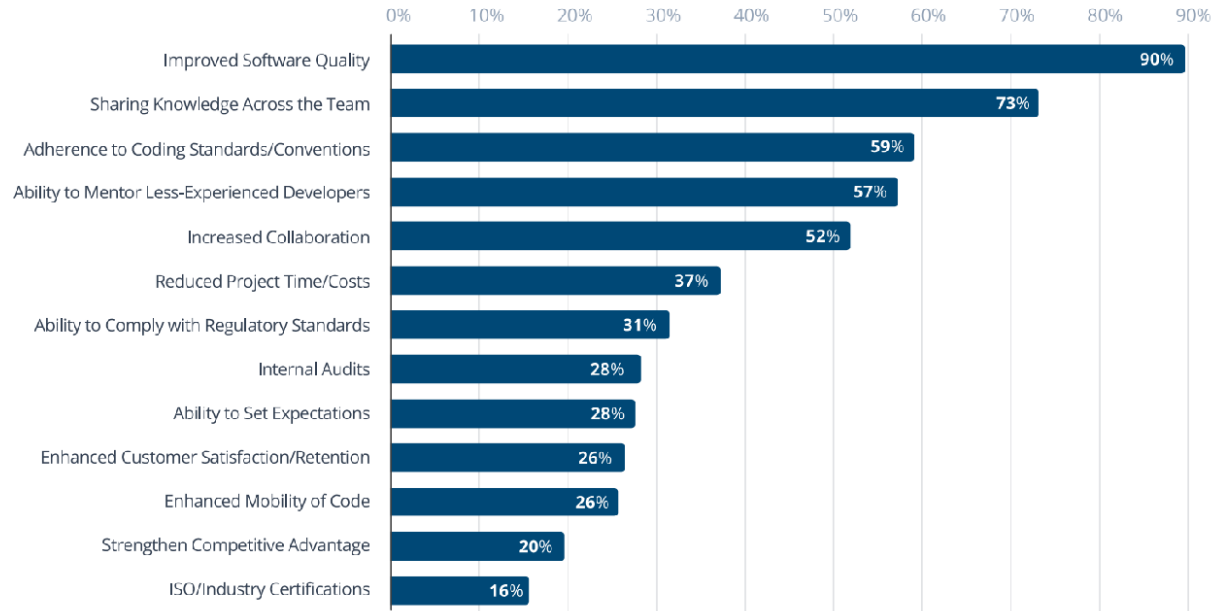
Caitlin Sadowski, Emma Söderberg, Luke Church, Michal Sipko, and Alberto Bacchelli. 2018. Modern Code Review: A Case Study at Google. International Conference on Software Engineering

# Reviewing Relationships



# 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
- CI can avoid embarrassment
- Identify defects, not alternatives; do not criticize coder
  - “*you* didn't initialize variable a” -> “I 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

# Risk Analysis:

- Probability a human makes a mistake: Very Likely
- Severity: ranges, but could be extensive

Solution:

Use CI to catch your mistakes, make you look better, and mitigate your risks!

Use Code review to teach and learn

