Métricas para avaliação de Linguagens de Modelação - UML

José Pedro Silva Pedro Faria Ulisses Araújo Costa

Resumo

A utilização de diagramas de *UML* é importante na moelação de uma arquitectura de um sistema de informação, muitas vezes estes diagramas podem ser muito completos e exaustivos de analizar, assim a análise automática de diagramas de *UML* é muito útil na medida em que pode ser um bom indicador da qualidade e complexidade de um sistema, para além de poder dar outras informações.
A teoria por trás da análise automática de diagramas *UML* foi largamente desenvolvida apartir da mesma avaliação que já era feita ao nível de código em linguagens de programação orientadas por objectos, um bom exemplo disso são as *CK metrics*.
Aqui explicamos a investigação que existe nesta área e ainda detalhamos as expressões matemáticas que servem de base para esta análise.

1 Métricas

As métricas para avaliação de software estão bem mais desenvolvidas que as métricas para linguagens de modelação. É então interessante ter em conta o estudo já realizado sobre métricas para avaliação de código, como é o caso das CK Metrics, como ponto de partida para métricas de modelação.
As CK Metrics, umas das primeiras métricas para o modelo Orientado a Objectos (OO), foram propostas por Chidamber e Kemerer [CK94]. O conjunto das CK Metrics consiste em seis métricas: Weighted Methods Per Class (WMC), Depth of Inheritance Tree (DIT), Number of Children (NOC), Coupling between Object Classes (CBO), Response For a Class (RFC), e Lack of Cohesion in Methods (LCOM). Estas métricas foram depois adaptadas para linguagens de modelação. De seguida, será explicado como são calculadas cada uma das métricas referidas.

1.1 Métricas CK

Weighted methods per class (WMC): Esta métrica diz respeito à complexidade que cada métodos tem para cada class. Assim, para se obter o valor desta métrica soma-se as complexidades que cada método pertencente a uma class tem. Se considerarmos a complexidade de cada método como medida unitária

então a métrica WMC para um classe é igual ao número de métodos definidos nessa classe, referimo-nos a isto como WMC1. A métrica WMC1 para uma classe pode ser obtida pelo diagrama de classes de um modelo UML, identificando a classe e contando o número de métodos que essa classe implementa. Aternativamente podemos considerar a complexidade de cada método como o McCabe Cyclomatic Complexity, que referimos como WMCcc. Os diagramas de actividade, sequencia e comunicação conteem informação relevante para o WMCcc, mas é igualmente plausibel que os diagrama de estado possam ser usados para calcular este valor para a casse como um todo.

Depth of inheritance tree (DIT): Esta é uma medida de profundidade da classe relativamente à sua árvore de herança. Esta métrica define-se por ser igual à distaância máxima desde a classe até á sua super classe root na árvore de herança. Esta métrica pode ser calculada para uma classe fazendo a união de todos os diagramas de classe num modelo *UML* e atravessando a hierarquia de herança desta classe.

Number of children (NOC): Esta métrica representa o número de decendentes imediatos de uma determinada classe. Esta métrica pode ser obtida ao juntar todos os diagramas de classes numa modelação *UML* e verificar todas as relações de herança da classe.

Coupling between object classes: Duas classes estão relacionadas se o método de uma classe usa uma variável de instância ou um método da outra classe. Uma estimativa desta métrica pode ser obtida a partir dos diagramas de classes, contando o número de classes relacionadas com a classe em questão e contando todos os tipos de referência dos atributos e todos os parâmetros dos métodos da classe. Para obter um valor mais fidedigno, pode ter em conta informação dada pelos diagramas comportamentais, de forma a obter mais informação sobre o uso das variáveis de instância e de métodos de invocação. O diagrama de sequência, por exemplo, oferece informação directa sobre interacções entre métodos de classes diferentes.

Response for a class (RFC) - esta métrica é a contagem do número de métodos que potencialmente poderão ser invocados por um objecto de uma dada classe. O número de métodos de uma classe pode ser obtido a partir de um diagrama de classes, mas o número de métodos de outras classes que são invocadas por cada um dos métodos da classe requer informação à cerca do comportamento dessa classe. Esta informação pode ser derivada a partir da inspecção de vários diagramas de comportamento (diagramas sequencias/diagramas de actividade), de modo a obter a identidade dos métodos invocados.

Lack of cohesion in methods (LCOM)- ou seja, falta de coesão entre métodos. Calcular esta métrica para uma dada classe envolve descobrir, para cada possível par de métodos, se os conjuntos de variáveis de instância acedidos por cada método têm uma interseção que não um conjunto vazio. Para ser

possível computar um valor para esta métrica, informação do modo de uso das variáveis de instância pelos métodos de uma classe é essencial. Esta informação não pode ser obtida através de uma diagrama de classes. No entanto, o valor máximo para esta métrica pode ser computado usando o número de métodos na classe. Diagramas contendo essa informação sobre o uso das variáveis, por exemplo, os diagramas sequenciais podem ser usados para calcular esta métrica.

O conjunto de métricas aqui definidas são referidos em vários papers e sem sombra de dúvidas são as mais estudadas, e também as mais utilizadas para avaliar modelos *UML*.
Estas focam-se mais nos diagramas de classes visto estes serem os que mais facilmente se relacionam com código e é preciso ter em consideração que como estas regras derivam directamente do paradigma Orientado-a-Objectos, é mais fácil aplicá-las aos diagramas de classes. Para além disso este tipo de diagrams do ponto de vista da implementação dão uma visão mais geral do sistema que modela.
Estas métricas em particular são detalhadas por McQuillan e Power em [MP06].
Também existe o SDMetrics software [SDM] que avalia além destas, um conjunto mais extenso de métricas, que analisam outros diagramas além do de classes, como por exemplo os diagramas de estados e de actividades.
Como grande parte das métricas derivam de fórmulas matemáticas, no capítulo seguinte serão apresentadas algumas que são essenciais para o cálculo dos valores das métricas apresentadas anteriormente.

2 Análise

Após lermos bastante documentação sobre as expressões matemáticas que definem este tipo de avaliação encontramos as medidas definidas por [PIB03] que definimos aqui e explicamos com algum detalhe.

Temos então quatro grandes grupos de medidas que pudemos usar para definir as fórmulas que vão ser usadas para análise dos diagramas e consequentemente extracção de uma métrica de qualidade para os diagramas UML.

Métricas primitivas que consistem na extracção bruta de informaçao no que diz respeito á quantidade de métodos, classes, parametros, etc. Temos que o número total de classes (*Total Number of Classes*) é

$$TNC = \sum_{i=1}^{n} tnc_i$$

o número total de relações herdadas (*Total Number of Inheritance Relationships*)

$$TNIR = \sum_{i=1}^{n} tnir_i$$

o número total de relações que ão sejam herdadas (*Total Number of Realization Relationships*)

$$TNRR = \sum_{i=1}^{n} tnrr_i$$

por isto entende-se uma relação entre dois elementos do modelo UML, entre dois diagramas de classes, cujo elemento cliente conhece o comportamento do outro elemento a que ele está ligado. Temos ainda a contagem do número total de relações que existem (*Total Number of Use Relationships*)

$$TNUR = \sum_{i=1}^{n} tnur_i$$

o número total de associações (*Total Number of Associations*)

$$TNA = \sum_{i=1}^{n} tna_i$$

Uma associação representa uma relação entre duas classes. Por exemplo, numa relação entre duas classes, podemos usar associações para mostrar as decisões de design que fizemos sobre a classe e para mostrar ainda que classe precisa dos atributos da outra. Número total de operações (*Total Number of Operation*)

$$TNO = \sum_{i=1}^{n} tno_i$$

Número total de parametros (*Total Number of Parameters*)

$$TNP = \sum_{i=1}^{n} tnp_i$$

Número total de atributos de uma classe (*Total Number of Class Attributes*)

$$TNCA = \sum_{i=1}^{n} tnca_i$$

Métricas de propensão a falhas (*Fault-Proneness Metrics*) este grupo de métricas tem este nome porque são muito orientadas a hierarquia dos diagramas de classes, assim sendo se houver um erro numa classe root esse erro será propagado para as classes filho que herdam. Temos três formulas que nos ajudam a ganhar conhecimento sobre a propensão a falhas que o nosso diagrama de classes tem: Peso de cada método por classe (*Weighted Method per Class*)

$$WMC = \sum_{i=1}^{n} c_i, \text{ onde } c_i \text{ é a complexidade dos métodos.}$$

Número de sub classes que cada classe tem (*Number of Children per Class*)

$$NOC = \sum_{i=1}^{n} sc_i, \text{ onde } sc_i \text{ é o número de subclasses imediatas.}$$

Profundidade da árvore de herança (*Depth of Inheritance Tree*)

$DIT = max\_leng$, onde max_leng é o comprimento máximo desde a raiz até à folha.

Métricas de qualidade (*Quality Measure Metrics*) Este grupo de métricas são as mais interessantes do ponto de vista da qualidade, visto que usam as anteriormente definidas como ferramenta para se chegar a uma conclusão sobre a qualidade do diagrama UML em questão. Os diagramas de UML, tal como as linguagens de programação que recorrem ao modelo orientado a objectos teem a noção de protecção de elementos que definem uma class, quer seja um método, uma variável de instância ou uma propriedade. Por isso é importante medir também este factor, assime existe o MHF (*Method Hiding Factor*), podemos dizer que esta métrica é a medida do uso de informação através de métodos e define-se por:

$$\frac{\sum_{i=1}^{rc} \sum_{m=1}^{Md(c_i)} (1 - V(M_{mi}))}{\sum_{i=1}^{rc} Md(c_i)}$$

onde:

$$V(M_{mi}) = \frac{\sum_{j=i}^{rc} is\_visible(M_{mi}, C_j)}{TC - 1}$$

$$is\_visible(M_{mi}, C_j) = \begin{cases} 1 & iff \begin{cases} j \neq i \\ C_j \texttt{may call} M_{mi} \end{cases} \\ 0 & otherwise \end{cases}$$

$TC =$ Número total de classes

$Md =$ Número total de métodos definidos

$V(M_{mi}) =$ A visibilidade de todas as classes onde o método M_{mi} é visível

De seguinda temos a métrica que mede a herdagem através de atributos (*Method Inheritance Factor*

$$MIF = \frac{\sum_{i=1}^{rc} M_i(C_i)}{\sum_{i=1}^{rc} M_a(C_i)}$$

onde:

$M_a(C_i) = Md(C_i) + M_i(C_i)$, é o número total de métodos disponíveis

De recordar que por métodos disponíveis entende-se os definidos localmente mais os herdados.

Por último temos o factor de herança de atributos (*Attribute Inheritance Factor*)

$$AIF = \frac{\sum_{i=1}^{rc} A_i(C_i)}{\sum_{i=1}^{rc} A_a(C_i)}$$

onde:

$A_a(C_i) = Ad(C_i) + A_i(C_i)$, é o número total de atributos disponíveis

Métricas sobre Use Cases (*Use Case Metrics*) embora estejamos mais focados para as métricas sobre diagramas de classes, também existem métricas que se aplicam a outros tipos de diagramas, como os Use Cases, assim é importante também explicar algumas elas relativamente a uma anãlise quantitativa do sistema no que diz respeito à informação que conseguimos extraír de um diagrama deste tipo. Assim conseguimos obter a complexidade do sistema através da informação obtida de um diagrama Use Case: Contabilizar o número de actores, para ter uma noção sobre a quantidade de grupos de pessoas que vão interagir com o sistema (*Number of actors*)

$$NOA = \sum_{i=1}^{n} noa_i$$

O número de Use Cases que o sistema modela (*Number of use cases*)

$$NOUC = \sum_{i=1}^{n} nouc_i$$

O númeor de Use Cases que cada actor tem diz-nos ou pode indicar a extensão com que cada utilizador pode usufruir do sistema (*Use cases per Actor*)

$$UCPA = \sum_{i=1}^{n} nouca_i$$

Referências

[CK94] S. R. Chidamber and C. F. Kemerer. A metrics suite for object oriented design. *IEEE Trans. Softw. Eng.*, 20:476–493, June 1994.

[MP06] Jacqueline A. Mcquillan and James F. Power. A definition of the chidamber and kemerer metrics suite for the unified modeling language. Technical report, 2006.

[PIB03] SangEun Kim Peter In and Matthew Barry. Uml-based object oriented metrics for arquitecture complexity analysis. Technical report, 2003.

[SDM] Sdmetrics. `http://www.sdmetrics.com/`.