Chidamber-Kemerer metrics Report

A métrica Chidamber-Kemerer (CK) tem como objetivo medir a complexidade do design e melhorar o desenvolvimento de software. Isto faz-se através da identificação de valores fora do aceitável para os parâmetros de avaliação da métrica.

As CK metrics consistem em seis parâmetros de avaliação:

**DIT- Depth of Inheritance Tree**

Representa o comprimento máximo desde o nó até à raiz da árvore. (nó = classe?)

Quanto maiores são os valores de DIT, significa que existe um número maior de métodos que é possível herdar dos seus antecessores tornando mais complexo prever o seu comportamento. Por outro lado, existe um maior potencial para reutilizar os métodos herdados.

Com valores mais pequenos de DIT, significa que ao desenvolver o software não damos tanta importância à reutilização em prol da facilidade de compreensão.

Portanto o objetivo deste parâmetro é um trade-off no sentido em que, se escolhe dar maior importância à reutilização, tornando o código mais complexo ou, se prefere facilitar a compreensão limitando a reutilização.

**NOC - Number of Children**

Na hierarquia de classes, representa o número de subclasses que são diretamente filhas de uma classe.

Quanto maior o valor de NOC, maior é a reutilização e maior é a probabilidade de existir abstração inapropriada da classe pai.

Valores baixos de NOC podem ser um indicador de falta de comunicação entre os desenvolvedores de diferentes classes.

Portanto o objetivo deste parâmetro é também um trade-off no sentido em que, se estivermos valores baixos, provavelmente não se trabalhou bem em equipa. Por outro lado, valores elevados podem significar abstração imprópria.

**CBO - Coupling Between Objects**

Representa o número de outras classes às quais está acoplado.

Valores baixos de CBO promovem a encapsulação, indicam independência da classe tornando-a mais fácil de reutilizar. Torna também mais fácil fazer a sua manutenção e criação de testes.

Portanto o objetivo é manter este valor baixo.

**RFC - Response for Class**

Representa a soma entre o número de métodos de uma classe mais o número de métodos chamados pelos métodos dessa classe.

Se o valor de RFC for elevado, então o número de métodos invocados também é elevado e, isto torna a manutenção e o teste da classe mais complexo.

Portanto o objetivo é manter este valor baixo.

**WMC - Weighted Methods per Class**

Representa a soma da complexidade dos métodos de uma classe.

Estima o tempo e o esforço necessário para desenvolver e manter a classe.

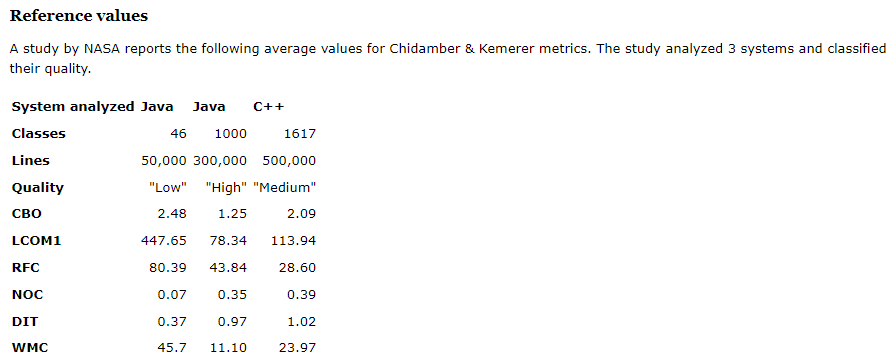
Portanto o objetivo ao desenvolver software é manter este valor baixo.

**LCOM - Lack of Cohesion of Methods**

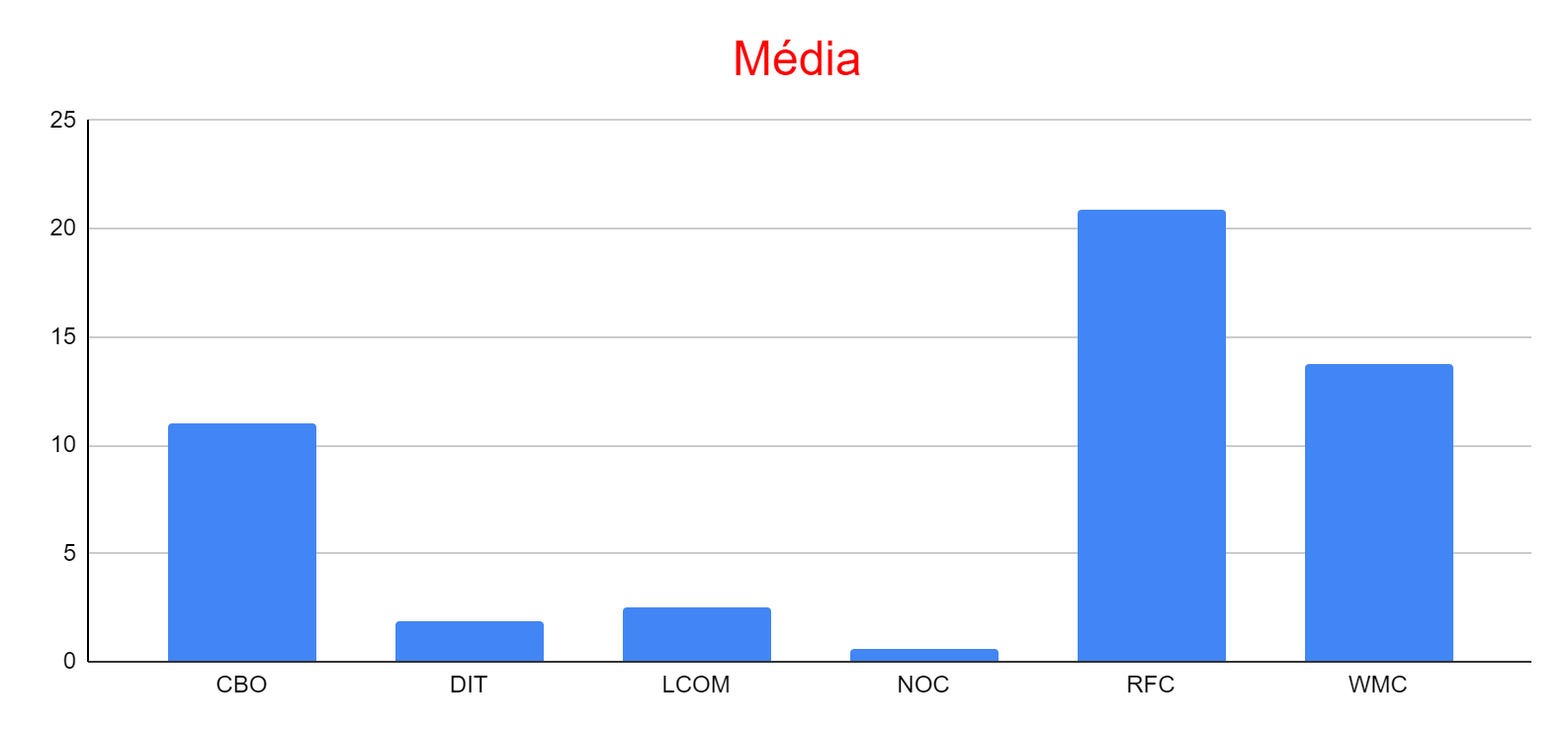
Representa a diferença entre o número de métodos que não têm variáveis de instância em comum, e o número de métodos que têm variáveis de instância em comum

Valores elevados de LCOM indicam elevada complexidade da classe. Quanto maior o valor maior a complexidade. Indicam também que não se promove a encapsulação. Este parâmetro ajuda a identificar design de baixo nível.

O objetivo é manter um valor baixo.

A tabela abaixo contém valores de referência para podemos classificar a qualidade do software e portanto, iremos considerar estes valores para podermos classificar o projeto que estamos a trabalhar.

<https://www.aivosto.com/project/help/pm-oo-ck.html>

Gráfico com os valores médios para a métrica Chidamber-Kemerer do ganttproject.

Nota: em todos os gráficos abaixo o valor médio estará representado por uma linha verde.

Começando pelo CBO, é o valor que se destaca de imediato dos outros valores. Isto é, comparando em proporção com os valores de referência pois, considerando os casos em que foi usada Java como linguagem, o valor de CBO é bastante mais baixo do que o nosso valor (cerca de 5 vezes no caso de baixa qualidade e 10 vezes no caso de alta qualidade).

Isto é um mau sinal pois nós queremos tentar baixar o coupling, ou seja, baixar o valor de CBO.

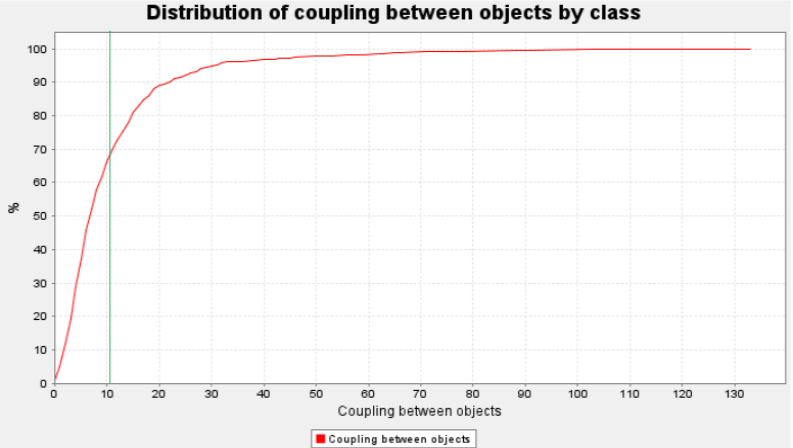
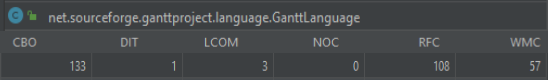


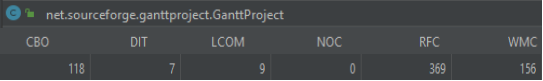
Gráfico com a distribuição de CBO entre as classes.

Analisando a classe com o valor de CBO mais elevado (133). (Classe A)



Esta classe realmente tem o valor mais elevado de CBO mas, se formos analisar a classe em si, verificamos que este valor é tão elevado pois utiliza bastantes classes da própria linguagem Java. Isto não é tão problemático como utilizar classes criadas pelo developer porque, a probabilidade de uma classe da linguagem Java ser alterada e isso fazer com que a classe que a utiliza deixe de funcionar corretamente é menor do que, utilizar várias classes criadas no âmbito de um projeto, e por necessidade, ter que alterar alguma delas e causar efeitos laterais negativos para uma classe que a utiliza.

Analisando a classe com segundo valor de CBO mais elevado (118). (Classe B)



Observamos que esta classe utiliza muitas classes criadas no âmbito deste projeto. Mesmo com um valor mais baixo de CBO em relação à outra classe analisada, o coupling nesta classe pode ser mais preocupante.

Se olharmos para o RFC entre as duas classes, podemos observar que a classe B tem maior complexidade de métodos utilizados (WMC) e tem também maior valor de RFC que a classe

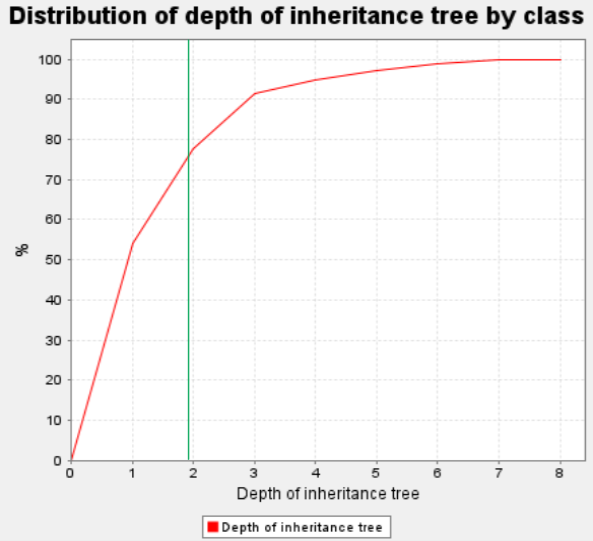


Gráfico com a distribuição de DIT entre as classes.

Podemos observar que em aproximadamente 75% das classes, o DIT está abaixo do valor médio. Em pouco mais de 50% o valor de DIT está abaixo de 1. Tendo em conta os valores de referência, podemos considerar que em 50% das classes o valor de DIT está de acordo com o pretendido. Mas como este indicador é um trade-off entre reutilização e facilidade de compreensão, podemos assumir que 75% das classes têm o valor de DIT menor que 2, ou seja, em 75% dos casos foi escolhido dar prioridade à compreensão e nos restantes 25% foi preferida a reutilização.

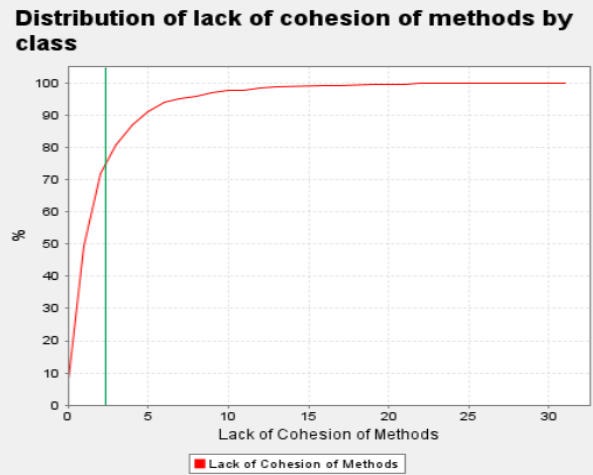


Gráfico com a distribuição de LCOM entre as classes.

O valor médio de LCOM é cerca de 2,5. Este valor é muito baixo quando comparado com o valor de referência que é 78,34. Outro bom indicador é que o valor máximo encontrado para LCOM é 31 que também está abaixo do valor de referência. Com estes valores podemos concluir que se promoveu a encapsulação e que se tentou baixar a complexidade das classes.

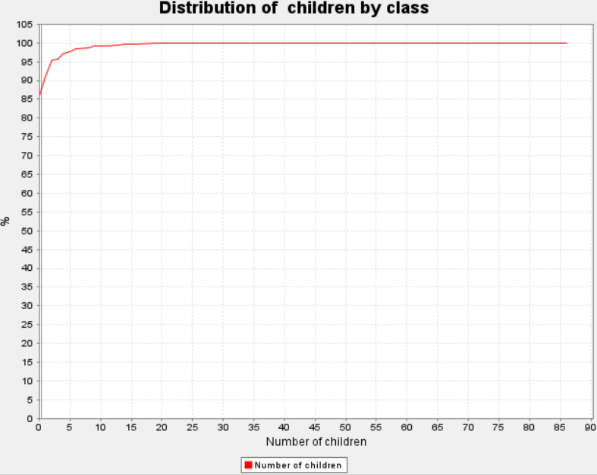


Gráfico com a distribuição de NOC entre as classes.

Tendo em conta o valor de referência (0.35), o valor médio de NOC 0.56 é quase o dobro. Em 85% das classes o valor de NOC está abaixo do valor médio, enquanto que nos 15% restantes, o valor de NOC está muito acima do valor médio e portanto do valor de referência. Estes valores tão elevados podem ser resultado de abstração indevida.

Por exemplo, a classe com maior valor de NOC (86).





Não analisando esta classe profundamente, podemos observar que esta classe tem 133 herdeiros. Ou seja, para além dos 86 herdeiros diretos, tem ainda 47 herdeiros por consequência. Provavelmente existem classes que poderiam estar modeladas de forma diferente de modo a diminuir o valor de NOC.

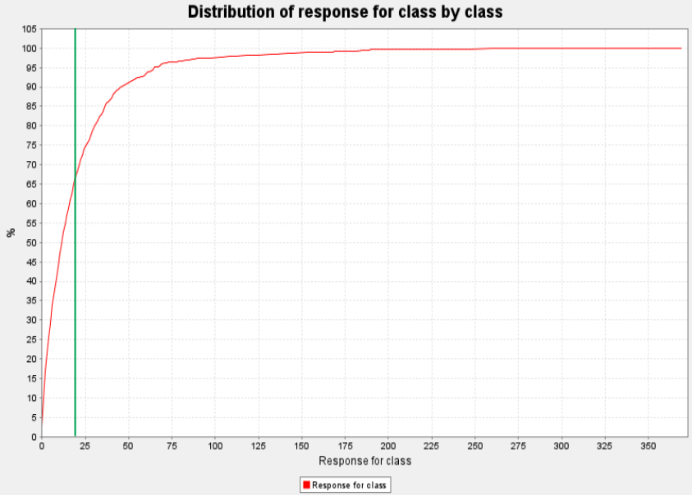


Gráfico com a distribuição de RFC entre as classes.

Este parâmetro tem o valor médio igual a 20.85. Este valor é 2 vezes menor que o valor de referência do projeto de alto nível o que é bastante positivo uma vez que se quer tentar manter este valor baixo.

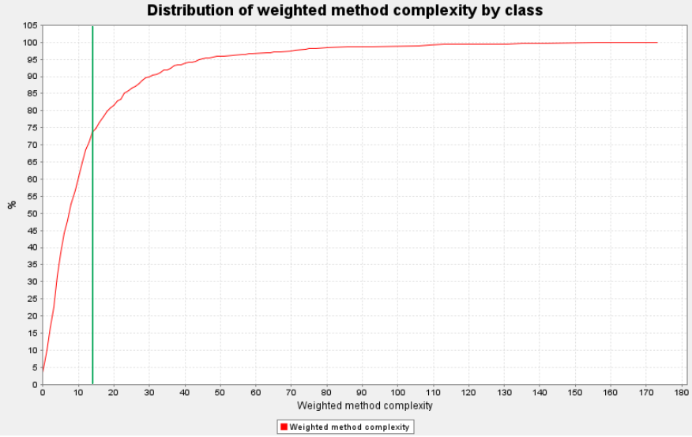


Gráfico com a distribuição de WMC entre as classes.

O valor médio de WMC é aproximadamente 14. Este valor está muito próximo do valor de referência do projeto de alto nível que é 11,1. Este valor significa que se tentou manter baixar a complexidade dos métodos utilizados nas classes.