MSE | MASTER OF SCIENCE IN ENGINEERING

MA_EmbReal

Robust Development Methodologies II

Version: 1.2

Serge Ayer – Luca Haab | 21.03.2023 | Cours MSE

SW Quality: what is it?

```
1  // Option 1
2  const result = Object.keys(obj).reduce(a, v) => ({ ...a, [v]: true }), {});
3
4  // Option 2
5  const result = {};
6  for (const key in obj) {
7    result[key] = true;
8  }
9
10 // Option 3
11 const result = {};
12 const keys = Object.keys(obj);
13 for (let i = 0; i < keys.length; ++i) {
14   const key = keys[i];
15   result[key] = true;
16 }
```

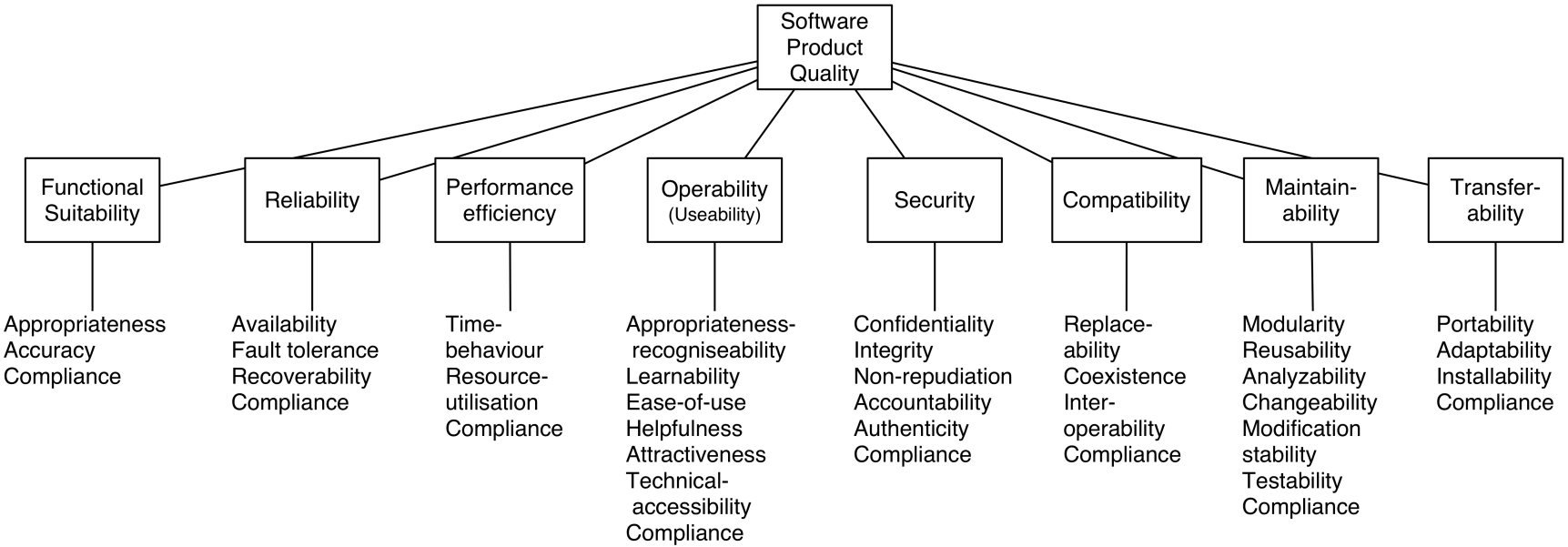
Looking at the example, which option is best in your opinion?

It depends...

Hint: the programming language is irrelevant for the response

Source: https://gist.githubusercontent.com/annaazzam/066077b3a82eb5a035c256134540e45c/raw/ef59982f9b2d8b8582985145978b94cc47d9a579/code_complexity.js

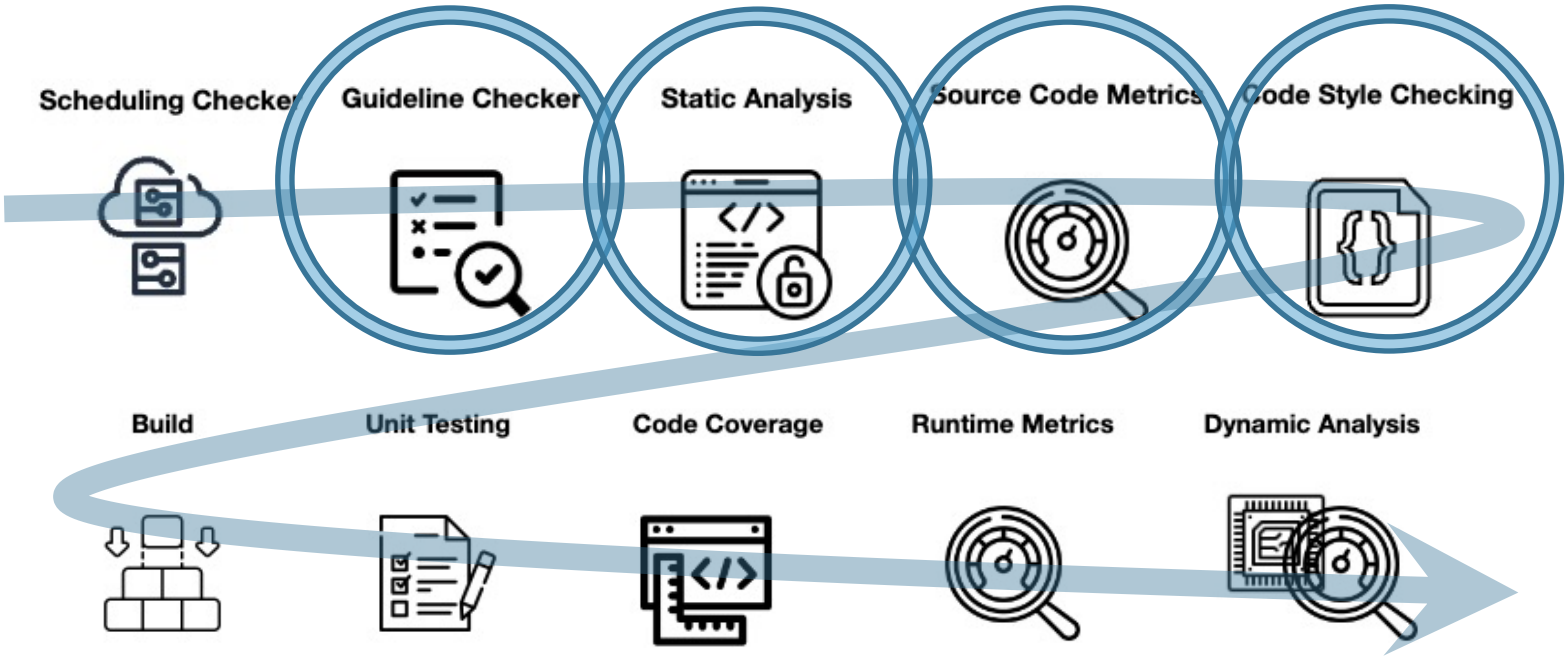
SW Quality: ISO 25010



Source: https://nocomplexity.com/wp-content/uploads/2016/08/ISO-25010-QualityTree.png

Ensure as many aspects as possible

Continuous Checking

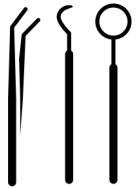


Recall: MISRA Goals

Staff training

Style guide enforcement

Metrics measurements



Tool management

Run-time behaviour

SW Quality: measure != KPI

- It is very important to have SW Quality Metrics as they:
  - Support decision making
  - Improve estimations
  - Increase visibility & code quality
- But…
  - May result in unintended effects (think of a case where “the more code, the higher the salary” – what would happen?)

SW Quality: KISS -> Volume

- #Files / Classes
- #includes
- Total Lines of code (LoC)
- LoC per file
- LoC per method

Complexity: the enemy

Complex systems are harder to test and therefore are more likely to have untested portions.

Complex systems have more interactions and therefore more security bugs

Complex systems have more lines of code and therefore security bugs

Complex systems are harder for users to understand

Complex systems are harder to design, implement, configure and use securely

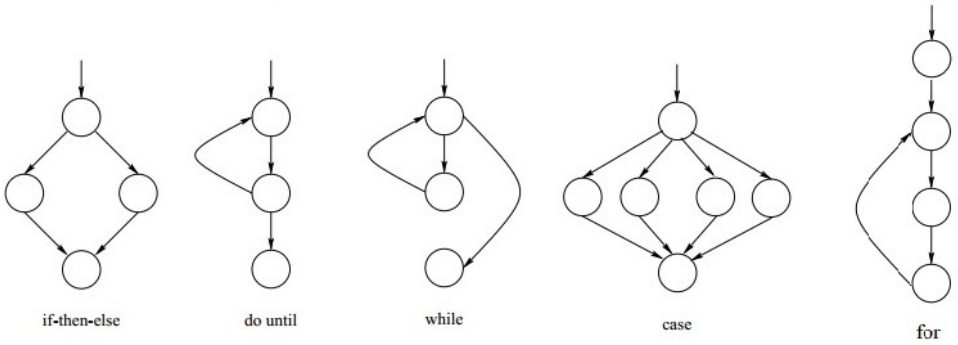
Source: Software Quality Metrics to Identify Risk - Tom McCabe

Essential vs Accidental Complexity

Essential complexity
Unavoidable complexity that crops up because of conscious decisions made in the development process.

Accidental complexity
Unintentional complexity that comes from sloppy coding or poor decision-making in the development process.

SW Quality: Cyclomatic Complexity



"**Cyclomatic complexity** is a measure of the logical complexity of a module and the minimum effort necessary to qualify a module.

Cyclomatic is the number of linearly independent paths and, consequently, the **minimum number of paths that one should** (theoretically) **test**."

Thomas McCabe Jr.

SW Quality: Cyclomatic Complexity

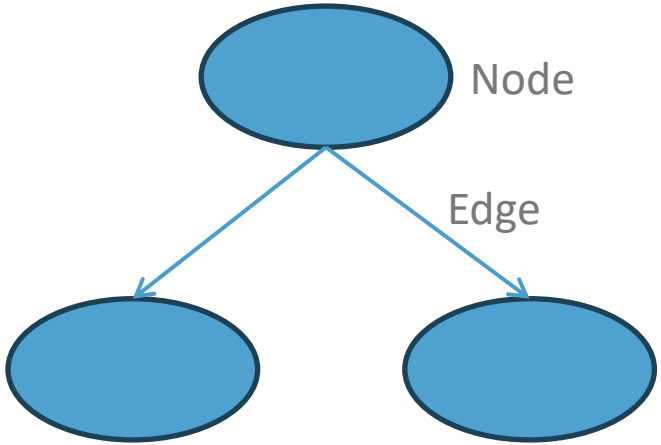
Cyclomatic Complexity
= E - N + 2*P

Where:

E: Edges

N: Nodes

P: Nodes with exit points



Alternatively: CC = D + 1
(D: decision points in control flow)

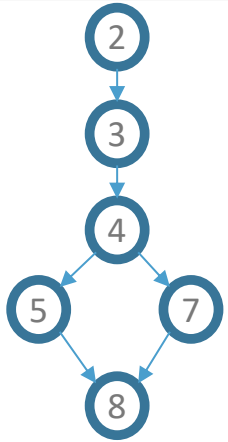
SW Quality: Cyclomatic Complexity

- N: 6
- E: 6
- P: 1

$\Rightarrow CC = 6 - 6 + 2 * 1 = 2$

$\Rightarrow$ Or D = 1 + 1

```
1  // Simple example for Cyclomatic Complexity
2  int a = 1;
3  int b = 2;
4  if (a>b){
5      printf("a is greater");
6  }else{
7      printf("b is greater");
8  }
```



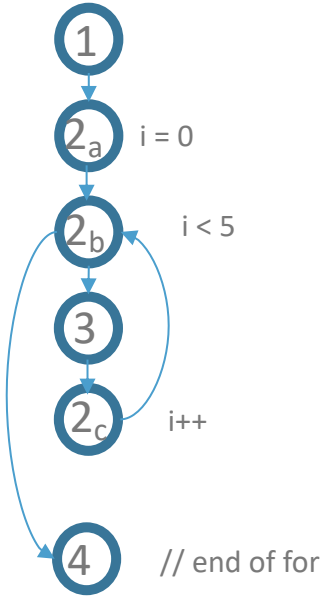
Number represents code line above

SW Quality: Cyclomatic Complexity

```
// simple sequential code
// CC = 2 - 3 + 2 * 1 = 1
// e = 2, n = 3, p = 1 (d = 0)
void a_sequence () {
    int a = 1;
    int b = 2;
    printf("a + b = %d\n", a+b);
}
```

SW Quality: Cyclomatic Complexity

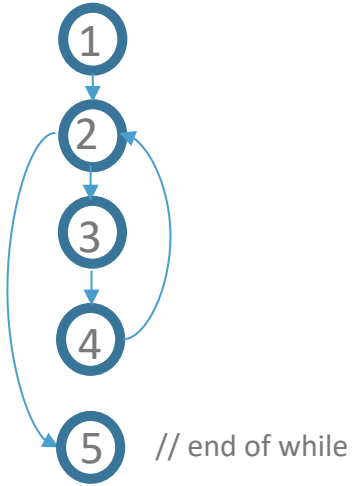
```
// for example
// CC = 6 - 6 + 2 * 1 = 2
// e = 6, n = 6, p = 1 (d = 1)
void a_for (){
    int i;
    for (i = 0; i < 5; i++) {
        printf("i: %d\n", i);
    }
}
```



SW Quality: Cyclomatic Complexity

```
// while example
// CC = 5 - 5 + 2 * 1
// e = 5, n = 5, p = 1 (d = 1)

void a_while (){
    int i;
    while (i < 5) {
        printf("i: %d\n", i);
        i++;
    }
}
```



SW Quality: Cyclomatic Complexity

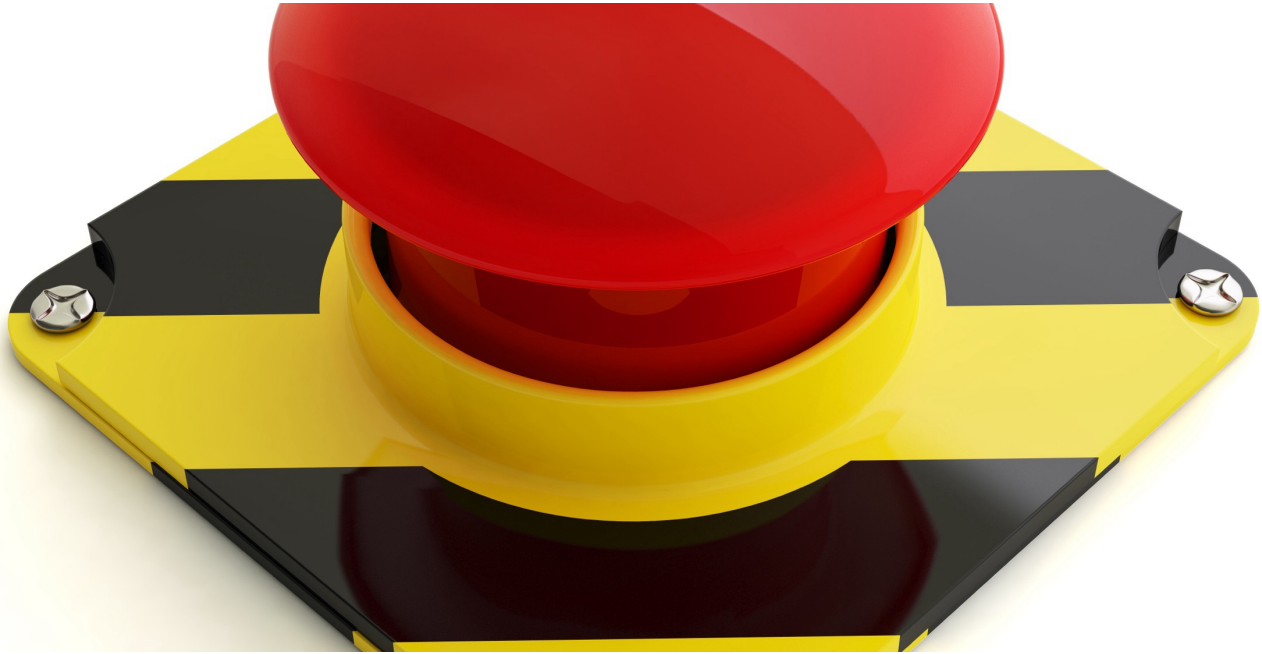
Complexity Number	Meaning	Reliability Risk	Likelihood of bugs*
1-10	**Structured and well written code** High Testability Cost and Effort is low	Little risk	5%
10-20	**Complex Code** Medium Testability Cost and Effort is medium	Moderate	10%
20-40	**Very complex Code** Low Testability Cost and Effort are high	High	30%
>40	**Not at all testable** Very high Cost and Effort	VERY HIGH	40%

*: introducing bugs while modifying (even slightly) the code

SW Quality: Cyclomatic Complexity

So:

1. Prefer Smaller Functions
2. Avoid Flag Arguments in Functions
3. Reduce the Number of Decision Points
4. Get Rid of Duplicated Code
5. Remove Obsolete Code
6. Don't Reinvent the Wheel (aka as “Use patterns”)



SW Quality: Cyclomatic Complexity

Word of caution

SW Quality: Cyclomatic Complexity

Addresses solely control flow – not other dimensions (e.g. data)

Results may differ slightly

Does not support latest programming language features

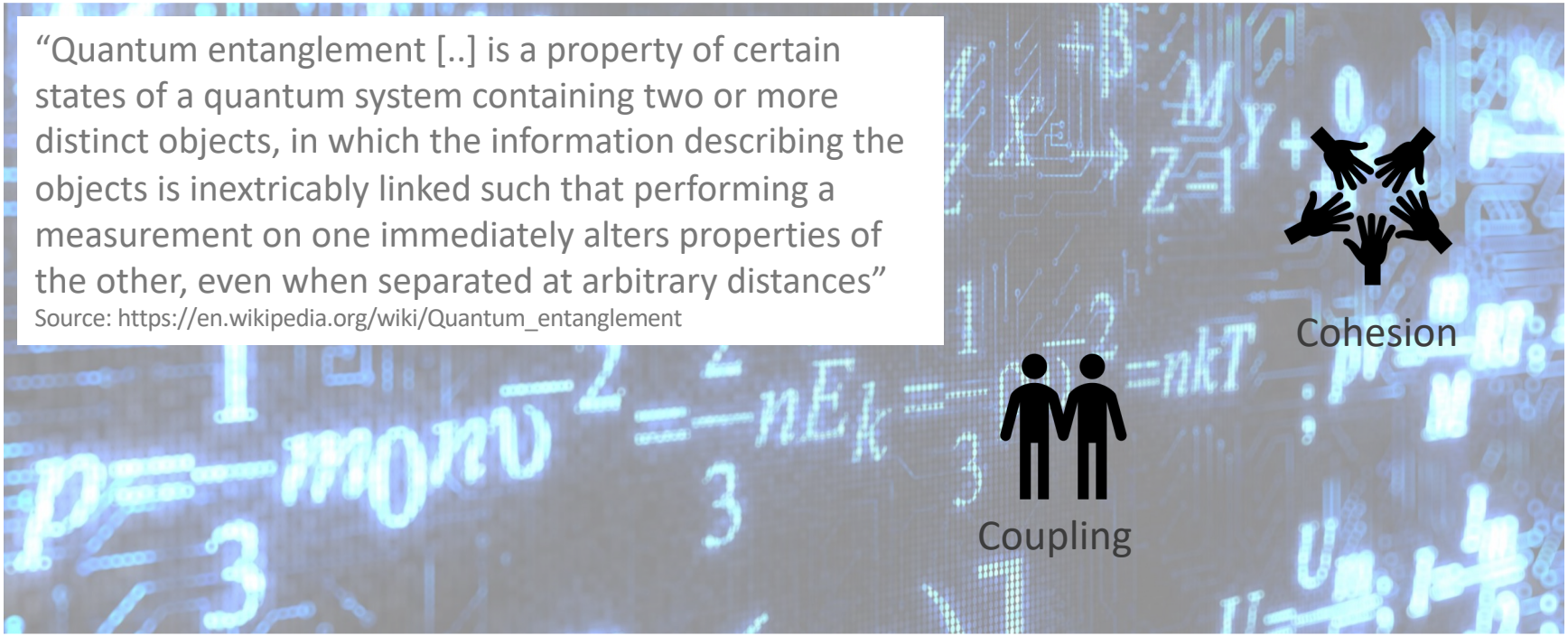
```
int sumOfPrimes(int max) {                  // +1
  int total = 0;
  OUT: for (int i = 1; i <= max; ++i) {     // +1
    for (int j = 2; j < i; ++j) {           // +1
      if (i % j == 0) {                     // +1
        continue OUT;
      }
    }
    total += i;
  }
  return total;
}                   // Cyclomatic Complexity 4
```

```
String getWords(int number) {  // +1
    switch (number) {
      case 1:                  // +1
        return "one";
      case 2:                  // +1
        return "a couple";
      case 3:                  // +1
        return "a few";
      default:
        return "lots";
    }
}      // Cyclomatic Complexity 4
```

Cognitive Complexity != Cyclomatic Complexity

Check Shepperd's 1988 paper and for SonarSource Cognitive Complexity a comprehensive critique of Cyclomatic Complexity

SW Quality: Entanglement



SW Quality: Guidelines

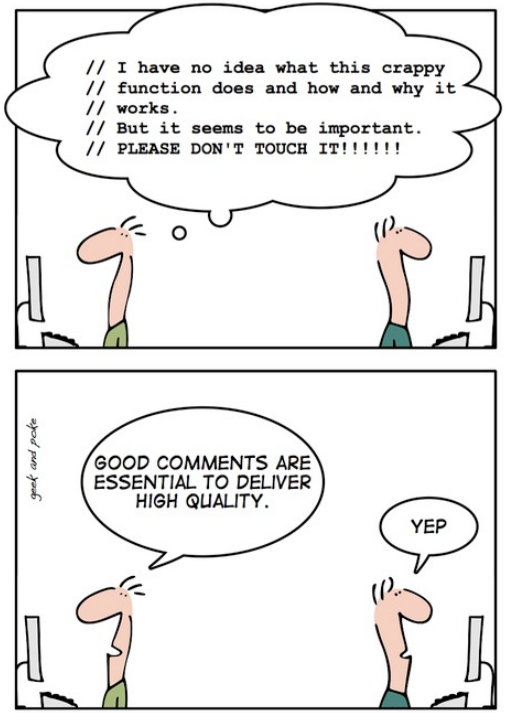
Rationale – in broad terms:

- rules to avoid dangers
- rules to enforce best practices
- rules to ensure consistency

Our style: https://google.github.io/styleguide/cppguide.html

More information: *Style Guides and Rules*, by author Shaindel Schwartz – chapter of "*Software Engineering at Google*" (ISBN: 9781492082798)

SW Quality: Comments



Rule 1: Comments should not duplicate the code.

Rule 2: Good comments do not excuse unclear code.

Rule 3: If you can't write a clear comment, there may be a problem with the code.

Rule 4: Comments should dispel confusion, not cause it.

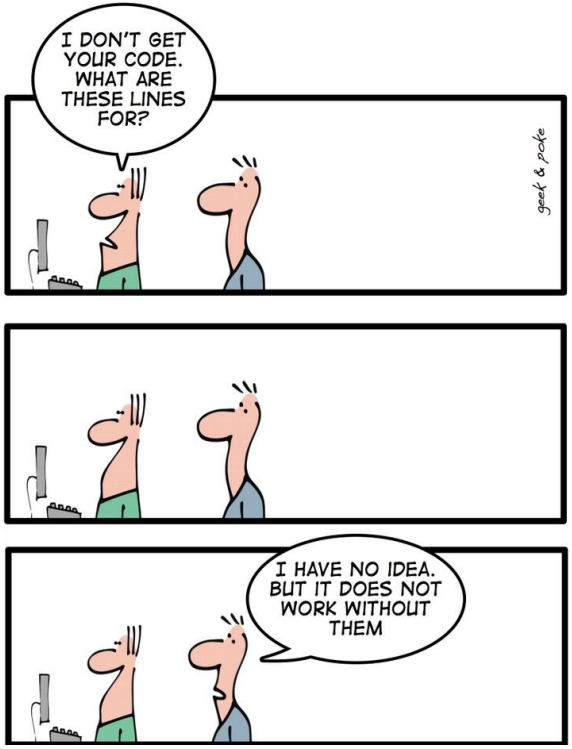
Rule 5: Explain unidiomatic code in comments.

Rule 6: Provide links to the original source of copied code.

Rule 7: Include links to external references where they will be most helpful.

Rule 8: Add comments when fixing bugs.

Rule 9: Use comments to mark incomplete implementations.

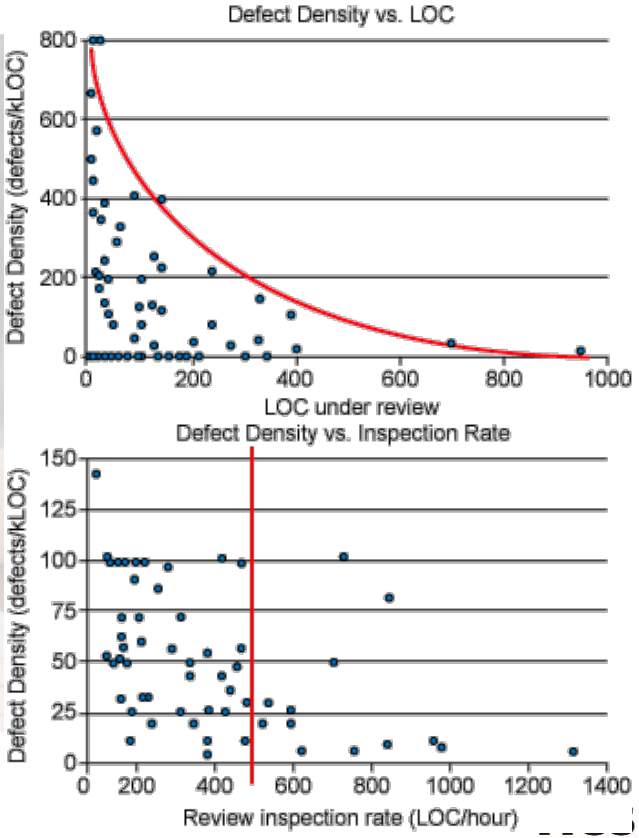


See more details under https://stackoverflow.blog/2021/12/23/best-practices-for-writing-code-comments/

SW Quality: One more thing

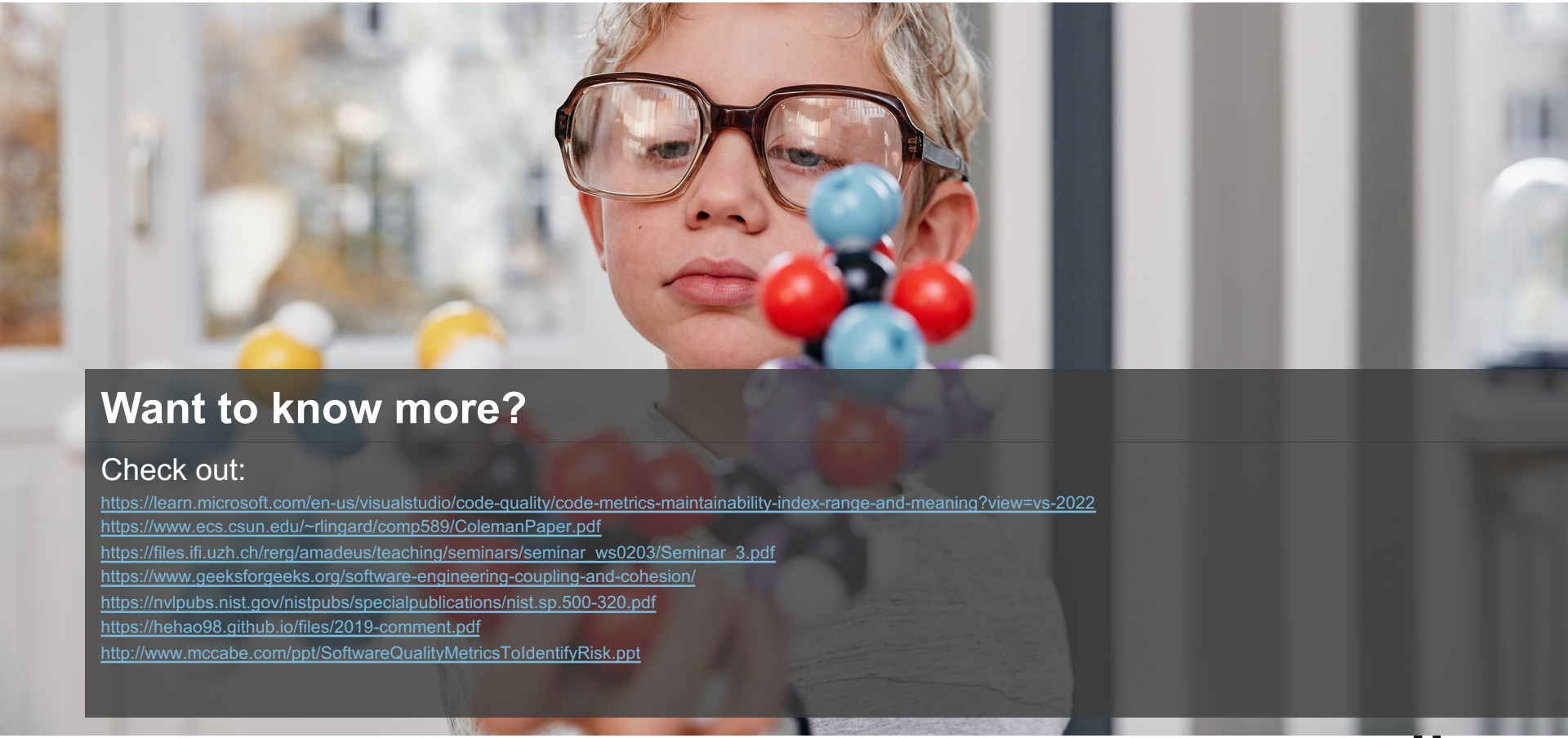
SW Quality: code peer review

1. Review fewer than 400 lines of code at a time
2. Inspection rates should under 500 LOC per hour
3. Do not review for more than 60 minutes at a time
4. Set goals and capture metrics
5. Establish a process for fixing defects found
6. Foster a positive code review culture
7. Embrace the subconscious implications of peer review
8. Practice lightweight code reviews
9. Use checklists



Source: https://smartbear.com/learn/code-review/best-practices-for-peer-code-review/

> “(…) The competent programmer is fully aware of the strictly limited size of his own skull; therefore he approaches the programming task in full humility, and among other things he avoids clever tricks like the plague. (…)”
>
> – Dijkstra (EWD340, 1972)



Want to know more?

Check out:

https://learn.microsoft.com/en-us/visualstudio/code-quality/code-metrics-maintainability-index-range-and-meaning?view=vs-2022
https://www.ecs.csun.edu/~rlingard/comp589/ColemanPaper.pdf
https://files.ifi.uzh.ch/rerg/amadeus/teaching/seminars/seminar_ws0203/Seminar_3.pdf
https://www.geeksforgeeks.org/software-engineering-coupling-and-cohesion/
https://nvlpubs.nist.gov/nistpubs/specialpublications/nist.sp.500-320.pdf
https://hehao98.github.io/files/2019-comment.pdf
http://www.mccabe.com/ppt/SoftwareQualityMetricsToIdentifyRisk.ppt