1.1 Lungimea claselor

Explicația simplă:

Prea multe linii de cod într-o clasă sugerează că se pot separa unele metode în clase diferite, făcând astfel codul mai ușor de gestionat și reutilizat.

Recomandare:

O anumită clasă face prea multe lucruri. Trebuie avut in vedere principiul „Single Responsibility” și factorizate clasele în consecință.

1.2 Lungimea metodelor

Explicația simplă:

Este o indicatie că o metodă ar putea face mai mult decât sugerează numele ei. Metodele lungi sunt dificil de citit și de înțeles.

Recomandare:

Trebuie redusa lungimea metodelor prin crearea de metode ajutătoare. Nu trebuie reutilizat cod vechi. (e.g.: cod copiat și lipit) De obicei, metodele ajutătoare sunt metode care nu folosesc atributele clasei lor.

1.3 Numar de metode din clasa

Max: nr maxim de metode dintr-o clasa
Sum: suma numerelor metodelor dintr-o clasa (=== nr total de metode din submisie)
Avg: numarul mediu de metode per clasa

Explicația simplă:

Numarul de metode definite intr-o clasa.

1.4 Numarul total de extinderi ale claselor

Explicația simplă:

- Această valoare reprezintă de câte ori a fost extinsa o clasa (de exemplu, de câte ori a fost folosit cuvântul "extends" într-un fragment de cod).

- Semnificatia depinde de problema care trebuie rezolvata.
- **O valoare apropiata de 0 (sau 0) sugereaza ca programatorul nu a facut o proiectare corecta.**

Recomandare:

Trebuie gasit un sablon de proiectare sau identificate relații între clase. Există elemente comune în mai multe clase distincte? Poate ar trebui creata o clasă părinte care să conțină acele elemente.

1.5 Implementari ale interfetelor

Explicația simplă:

- Valoarea reprezinta de cate ori au fost implementate interfete (e.g. de cate ori a fost utilizata constructia „implements” in cod.)
- Semnificatia depinde de problema care trebuie rezolvata.
- **O valoare apropiata de 0 (sau 0) sugereaza ca programatorul nu a facut o proiectare corecta.**

Recomandare:

Trebuie luata in considerare o funcționalitate comună pe care o au entitățile. De exemplu, exista o metodă numită "luptă()" în mai multe clase, dar implementată în moduri diferite? Poate ar fi util să fie creata o interfață "Luptător" și implementata.

1.6 Complexitate ciclomatica

Explicația simplă:

Măsoară numărul de puncte de decizie (modificare a fluxului de control) dintr-o metodă. Dacă metoda are prea multe căi, atunci devine mai dificil de testat și întreținut.

Recomandare:

Trebuie redus numarul de puncte de modificare a fluxului de control (e.g if, for, switch) din metode.
Sugestie: metodele care contin multe astfel de puncte trebuie sparte in metode mai mici.

1.7 Complexitate cognitiva

Explicația simplă:

Complexitatea cognitiva masoara cat de greu este de citit si inteles un cod.

Complexitatea ciclomatica adreseaza **testabilitatea** in timp de complexitatea cognitiva adreseaza **intelegerea.**

Recomandare:

Trebuie redus numarul de blocuri imbricate din metode (e.g un if in interiorul unui if in interiorul unui ciclu while)
Trebuie redus numarul de puncte de decizie din metode.
Sugestie: metodele care contin multe astfel de puncte trebuie sparte in metode mai mici.

1.8 Cuplare

Explicația simplă:

Numar de tipuri diferite utilizate in interiorul unei clase (e.g: Caine, Pisica, Calculator)

1.9 Procent comentarii