SOFT
DISCIPLINA: Engenharia de Software **DATA: _____/_____/_____**
AULA NÚMERO: 16 **PROFESSOR: Andrey**

APRESENTAÇÃO

O objetivo desta aula é apresentar, discutir o conceito de métricas de software.

DESENVOLVIMENTO

Métricas de Software

Métrica é uma medida quantitativa do grau em que um sistema, componente ou processo possui um determinado atributo.

Métricas de Software devem ser:
- Simples e computáveis
- Empíricas e intuitivamente persuasivas
- Consistentes e objetivas
- Consistentes nas unidades e dimensões
- Independente de linguagem
- Mecanismo efetivo para realimentação da alta qualidade.

O software é medido por muitas razões:
- Para indicar a qualidade do produto;
- Avaliar a produtividade das pessoas que produzem o produto;
- Avaliar benefícios derivados de novos métodos e ferramentas de software;
- Formar uma linha básica para estimativas;
- Ajudar a justificar os pedidos de novas ferramentas ou treinamento adicional.

De forma análoga a outras grandezas do mundo físico, as medições de software podem ser classificadas em duas categorias principais:

- As **medições diretas**, por exemplo, o número de linhas de código (LOC) produzidas, o tamanho de memória ocupado, a velocidade de execução, o número de erros registrados num dado período de tempo, etc.
- As **medições indiretas**, as quais permitem quantizar aspectos como a funcionalidade, complexidade, eficiência, manutenibilidade, etc...

As medições de software podem ser organizadas em outras classes, as quais serão definidas a seguir:

- **Métricas da produtividade**, baseadas na saída do processo de desenvolvimento do software com o objetivo de avaliar o próprio processo;
- **Métricas da qualidade**, que permitem indicar o nível de resposta do software às exigências explícitas e implícitas do cliente;
- **Métricas técnicas**, nas quais encaixam-se aspectos como funcionalidade, modularidade, manutenibilidade, etc...

Sob uma outra ótica, é possível definir uma nova classificação das medições:

- **Métricas orientadas ao tamanho**, baseadas nas medições diretas da saída e da qualidade da engenharia de Software;
- **Métricas orientadas à função**, que oferecem medidas indiretas;
- **Métricas orientadas às pessoas**, as quais dão indicações sobre a forma como as pessoas desenvolvem os programas de computador e percepções humanas sobre a efetividade das ferramentas e métodos.

Principais métricas de Software:

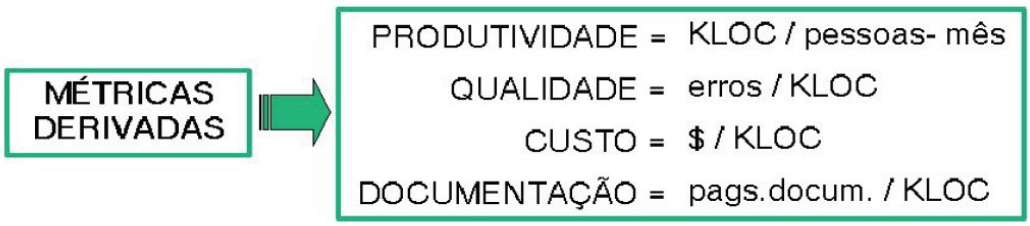
- Pontos por função
- Pontos por caso de uso
- Independência funcional (coesão e acoplamento)
- Métricas OO (Chidamber e Kemerer, Lorenz e Kidd)
- Linhas de código (LOC)
- Complexidade Ciclomática

Métricas Orientadas ao Tamanho

Métricas de software orientadas ao tamanho são medidas diretas do software e do processo por meio do qual ele é desenvolvido. Se uma organização de software mantiver registros simples, uma tabela de dados orientada ao tamanho poderá ser criada. A tabela relaciona cada projeto de desenvolvimento de software que foi incluído no decorrer dos últimos anos aos correspondentes dados orientados ao tamanho deste projeto. A partir dos dados brutos contidos na tabela, um conjunto de métricas de qualidade e de produtividade orientadas ao tamanho pode ser desenvolvido para cada projeto. Médias podem ser computadas levando-se em consideração todos os projetos.

Exemplo:

projeto	esforço	$	KLOC	pags.docum.	erros	pessoas
projA-01	24	168	12.1	365	29	3
projB-04	60	440	27.2	1224	86	5
projC-03	48	314	20.2	1050	64	6



A seguir constam as vantagens de desvantagens do uso da métrica orientada ao tamanho:

Vantagens:

- Fáceis de serem obtidas;
- Vários modelos de estimativa baseados em LOC ou KLOC.

Desvantagens:

- LOC depende da linguagem de programação;
- Penalizam programas bem projetados, mas pequenos;
- Não se adaptam às linguagens não procedimentais; e
- Difícil de obter em fase de planejamento.

Segundo Pressman [PRESSMAN, 2005], a mais amplamente referenciada dentre as métricas orientadas a objetos é o conjunto de métricas criado por Chidamber e Kemerer. Este conjunto tem como vantagem medir as características de classes individuais como número e tamanho dos métodos, interações com outras classes, herança e encapsulamento, entre outras. Como os requisitos funcionais neste nível são serviços técnicos e os parâmetros de projeto são objetos ou variáveis é necessário usar uma métrica que contemple características de classes, atributos e métodos. Nesta métrica são definidas seis grandezas que serão medidas:

1. Métodos ponderados por classe
2. Profundidade da árvore de herança
3. Número de subclasses
4. Acoplamento entre objetos
5. Respostas para a classe
6. Falta de coesão nos métodos

Métodos Ponderados por Classe

A grandeza "métodos ponderados por classe" (WMC, do inglês weighted methods per class) representa o total das complexidades dos métodos de uma classe. O valor é dado pela soma das complexidades de cada método. Uma forma simplificada de calcular o WMC é considerar os métodos com o mesmo valor para complexidade, sendo este valor igual a 1. Neste caso, o valor do WMC é igual ao número de métodos da classe. No caso de métodos maiores, pode-se calcular a complexidade do método usando-se métricas de tamanho de programa. A mais fácil de ser computada é o número de linhas de código (LOC[1]). Pode-se, também, calcular a complexidade de cada método usando a complexidade ciclomática.

O número de métodos de uma classe e a complexidade destes ajuda a estimar o tempo e o esforço para desenvolver e manter a classe. Classes com um número grande de métodos são mais específicas para determinadas aplicações e limitam a possibilidade de reuso. Sendo c_i a complexidade do método i de uma classe, WMC pode ser calculado pela Equação (19).

$$WMC = \sum_{n}^{i=1} c_i \qquad (1)$$

Profundidade da Árvore de Herança

A profundidade da árvore de herança (DIT, do inglês depth of the inheritance tree), para uma classe, é definida como sendo o comprimento máximo do nó que representa a classe até a raiz da árvore (classes mais abstratas). Árvores de herança muito profundas geram muita complexidade no projeto pelo fato de mais classes e métodos estarem envolvidos. A herança, ou generalização, pode aumentar a complexidade de uma classe pois o projetista deve conhecer, além dos métodos da própria classe, todos os métodos e atributos relacionados que esta classe herda. Neste caso, quanto mais profunda for a árvore de herança, maior o número de métodos a ser considerado. A Figura 1 ilustra uma árvore de herança. Neste exemplo, o valor de DIT para a classe "AlunoGraduacao" é igual a 2.

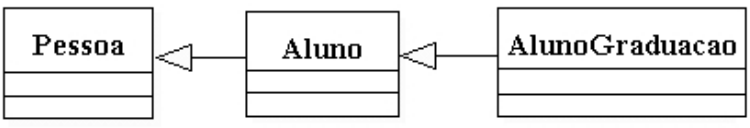


Figura 1 - Profundidade da árvore de herança (DIT)

Número de Subclasses

O número de subclasses (NOC, do inglês number of children) é o número de subclasses diretas de uma classe. Segundo CHIDAMBER e KEMERER (1994), quanto maior o número de

[1]do inglês, *Lines of Code*

subclasses diretas de uma classe, maior a possibilidade de que a herança tenha sido usada incorretamente para esta classe, em outras palavras, é provável que estejam faltando níveis da árvore de herança. Uma classe com muitas subclasses diretas possui um potencial muito grande de propagação dos efeitos da mudança em um dos seus métodos, exigindo mais testes. A Figura 2 ilustra uma árvore de hierarquia. O valor para NOC para a classe "Obra" é igual a 4, pois ela tem 4 subclasses diretas.

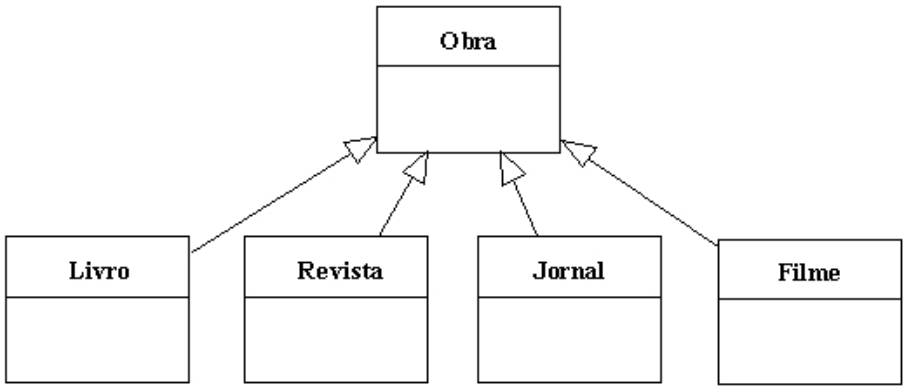


Figura 2 - Número de subclasses (NOC)

Acoplamento entre Objetos

A grandeza "acoplamento entre objetos" (CBO, do inglês coupling between objects) de uma classe é definida como sendo o número de outras classes com as quais esta classe está acoplada (relacionada através de uma associação). Dois objetos estão acoplados quando os métodos de um objeto usam métodos ou variáveis de instância do outro. Em outras palavras, o número de classes das quais esta classe usa métodos ou variáveis de instância. Uma maneira prática para determinar o CBO é usando os cartões classe-responsabilidade-colaborador (CRC). Um colaborador é uma classe que presta serviços para que uma outra classe consiga realizar suas responsabilidades.

Acoplamento excessivo entre objetos de um sistema prejudica o desenvolvimento modular, além de dificultar o reuso. Mesmo que esse acoplamento seja através de chamadas de métodos, ele faz com que o projetista precise se concentrar em várias outras classes além daquela que está projetando. Encapsulamento é uma forma de prevenir o acoplamento. Outra forma é projetar as classes de forma que o projeto satisfaça o Axioma da Independência.

Resposta para uma Classe

A grandeza "resposta para uma classe" (RFC, do inglês response for a class) é definida como a cardinalidade do conjunto de respostas de uma classe.

O conjunto de respostas de uma classe é o conjunto de métodos de uma classe que podem potencialmente ser executados em resposta a uma mensagem recebida. Quanto maior o conjunto de métodos que podem ser chamados de uma classe, maior a complexidade da classe. Se um grande número de métodos de uma classe pode ser chamado por outras classes, as operações de teste e correção tornam-se mais complexas, exigindo maior experiência por parte do desenvolvedor. Sendo RS, o conjunto de resposta da classe, $\{R_i\}$, o conjunto dos métodos da classe chamados pelo método i e {M}, o conjunto de todos os métodos da classe. O conjunto de resposta para cada método M_i corresponde ao método M_i e todos os métodos da classe chamados por M_i $\{R_i\}$. O conjunto de resposta de uma classe, RS, é calculado pela união dos conjuntos de respostas de todos os métodos da classe.

$$RS = \{M\} \cup_{all\,i} \{R_i\} \quad (2)$$

Falta de Coesão nos Métodos

Considerando uma classe com um conjunto de métodos M_1, M_2, M_3, ..., M_n e $\{I_j\}$ o conjunto de

variáveis de instância usados pelo método M_j. O conjunto P é definido como sendo o conjunto de pares de métodos (M_i, M_j) tal que $\{I_i\} \cap \{I_j\} = \emptyset$. O conjunto Q é definido como sendo o conjunto de pares de métodos (M_i, M_j) tal que $\{I_i\} \cap \{I_j\} \neq \emptyset$.

$$LCOM = |P| - |Q|, se |P| > |Q|$$
$$LCOM = 0, caso\ contrário \quad (3)$$

A grandeza LCOM é a diferença entre o número de pares de métodos de uma classe que não compartilham um mesmo conjunto de variáveis de instância (atributos) e o número de pares que compartilham. LCOM mede a coesão entre os métodos de uma classe. Se LCOM é alto, isto pode significar que a classe pode ser dividida em duas ou mais sub-classes. Como visto no capítulo 2, a coesão de um módulo de software mede o grau com que as tarefas executadas por este módulo se relacionam entre si. Quando dois métodos, de uma mesma classe, não compartilham atributos entre si, pode-se dizer que o grau de relacionamento entre eles é baixo e, portanto, a coesão entre eles é baixa.

Como exemplo de valores obtidos para as métricas de Chidamber e Kemerer, são apresentados dois estudos realizados. O primeiro estudo foi realizado por Chidamber e Kemerer e compara os valores obtidos para as métricas em duas organizações diferentes. A primeira organização, chamada site A, desenvolve software para vendas e possui uma coleção de bibliotecas de classes C++. O estudo nesta organização contempla 634 classes de duas bibliotecas usadas para desenvolver interfaces gráficas para usuários (GUI). A segunda organização deste estudo é um fabricante de semi-condutores que usa Smalltalk para desenvolver sistemas de controle flexíveis de manufatura. Para esta organização foram analisadas 1459 classes.

Tabela 1 - Valores comparativos para as métricas CK entre classes em C++ e Smalltalk

	WMC, $c_i = 1$	DIT	NOC	CBO	RFC	LCOM
Site A - C++ 634 classes	5	1	0	0	6	0
Site B - Smalltalk 1459 classes	10	3	0	9	29	2
Valores relativos às medianas de cada amostra.						

Na Tabela 1, são mostrados os valores para as métricas de Chindamber e Kemerer para o estudo realizado em. É necessário considerar as diferenças entre as linguagens de programação e principalmente as diferenças entre os tipos e propósitos das classes analisadas. Outro detalhe importante a ser considerado é que neste estudo os valores para a métrica "métodos ponderados por classe" (WMC) levam em consideração apenas o número de métodos de cada classe ($c_i = 1$).

Num outro estudo, Verves, van Dalen e van Katwijk obtiveram valores para algumas métricas orientadas a objetos, entre elas 5 das 6 métricas de Chidamber e Kemerer [VERVERS; van DALEN; van KATWIJK, 1996]. Foram medidos valores para 958 classes escritas na linguagem Objective C.

Tabela 2 - Valores das métricas CK no estudo de Ververs, vDalen e vKatwijk (1996)

	WMC ciclomático	Linhas p/ método	Complexidade ciclomática	DIT	NOC	CBO	LCOM
Média	59,27	9,44	2,75	3,4	1	5,45	286,48
Desvio padrão	76,22	6,38	1,52	2	8,21	7,22	1279,81
Mediana	37	7,95	2,39	3	0	3	11

Os valores obtidos para a métrica WMC, apresentados na Tabela 2, levam em consideração a complexidade de cada método. Para obter esta complexidade foi usada a métrica da complexidade ciclomática. Além disso foram obtidos os valores médios para número de linhas de código por método, que é uma medida de complexidade mais simples de ser obtida.

ATIVIDADE

1. Utilizando métricas orientada ao tamanho, encontre as medidas para os seguintes elementos:
- Produtividade;
- Custo; e
- Qualidade.

Para o cálculo, use os seguintes valores para cada software:

Projeto	*Esforço*	*Valor ($)*	*Pág. Docum.*	*KLOC*	*Erros*	*Pessoas*
Zzs-e	12	125	1569	12,5	235	5
Ddf-98	16	253	3268	16,3	165	6
Kjf0-8	22	572	12487	25,6	632	8
73pe-u	10	322	5487	8,2	197	4

BIBLIOGRAFIA BÁSICA

PRESSMAN, R. S.. *Engenharia de Software*. Makron Books. 1995

PRESSMAN, R. S.. Software Engineering: a practitioner's approach. 6a. ed .New York, EUA: McGraw-Hill, 2005.

CHIDAMBER, S. R.; KEMERER, C. F. A Metrics Suite for Object-Oriented Design. IEEE Transactions on Software Engineering, New York, EUA, v. SE-20, n. 6, p. 476-493, jun. 1994.

VERVERS, F; van DALEN, P; van KATWIJK, J. A case study in the application of object oriented metrics. Anais do 9th World Conference on Integrated Design and Process Technology. Austin, EUA: 1996.