



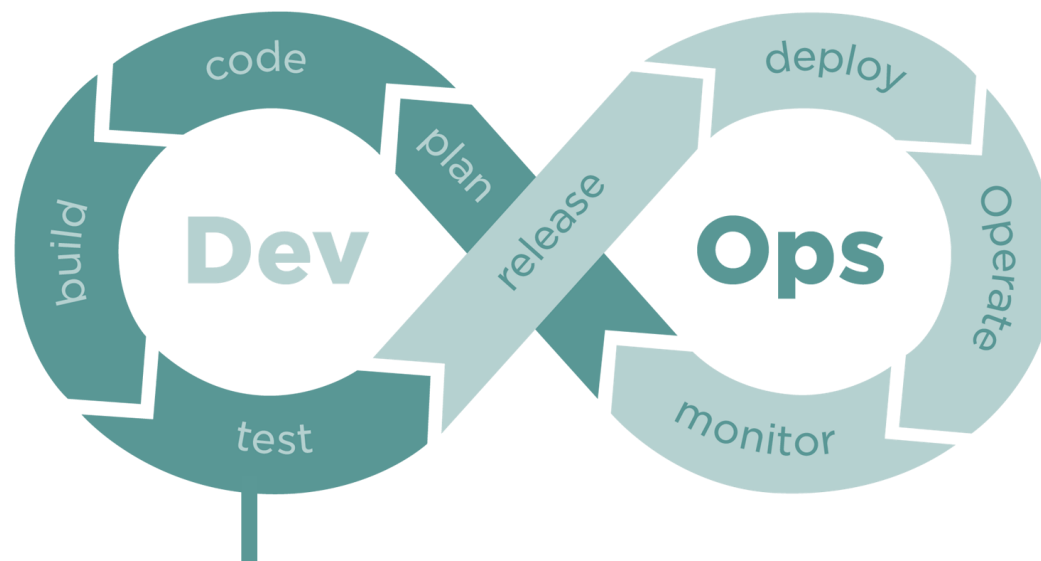
南方科技大学  
SOUTHERN UNIVERSITY OF SCIENCE AND TECHNOLOGY

# CS304 SOFTWARE ENGINEERING

Yida Tao

taoyd@sustech.edu.cn

# WHERE ARE WE NOW?



Quality control: **code quality**, testing



# WHERE ARE WE NOW?

## Quality control

- Code quality
- Testing

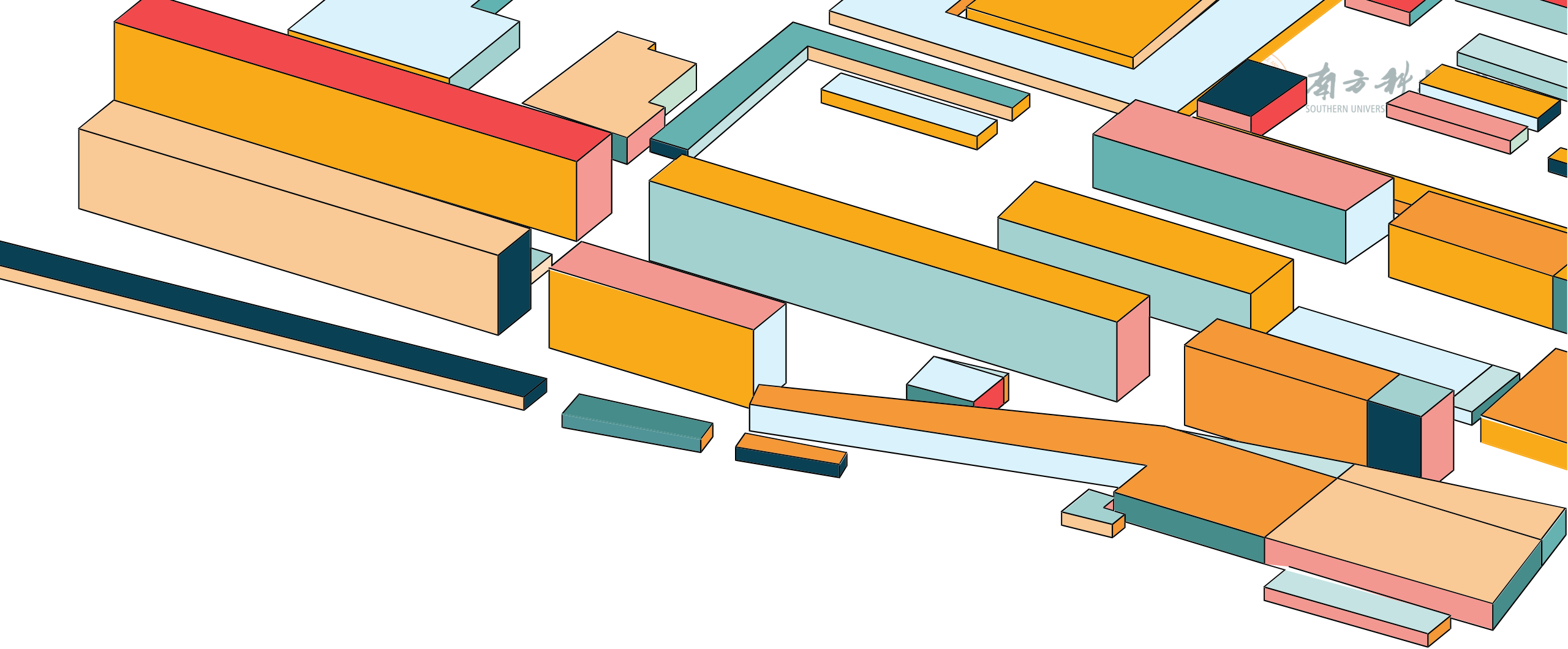
## Software quality

- Correctness (正确性)
- Readability (可读性)
- Maintainability (可维护性)
- Reliability (可靠性)
- Security (安全性)
- Efficiency (高效性)
- Portability (可移植性)



# LECTURE 7

- Motivating examples
- Metrics
- Linters
- Code review



# **CASE STUDY: BORROW BOOK**



# EXAMPLE OF BAD CODE: BORROW BOOK

```
1      public void borrowBook(String id1, String id2) {
2          if (! id1.equals("")) {
3              if (checkBooks(id1) < 5) {
4                  if (!(hasOverdueBooks(id1))) {
5                      if (! id2.equals("")) {
6                          if (! getBook(id2).isAllOut() && getBook(id2).blocked())
7                              updateStudentStatus(id1, id2); updateBookStatus(id2, id1);
8                          throw new IllegalArgumentException("This book can't be borrowed");
9                      } else
10                         throw new IllegalArgumentException("Book id can't be empty");
11                  }
12                  else
13                      throw new IllegalArgumentException("Overdue books exist");
14              } else
15                  throw new IllegalArgumentException("The number of books borrowed exceeds the limit");
16          } else
17              throw new IllegalArgumentException("Student id can't be empty");
18      }
```



# EXAMPLE OF BAD CODE: BORROW BOOK

```
1      public void borrowBook(String id1, String id2) {
2          if (! id1.equals("")) {
3              if (checkBooks(id1) < 5) {
4                  if (!(hasOverdueBooks(id1))) {
5                      if (! id2.equals("")) {
6                          if (! getBook(id2).isAllOut() && getBook(id2).blocked())
7                              updateStudentStatus(id1, id2); updateBookStatus(id2, id1);
8                          throw new IllegalArgumentException("This book can't be borrowed");
9                      } else
10                         throw new IllegalArgumentException("Book id can't be empty");
11                  }
12                  else
13                      throw new IllegalArgumentException("Overdue books exist");
14              } else
15                  throw new IllegalArgumentException("The number of books borrowed exceeds the limit");
16          } else
17              throw new IllegalArgumentException("Student id can't be empty");
18      }
```

Confusing identifiers: which is student ID, and which is book ID?



# EXAMPLE OF BAD CODE: BORROW BOOK

```
1      public void borrowBook(String id1, String id2) {
2          if (! id1.equals("")) {
3              if (checkBooks(id1) < 5) {
4                  if (!(hasOverdueBooks(id1))) {
5                      if (! id2.equals("")) {
6                          if (! getBook(id2).isAllOut() && getBook(id2).blocked())
7                              updateStudentStatus(id1, id2); updateBookStatus(id2, id1);
8                          throw new IllegalArgumentException("This book can't be borrowed");
9                      } else
10                         throw new IllegalArgumentException("Book id can't be empty");
11                  }
12                  else
13                      throw new IllegalArgumentException("Overdue books exist");
14              } else
15                  throw new IllegalArgumentException("The number of books borrowed exceeds the limit");
16          } else
17              throw new IllegalArgumentException("Student id can't be empty");
18      }
```

Misleading indentation: if statements should be nested, instead of the same level





# EXAMPLE OF BAD CODE: BORROW BOOK

```
1      public void borrowBook(String id1, String id2) {  
2          if (! id1.equals("")) {  
3              if (checkBooks(id1) < 5) {  
4                  if (!(hasOverdueBooks(id1))) {  
5                      if (! id2.equals("")) {  
6                          if (! getBook(id2).isAllOut() && getBook(id2).blocked())  
7                              updateStudentStatus(id1, id2); updateBookStatus(id2, id1);  
8                              throw new IllegalArgumentException("This book can't be borrowed");  
9                          } else  
10                             throw new IllegalArgumentException("Book id can't be empty");  
11                      }  
12                      else  
13                         throw new IllegalArgumentException("Overdue books exist");  
14                  } else  
15                     throw new IllegalArgumentException("The number of books borrowed exceeds the limit");  
16              } else  
17                 throw new IllegalArgumentException("Student id can't be empty");  
18          }
```

Deep nesting: 5 nested if statements, hard to understand



# EXAMPLE OF BAD CODE: BORROW BOOK

```
1      public void borrowBook(String id1, String id2) {
2          if (! id1.equals("")) {
3              if (checkBooks(id1) < 5) {
4                  if (!(hasOverdueBooks(id1))) {
5                      if (! id2.equals("")) {
6                          if (! getBook(id2).isAllOut() && getBook(id2).blocked())
7                              updateStudentStatus(id1, id2); updateBookStatus(id2, id1);
8                          throw new IllegalArgumentException("This book can't be borrowed");
9                      } else
10                     throw new IllegalArgumentException("Book id can't be empty");
11                  }
12                  else
13                     throw new IllegalArgumentException("Overdue books exist");
14              } else
15                 throw new IllegalArgumentException("The number of books borrowed exceeds the limit");
16          } else
17             throw new IllegalArgumentException("Student id can't be empty");
18      }
```

Improper exception type: only line 10 and line 17 are IllegalArgumentException.



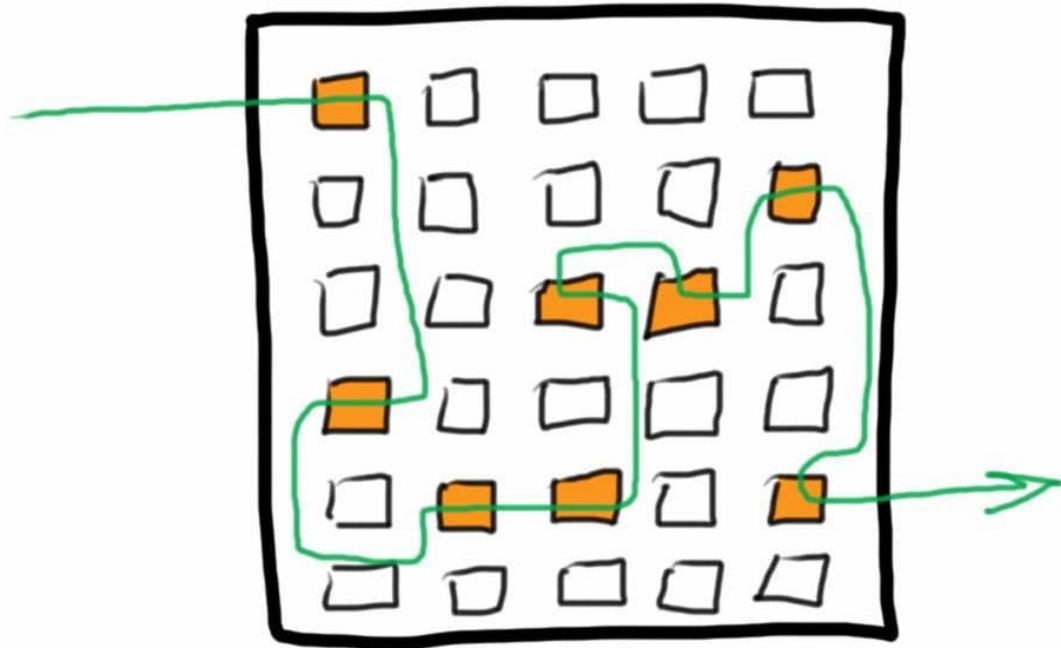
# EXAMPLE OF BAD CODE: BORROW BOOK

```
1      public void borrowBook(String id1, String id2) {  
2          if (! id1.equals("")) {  
3              if (checkBooks(id1) < 5) {  
4                  if (!(hasOverdueBooks(id1))) {  
5                      if (! id2.equals("")) {  
6                          if (! getBook(id2).isAllOut() && getBook(id2).blocked())  
7                              updateStudentStatus(id1, id2); updateBookStatus(id2, id1);  
8                          throw new IllegalArgumentException("This book can't be borrowed");  
9                      } else  
10                         throw new IllegalArgumentException("Book id can't be empty");  
11                  }  
12                  else  
13                      throw new IllegalArgumentException("Overdue books exist");  
14              } else  
15                  throw new IllegalArgumentException("The number of books borrowed exceeds the limit");  
16          } else  
17              throw new IllegalArgumentException("Student id can't be empty");  
18      }
```

Can you spot at least 2 bugs in the code?

# CLEAN CODE VS. BAD CODE

Finding our way  
through clean code



Finding our way  
through bad code



<https://lvivivity.com/how-to-write-good-code>



# HOW TO EVALUATE CODE QUALITY?

- Metrics
- Linters
- Code review



# WE NEED UNBIASED, QUANTIFIABLE INFORMATION



<https://thevaluable.dev/complexity-metrics-software/>



# LINES OF CODE

- LoC is easy to measure (`> wc -l` command)
- All software products produce LoC

Potential problems with the LoC metric?

| LOC           | projects             |
|---------------|----------------------|
| 450           | Expression Evaluator |
| 2,000         | Sudoku               |
| 100,000       | Apache Maven         |
| 500,000       | Git                  |
| 3,000,000     | MySQL                |
| 15,000,000    | gcc                  |
| 50,000,000    | Windows 10           |
| 2,000,000,000 | Google (MonoRepo)    |





# LINES OF CODE - NORMALIZING

```
for (i = 0; i < 100; i += 1) printf("hello");
```

- LoC needs to be **normalized** to be meaningful and comparable

```
for (  
    i = 0;  
    i < 100;  
    i += 1  
){  
    printf("hello");  
}
```

- Ignore comments and empty lines
- Count statement (logical lines) instead of textual lines
- See cLoc  
(<https://github.com/AIDanial/cloc>)



Higher-level languages are more expressive than lower-level languages. Each line of code says more.

- [illegible]



# CYCLOMATIC COMPLEXITY

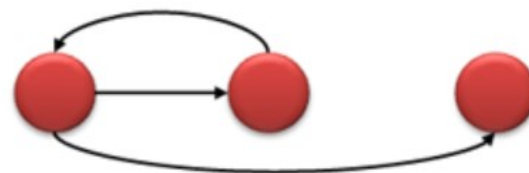
- Cyclomatic complexity (圈复杂度) is a software metric used to indicate the logic complexity of a program.
- Cyclomatic complexity is computed using the control-flow graph (控制流图) of the program

Core idea: the complexity of code depends on the number of decisions in the code (if, while, for)

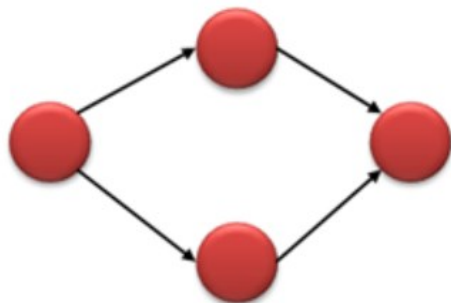
Sequence



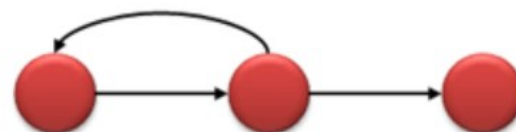
While



If-then-else



Until



# CONTROL FLOW GRAPH

Decisions are **cycles**



# CALCULATE CYCLOMATIC COMPLEXITY

Approach 1:  $V(G) = P + 1$

P: Number of branch nodes (e.g., if, for, while, case)

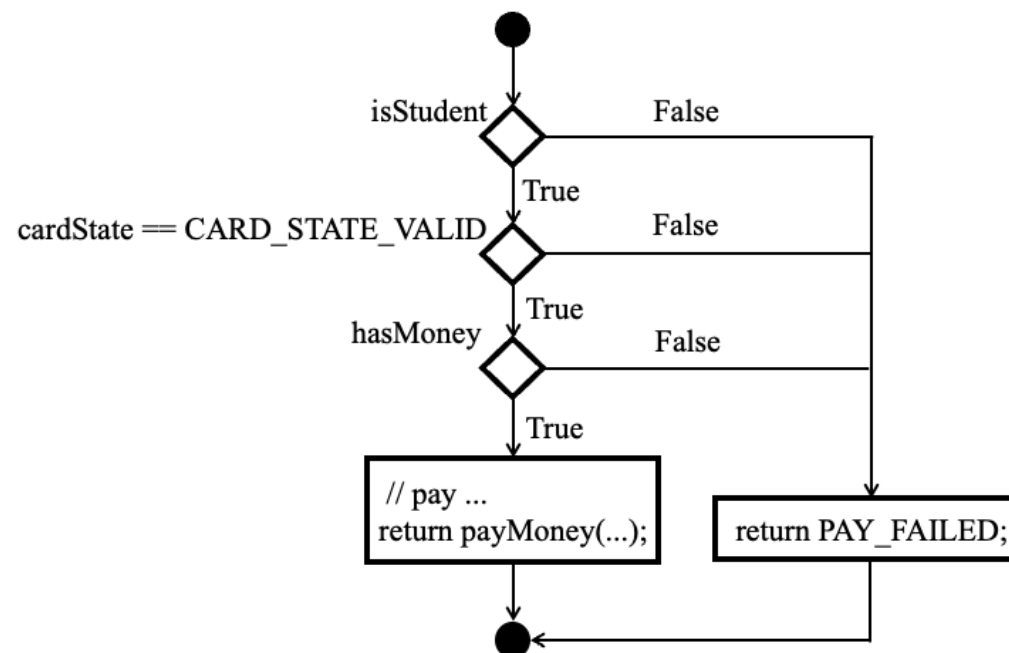
Approach 2:  $V(G) = E - N + 2$

E: Number of edges

N: Number of nodes

# CALCULATE CYCLOMATIC COMPLEXITY

```
1  if (isStudent) {  
2    if (cardState == CARD_STATE_VALID) {  
3      if (hasMoney) {  
4        // pay ...  
5        return payMoney(...);  
6      }  
7    }  
8  }  
9  return PAY_FAILED;
```



Cyclomatic complexity = 3 + 1 = 4



# INTERPRETING CYCLOMATIC COMPLEXITY

| Cyclomatic complexity | Code                        | Testability | Maintenance cost |
|-----------------------|-----------------------------|-------------|------------------|
| 1-10                  | Structured and well written | High        | Low              |
| 10-20                 | Complex                     | Medium      | Medium           |
| 20-40                 | Very complex                | Low         | High             |
| >40                   | Unreadable                  | Very low    | Very high        |

<https://www.guru99.com/cyclomatic-complexity.html>



# PITFALLS ON CYCLOMATIC COMPLEXITY

High cyclomatic complexity means poor readability?

- The code got the cyclomatic complexity of 14
- Seems readable

```
String getMonthName (int month) {  
    switch (month) {  
        case 0: return "January";  
        case 1: return "February";  
        case 2: return "March";  
        case 3: return "April";  
        case 4: return "May";  
        case 5: return "June";  
        case 6: return "July";  
        case 7: return "August";  
        case 8: return "September";  
        case 9: return "October";  
        case 10: return "November";  
        case 11: return "December";  
        default:  
            throw new IllegalArgumentException();  
    }  
}
```



# PITFALLS ON CYCLOMATIC COMPLEXITY

```
String getWeight(int i) {  
    if (i <= 0) {  
        return "no weight";  
    }  
    if (i < 10) {  
        return "light";  
    }  
    if (i < 20) {  
        return "medium";  
    }  
    if (i < 30) {  
        return "heavy";  
    }  
    return "very heavy";  
}
```

```
int sumOfNonPrimes(int limit) {  
    int sum = 0;  
    OUTER: for (int i = 0; i < limit; ++i) {  
        if (i <= 2) {  
            continue;  
        }  
        for (int j = 2; j < i; ++j) {  
            if (i % j == 0) {  
                continue OUTER;  
            }  
        }  
        sum += i;  
    }  
    return sum;  
}
```

Both have a cyclomatic complexity of 5. Same readability & maintainability?





# OO METRICS

ckjm — Chidamber and Kemerer Java Metrics

- Weighted Methods per Class (WMC)
- Depth of Inheritance Tree (DIT)
- Number of Children (NOC)
- Coupling between Object Classes (CBO)
- Lack of Cohesion in Methods (LCOM)
- .....



# WEIGHTED METHODS PER CLASS

- This metric is the sum of complexities of methods defined in a class.
- It therefore represents the complexity of a class as a whole
- Possible method complexities
  - 1 (# of methods)
  - LoC
  - # of method calls
  - Cyclomatic complexity

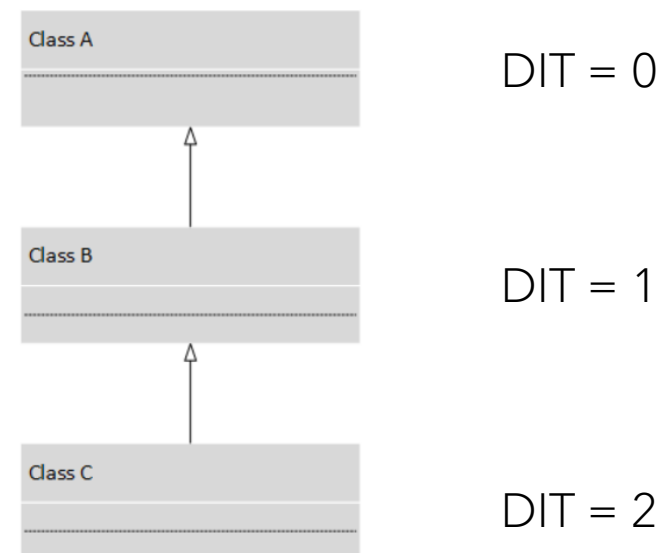
**WMC can be used to indicate the development and maintenance effort for the class.**



# DEPTH OF INHERITANCE TREE

DIT measures the maximum length between a node and the root node in a class hierarchy

The deeper a class is in the hierarchy, the more methods it inherits and so it is harder to predict its behavior





# NUMBER OF CHILDREN

- NOC indicates the number of immediate subclasses
- A class with a large NOC is probably very important and needs a lot of testing

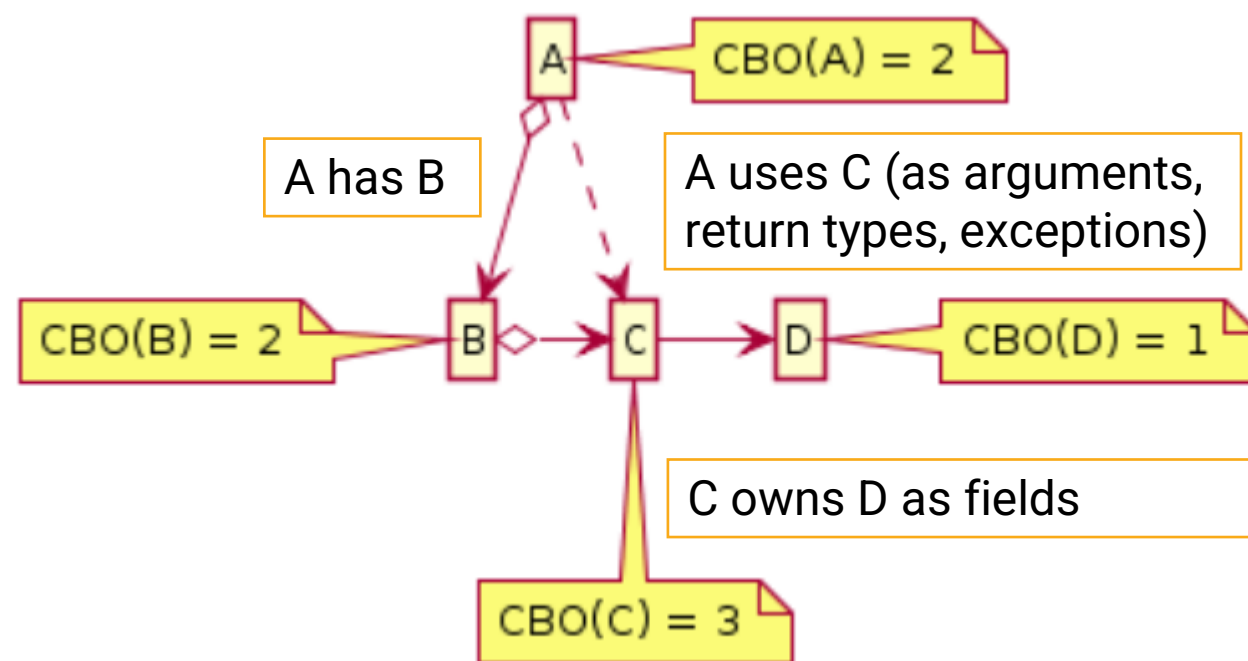


# COUPLING AND COHESION

- Dependences
  - Call methods, refer to classes, share variables
- Coupling
  - Dependences between modules (bad)
- Cohesion
  - Dependences within modules (good)

# COUPLING BETWEEN OBJECT CLASSES

- CBO represents the number of classes coupled to a given class.
- This coupling can occur through method calls, field accesses, arguments, return types, and exceptions.
- CBO doesn't care about the type or direction of a dependency





# COUPLING BETWEEN OBJECT CLASSES

- CBO=0: a class has no relationship to any other class in the system, and therefore should not be part of the system.
- CBO=1~4 is good, since it indicates that the class is loosely coupled.
- A higher CBO: the class is **too tightly coupled with** other classes, which would complicate testing and modification, and limit the possibilities of re-use. Consider **de-coupling** this class to become more independent.



# LACK OF COHESION IN METHODS

- LCOM counts the sets of methods **in a class** that are **not related** through the sharing of some of the class's fields.
- Considers all pairs of a class's methods:
  - Q: In some pairs, both methods access at least one common field of the class
  - P: In other pairs, the two methods do not share any common field accesses

$$LCOM = P - Q$$





# LACK OF COHESION IN METHODS

- Cohesiveness of methods is a sign of encapsulation
- A high LCOM could indicate that the design of the class is poor; it might be worthwhile to split the class into two or more classes.

$$LCOM = P - Q$$



# MEASUREMENT IS DIFFICULT





# HOW TO EVALUATE CODE QUALITY?

- Metrics
- Linters
- Code review



# LINTERS

- A linter (静态语法检查器) is a tool that analyzes **source code** for style, syntax, and potential errors that can lead to bugs and vulnerabilities
  - Enforces coding standards
  - Detect errors early
  - Ensure consistency and best practices
  - Improves readability and maintainability
- Rule-based: classic linters use predefined rules to identify issues in the code automatically

# LINTERS

## PMD 7.11.0

Release date: 28-February-2025

|                              |   |
|------------------------------|---|
| About                        | ▼ |
| User Documentation           | ▼ |
| Rule Reference               | ▲ |
| Apex Rules                   | ▼ |
| HTML Rules                   | ▼ |
| Java Rules                   | ▲ |
| Index                        |   |
| Best Practices               |   |
| Code Style                   |   |
| Design                       |   |
| Documentation                |   |
| Error Prone                  |   |
| Multithreading               |   |
| Performance                  |   |
| Security                     |   |
| Java Server Pages Rules      | ▼ |
| JavaScript Rules             | ▼ |
| Kotlin Rules                 | ▼ |
| Maven POM Rules              | ▼ |
| Modelica Rules               | ▼ |
| PLSQL Rules                  | ▼ |
| Salesforce Visualforce Rules | ▼ |
| Scala Rules                  | ▼ |


```
<rule ref="category/java/design.xml/CollapsibleIfStatements" />
```

## CouplingBetweenObjects

**Since:** PMD 1.04

**Priority:** Medium (3)

This rule counts the number of unique attributes, local variables, and return types within an object. A number higher than the specified threshold can indicate a high degree of coupling.

**This rule is defined by the following Java class:** [net.sourceforge.pmd.lang.java.rule.design.CouplingBetweenObjectsRule](#) 

### Example(s):

```
import com.Blah;
import org.Bar;
import org.Bardo;

public class Foo {
    private Blah var1;
    private Bar var2;

    //followed by many imports of unique objects
    ObjectC doWork() {
        Bardo var55;
        ObjectA var44;
        ObjectZ var93;
        return something();
    }
}
```

**This rule has the following properties:**

| Name      | Default Value | Description                     |
|-----------|---------------|---------------------------------|
| threshold | 20            | Unique type reporting threshold |



南方科技大学  
SOUTHERN UNIVERSITY OF SCIENCE AND TECHNOLOGY

### Table of Contents

- AbstractClassWithoutAnyMethod
- AvoidCatchingGenericException
- AvoidDeeplyNestedIfStmts
- AvoidRethrowingException
- AvoidThrowingNewInstanceOfSameException
- AvoidThrowingNullPointerException
- AvoidThrowingRawExceptionTypes
- AvoidUncheckedExceptionsInSignatures
- ClassWithOnlyPrivateConstructorsShouldBeFinal
- CognitiveComplexity
- CollapsibleIfStatements
- CouplingBetweenObjects
- CyclomaticComplexity
- DataClass
- DoNotExtendJavaLangError
- ExceptionAsFlowControl
- ExcessiveImports
- ExcessiveParameterList
- ExcessivePublicCount
- FinalFieldCouldBeStatic
- GodClass
- ImmutableField
- InvalidJavaBean
- LawOfDemeter



# LINTERS

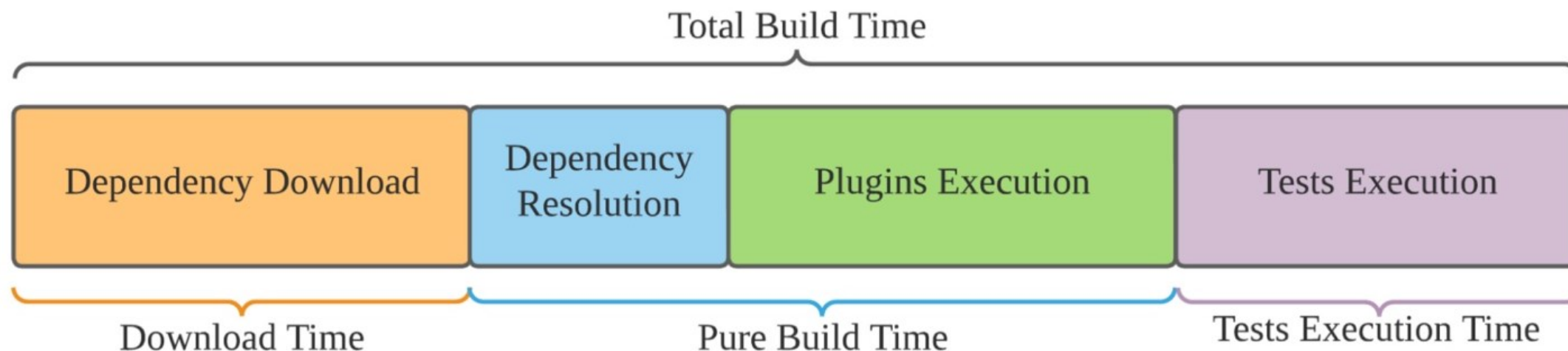
| Linters             | Programming Languages  |
|---------------------|------------------------|
| ESLint              | JavaScript, TypeScript |
| Infer               | Java, C++, Objective-C |
| FindBugs (SpotBugs) | Java                   |
| PyLint              | Python                 |
| Roslyn Analyzers    | C#                     |

Linters are especially useful for dynamically typed languages like JavaScript and Python, cuz the compilers of such languages typically do not enforce as many and as strict rules **prior to execution**



# LINTERS

Linters are often integrated in the build process as a step of quality checks.  
Linters are **static (work on source or byte code)**, without having to execute code.





# HOW TO EVALUATE CODE QUALITY?

- Metrics
- Linters
- Code review





# WHAT IS CODE REVIEW?

Code review (sometimes referred to as peer review) is a software quality assurance activity in which one or several people check a program mainly by viewing and reading parts of its source code (changes/commits)



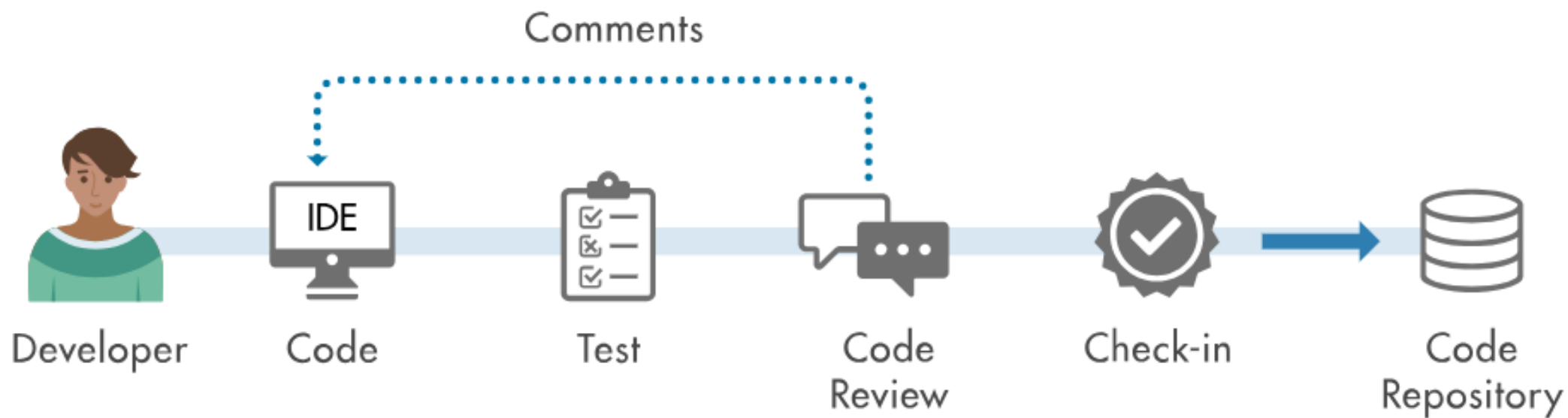
<https://smartbear.com/learn/code-review/what-is-code-review/>



# TYPES OF CODE CHANGES TO BE REVIEWED

- Modifications to existing code
  - Behavioral changes, improvements, optimizations
  - Bug fixes and rollbacks
- Entirely new code
  - Greenfield code review: review entirely new code
- Automatically generated code
  - E.g., refactoring tools, AI code generators

# CODE REVIEW WORKFLOW



<https://www.mathworks.com/discovery/code-review.html>



# CODE REVIEW GOALS

- The code is well-designed.
- The functionality is good for the users of the code.
- Any UI changes are sensible and look good.
- Any parallel programming is done safely.
- The code isn't more complex than it needs to be.
- Code has appropriate well-designed unit tests

<https://google.github.io/eng-practices/review/reviewer/looking-for.html>



# CODE REVIEW GOALS

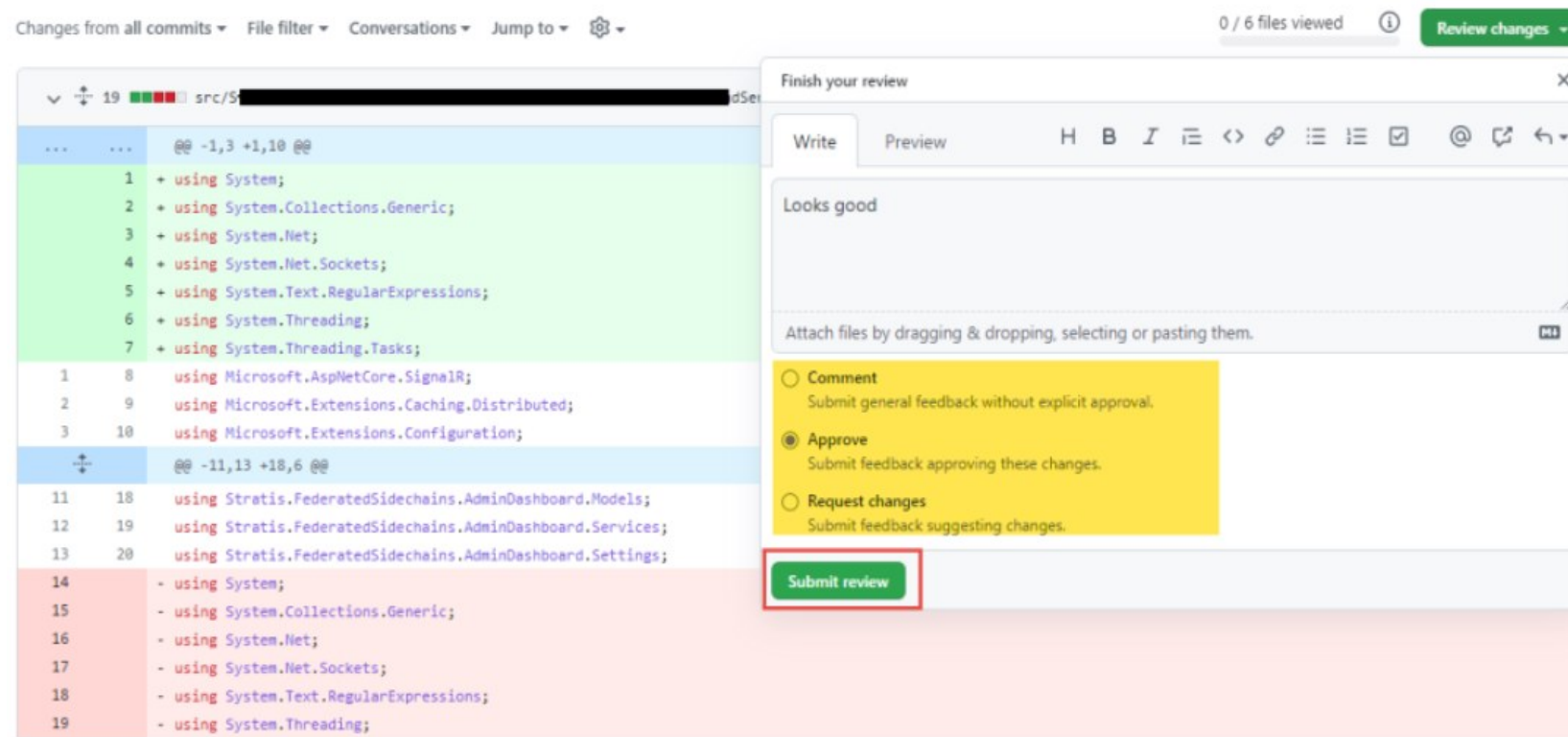
- The developer used clear names for everything.
- Comments are clear and useful, and mostly explain why instead of what.
- Code is appropriately documented
- The code conforms to the company's style guides.

Most style checks and correctness checks are performed automatically through techniques such as linters and automated testing

<https://google.github.io/eng-practices/review/reviewer/looking-for.html>

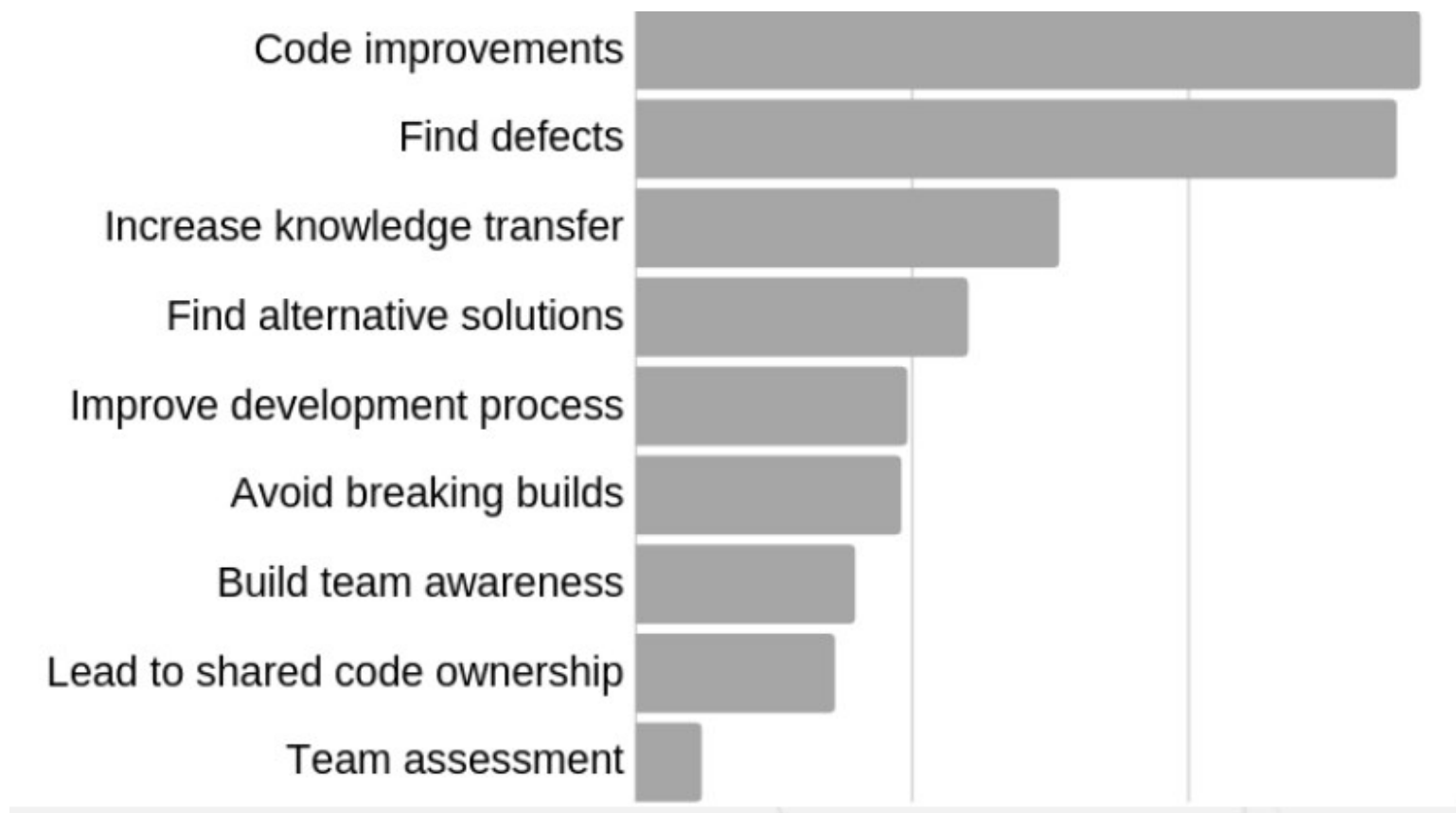


# CODE REVIEW TOOLS





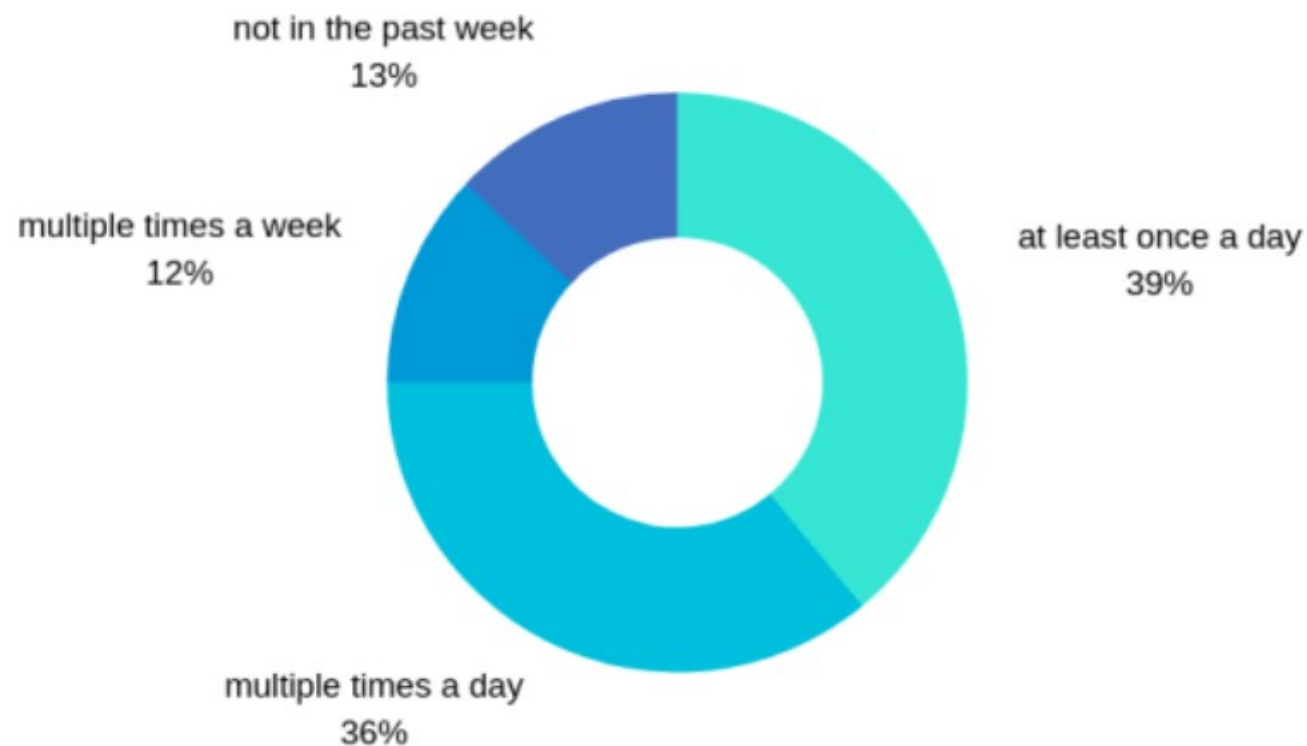
# CODE REVIEW BENEFITS



<https://www.freecodecamp.org/news/how-code-reviews-work-at-microsoft-4ebdea0cd0c0/>



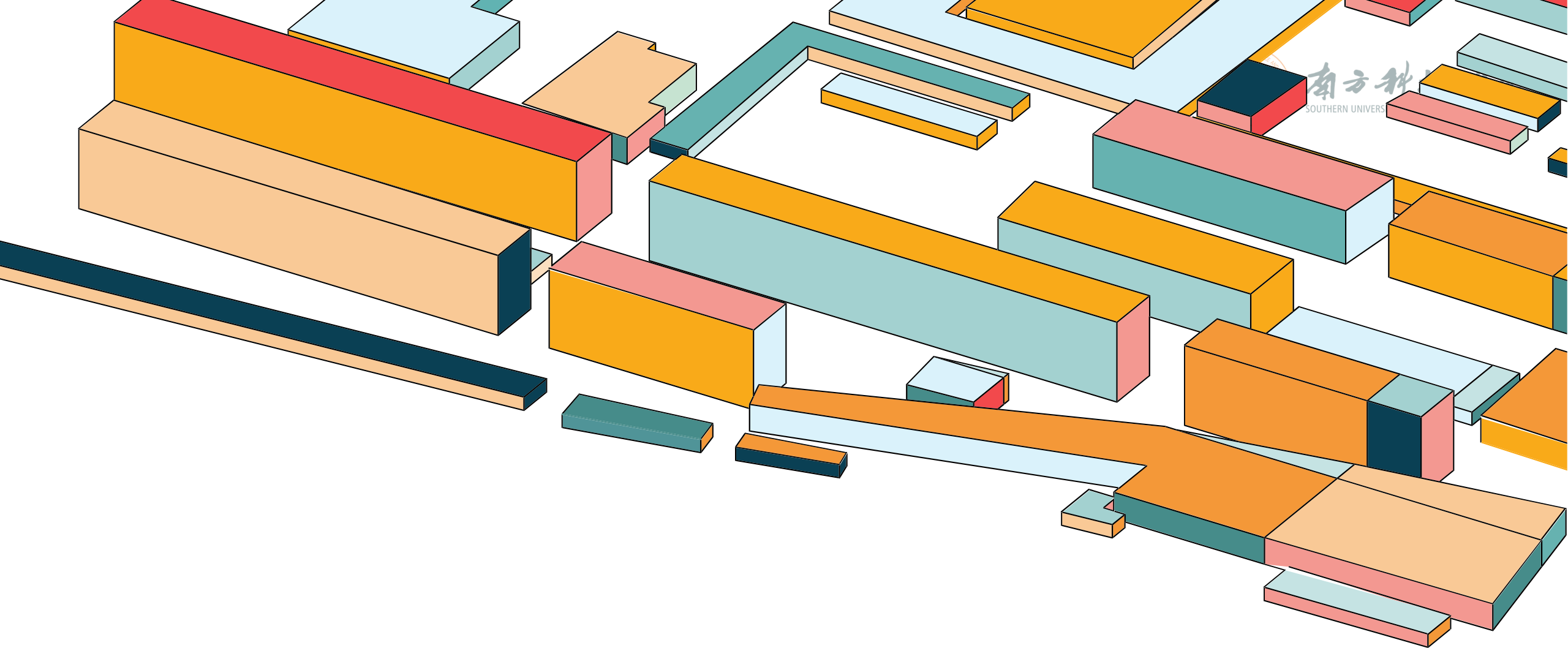
# CODE REVIEW FREQUENCY



How often do Microsoft developers review code?

<https://www.freecodecamp.org/news/how-code-reviews-work-at-microsoft-4ebdea0cd0c0/>





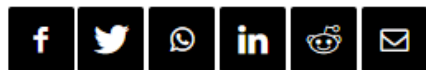
# AI AND CODE QUALITY



# CODE QUALITY ON AI GENERATED CODE

## AI assistance is leading to lower code quality, claim researchers

By Tim Anderson - January 24, 2024



### 代码屎山噩梦加速来袭，都是AI生成代码的锅？

AI前线 · 2024-01-25 17:47

关注

无脑用AI生成的代码，只会让人更累！

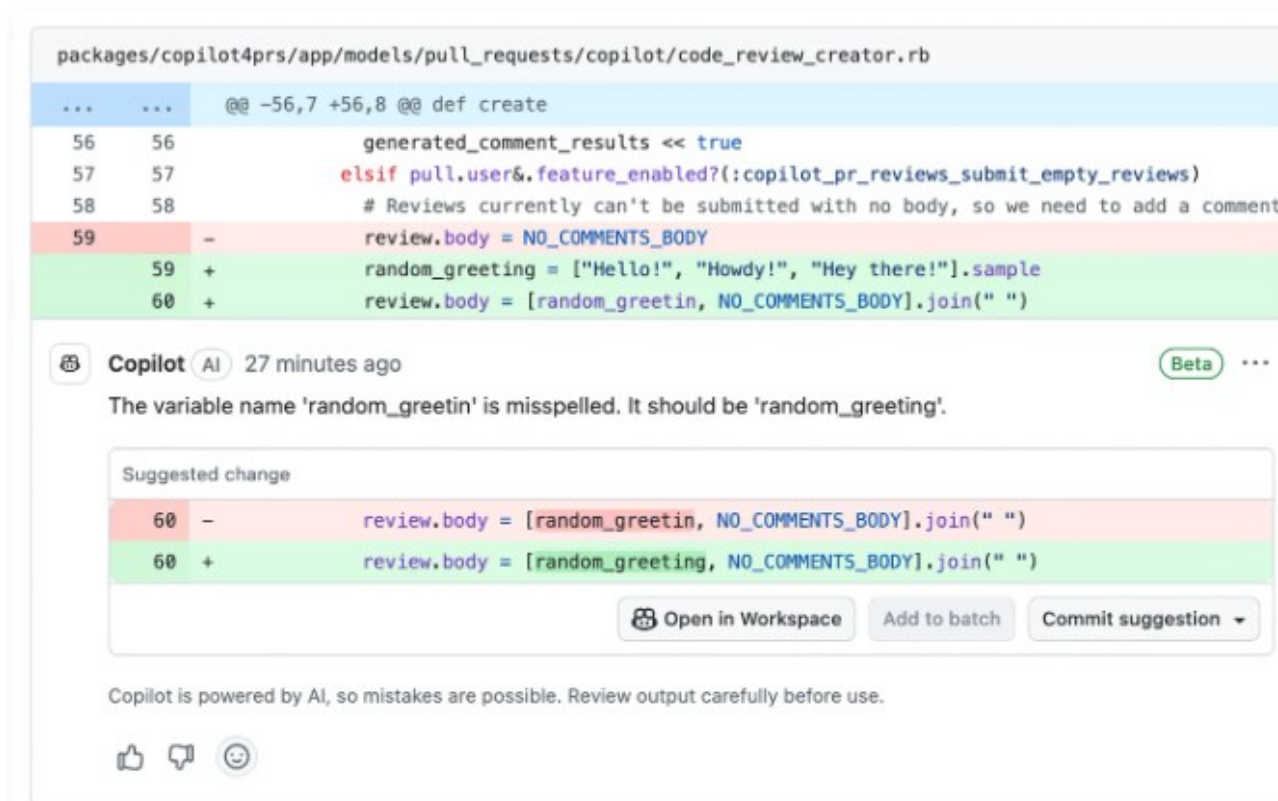
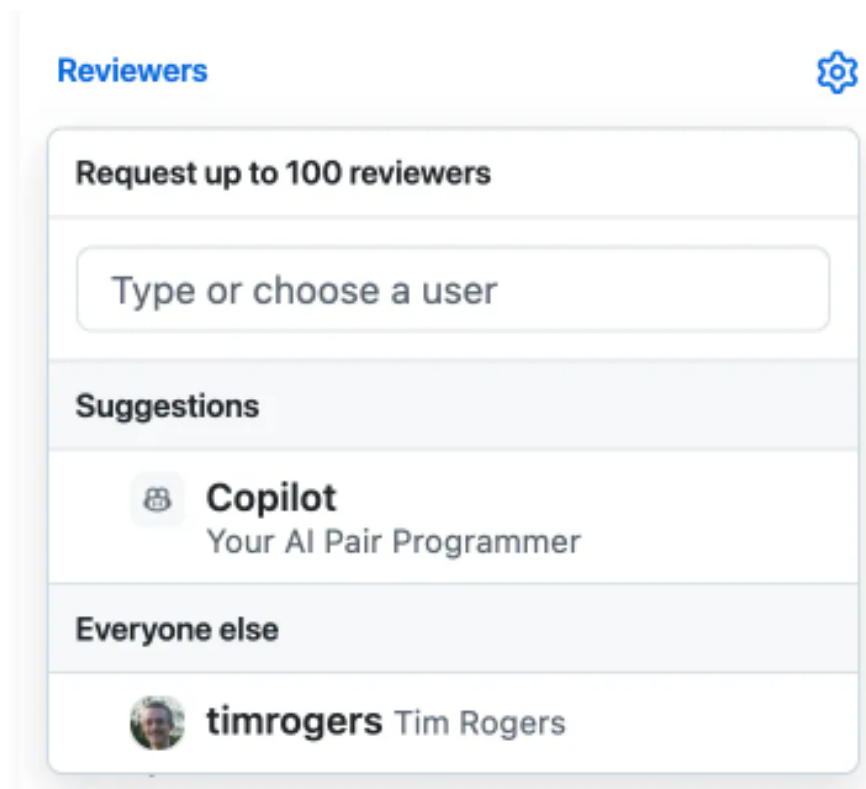
### Metrics that should be used to measure performance if AI coding tools are used

Top responses shown, N=500



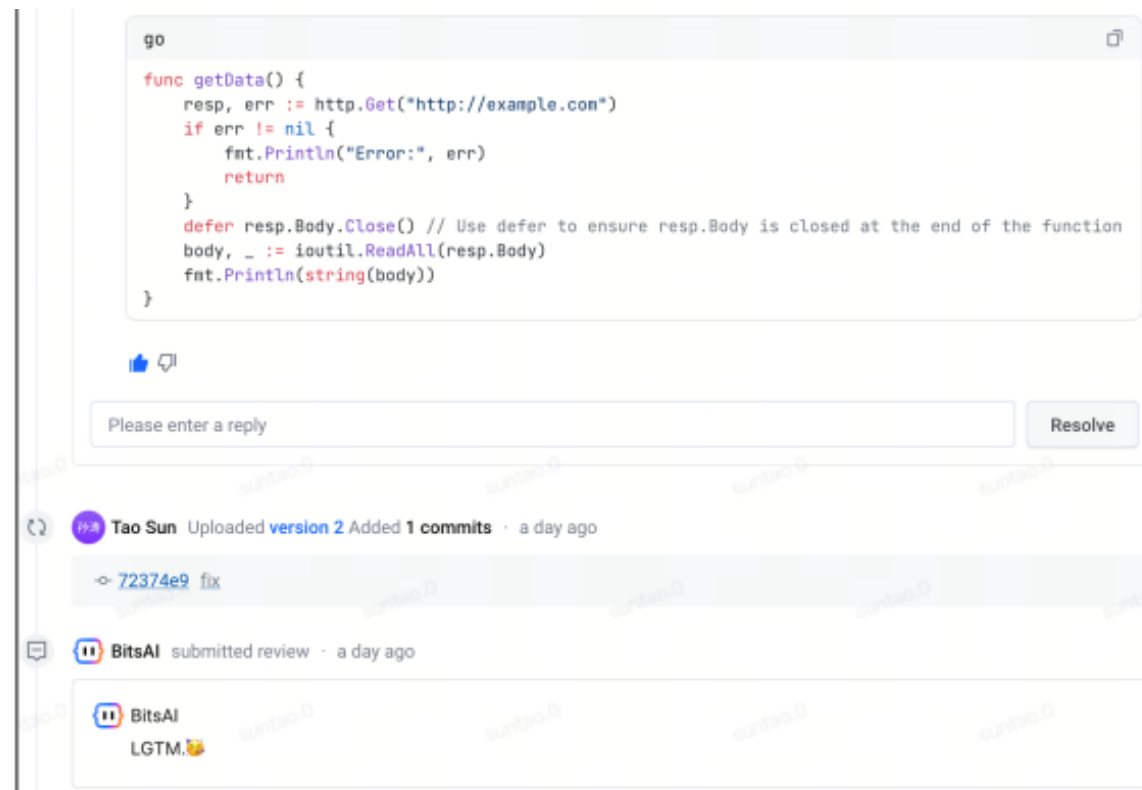
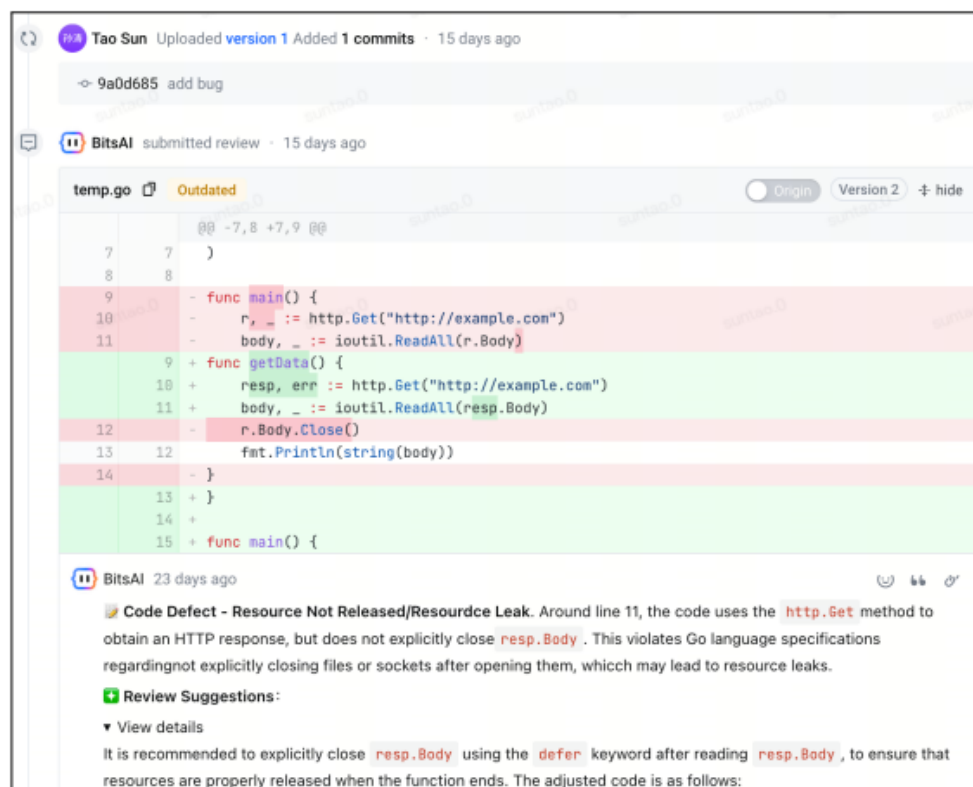
<https://github.blog/2023-06-13-survey-reveals-ais-impact-on-the-developer-experience/>

# AI CODE REVIEWER - GITHUB COPILOT





# AI CODE REVIEWER - BYTEDANCE

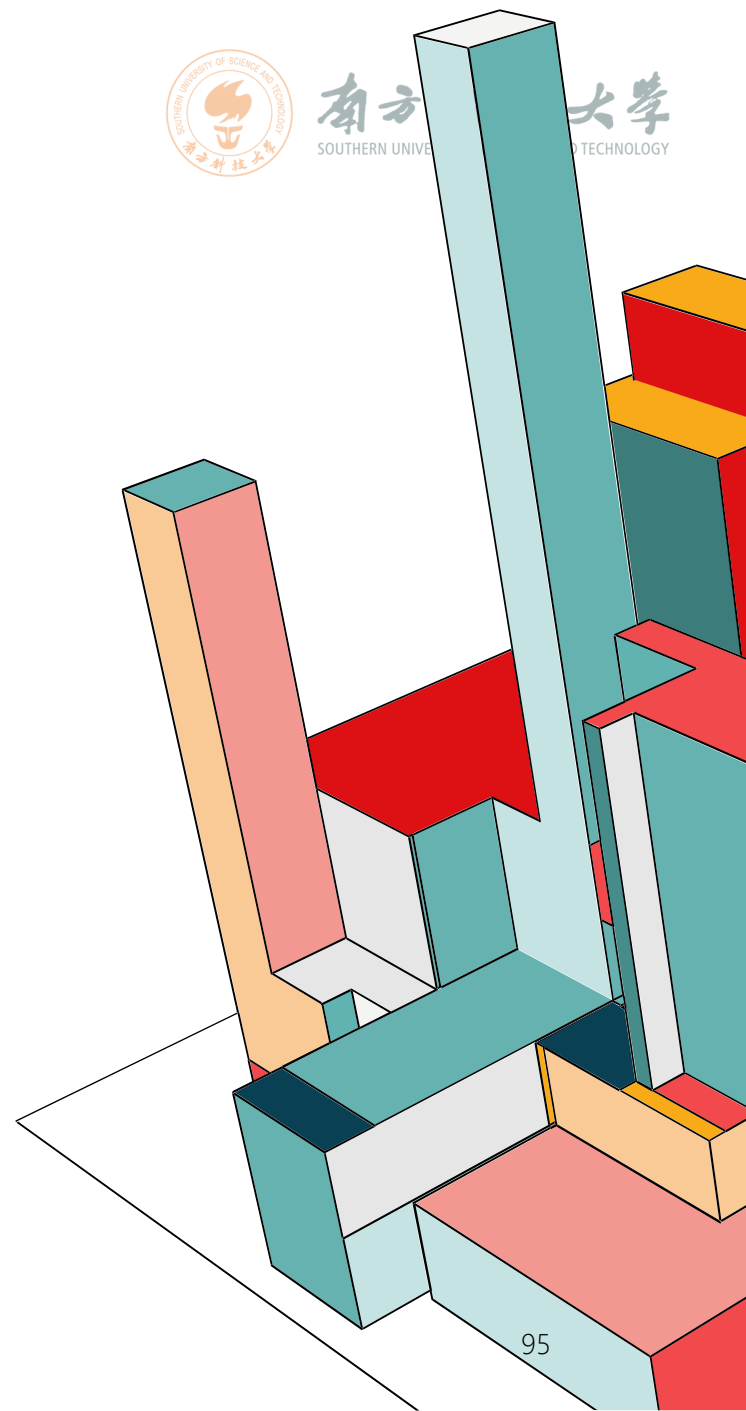


<https://arxiv.org/pdf/2501.15134>



# READINGS

- Chapter 19, 20, and 21. Software Engineering: A Practitioner's Approach. Roger Pressman and Bruce Maxim, 8<sup>th</sup> edition.
- Chapter 14. Software Engineering at Google by Titus Winters et al.





# NEXT

- Software Testing