

Analisi statica codice per code smells

Mattia Corradi

Esistono degli strumenti che ci possono aiutare a identificare i code smells in una grande codebase?

Sonar Cloud

The screenshot displays the Sonar Cloud Quality Gate interface. At the top left, a red square with a white 'X' icon is next to the text 'Quality Gate' and a question mark. Below this, the word 'Failed' is prominently displayed in large white font. To the right of 'Failed', the text '3 Failing Conditions' is shown. Further right, there are two buttons: 'New Code' with a '3' next to it, and 'Overall Code'. A large white arrow points downwards from the '3 Failing Conditions' text to the 'Maintainability' section.

The main content area is divided into six sections, each with a progress bar and a grade indicator:

- Reliability**: Grade C (Yellow circle). Sub-section: Bugs.
- Maintainability**: Grade A (Green circle). Sub-section: Code Smells. This section is highlighted by a white arrow.
- Security**: Grade E (Red circle). Sub-section: Vulnerabilities.
- Security Review**: Grade E (Red circle). Sub-section: Security Hotspots (0.0% Reviewed).
- Coverage**: Grade E (Red circle). Sub-section: Coverage.
- Duplications**: Grade A (Green circle). Sub-section: Duplications.

Cyclomatic Complexity

```
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";
        default: // +1
            return "lots";
    }
} // Cyclomatic Complexity 4
```

Cognitive complexity

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

Eslint Rules: **complexity**

Questa regola di ESLint misura la complessità ciclomatica del codice.

Un valore alto può indicare che il codice è difficile da testare e mantenere.

Eslint Rules: **max-lines**

Impone un limite massimo al numero di righe per file. File troppo lunghi possono essere un segno di un'eccessiva complessità e possono rendere il codice più difficile da comprendere e mantenere

ESLint Rules: **max-lines-per-function**

Impone un limite massimo al numero di righe per file. File troppo lunghi possono essere un segno di un'eccessiva complessità e possono rendere il codice più difficile da comprendere e mantenere.

Eslint Rules: **max-depth**

Limita il numero massimo di blocchi annidati. Una profondità eccessiva può rendere il codice più difficile da leggere e seguire.

ESLint Rules: **max-params**

Impone un limite al numero di parametri per le funzioni. Un numero elevato di parametri può indicare che una funzione sta facendo troppo o che i suoi input potrebbero essere meglio strutturati.

ESLint Rules: **no-unused-vars**

Questa regola aiuta a identificare le variabili, funzioni e import non utilizzati, che possono indicare codice inutile o dimenticato.

Bibliografia

- <https://www.sonarsource.com/blog/cognitive-complexity-because-testability-understandability/>

Grazie

m.corradi@davinci.care