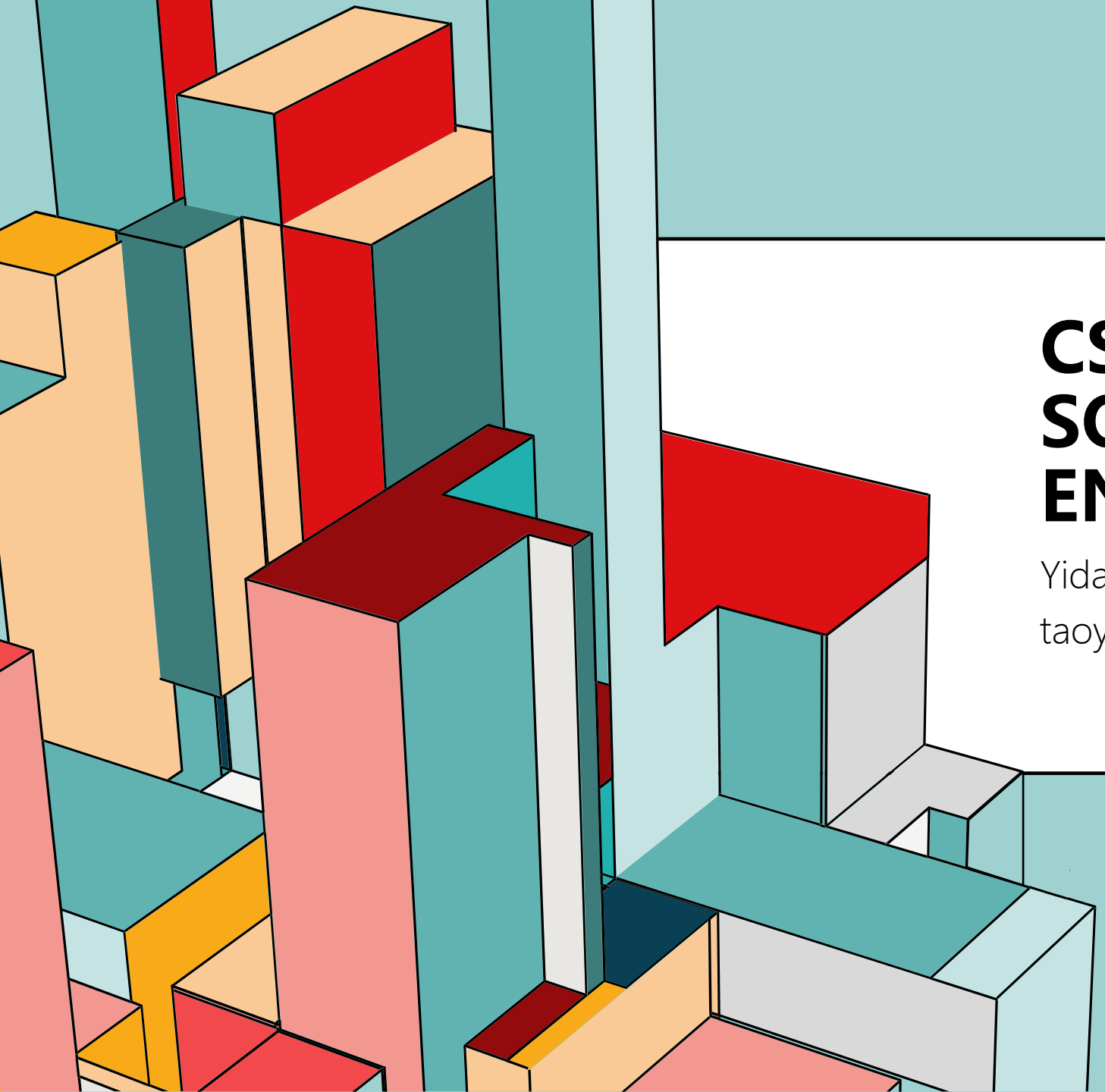


CS304 SOFTWARE ENGINEERING

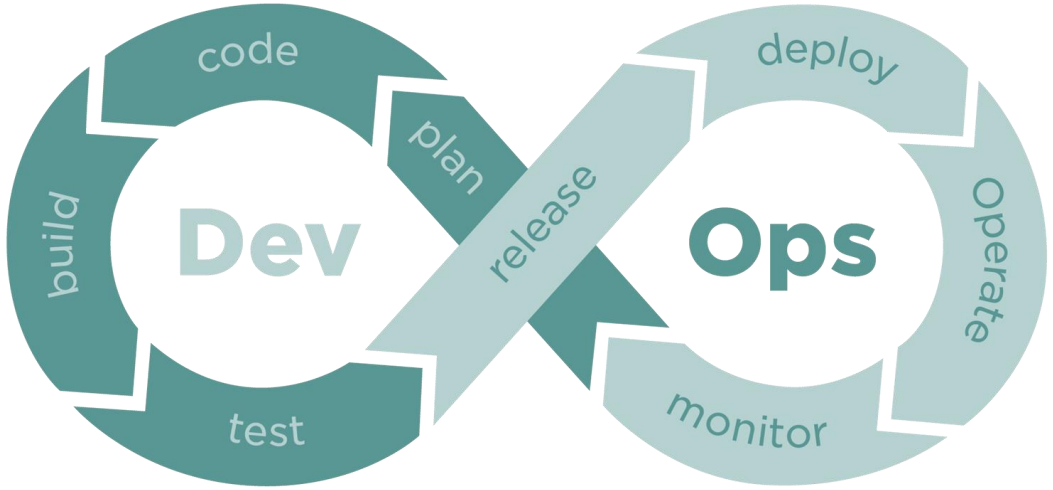
Yida Tao

taoyd@sustech.edu.cn

WHERE ARE WE NOW?

Quality control

- **Code quality**
- Testing



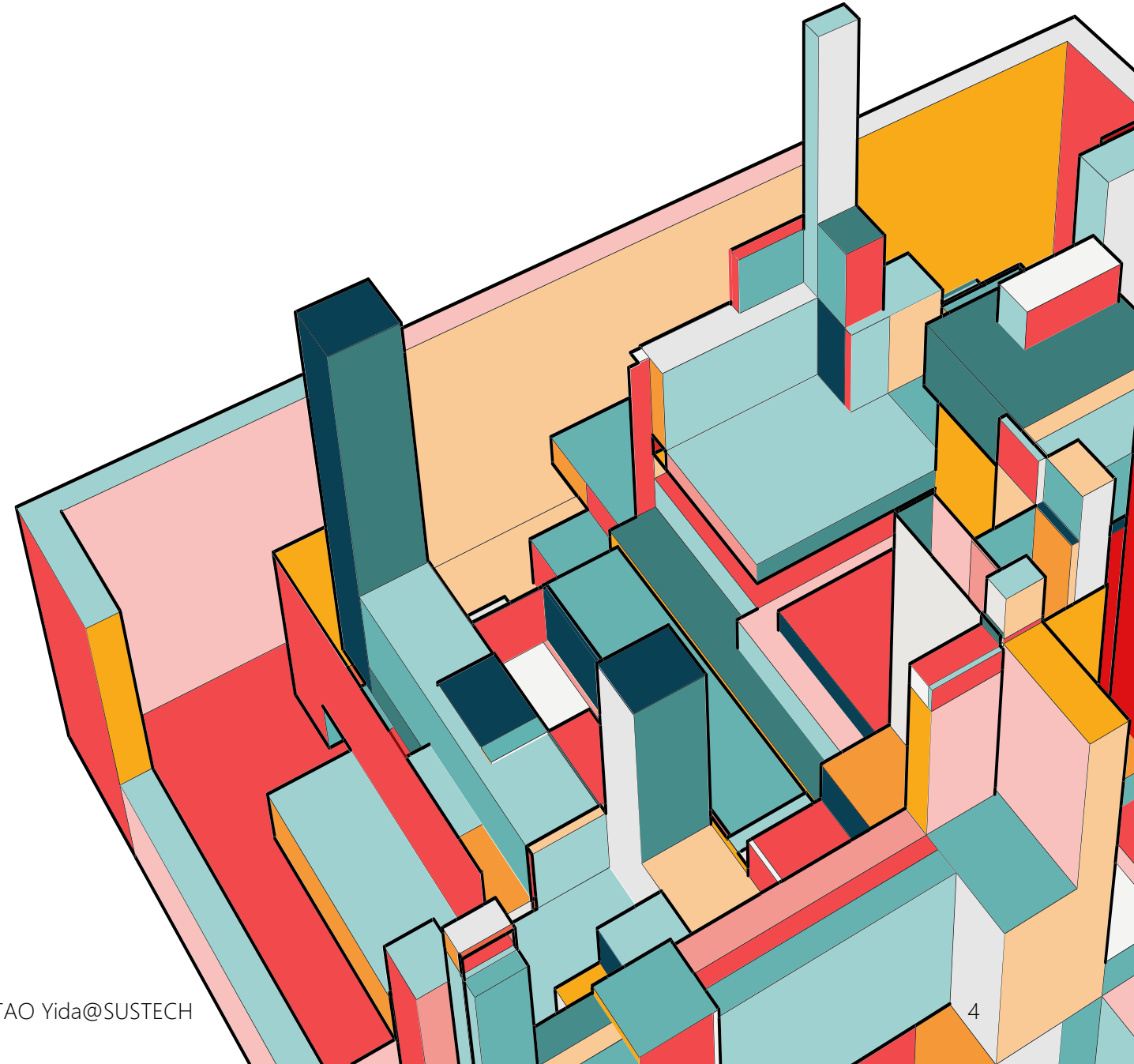
WHERE ARE WE NOW?

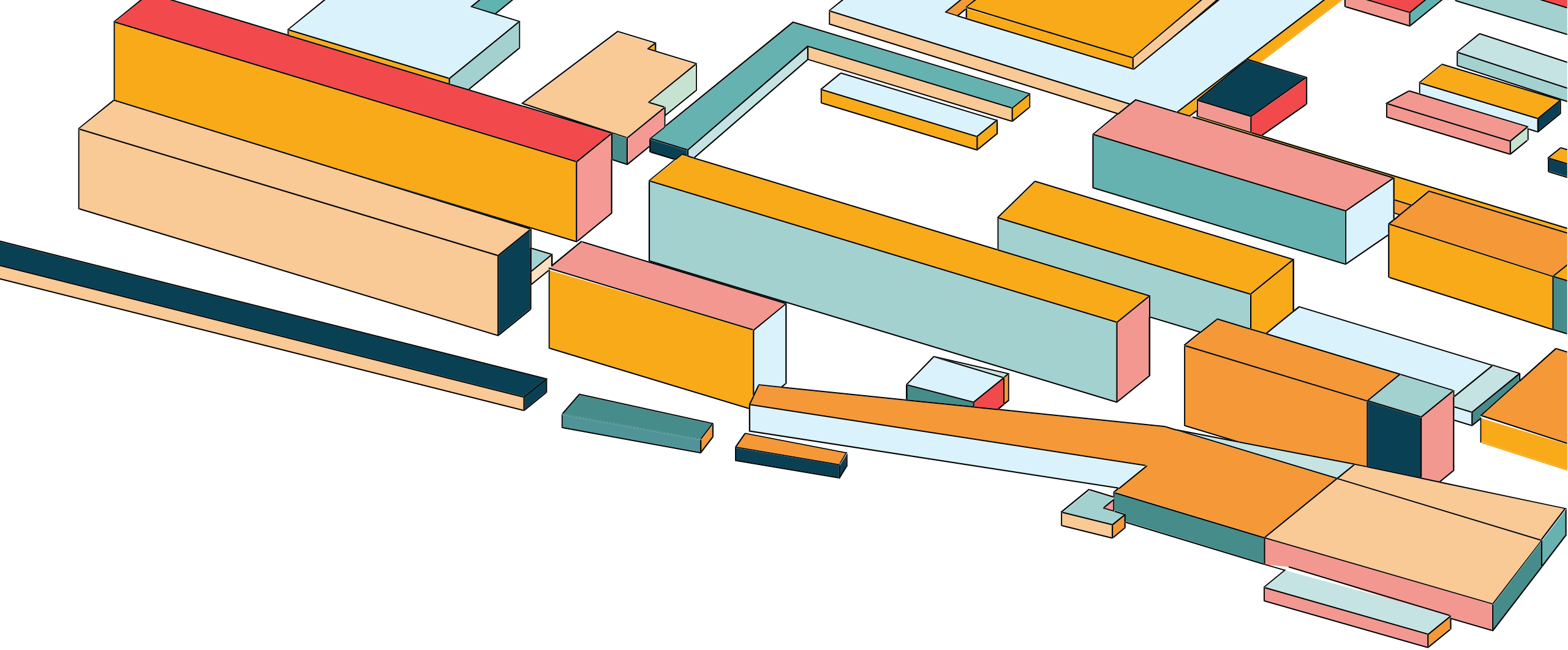
Quality control
- **Code quality**
- Testing

Software quality
- Correctness (正确性)
- **Understandability (可理解性)**
- **Maintainability (可维护性)**
- Reliability (可靠性)
- Security (安全性)
- Efficiency (高效性)
- Portability (可移植性)

LECTURE 7

- Code quality overview
- Linters
- Metrics
- 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  }
```

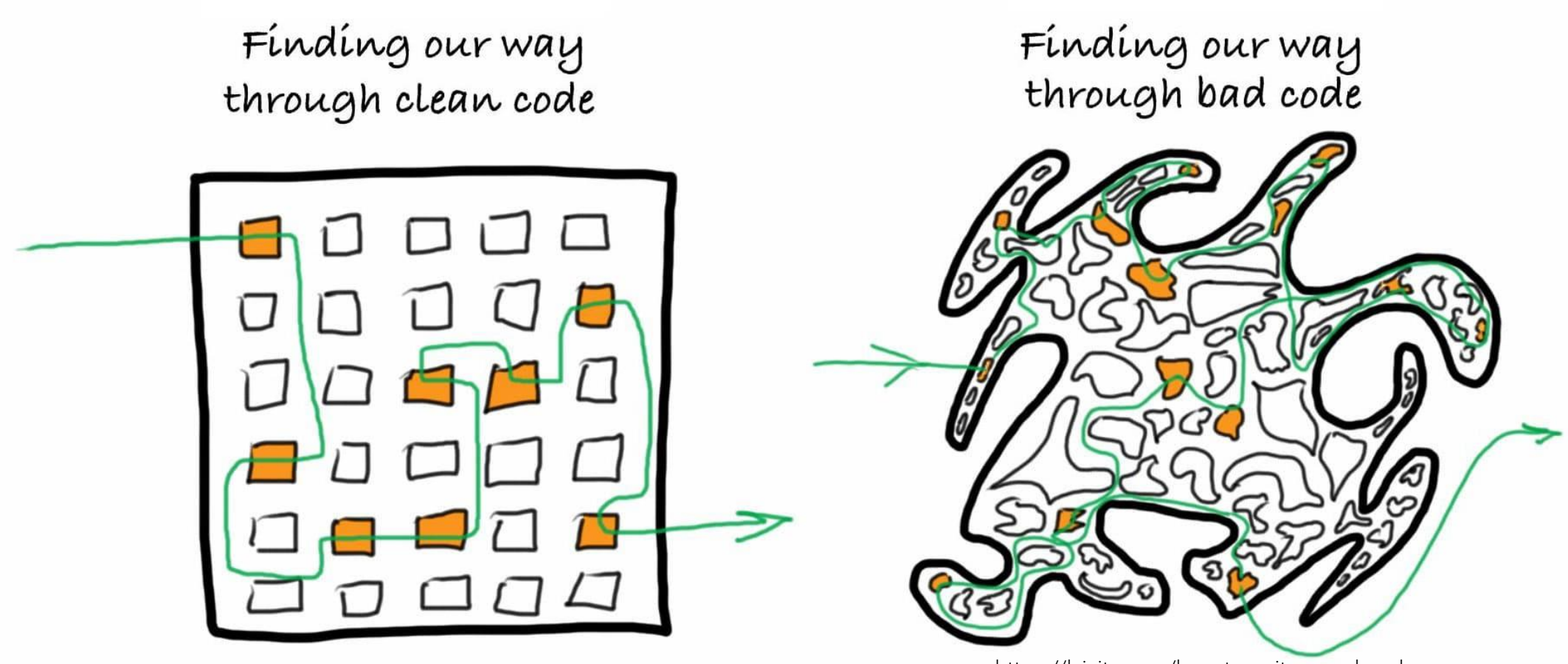
Improper exception type: only line 10 and line 17 are IllegalArgumentException.

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?

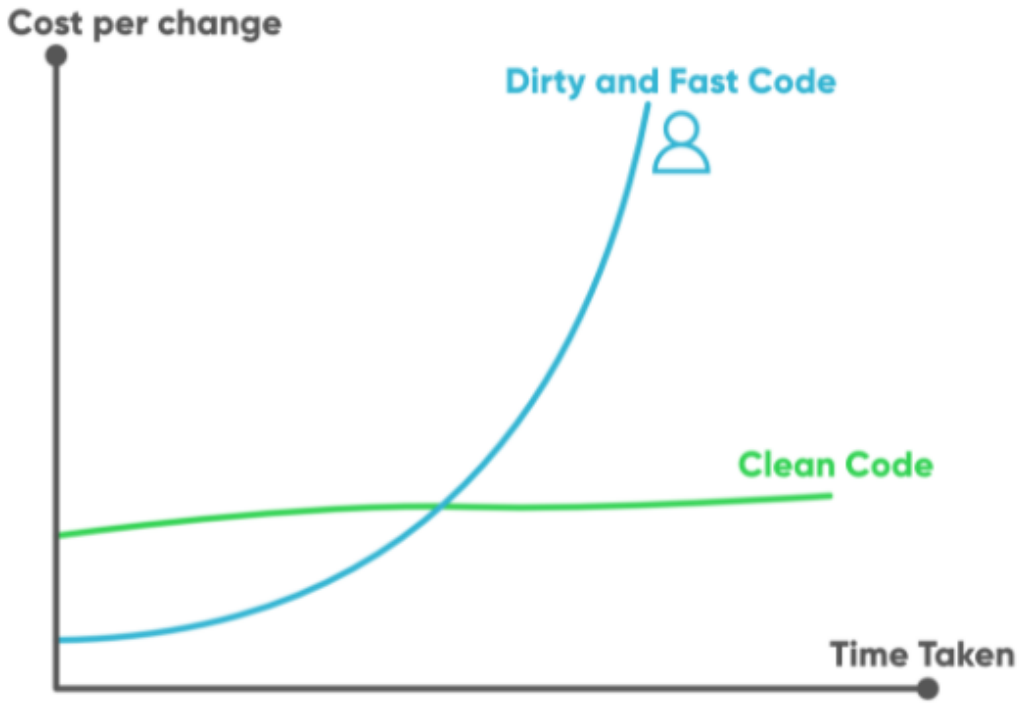
CLEAN CODE VS. BAD CODE



https://lvivity.com/how-to-write-good-code

CLEAN CODE

- Software engineers not only write code, but also read, maintain, and extend code
- Good software engineers should aim for writing clean code, instead of fast but dirty code/workaround



CLEAN CODE

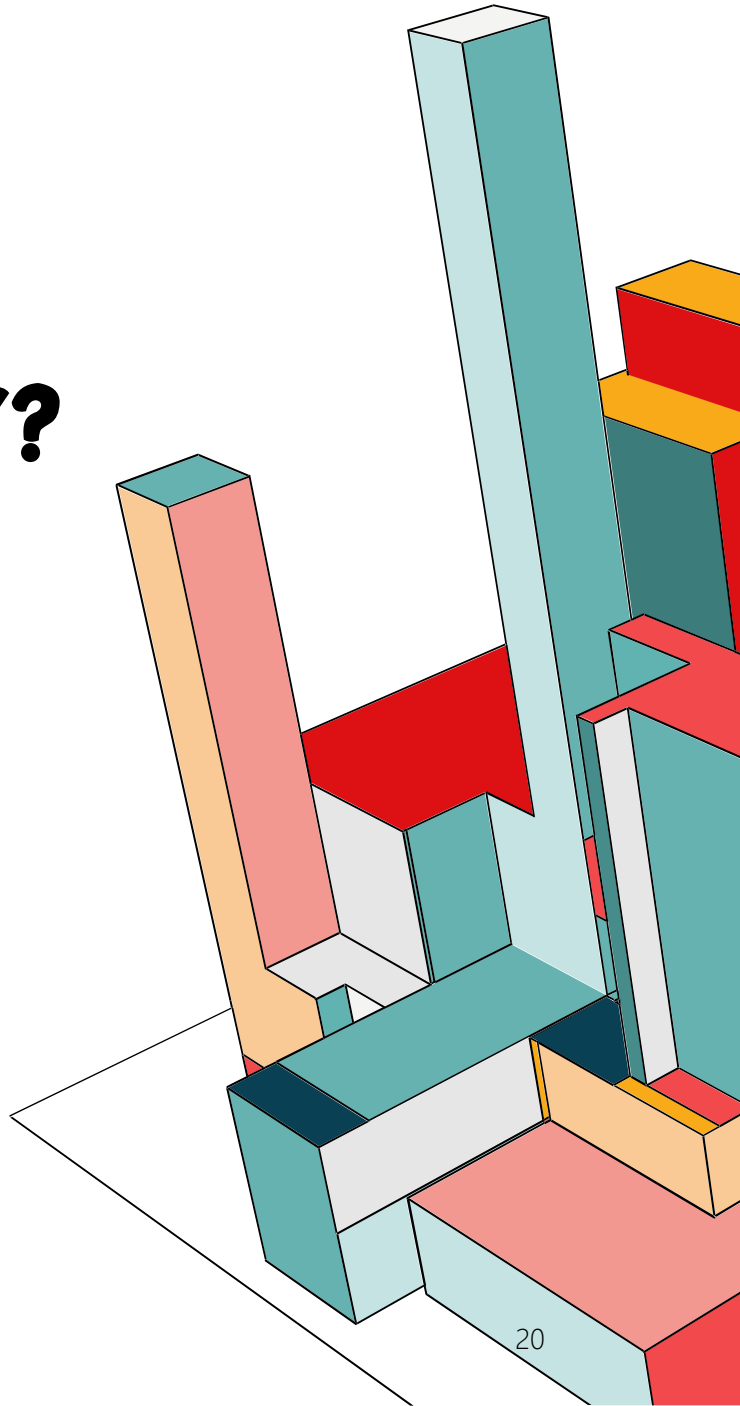
Factors for building a software

- Teamwork
- Time
- Scale
- ……

Big tech companies have strict requirements on code quality!

How to evaluate code quality?

- Linters
- Metrics
- Code review



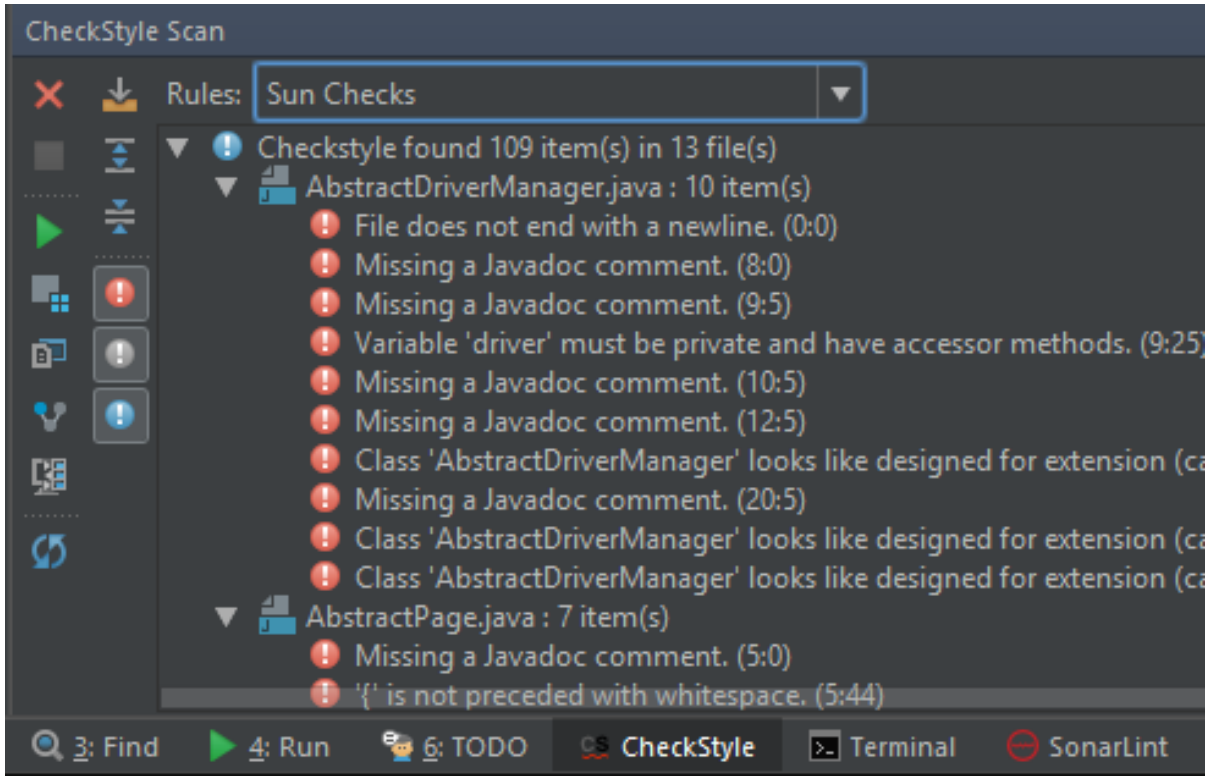
LINTERS

- A linter (语法检查器) serves as a valuable developer tool for improving and enhancing code quality.
- It scans **source code** looking for errors, defects, stylistic issues, and questionable constructs that can lead to bugs, vulnerabilities, and code smells.
- Today, modern linters are available for almost all programming languages

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

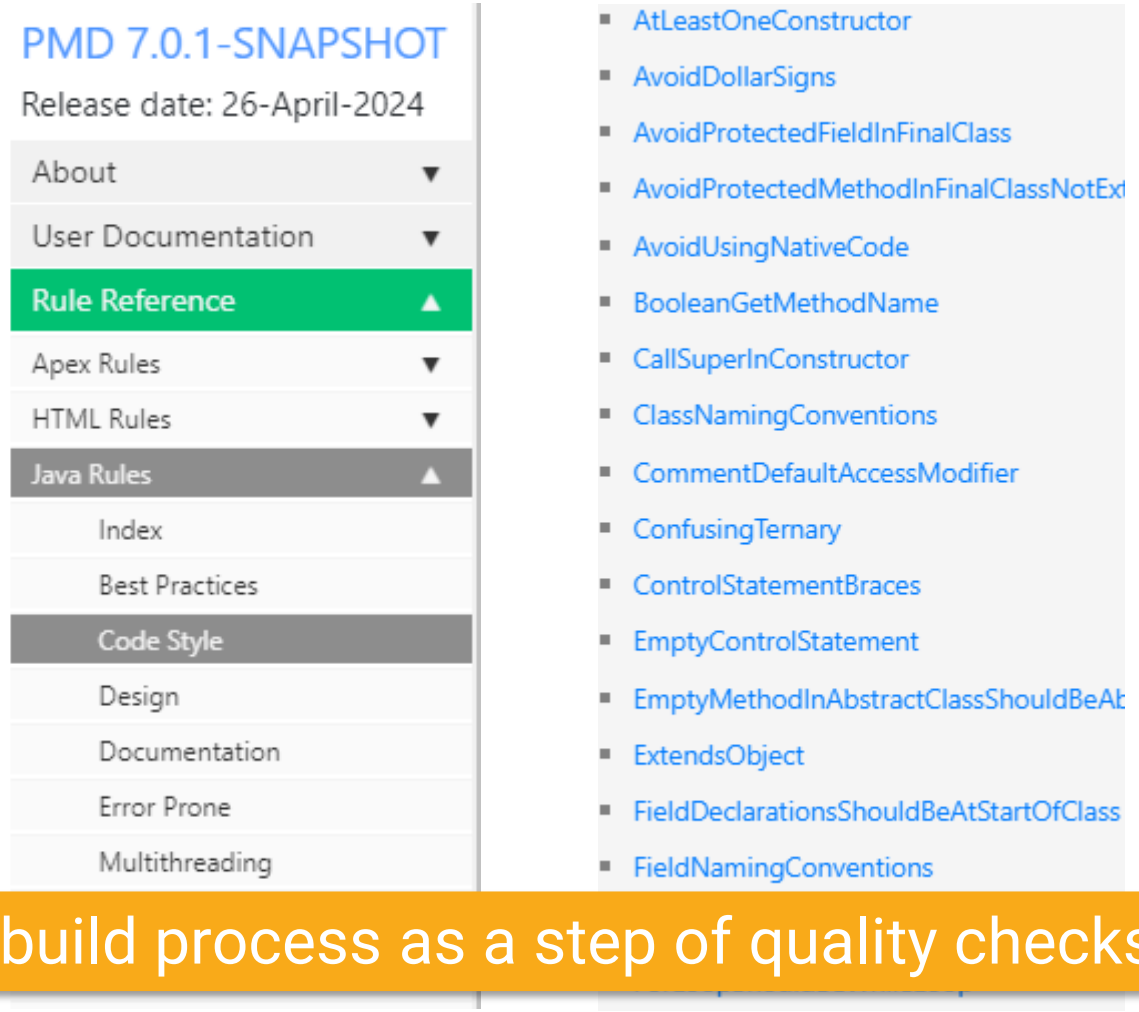
LINTERS

- Rule-based linters: use predefined rules to identify issues in the code automatically
- Tools: CheckStyle, PMD, etc.



LINTERS

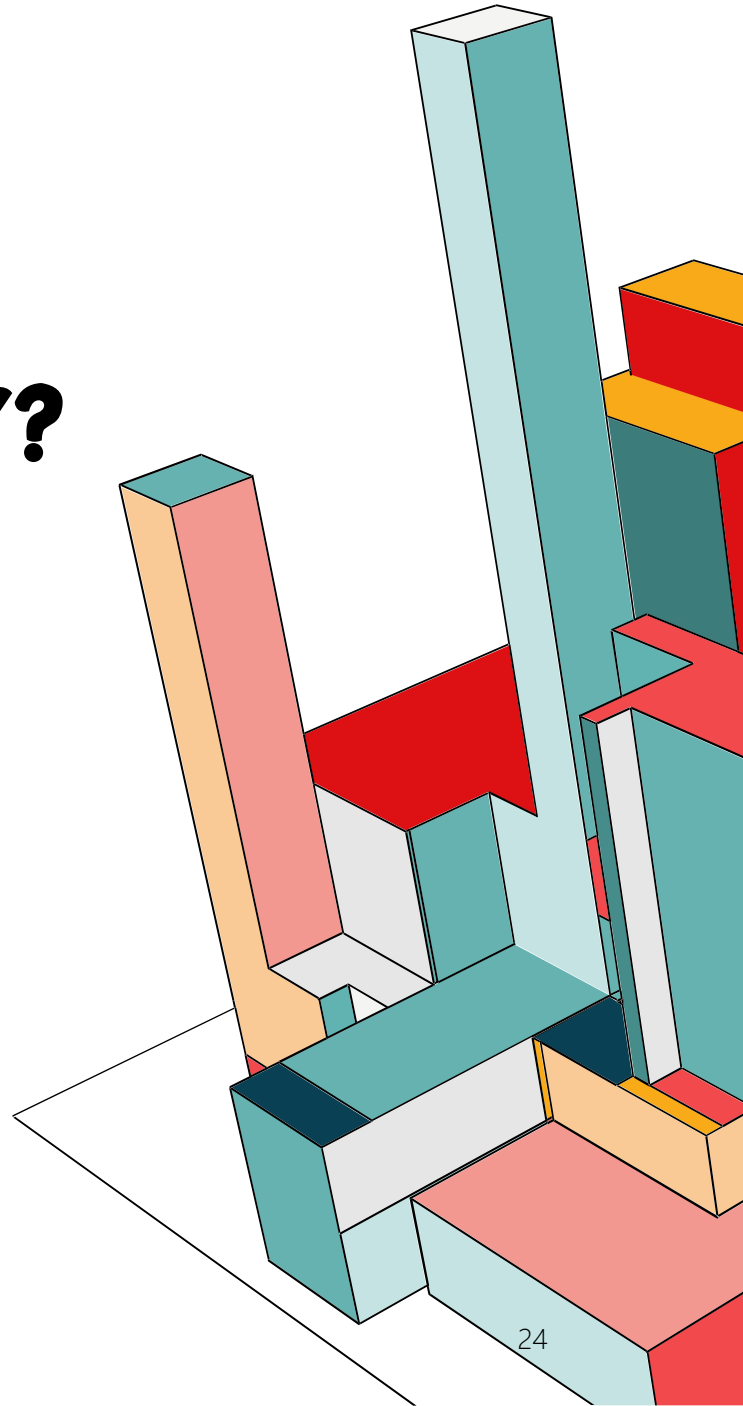
- Rule-based linters: use predefined rules to identify issues in the code automatically
- Tools: CheckStyle, PMD, etc.



Linters can also be integrated in the build process as a step of quality checks

How to evaluate code quality?

- Linters
- Metrics (for code complexity)
- 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");
```

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

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

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

LINES OF CODE - LANGUAGE MATTERS

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

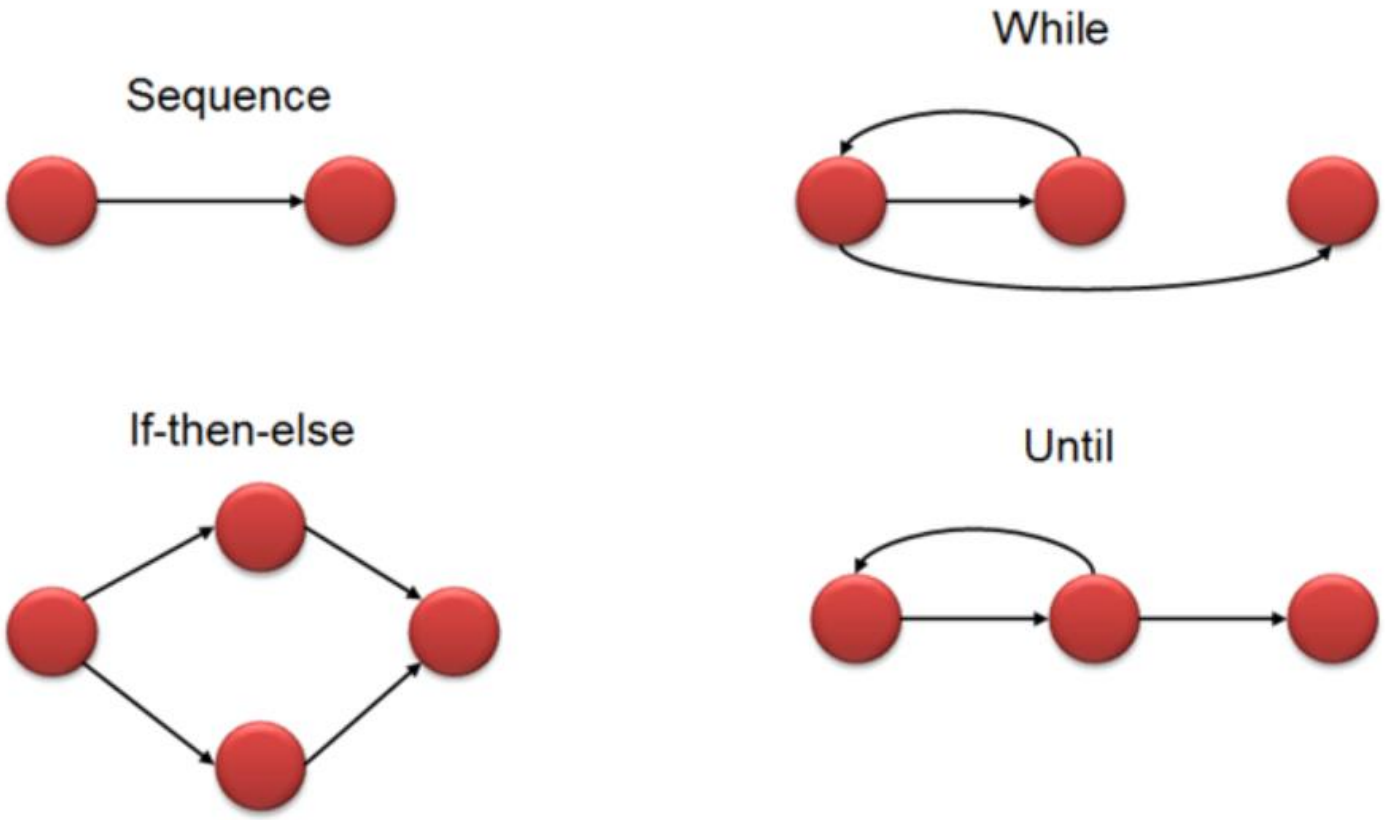
- Assembly code may be 2-3X longer than C code
- C code may be 2-3X longer than Java code
- Java code may be 2-3X longer than …

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)

CONTROL FLOW GRAPH



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

Decisions are **cycles**

CALCULATE CYCLOMATIC COMPLEXITY

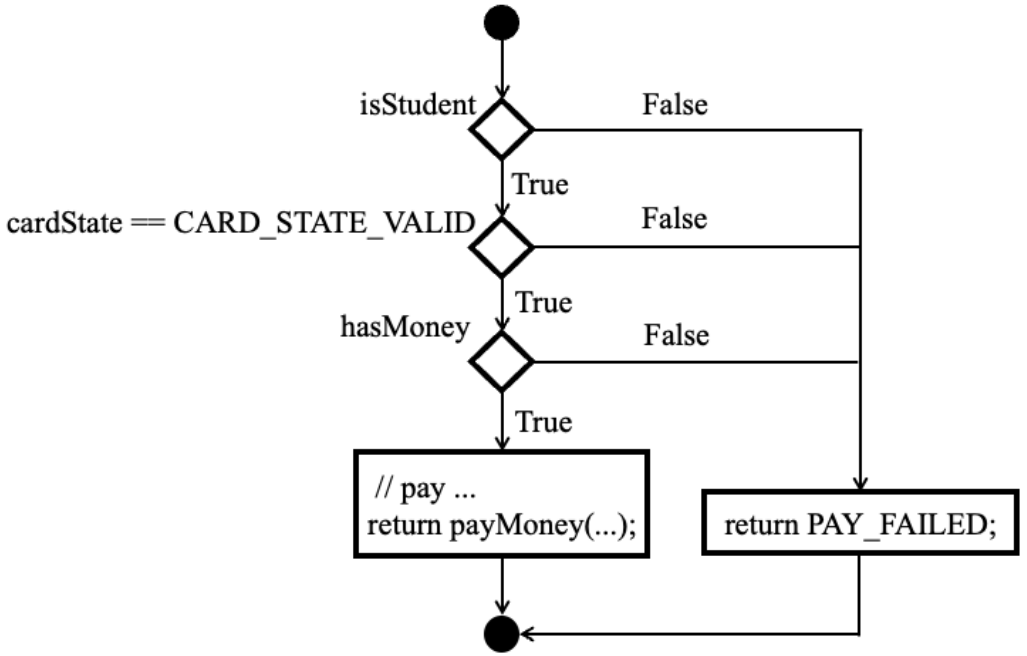
Approach 1: `V (G) = P + 1`
P: Number of branch nodes (e.g., if, for, while, case)

Approach 2: `V(G) = E - N + 2`
E: Number of edges
N: Number of nodes

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

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();
    }
}
```

https://www.cqse.eu/en/news/blog/mccabe-cyclomatic-complexity/

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;
}
```

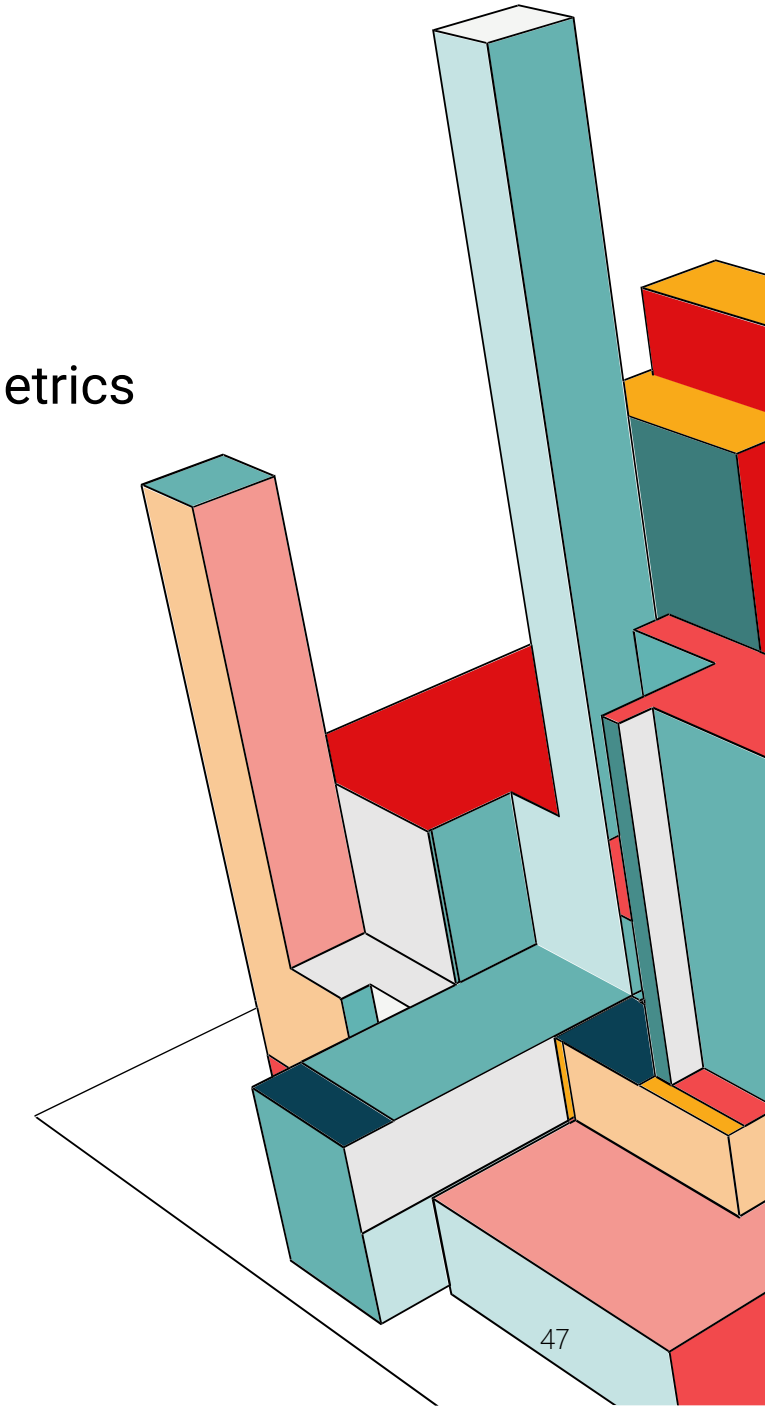
Both have a cyclomatic complexity of 5. Same readability & maintainability?

https://www.cqse.eu/en/news/blog/mccabe-cyclomatic-complexity/

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)
- Response for a Class (RFC)
- ……



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

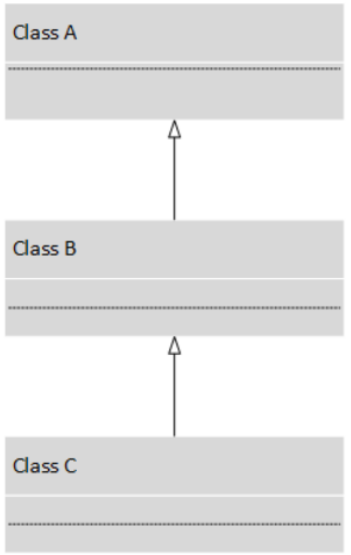
- Possible method complexities
  - 1 (# of methods)
  - LoC
  - # of method calls
  - Cyclomatic complexity

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



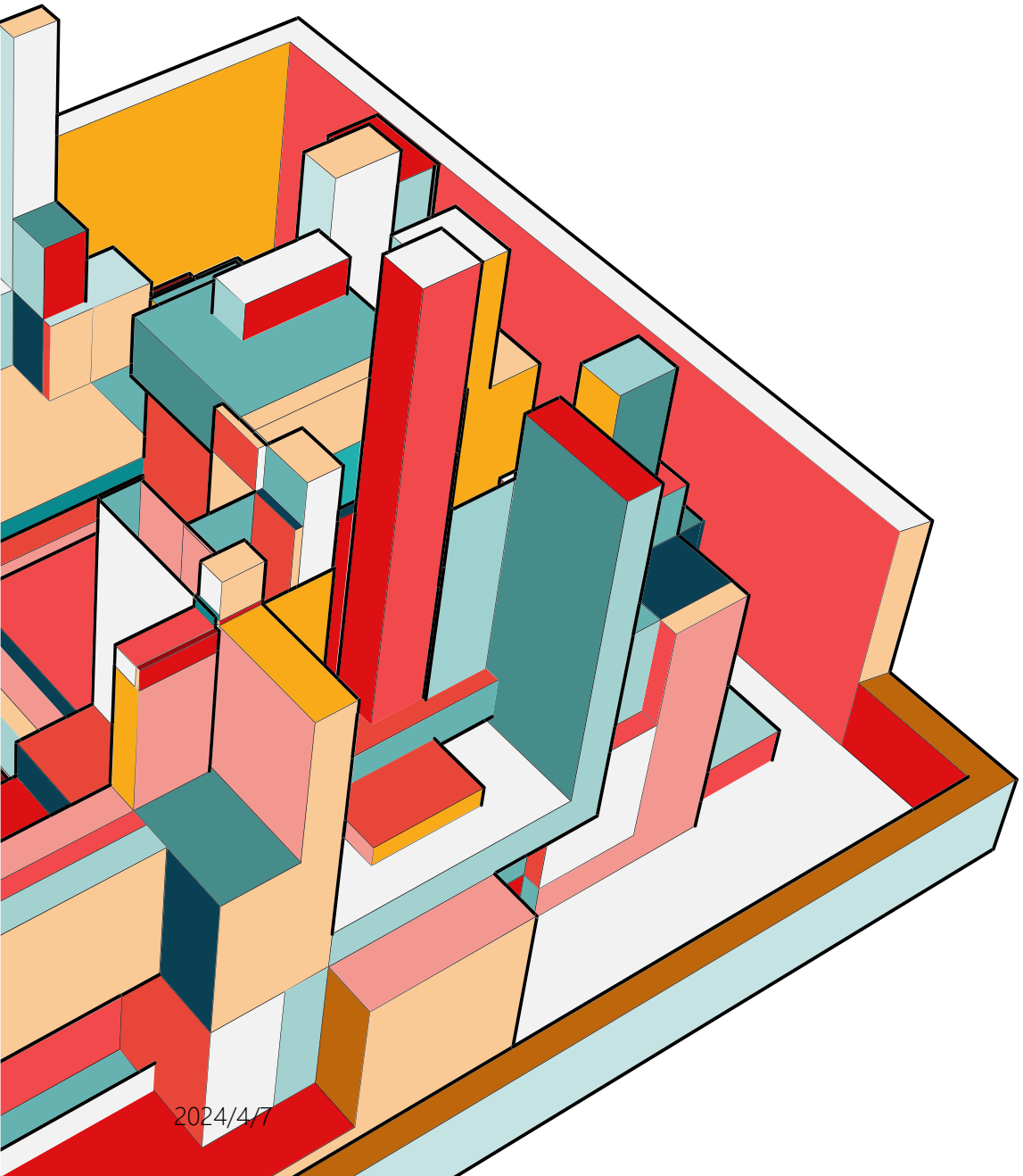
DIT = 0

DIT = 1

DIT = 2

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

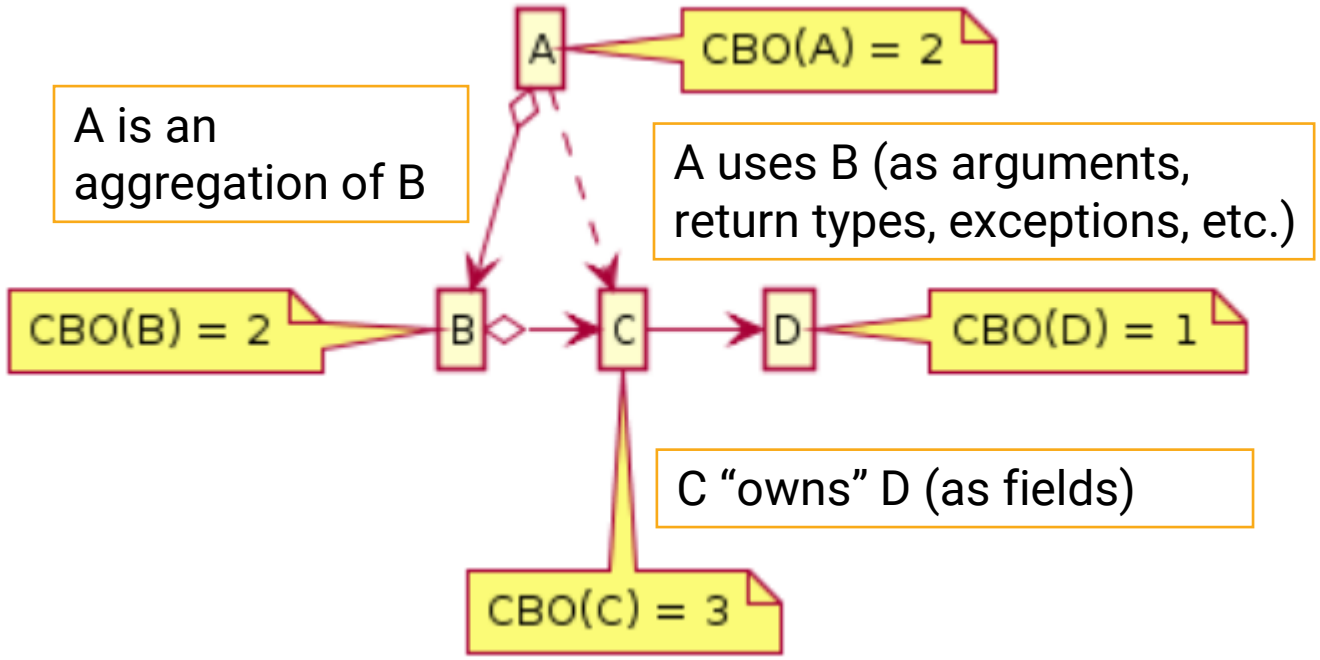


COUPLING AND COHESION

- Dependences
  - Call methods, refer to classes, share variables
- Coupling
  - Dependences among 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



https://stackoverflow.com/questions/27515541/cbo-coupling-between-object

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.

$$LCOM = P - Q$$

RESPONSE FOR A CLASS

- RFC measures the number of different methods that can be executed when an object of that class receives a message (when a method is invoked for that object).
- If a large number of methods can be invoked in response to a message, testing becomes more complicated
- The more methods that can be invoked from a class, the greater the complexity of the class

USING CODE QUALITY METRICS

AI assistance is leading to lower code quality, claim researchers

By **Tim Anderson** - January 24, 2024

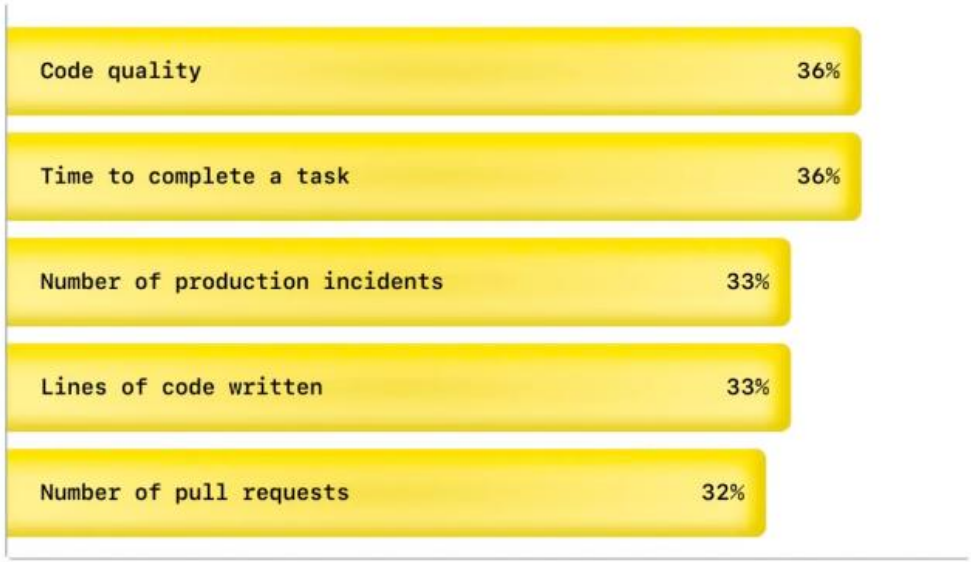
代码屎山噩梦加速来袭，都是AI生成代码的锅?

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*

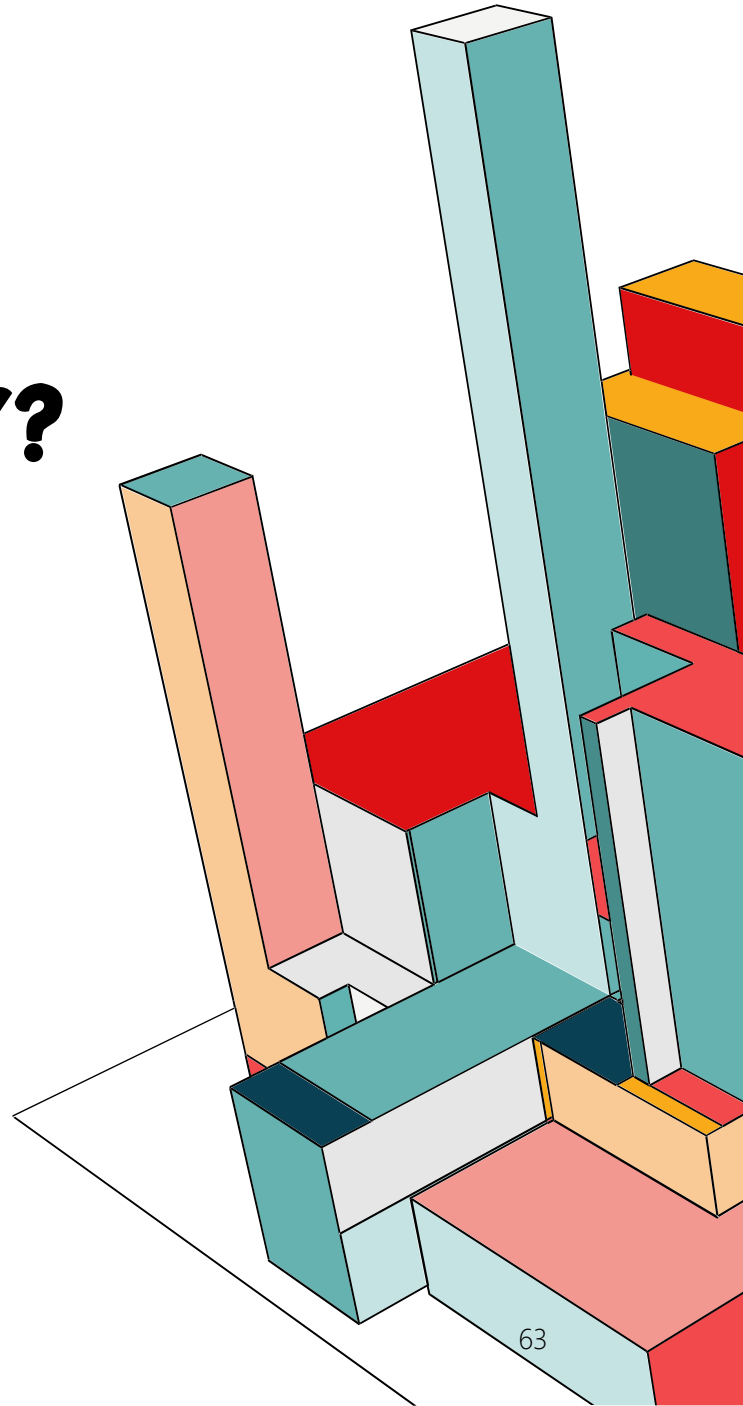
Top responses shown, N=500



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

How to evaluate code quality?

- Linters
- Metrics (for code complexity)
- Code review



WHAT IS CODE REVIEW?

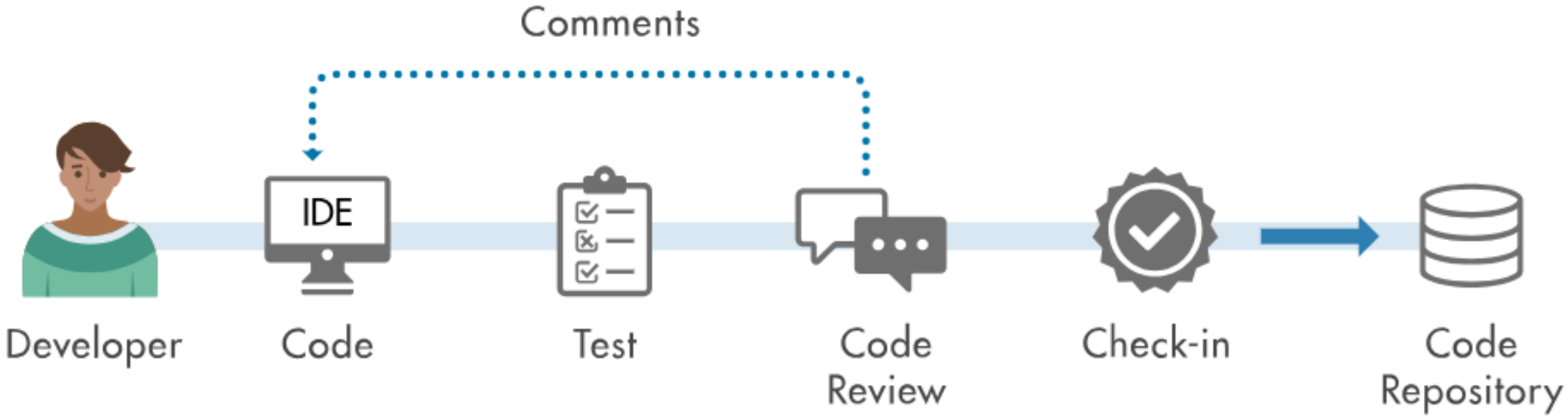
Code review (sometimes referred to as peer review) is a software quality assurance activity in which one or several people check a program mainly by viewing and reading parts of its source code (changes/commits)

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

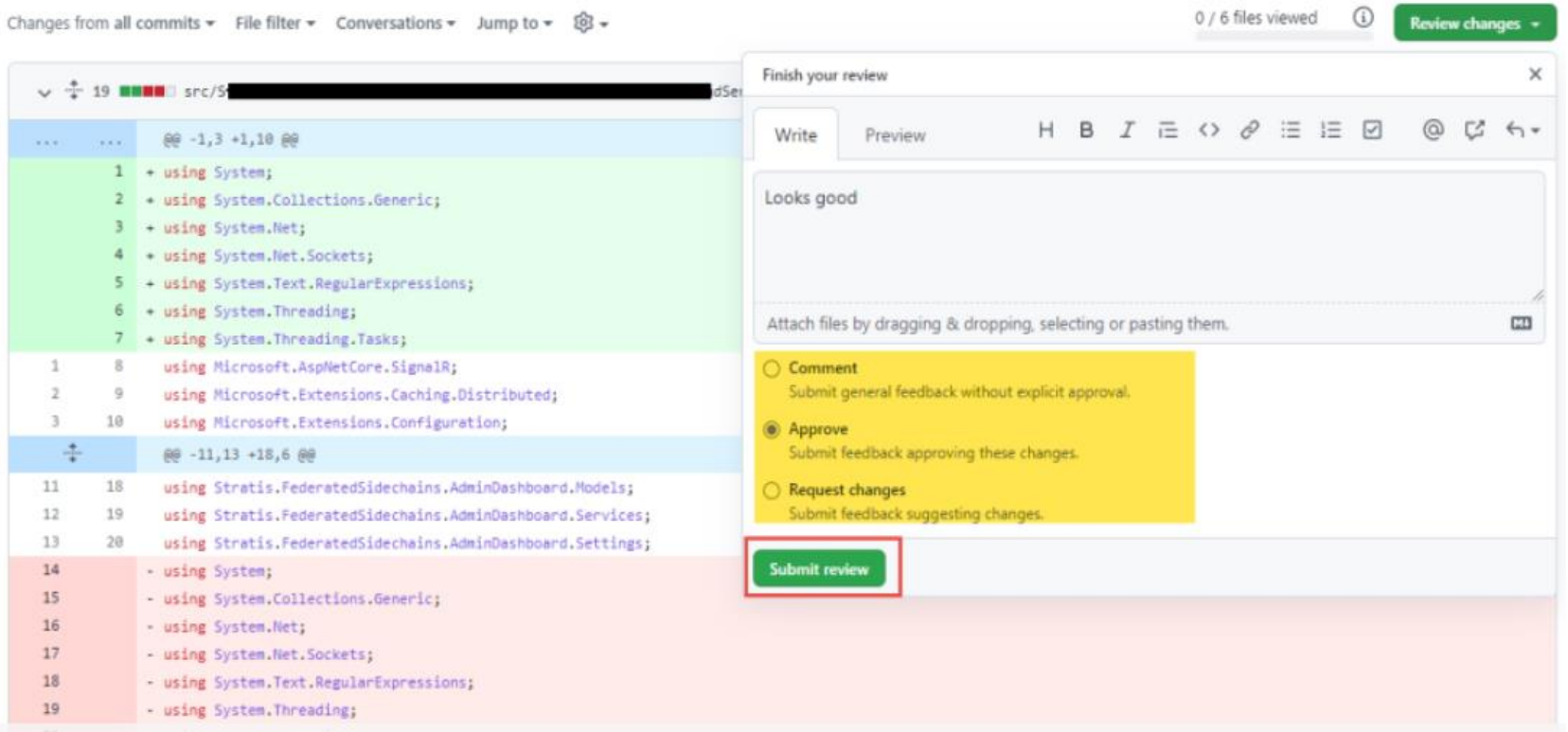
CODE REVIEW GOALS

- 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.

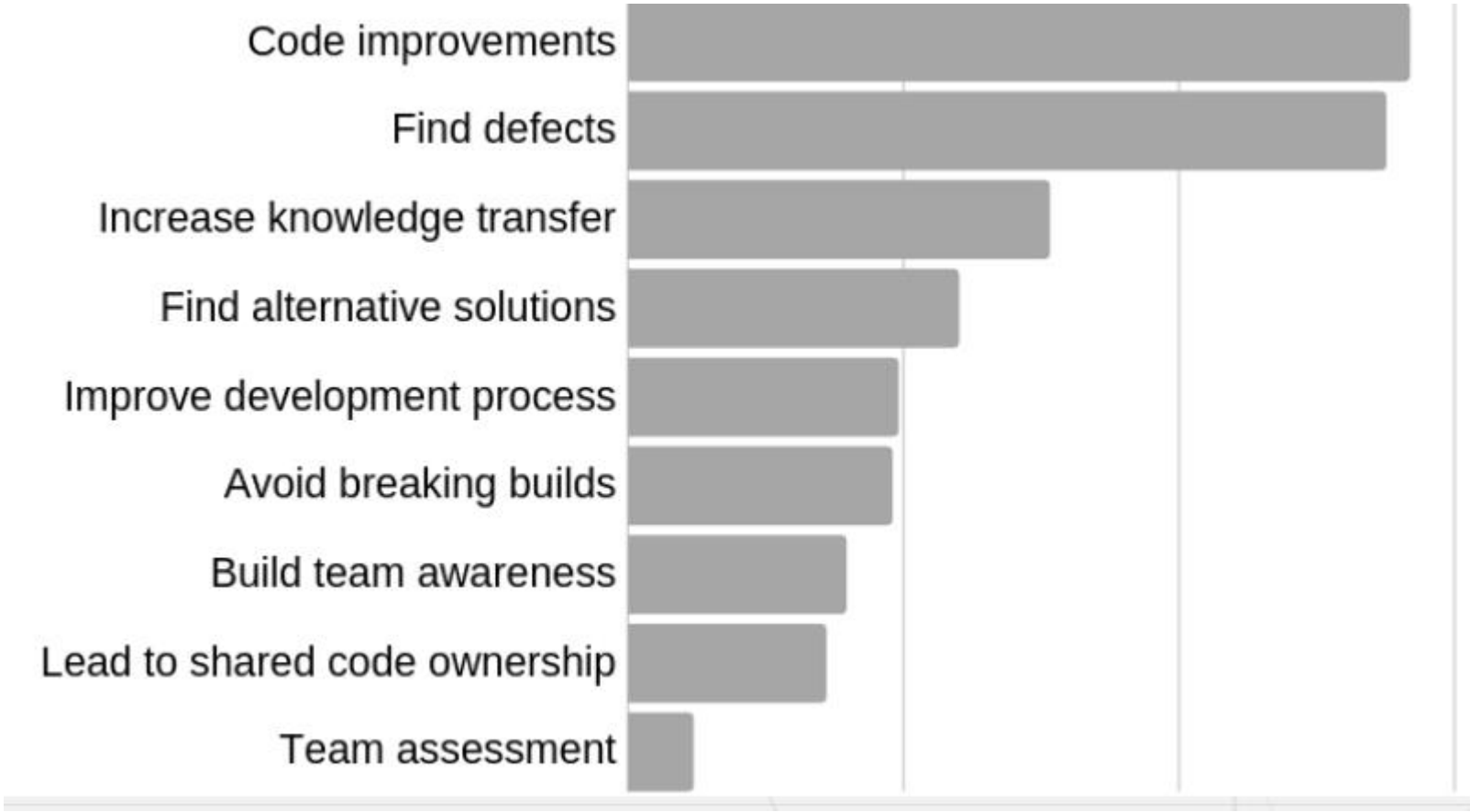
Most style checks and correctness checks are performed automatically through techniques such as linters and automated testing

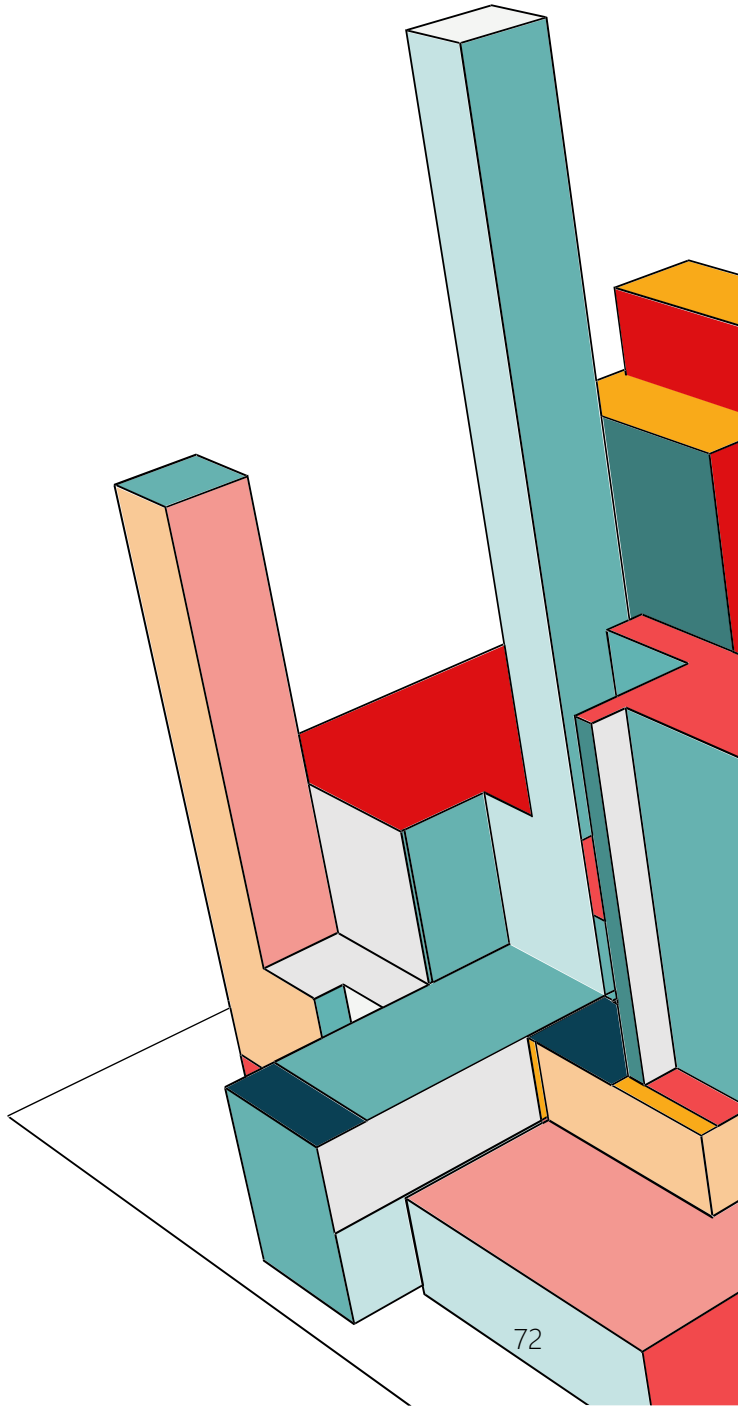
https://google.github.io/eng-practices/review/reviewer/looking-for.html

CODE REVIEW TOOLS



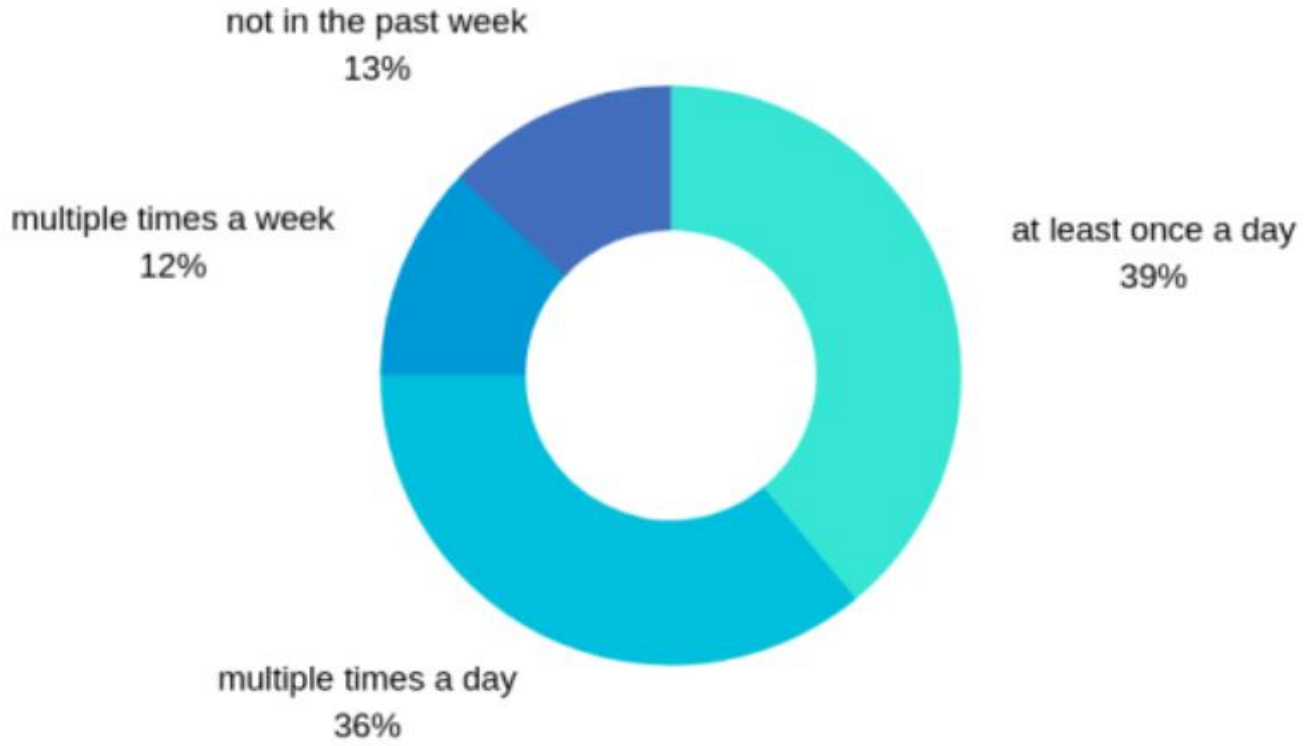
CODE REVIEW BENEFITS



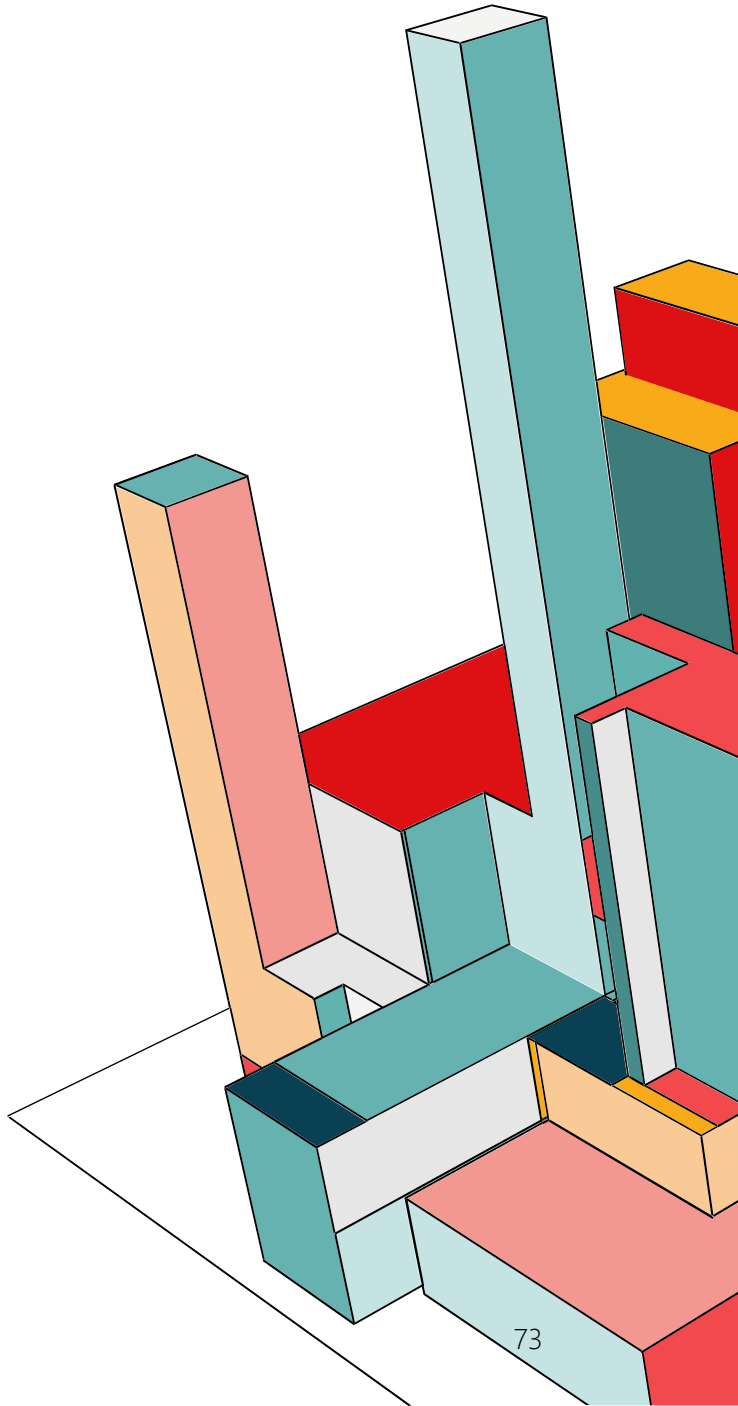


https://www.freecodecamp.org/news/how-code-reviews-work-at-microsoft-4ebdea0cd0c0/

CODE REVIEW FREQUENCY



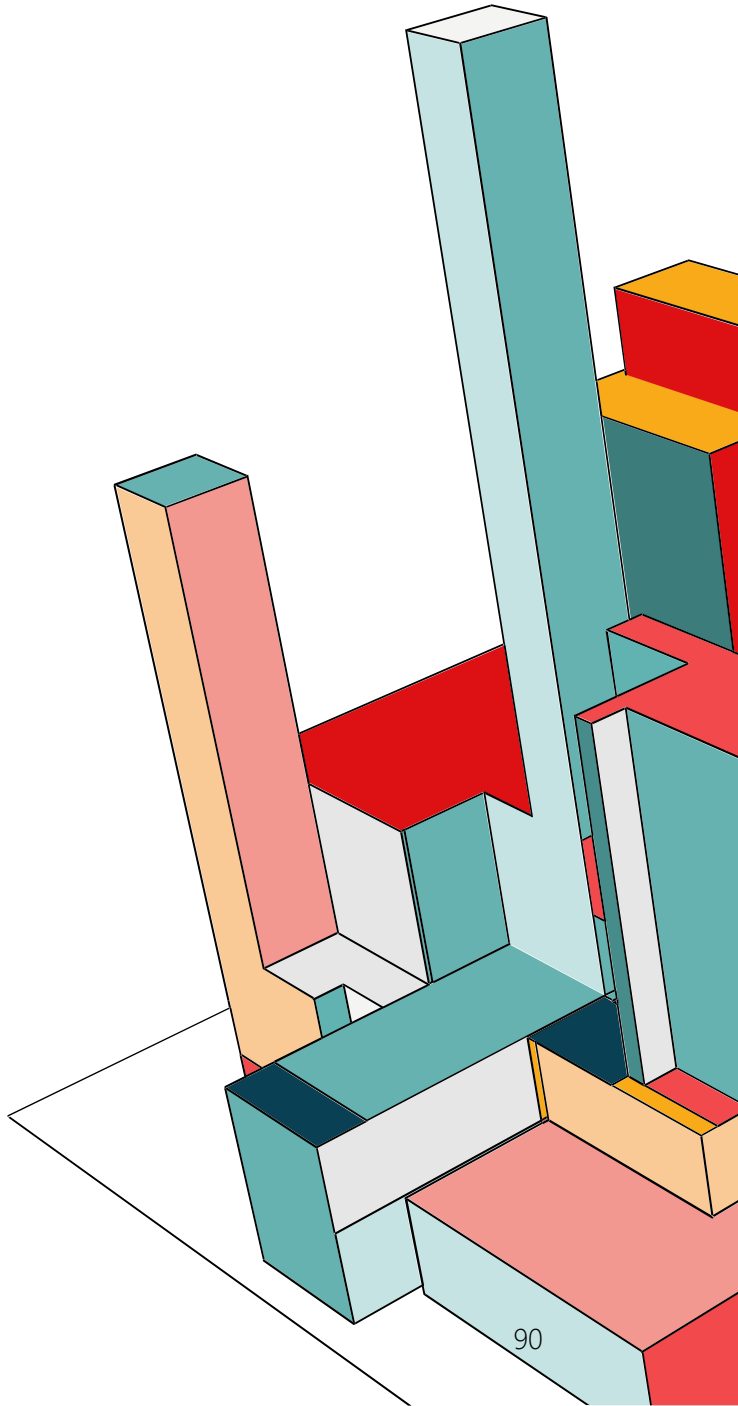
https://www.freecodecamp.org/news/how-code-reviews-work-at-microsoft-4ebdea0cd0c0/



READINGS

- Chapter 19, 20, and 21. Software Engineering: A Practitioner's Approach. Roger Pressman and Bruce Maxim, 8th edition.

- Chapter 14. Software Engineering at Google by Titus Winters et al.



NEXT

- Software Testing