Refactoring with Cognitive Complexity

the New Option for Measuring Understandability

G. Ann Campbell
@GAnnCampbell

Cognitive Complexity

- Introduced Dec. 2016
- Available as a rule in SonarQube analyzers
  - SonarC#
  - SonarCFamily
  - SonarJava
  - SonarJS
  - …
- https://www.sonarsource.com/docs/CognitiveComplexity.pdf

Cyclomatic Complexity

Intended to measure

- Testability
- Maintainability

Cyclomatic Complexity

```
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
```

```
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
```

Cognitive Complexity Motivations

1. Measure “understandability”

2. Incent good code

Guiding Principles

Guiding Principles

1. Ignore readable shorthand structures

Guiding Principles

1. **Ignore readable shorthand structures**
   - Method Structure
   - Null-coalescing operators

Guiding Principles

1. Ignore readable shorthand structures

- Method Structure
- Null-coalescing operators

```
MyObj myObj = null;
if (a != null) {
 myObj = a.myObj;
}
```

```
MyObj myObj = a?.myObj;
```

Guiding Principles

1. Ignore readable shorthand structures
2. **Increment for breaks in linear flow**

Guiding Principles

1. Ignore readable shorthand structures
2. **Increment for breaks in linear flow**
   - `if`, `else if`, `else`, ternary operators
   - `switch`
   - `catch`
   - Loops
   - Sequences of logical operators (`a && b && c && d // +1`)
   - Jumps to labels
   - Recursion

Guiding Principles

1. Ignore readable shorthand structures
2. Increment for breaks in linear flow
3. **Increment for nesting**

Guiding Principles

1. Ignore readable shorthand structures
2. Increment for breaks in linear flow
3. **Increment for nesting**
   - Loops
   - `switch`
   - `catch`
   - `if` and ternary

Guiding Principles

1. Ignore readable shorthand structures
2. Increment for breaks in linear flow
3. **Increment for nesting**
   - Loops
   - `switch`
   - `catch`
   - `if` and ternary
   - *`else if`, `else` increment nesting level*
   - *Nested method / lambda increments nesting level*

Cognitive Complexity

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

```
int sumOfPrimes(int max) {
  int total = 0;
  OUT: for (int i = 1; i <= max; ++i) { // +1
    for (int j = 2; j < i; ++j) {       // +2
      if (i % j == 0) {                 // +3
        continue OUT;                   // +1
      }
    }
    total += i;
  }
  return total;
}                    // Cognitive Complexity 7
```

Real life code

Recap

- Cognitive Complexity: *understandability*
  - Increments for breaks in linear flow
  - Increments for nesting
  - Ignores readable shorthand structures
- High method Cognitive Complexity indicates a need for refactoring

Questions

@GAnnCampbell