Lecture 12: Code review

CS 5150, Spring 2025

Administrative Reminders

- Peer reviews: Due Mar 7
- Presentation scheduling
  - Schedule with client by March 25
- Report 3: Due Mar 21
- In-class exam 1: Thurs, Mar 27

Previously in 5150 ….

- Version Control Systems
- Git Basics
- Branching
- Git/VCS Terms

Code reviews

Can we catch human errors?

Can we catch human error before we ship our code?

Can we automate tasks to prevent problems?

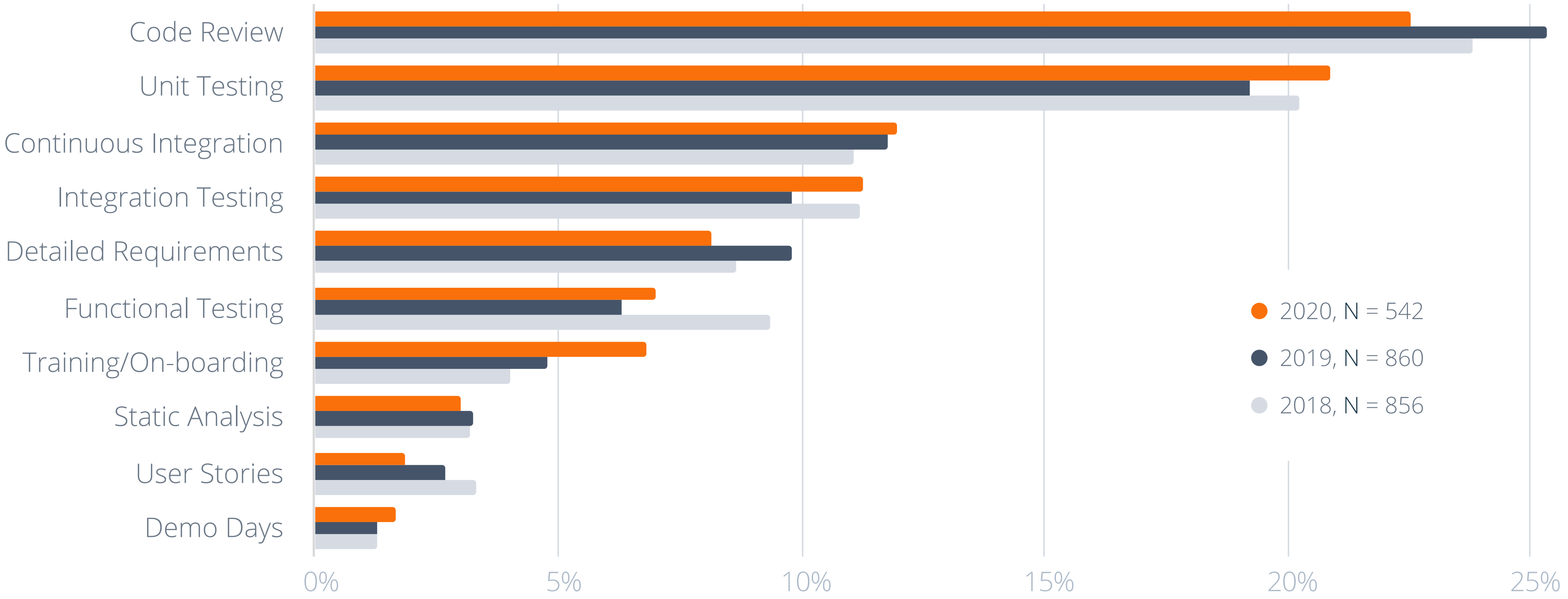
Approach

Automate what we can

Review what we cannot

Development processes

What is the number one thing a company can do to improve code quality? *fig.4*

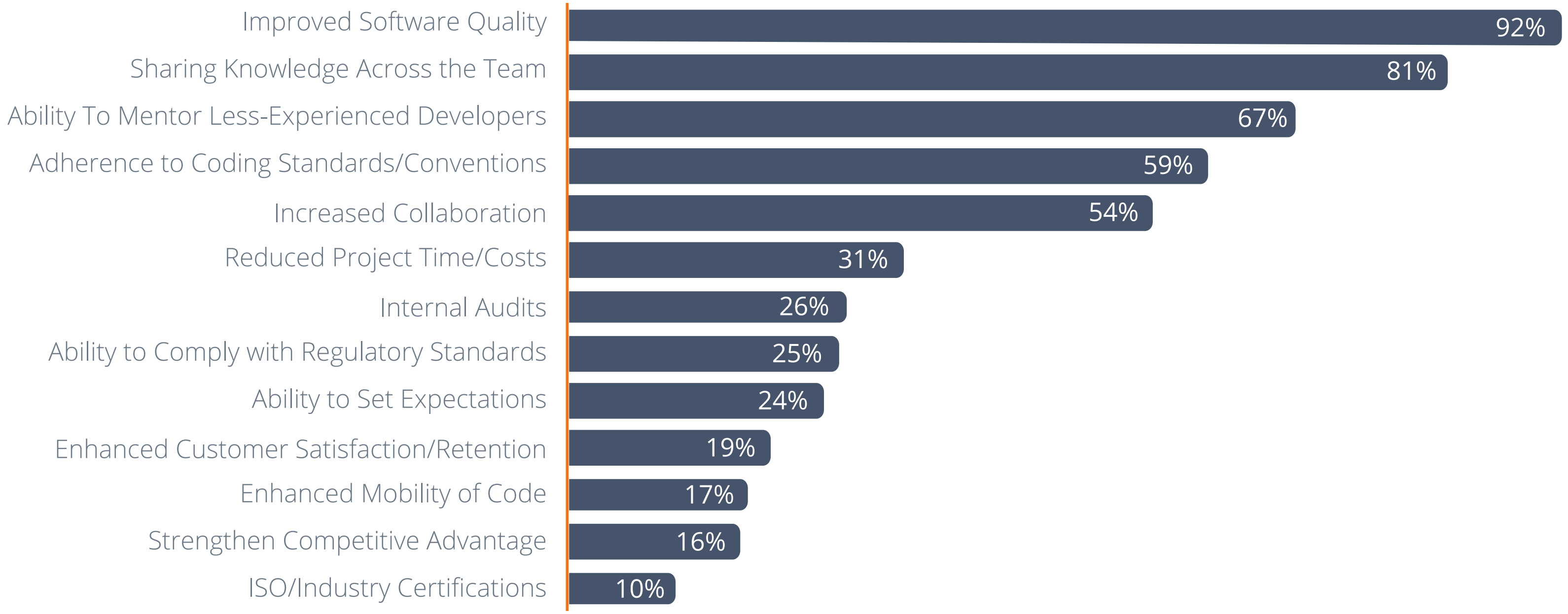


Beyond quality

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

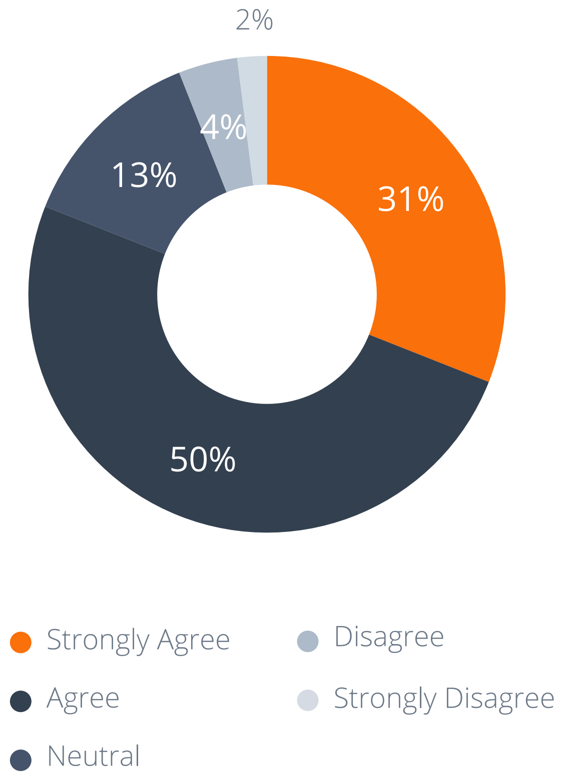
Select all that apply *fig.9*

N = 735



I often learn from others when I participate in code reviews. *fig.12*

N = 674



Code Review at Microsoft

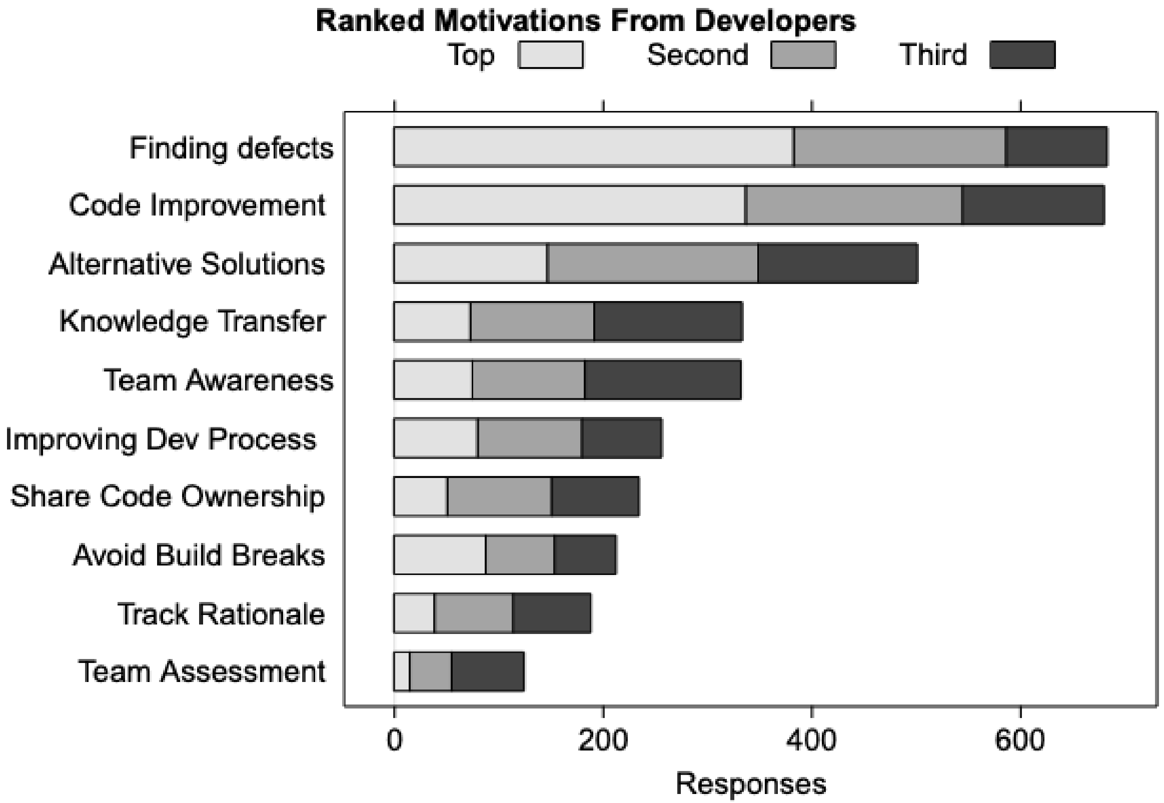


Fig. 3. Developers' motivations for code review.

Expectations, Outcomes, and Challenges of Modern Code Review. Bacchelli and Bird. ICSE 2013

Review spectrum

- Pair programming (XP)
  - Lacks independence
- Tool-assisted peer review
  - Asynchronous
  - Postpones structured collaboration
- Formal inspections
  - Maximizes benefits
  - Expensive

- Review all artifacts! (not just code)
  - *"If it is worth writing down and keeping, it is worth reviewing"*
  - Leverage collaborative documents, "track changes", etc.

  - Requirements, architecture, design, test plan, test results, ticket backlog, user manual, presentation slides, marketing materials, project plan, …

Writing reviewable code

- Keep changes small
- Clean branch history
  - Don't base on unmerged branches
  - Avoid intermediate back-merges
  - Commits should be logical, self-contained
- Don't mix reformatting, refactoring, and functional changes

- Style tips
  - Trailing commas (when allowed)
  - Arguments on separate lines
  - Autoformatting, static analysis
    - Help reviewer focus on content, not style/syntax

Reviewing code

- Review in context of purpose
  - Ideally traced to a ticket
  - Review documentation of context
- Understand existing code first
- Focus on correctness, broader implications
  - Hopefully leave details to tools
- Review testcases
- Ask questions, demand clear answers
  - Ensure issues are fully resolved
- Don't feel rushed/pressured

- Inspect the item, not the author
  - Shared ownership of total product
- Justify defects, refrain from neutral alternatives
- Allow author to decide how defects are resolved
- Avoid debates
  - If code is correct and consistent with team guidelines, allow it
  - If debate is necessary, resolve synchronously, then summarize
- Use a checklist

AI-Based Code Review

- **Github Copilot:** https://github.blog/changelog/2024-10-29-github-copilot-code-review-in-github-com-private-preview

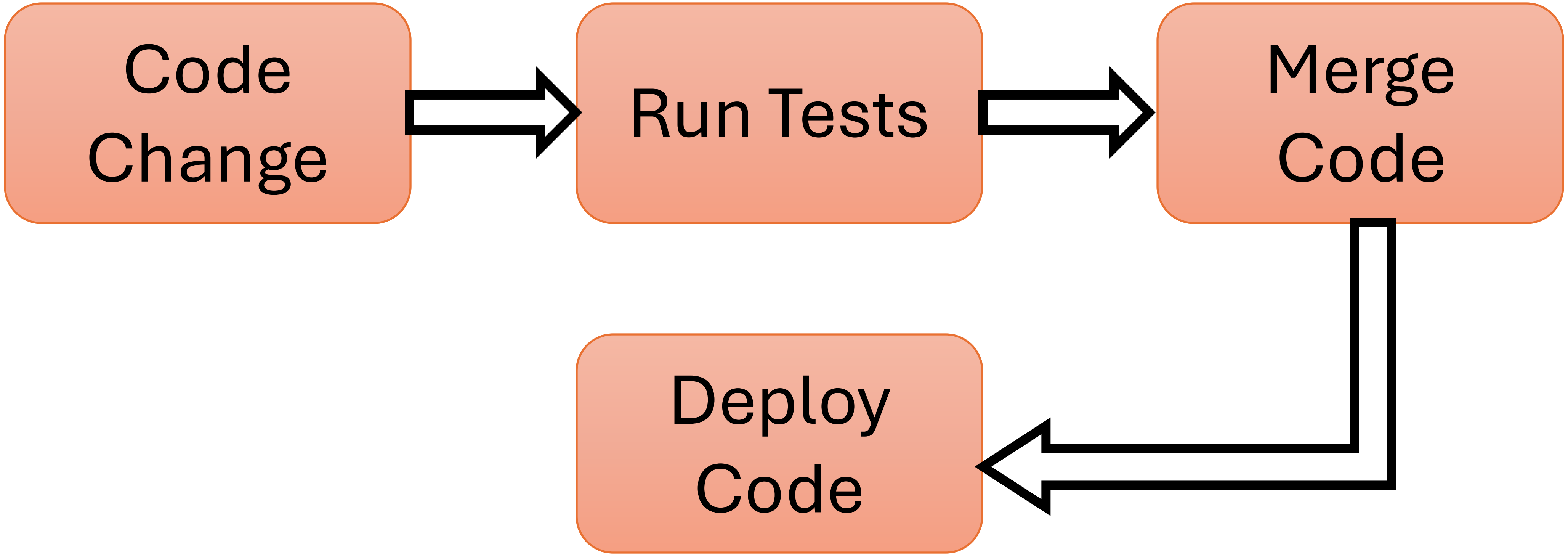
- Other solutions:
  - https://www.ibm.com/think/insights/ai-code-review (Watsonx)
  - Code Rabbit: https://www.coderabbit.ai

- Can AI check for: Correctness, Security, Performance, …?
- Can it replace static and dynamic analysis tools?

How to automate finding bugs?

CI/CD Pipelines

- Continuous Integration/Continuous Delivery
- Catch mistakes before you push code


Code Change

Run Tests

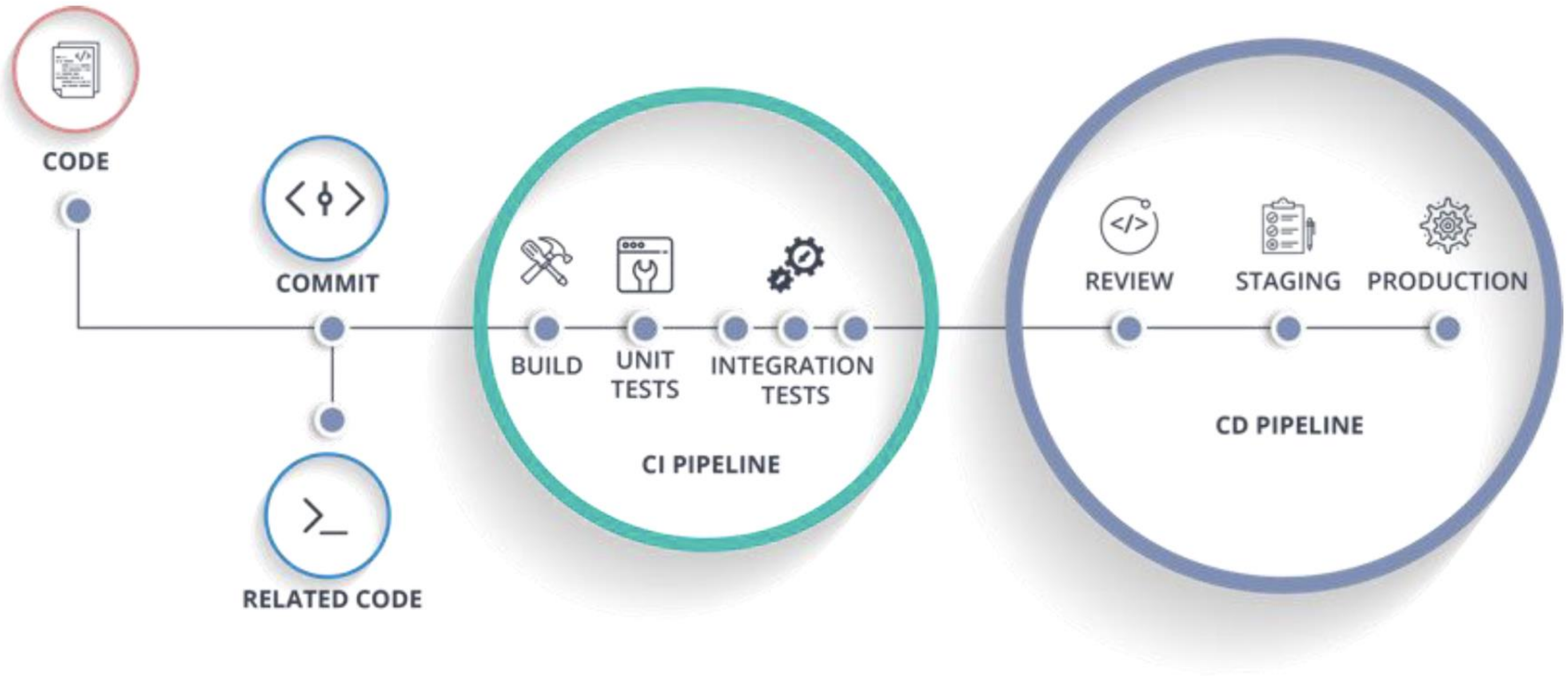
Merge Code

Deploy Code

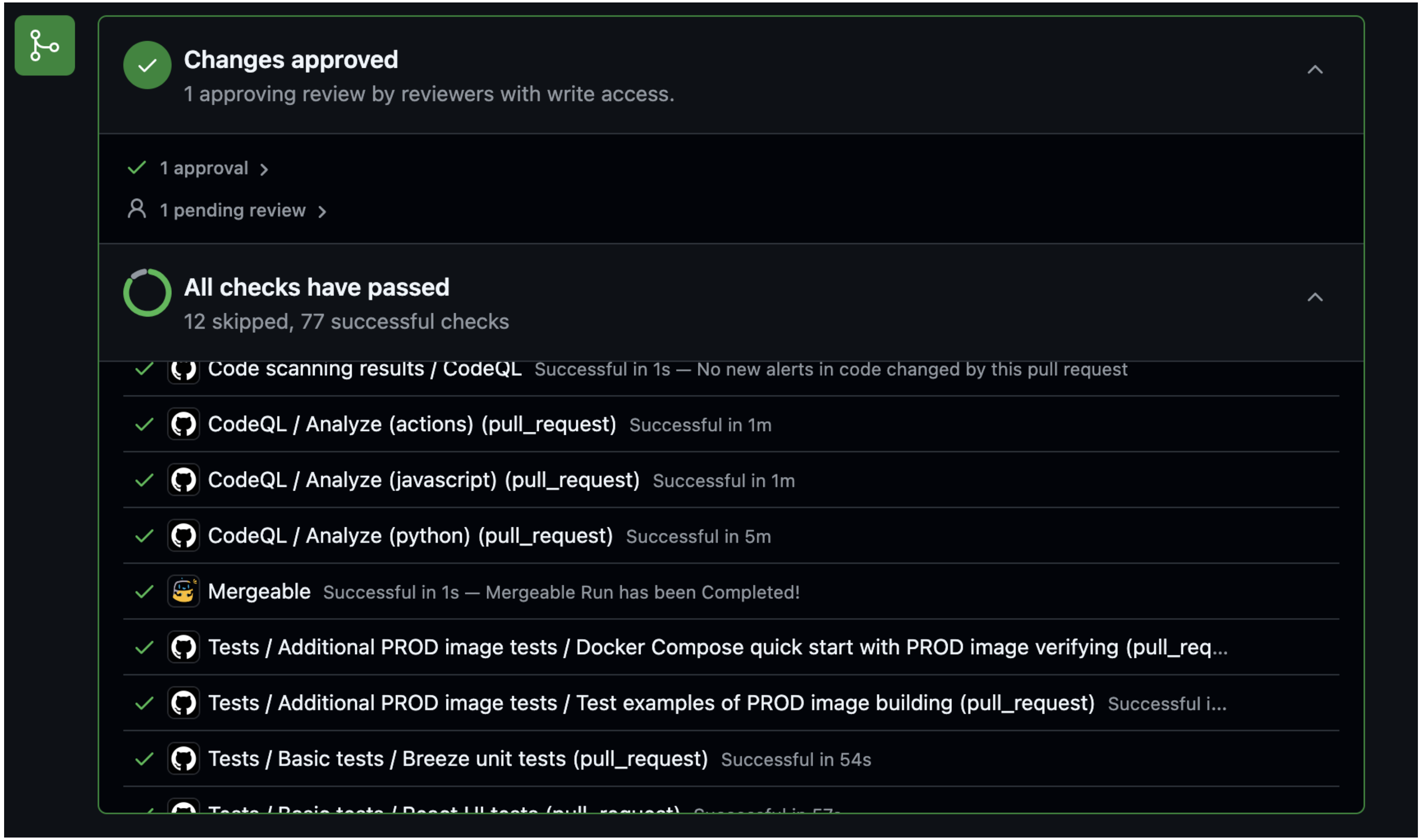
History of CI

- **1999**: Extreme Programming (XP) rule: Integrate Often
- **2000**: Martin Fowler posts “Continuous Integration” blog
- **2001**: First CI tool: Cruise Control
- **2005**: Hudson/Jenkins
- **2011**: Travis CI
- **2019**: Github Actions

Example CI/CD Pipeline



CI can run static and dynamic analysis



Changes approved
1 approving review by reviewers with write access.

✓ 1 approval ›
1 pending review ›

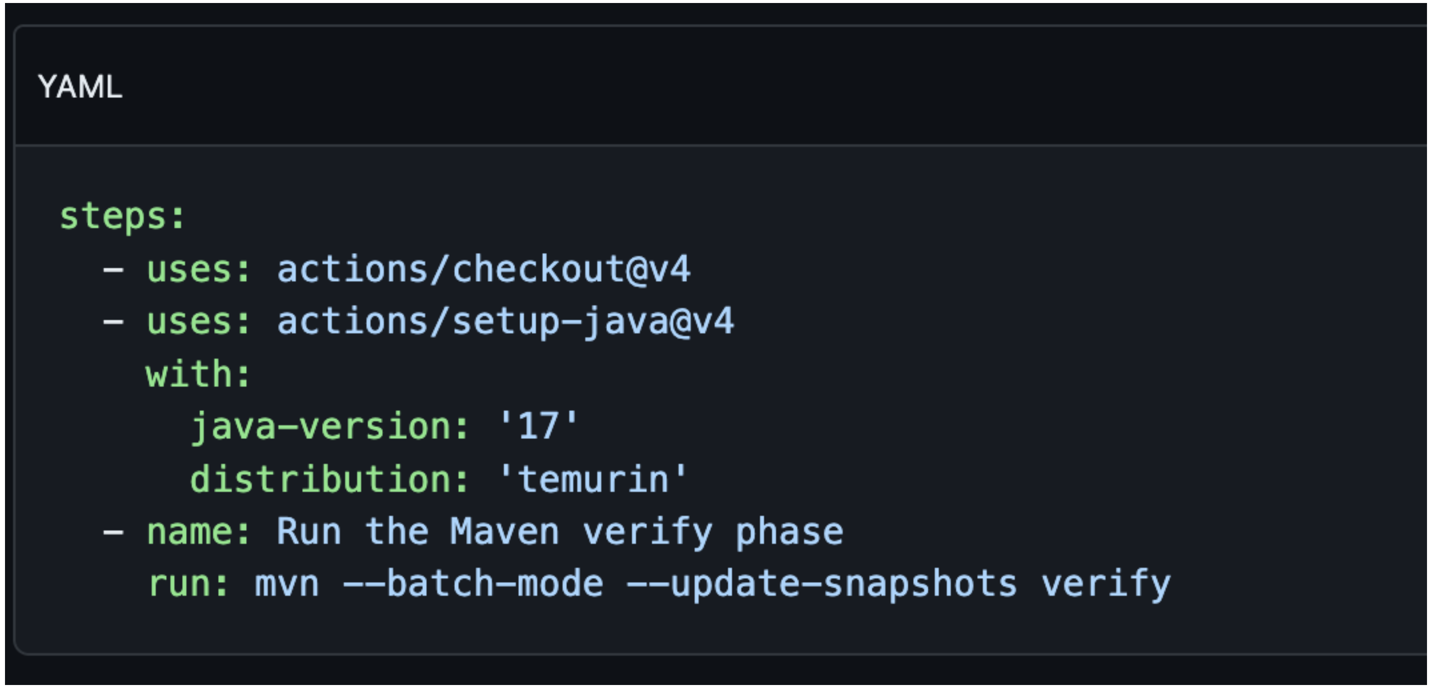
All checks have passed
12 skipped, 77 successful checks

✓ Code scanning results / CodeQL Successful in 1s — No new alerts in code changed by this pull request
✓ CodeQL / Analyze (actions) (pull_request) Successful in 1m
✓ CodeQL / Analyze (javascript) (pull_request) Successful in 1m
✓ CodeQL / Analyze (python) (pull_request) Successful in 5m
✓ Mergeable Successful in 1s — Mergeable Run has been Completed!
✓ Tests / Additional PROD image tests / Docker Compose quick start with PROD image verifying (pull_req...
✓ Tests / Additional PROD image tests / Test examples of PROD image building (pull_request) Successful i...
✓ Tests / Basic tests / Breeze unit tests (pull_request) Successful in 54s
✓ Tests / Basic tests / React UI tests (pull_request) Successful in 57s

https://github.com/apache/airflow/pull/47051

CI Configs: Demo

- Example for apache airflow
- https://github.com/apache/airflow/tree/main/.github/workflows
- Build and test java with maven



```yaml
steps:
  - uses: actions/checkout@v4
  - uses: actions/setup-java@v4
    with:
      java-version: '17'
      distribution: 'temurin'
  - name: Run the Maven verify phase
    run: mvn --batch-mode --update-snapshots verify
```

Poll: PollEv.com/cs5150sp25

- You are working on a feature branch for a critical bug fix in a team project. While developing, you realize that your current work is not yet ready to be committed (unsaved). An urgent hotfix from the main branch needs to be integrated into your feature branch. You want to ensure your unfinished work remains intact during this process.
- Identify the git operation(s) you would use to solve this.

Coding Conventions

Beyond code review

- How to ensure a healthy body of source code and preserve quality over time?
  - Explicit style guides and rules
  - Static analysis
  - Continuous enforcement

Past CS 5150 advice

- Write simple code
- Avoid risky programming constructs
- If code is difficult to read, rewrite it
- Include runtime verification
  - Verify class/data invariants after modification
  - Verify preconditions for parameter values
- Eliminate all warnings from source code
- Have a thorough set of test cases
- Expect to take longer to write and test production code in a production environment than in an academic one

Style guides

- Improve consistency of code
- Avoid unproductive arguments
- Guido van Rossum: Code is read much more than it is written!
- **Linters**: Black (python), Eslint (JS), CPPLint, Checkstyle (Java)

- https://google.github.io/styleguide/cppguide.html#Comments
- https://google.github.io/styleguide/pyguide#38-comments-and-docstrings
- https://google.github.io/styleguide/javaguide.html#s7-javadoc
- https://www.python.org/dev/peps/pep-0008
- https://github.com/airbnb/javascript
- Linux kernel style guide

Who writes 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

Static analysis

- Checks that can be done on the source code (without running it)
  - Syntax errors during compilation
  - Linters & compiler warnings
  - Style checks
  - Complexity measurement
- Notable tools
  - clang-static-analyzer (C++)
  - Semgrep, ErrorProne (Java), CodeQL
  - SonarQube
- Keep false positives low (ideally zero)
  - Allows checks to be run continuously without risking desensitization

What bugs can static analysis find?

- Dead code
  - Many subtle ways to introduce (bad ordering of if-statements, poorly-scoped early returns)
- Typos in names (indicated by unused parameters)
- Misleading indentation
- Unintentional overloads, risky implicit conversions (abs vs. std::abs)
- Unhandled cases, unintended fallthrough in switch statements
- Use of deprecated functionality
- Common mistakes
  - Using == when operand types override equals()
  - delete vs. delete[]
- Missing null pointer checks
- …

Style automation

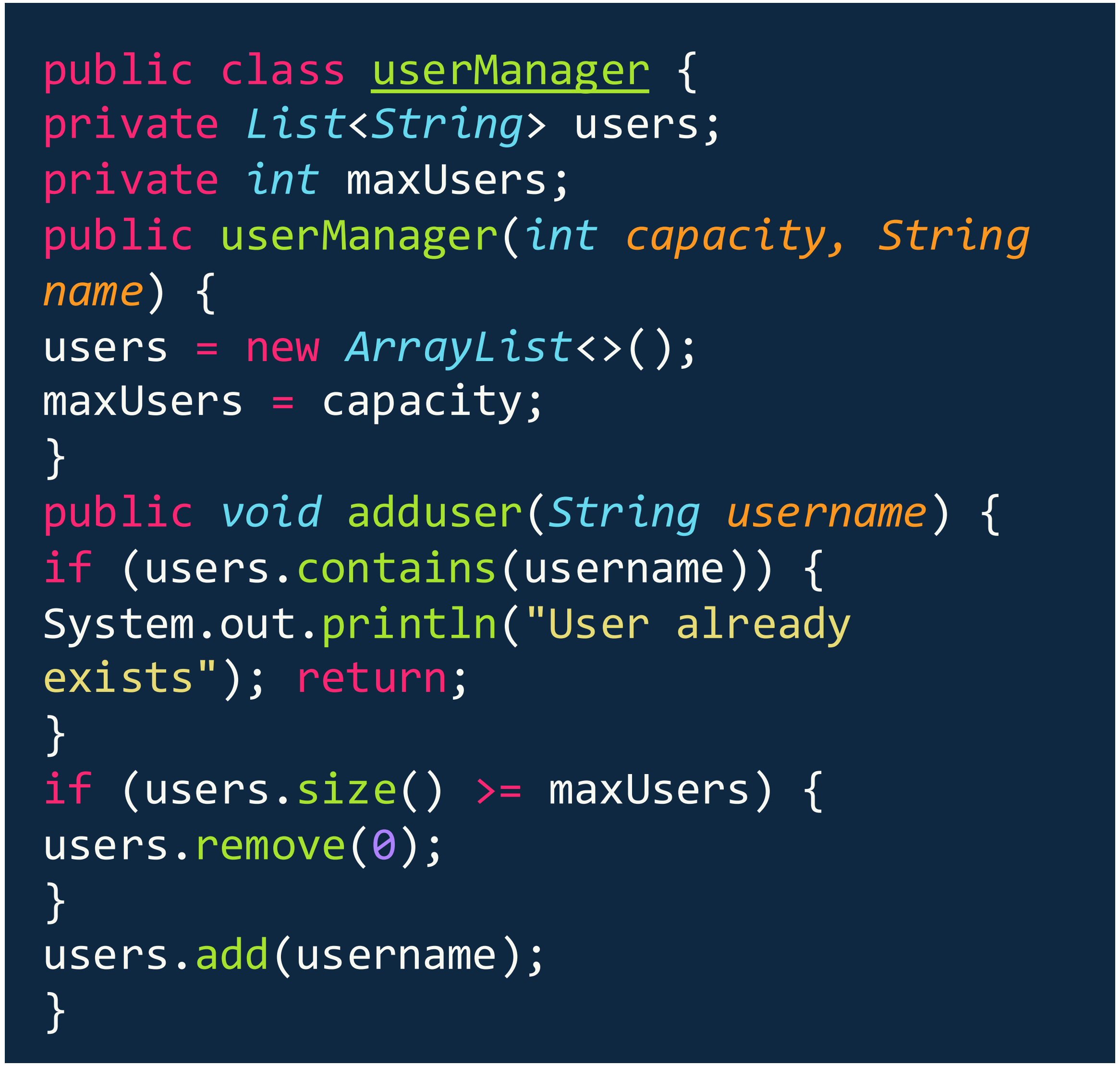
Advantages

- Zero human effort
- Uniform enforcement
- Prevent accidentally misleading style
- Can be applied after refactoring, synthesizing code
- Can update entire codebase when style rules change

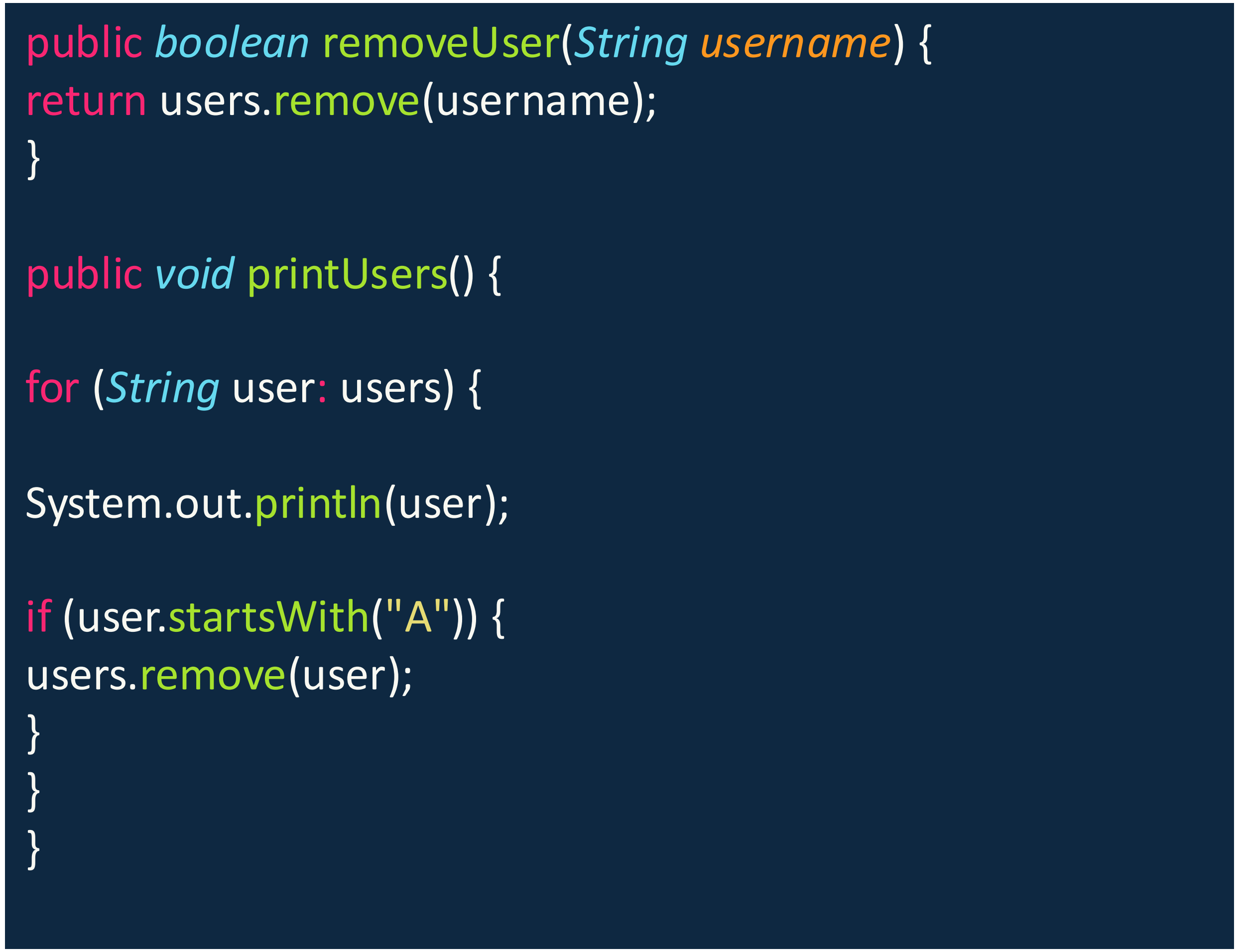
Disadvantages

- Can't reproduce all reasonable style rules
- Special-case exceptions are awkward
- Reformatting pollutes blame history

Discuss: Review this piece of code!



```java
public class userManager {
private List<String> users;
private int maxUsers;
public userManager(int capacity, String
name) {
users = new ArrayList<>();
maxUsers = capacity;
}
public void adduser(String username) {
if (users.contains(username)) {
System.out.println("User already
exists"); return;
}
if (users.size() >= maxUsers) {
users.remove(0);
}
users.add(username);
}
```



```java
public boolean removeUser(String username) {
return users.remove(username);
}

public void printUsers() {

for (String user: users) {

System.out.println(user);

if (user.startsWith("A")) {
users.remove(user);
}
}
}
```

Bugs

- No Null Checks in addUsers
- Case sensitive comparison of user name
- No input validation or sanitization
- Unsafe list manipulation
- Concurrent Modification of List

- Conventions: Inconsistent case for identifiers, unused parameters

Resources

- Read Software Engineering at Google:
  - Chapter 8: Style Guides and Rules
  - Chapter 9: Code Review

- For more on version control, read:
  - Chapter 16: Version Control and Branch Management