Engenharia de Software Orientada a Objetos

Avaliação da Qualidade: medindo o produto

Guilherme H. Travassos

Programa de Engenharia de Sistemas e Computação
ght@cos.ufrj.br
http://www.cos.ufrj.br/~ght

COPPE
UFRJ

Engenharia de Software OO

- Bibliografia

[Pfleeger00]
Shari Lawrence Pfleeger; Software Engineering: Theory and Practice, Second Edition, Prentice Hall, 2000 Chapter 6

[Travassos et all01]
Travassos, G.H., Shull, F. and Carver, J.; Working with UML: A Software Design Process Based on Inspections for the Unified Modeling Language, in Advances in Computers, vol. 54, Academic Press, 2001

Chidamber, S.R. and Kemerer, C. F.; A Metrics Suite for Object Oriented Design, IEEE Transactions on Software Engineering, vol.20, no6, June 1994.

Basili, Victor R.; Briand, Lionel and Melo, Walcelio; A validation of Object-Oriented design metrics, Technical Report, Univ. of Maryland, Dep. Of Computer Science, CS-TR-3443, May, 1995

W. Lie and S. Henry (1993). Object-oriented metrics that predict maintainability, Journal of Systems and Software. 23(2):111-122.

Travassos, G. H. and Andrade, R. S., Combining Metrics, Principles and Guidelines for Object Oriented Design, Workshop on Quantitative Approaches on Object Oriented Software Engineering, ECOOP'99, Lisbon, Portugal, 1999 (http://www.esw.inesc.pt/ftp/pub/esw/mood/ecoop99/)

Medição do Produto

- **Modelos de Software e Medidas: Escopo**
- Precisamos definir modelos para
  - Ajudar a entender o que estamos fazendo
  - Fornecer uma base para definir objetivos
  - Fornecer uma base para medição
- Precisamos modelos de:
  - pessoas, i.e., cliente, gerente, desenvolvedor
  - processos, i.e., um ciclo de vida, método, técnica
  - produtos, o sistema,um componente, um plano de teste
- Precisamos estudar as interações entre estes modelos
  - Qual o efeito da modificação de um processo no produto?
- Precisamos associar métricas a estes modelos
  - Como medir processo?

Medição do Produto

- **Modelos de Software e Medidas**
- O que podemos medir?
  - Dados sobre recursos:
    - Esforço por atividade, fase, tipo de pessoal, tempo de computação, tempo de desenvolvimento
  - Dados sobre alterações/defeitos:
    - Alterações e defeitos de acordo com diferentes esquemas de classificação
  - Dados sobre processo:
    - Definição do processo, Conformidade com o processo, compreensão do domínio
  - Dados sobre produto:
    - Características do produto
      - Lógicas, i.e., domínio de aplicação, função
      - Físicas, i.e., tamanho, estrutura
      - Operacional, i.e., confiabilidade
    - Informação de uso e contexto, i.e., método de projeto utilizado

Medição do Produto

- **Modelos de Software e Medidas**
  - Recursos
    - modelos de custo locais, modelos de alocação de recursos
  - Modificações e Defeitos
    - modelos de predição de defeitos, tipos de defeitos esperados para a aplicação
  - Progresso do Produto
    - tamanho de produto atual vs. esperado, acesso a bibliotecas, excesso de tempo
  - Processos
    - modelo de processo para Cleanroom, modelo de processo para OO
  - Avaliações de Métodos e Técnicas
    - melhor método para encontrar falhas em interface
  - Produtos
    - compomentes genéricos Ada para simulação de órbita de satélites
  - Qualidade
    - modelos de confiabilidade, modelos de facilidade de alteração
  - Lições Aprendidas
    - riscos associados com desenvolvimento C++

Medição do Produto

- **Modelos de Software e Medidas: Perspectivas**
  - De que ponto de vista estamos medindo?
    - Existe uma variedade de pontos de vista que determinam o que medimos
      - Gerenciamento
      - Cliente
      - Usuário
      - Organização
      - Desenvolvedor
  - Por que estamos medindo?
    - Existem várias razões para medir que ajudam a definir o que medimos, i.e.:
      - Caracterização e Compreensão
      - Identificação (Assessment) e Avaliação
      - Predição e Controle
      - Motivação e Prescrição
      - Melhoria

Medição do Produto

- **Modelos de Software e Medidas**
  - Caracterizar
    - Descrever e diferenciar processos e produtos de software
    - *Construir modelos descritivos e "baselines"*
  - Compreender
    - Explicar associações/dependências entre processos e produtos
    - Descobrir relacionamentos causais
    - *Analisar modelos*
  - Avaliar
    - Identificar a realização dos objetivos de qualidade
    - Identificar o impacto da tecnologia nos produtos
    - *Comparar modelos*
  - Predizer
    - Estimar a qualidade esperado do produto e consumo de recursos no projeto
    - *Construir modelos de predição*
  - Motivar
    - Descrever o que precisamos fazer para controlar e gerenciar o software
    - *Construir modelos prescritivos*

Medição do Produto

- **Modelos de Software e Medidas**
  - O que podemos fazer com as medidas?
    - **Criar uma memória corporativa –** "baselines"/modelos de práticas correntes
      - quanto um novo projeto irá custar?
    - **Determinar os pontos fortes e fracos do processo corrente e produto**
      - certos tipos de erros sempre ocorrem?
    - **Desenvolver critérios (razões) para adoção/refinamento de técnicas**
      - que técnicas reduzirão os problemas, modificando as "baselines"?
    - **Identificar o impacto das técnicas**
      - aplicar testes funcionais reduz certas classes de erro?
    - **Avaliar a qualidade do processo/produto**
      - estamos aplicando inspeções adequadamente?
      - qual a confiabilidade do produto após a entrega?

Medição do Produto

- **Modelos de Software e Medidas**
- Contribuindo para:
  - Planejar o desenvolvimento do software e o processo de manutenção de forma que
    - Recursos adequados estejam disponíveis quando necessários
    - Análise de custo/benefício e identificação de riscos possam ser executadas
  - Monitorar o processo para prevenir ou reduzir dificuldades quando ainda possível
  - Controlar o processo através de ações corretivas ou preventivas baseadas na análise quantitativa
  - Avaliar a eficiência das fases e atividades de desenvolvimento ou processo de manutenção baseado em informação objetiva
  - Refinar os processos de desenvolvimento e manutenção

Medição do Produto

- **Modelos de Software e Medidas** [Basili99]
- **Visões de Métricas de Software**
  - Existem várias maneiras de se discutir métricas:
    - Nível de precisão, e.g., objetiva, subjetiva
    - Escalas de medição , e.g., nominal, ordinal, intervalo, razão
    - Objeto de medição, e.g., processo ou produto
  - **Métrica Objetiva**:
    - Uma medida absoluta extraída do produto ou do processo
    - Usualmente representada num intervalo ou escala razão
    - Exemplos: tempo de desenvolvimento, número de linhas de código, número de erros ou modificações
  - **Métrica Subjetiva**:
    - Uma estimativa de extensão ou grau de aplicação de alguma técnica
    - Uma classificação ou qualificação do problema ou experiência
    - Normalmente feita numa escala nominal ou ordinal
    - Exemplos: qualidade do uso de um método ou técnica, experiência dos programadores na aplicação ou processo

Medição do Produto

- Modelos de Software e Medidas
  - Visões de Métricas de Software
  - Escalas de Medição

Escala	**Operações Básicas**	**Exemplos Típicos**
Nominal	Determinação de igualdade	Áreas de aplicação Tipos de defeitos
Ordinal	Determinação de maior ou menor	Nível de treinamento ou compreensão
Intervalo	Determinação de igualdade de intervalos ou diferenças	Datas de calendário
Razão	Determinação de igualdade de razões	Linhas de código Número de defeitos Complexidade do código

Medição do Produto

- Métricas para qualquer produto “engenheirado” são governadas pelas características do produto
- Assim como para software convencional, métricas OO objetivam:
  - Melhor compreender a qualidade do produto
  - Identificar a eficiência do processo
  - Melhorar a qualidade do trabalho sendo realizado

O que faz um produto de software OO ser diferente de um software convencional?

Como medimos a qualidade de um sistema OO?

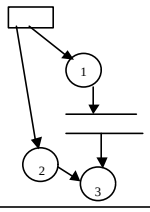
Que características do modelo de projeto podem ser identificadas para determinar se um sistema será fácil de implementar, ameno de ser testado, simples de modificar, e mais importante, será aceito pelos usuários?

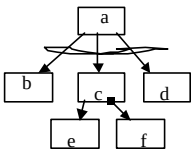
Medicção do Produto

- "Engenheirando" produtos de software convencionais
  - Solução está centrada na função ou nos dados
  - Utiliza algoritmos, procedimentos (com diferentes notações ao longo do ciclo de vida) e organização de dados para descrever o problema e organizar a informação
  - Basicamente, módulos organizam a arquitetura do software
  - Tecnologia de estruturação da informação (modelo relacional, por exemplo) determina como a informação será armazenada e avaliada.

Medição do Produto

- "Engenheirando" produtos de software convencionais
  - Decomposição funcional (desenvolvimento estruturado) e algumas métricas possíveis






```
Program a
do while X
call b;
call c;
call d
end while;
```

```
Subroutine c
call e;
if y then call f
end;
```

Especificação	Projeto	Implementação	Testes
Pontos por função Páginas de documentação Linhas de código estimadas Tempo (calendário) …	Coesão Acoplamento Complexidade Número de Módulos ...	Complexidade Ciclomática Teoria de Halstead Linhas de código Ligações de Dados ...	Cobertura Confiabilidade ...

Medição do Produto

- Paradigma Orientado a Objetos:
  - Significa organizar o software como uma coleção de objetos discretos que incorporam conjuntamente a estrutura dos dados e comportamento
  - características:

identidade	herança
abstração	polimorfismo
classificação	persistência
encapsulamento	

Medição de Software OO

- Modelos e Métricas: Algumas definições

Acoplamento

Intuitivamente, refere-se ao grau de interdependência entre diferentes pedaços de um projeto

Desde que objetos de uma mesma classe possuem as mesmas propriedades, duas classes estão acopladas quando métodos declarados numa classe utilizam métodos ou variáveis de instância de outra classe.

Coesão

Intuitivamente, refere-se à consistência interna dos pedaços de um projeto

O *grau de similaridade* de métodos pode ser visto como o maior aspecto de coesividade das classes/objetos. Se uma classe/objeto tem diferentes métodos executando diferentes operações para um mesmo conjunto de variáveis de instância, a classe é coesa.

Medição de Software OO

- Modelos e Métricas
- **Tamanho (descrição de requisitos, projeto de alto e baixo nível)**
  Lorenz e Kidd [Lorenz94]
  - Número de descrição de cenários (use-cases) (NSS)
    - Diretamente relacionada ao tamanho da aplicação e ao número de casos de teste
  - Número de classes chave (classes de domínio) (NKC)
    - Relacionada com o projeto de alto nível e o montante de esforço necessário para implementar o sistema e a reutilização necessária
  - Número de classes de suporte (NSC)
    - Referente ao projeto de baixo nível e o montante de esforço necessário para implementar o sistema e a reutilização necessária
  - Número médio de classes de suporte por classe chave (ANSC)
  - Número de subsistemas (NSUB)

Medição de Software OO

- Modelos e Métricas
- **Tamanho** (descrição de requisitos, projeto de alto e baixo nível)
  Lorenz e Kidd [Lorenz94]
  - Tamanho da Classe (CS)
    - Número total de operações + número de atributos (ambos incluindo características herdadas)
  - Número de operações especializadas por uma subclasse (NOO)
  - Número de operações acrescentadas por uma sub classe (NOA)
  - Indíce de Especialização (SI)
    - SI = [NOO x nível] / (Métodos Totais da Classe)

Medição de Software OO

- Modelos e Métricas
- Tamanho (descrição de requisitos, projeto de alto e baixo nível)
  Lorenz e Kidd [Lorenz94]

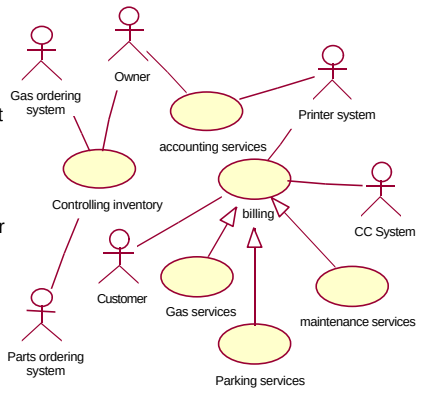
Fase **Métrica**	Descrição Requisitos	Projeto Alto Nível	Projeto Baixo Nível	Codificação	Testes
NSS					
NKC					
NSC					
ANSC					
NSUB					
CS					
NOO					
NOA					
SI					

Medição de Software OO

- Modelos e Métricas

The Gas Station problem

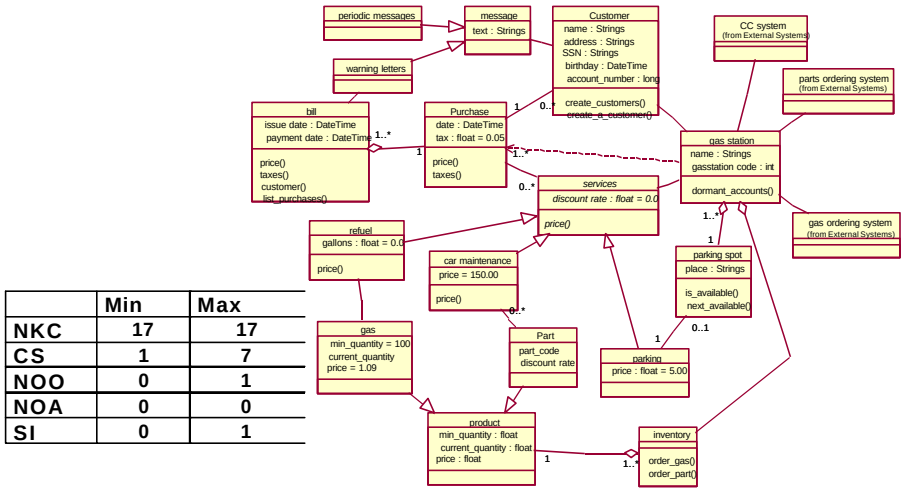
Requirements:
1 – A customer has the option to be billed automatically at the time of purchase (gas, car maintenance or parking spots) or to be sent a monthly paper bill. Customers can pay via cash, credit card or personal check. Gas Station services have a fixed price (gas: US$ 1.09 gallon, car maintenance: US$ 150.00 and parking spot: US$ 5.00 per day). The tax is 5% added to the final price of the purchase. Sometimes, the Gas Station owner can define discounts to those prices.
2 – The system has to track bills on a month-to-month basis and the services performed by the gas station on a day-to-day basis. The results of this tracking can be reported to the owner upon request.
3 – The gas station owner can use the system to control inventory. The system will either warn of low inventory or automatically order new parts and gas.
...



NSS = 6

Medição de Software OO

- Modelos e Métricas



	Min	Max
NKC	17	17
CS	1	7
NOO	0	1
NOA	0	0
SI	0	1

Medição de Software OO

- Modelos e Métricas
- Produto (projeto de alto e baixo nível)

  Chidamber e Kemerer Metrics Suite [Chidamber94]
  - Métodos Pesados por Classe (WMC)
  - Profundidade de Herança (DIT)
  - Número de Filhos (NOC)
  - Acoplamento entre Objetos (CBO)
  - Resposta de uma classe (RFC)
  - Perda de Coesão em Métodos (LCOM)

Fase **Métrica**	Projeto Alto Nível	Projeto Baixo Nível	Codificação	Testes
WMC				
DIT				
NOC				
CBO				
RFC				
LCOM				

Medição de Software OO

- Modelos e Métricas
- Produto
  - **Métodos Pesados por Classe (WMC)**

Considere uma classe C_1, com métodos $M_1,\ldots, M_n$ que são definidos pela classe. Faça $c_1,\ldots,c_n$ ser a complexidade dos métodos. Então:

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

Se todas as complexidades dos métodos são consideradas ser 1, então WMC = n, o número de métodos.

Pontos de Vista:

1) Número de métodos e a complexidade de métodos relacionam-se com quanto tempo e esforço são necessários para desenvolver e manter a classe
2) Quanto maior o número de métodos numa classe, maior o potencial de impacto nos filhos, desde que os filhos herdam os métodos da classe
3) Classes com grande número de métodos tendem a ser mais específicas da aplicação, limitando sua possibilidade de reutilização

Medição de Software OO

- Modelos e Métricas
- Produto
  - **Profundidade de Herança (DIT)**

Profundidade de herança da classe é a métrica DIT para a classe. Em casos envolvendo múltipla herança, a DIT será a extensão máxima do nó até a raiz da árvore.

Pontos de vista:

1) Quanto mais profunda uma classe está na hierarquia, maior o número de métodos que provavelmente herdará, fazendo ser mais difícil predizer seu comportamento.
2) Árvores profundas constituem maior complexidade de projeto, desde que mais métodos e classes estão envolvidos.
3) Quanto mais profunda está uma classe particular na hierarquia, maior o potencial de reutilização dos métodos herdados.

Medição de Software OO

- Modelos e Métricas
- Produto
  - **Número de filhos (NOC)**

  NOC = número de sub classes imediatas e subordinadas a uma classe na hierarquia

Pontos de Vista:

1) Maior o número de filhos, maior a reutilização, desde que herança é uma forma de reutilização
2) Maior o número de filhos, maior a probabilidade de abstração imprópria para a classe pai. Se uma classe tem um grande número de filhos, isto pode provocar uso inadequado da especialização.
3) O número de filhos dá uma idéia da potencial influência que uma classe tem no projeto. Se uma classe tem um grande número de filhos, isto pode requerer mais teste dos métodos desta classe.

Medição de Software OO

- Modelos e Métricas
- Produto
  - **Acoplamento entre Objetos (CBO)**

  CBO para uma classe é a contagem do número de outras classes para as quais ela está acoplada (associada)

Pontos de vista:

1) Acoplamento excessivo entre classes/objetos é prejudicial ao projeto modular e previne a reutilização. Quanto mais independente uma classe, mais fácil será sua reutilização em outra aplicação.
2) Em ordem de aumentar a modularidade e promover o encapsulamento,o acoplamento entre objetos deve ser mantido no mínimo possível. Quanto maior o número de acoplamenos, maior sensibilidade a trocas em outras partes do projeto, e por isto a manutenção se torna mais difícil.
3) Uma medida de acoplamento é útil para determinar quão provavelmente complexo será o teste de várias partes do projeto. Quanto maior o acoplamento entre objetos, mais rigoroso o teste precisa ser.

Medição de Software OO

- Modelos e Métricas
- Produto
  - **Resposta para uma classe (RFC)**

  O conjunto de respostas para uma classe é o conjunto de métodos que podem potencialmente ser executados em resposta a uma mensagem recebida por um objeto da classe.

  RFC = |RS| onde RS é a reposta para a classe.

  RS = {M} Uall I {Ri}

  onde {Ri} = conjunto de métodos chamados pelo método I e

  {M} = conjunto de todos os métodos da classe

Pontos de Vista:

1) Se um grande número de métodos podem ser chamados em resposta a uma mensagem , o teste e depuração da classe se torna mais complicado desde que requer um grande nível de compreensão por parte do testador.
2) Quanto maior o número de métodos que podem ser chamados para uma classe, maior a complexidade da classe.
3) O valor de pior caso para possíveis respostas apoiará na alocação apropriada do tempo/esforço para testes

Medição de Software OO

- Modelos e Métricas
- Produto
  - **Perda de Coesão em Métodos (LCOM)**

  Considere uma classe C1 com n métodos $M_1, M_2, \ldots, M_n$. Faça $\{I_j\}$ = conjunto de variáveis de instância usadas pelo método M_i.

  There are n such sets $\{I_1\}, \ldots, \{I_n\}$. Let $P = \{(I_i, I_j) | I_i \cap I_j = 0\}$
  and $Q = \{(I_i, I_j) | I_i \cap I_j \neq 0\}$. If all n sets $\{I_1\}, \ldots, \{I_n\}$ are 0 then let $P = 0$.
  $LCOM = |P| - |Q|$, if $|P| > |Q|$
  $LCOM = 0$ otherwise

Pontos de Vista:

1) Coesão de métodos dentro de uma classe é desejável, desde que promove o encapsulamento
2) Perda de coesão implica que classes provalmente deveriam ser divididas em duas ou mais classes ou subclasses
3) Qualquer medida de inadequação de métodos contribui para identificar problemas no projeto das classes
4) Baixa coesão aumenta a complexidade, com isto aumentando a probabilidade de erros durante o processo de desenvolvimento.

Medição de Software OO

- Modelos e Métricas
- **Produto**

  Lie e Henry [Lie93]
  - Métricas validadas através do estudo do número de modificações executadas em dois sistemas comerciais implementaddos com um dialeto Ada OO
  - Adiciona duas métricas ao conjunto proposto por Chidamber e Kemerer:
    - Acoplamento de Passagem de Mensagem (MPC): é calculado como o número de declarações de envio definido na classe
    - Acoplamento de Abstração de Dados (DAC): calculado como o número de tipos abstratos de dados usados na classe medida e definidos em outra classe do sistema
  - Modelo foi adequado para predizer o tamanho das modificações nas classes durante a fase de manutenção

Medição de Software OO

- Modelos e Métricas

  Basili, Briand e Melo [Basili95]
  - Baseado em Chidamber e Kemerer, empiricamente validaram a adequabilidade destas métricas para predição da probabilidade de identificar classes sujeitas a falha em softwares baseados em C++
  - Ajustaram levemente algumas das métricas para capturar algumas das características de C++ (as métricas não são independentes da linguagem)

WMC: todos os métodos tem complexidade 1 e operadores "friend" não são contados

DIT: mede o número de ancestrais de uma classe

NOC: número de descendentes diretos para cada classe

CBO: uma classe está acoplada a outra se utiliza suas funções membro e/ou variáveis de instância

RFC: número de funções diretamente chamadas por funções membro ou operadores da classe

LCOM: número de pares de funções membro sem variáveis de instância compartilhadas, menos o número de pares de funções membro com variáveis de instância compartilhadas. Assume valor 0 se a subtração é negativa.

Medição de Software OO

- Modelos e Métricas
  Pontos de vista Basili, Briand e Melo :

WMC: uma classe com significativamente mais funções membro que seus pares é mais complexa, em consequência sendo mais propensa a falhas

DIT: sistemas OO bem projetados são aqueles estruturados como florestas de classes, ao invés daqueles estruturados como uma grande estrutura de herança

NOC: classes com grande número de filhos são difíceis de modificar e usualmente requerem mais teste porque a classe potencialmente afeta todos os seus filhos

CBO: classes fortemente acopladas são mais propensas a falha que classes fracamente acopladas

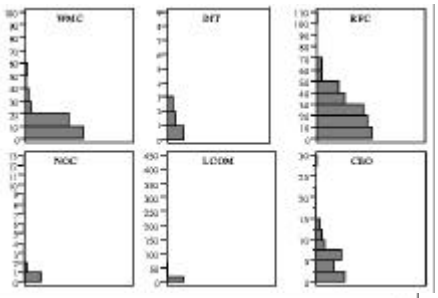
RFC: quanto maior a resposta de uma classe, maior a complexidade da classe, e mais propensa a falha e dificil de modificar

LCOM: uma classe com baixa coesão entre seus métodos sugere um projeto não apropriado (i.e., o encapsulamento de objetos de programa e funções membro não relacionadas que não deveriam estar juntos), provável de ser propenso a falha.

Medição de Software OO

- Modelos e Métricas
  Análise das Distribuições:

Métricas OO baseadas em 180 classes:
DIT indica que hierarquias de herança são de alguma forma planas e **NOC** que classes tem, em geral, poucos filhos. Classes parecem ter alta coesão (LCOM perto de 0)



	WMC	DIT	RFC	NOC	LCOM	CBO
Maximum	99.00	9.00	105.00	13.00	426.00	30.00
Minimum	1.00	0.00	0.00	0.00	0.00	0.00
Median	9.50	0.00	19.50	0.00	0.00	5.00
Mean	13.40	1.32	33.91	0.23	9.70	6.80
Std Dev	14.90	1.99	33.37	1.54	63.77	7.56

Medição de Software OO

- Modelos e Métricas

  resultados relativos a probabilidade de identificação de falhas numa classe durante a fase de teste:

WMC: quanto **maior** WMC, **maior** a probabilidade de identificação de falha

NOC: quanto **maior** o NOC, **menor** a probabilidade de identificação de falhas

LCOM: não significante

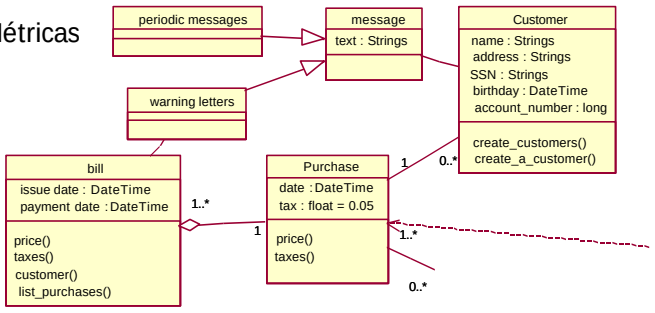
DIT: quanto **maior** DIT, **maior** a probabilidade de identificação de falhas

RFC: quanto **maior** RFC, **maior** a probabilidade de identificação de falhas

CBO: mais significante para classes de IU que para classes de BD (não foi encontrada explicação satisfatória para as diferenças nos padrões entre estes tipos de classe)

Medição de Software OO

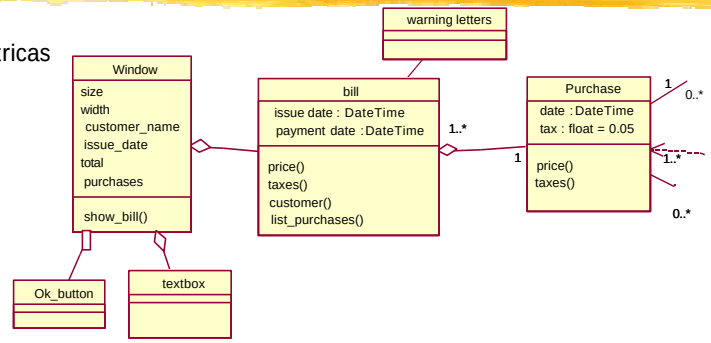
- Modelos e Métricas (alto nível)



	Bill	Purchase	Warning Letters	Periodic Messages	Message	Customer
WMC	4	2	0	0	0	2
NOC	0	0	0	0	2	0
DIT	0	0	1	1	0	0
RFC	--	--	--	--	--	--
CBO	2	4	2	1	1	2
LCOM	--	--	--	--	--	--

Medição de Software OO

- Modelos e Métricas (baixo nível)



	Bill	Purchase	Warning Letters	Window	Ok button	Textbox
WMC	4	2	0	1	0	0
NOC	0	0	0	0	0	0
DIT	0	0	1	1	0	0
RFC	--	--	--	--	--	--
CBO	3	4	2	3	1	1
LCOM	--	--	--	--	--	--

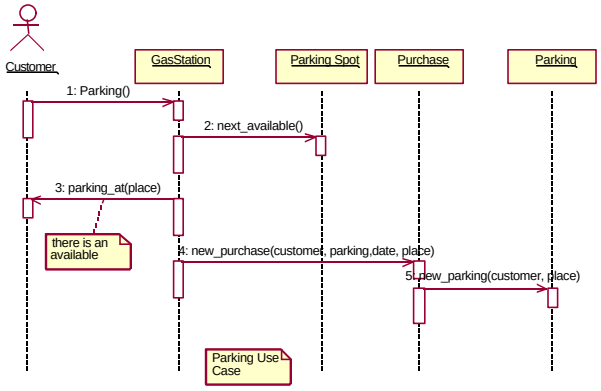
Medição de Software OO

- Modelos e Métricas
  Outras Métricas possíveis
  - Tamanho médio do Serviço (OS_{avg})
    - O número de mensagens envolvidas num serviço
  - Número médio de parâmetros por operação (NP_{avg})
  - Complexidade da Operação (OC)
    - Pode ser calculado usando qualquer das métricas de complexidade propostas para software convencional
  - Percentagem pública e protegida (PAP)
  - Acesso público aos membros de dados (PAD)
  - Número de Classes Raiz (NOR)
    - Mostra o número de hierarquias de classe que existem no sistema. Valores grandes devem ser evitados.
  - Fan in (FIN)
    - FIN > 1 indica múltipla herança

Medição de Software OO

- Modelos e Métricas

	Min	Max
OS	5	5
NP	0	4



Medição de Software OO

Class name: refuel
Category: Service
External Documents:
Export Control: Public
Cardinality: n
Hierarchy:
Superclasses: services
Associations:
<no rolename> : gas in association <unnamed>

Public Interface:
Operations:
price
Private Interface:
Attributes:
gallons
price = 1.09
Implementation:
Attributes:
gallons
price = 1.09
State machine: No
Concurrency: Sequential
Persistence: Transient

	Value
CS	4
PAD	0
PAP	0

Operation name: price
Public member of:refuel
Documentation:
// Calculates the final price of the fuel
Preconditions:
gallons > 0
Object diagram: (Unspecified)
Semantics:
final_price = gallons * price
Object diagram: (Unspecified)
Concurrency: Sequential

Medição de Software OO

• Modelos x Métricas:
Onde medir/visualizar métricas OO

Model Metric	Use Cases	Class Diagrams	Interaction Diagrams	Class Descriptions	State Diagrams	Package Diagrams	Deployment Diagrams
NSS							
NKC							
NSC							
ANSC							
NSUB							
CS							
NOO							
NOA							
SI							

Medição de Software OO

Modelos x Métricas:
Onde medir/visualizar métricas OO

Model Metric	Use Cases	Class Diagrams	Interaction Diagrams	Class Descriptions	State Diagrams	Package Diagrams	Deployment Diagrams
WMC							
DIT							
NOC							
CBO							
RFC							
LCOM							

Model Metric	Use Cases	Class Diagrams	Interaction Diagrams	Class Descriptions	State Diagrams	Package Diagrams	Deployment Diagrams
OS_{avg}							
NP_{avg}							
OC							
PAP							
PAD							
NOR							
FIN							

Medição de Software OO

- **Modelos de Software e Medidas: Automação**
  - Que aspectos da medição podem ser automatizados?
    - Existe uma variedade de ferramentas disponíveis comercialmente que geram várias métricas de código para uma variedade de linguagems
      - Exemplos: Code Metric Tools such as Analysis of Complexity Tool (ACT) and BattleMap (McCabe&Associates) and Logiscope (Verilog)
    - Existem vários ambientes de medição que vão além das métricas de código e suportam várias funções de gerenciamento. Estes utilizam dados históricos do ambiente ou de outros ambientes.
      - Exemplos: SPQR AND Checkpoint (Software Productivity Research, Inc.) and PADS (Quantitative Software Management, Inc.)

Medição de Software OO

- **Modelos de Software e Medidas: Aplicação**
  - Qual o estado da prática?
    - Muitas companhias possuem programas de medição em escala completa, não apenas no nível de projeto mas a nível de divisão e corporação.
    - Tipicamente, as mais avançadas utilizam algum tipo de infraestrutura ou modelo.
    - Exemplos:
      - HP utilizando uma versão preliminar do GQM
      - Motorola empregando GQM e Quality Improvement Paradigm
      - NEC empregando SQMAT, uma melhoria sobre SQM, e Plan-Do-Check-Act
      - AT&T utilizando QFD, adaptado para software

Medição de Software OO

- Como desenvolvedores utilizam métricas OO:
  - Avaliando qualidade do software
  - Compreendendo o processo de projeto
  - Identificando problemas no produto
  - Melhorando soluções
  - Adquirindo conhecimento de projeto
- Por exemplo
  - Utilizando métricas para avaliar complexidade estrutural OO
    - Abordagem aplicada a dois domínios de aplicação diferentes :
      - Software Telecomunicações (C++)
      - Ambientes de Desenvolvimento de Software (Eiffel)

Medição de Software OO

- Utilizando métricas para avaliar a complexidade estrutural OO [Travassos99]
- Desenvolvedores podem combinar:
  - Heurísticas aplicadas para reduzir a complexidade do software durante atividades de projeto (Diretrizes para reduzir a complexidade de projeto OO)
  - Conhecimento sobre experiências de desenvolvimento (Princípios de Projeto OO)
  - Métricas para caracterizar como as solucões foram projetadas.

G1	Classes Restructuring Guideline (set of 10 procedures)
G2	Adapter Class Guideline
G3	Bridge Class Guideline
G4	Facade Class Guideline
G5	Multiple Inheritance Elimination Guideline
G6	Large Hierarchies Redefinition Guideline

Heurísticas

M1	**DIT**
M2	**NOC**
M4	**Coupling**
M5	**Class Size**
M6	**Number of Multiple Inheritance**
M11	**Number of Abstract Classes**

Métricas

Medição de Software OO

- Utilizando métricas para avaliar a complexidade estrutural OO [Travassos99]

Princípios

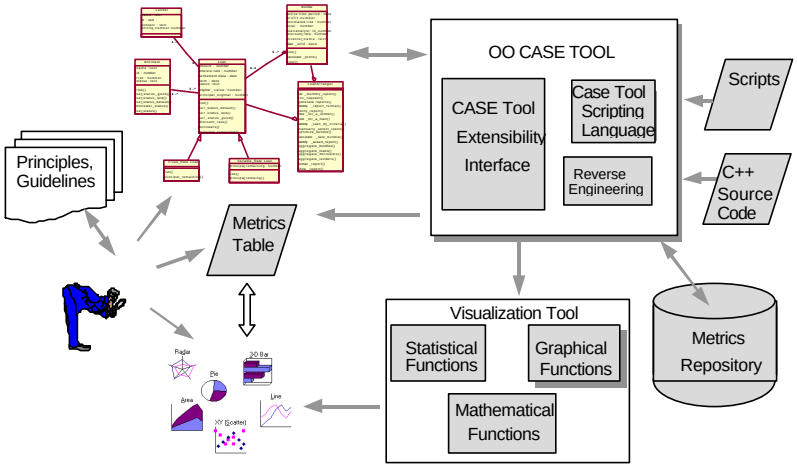
P1	Few Interfaces Principle : “All module (class) should communicate with as few others as possible”.
P2	Small Interfaces Principle: “If any two modules (classes) communicate at all, they should exchange as little information as possible”.
P3	Explicit Interfaces Principle: “Whenever two modules (classes) *A* and *B* communicate, this must be obvious from the text of *A* or *B* or both”.
P4	Information Hiding Principle: “All information about a module (class) should be private to the module (class) unless it is specifically declared public”.
P5	The Open-Closed Principle: “Software entities (classes, modules, etc.) should be open for extension but closed for modification”.
P6	The Liskov Substitution Principle: “It is only when derived types are completely substitutable for their base types that functions with use those base types can be reused with impunity, and the derived types can be changed with impunity”.
P7	The Dependency Inversion Principle: “Details should depend on abstractions. Abstractions should not depend on details. High level policies should not depend on low-level implementations; rather both should depend on abstractions”.
P8	The Interface Segregation Principle: “Clients should not be forced to depend on interfaces that they do not use”.

Medição de Software OO

	P1	P2	P3	P4	P5	P6	P7	P8
G1	M3, M7, M9, M10, M12	M8	M14	M14	M1, M2, M3, M6, M11	M1, M2, M6	M1, M2, M11	M1, M2, M4, M5
G2		M13						M5
G3					M11			
G4	M3				M3			
G5	M3				M3, M6			
G6	M3				M1, M2, M3	M1, M2	M1, M2	M1, M2, M4

Relação entre Diretrizes, Princípios e Métricas

Medição de Software OO



Automatizando esta abordagem

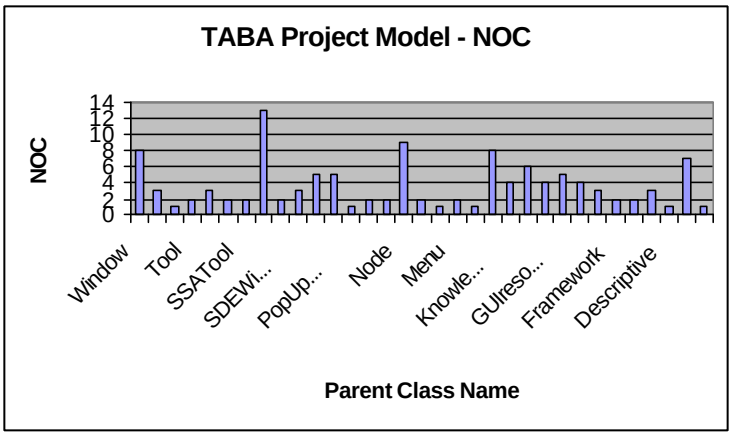
Medição de Software OO

Uma tabela de métricas produzida com ferramentas CASE

```
**
** Legend: A: Attribute, B: Behavior,
**         IA: Inherited Attribute, IB: Inherited Behavior,
**         TA: Total Attributes, TB: Total Behavior,
**         DIT: Hierarchical Level, NOC: Number of Children
**
**   Normal NOC for this model.....: 3
**   Number of leaf classes.....: 130
**   Number of Non-Hierarquical Classes..: 11
**   Number of Abstract Classes.....: 33
**   Number of Classes.....: 163
**
```

Parent Class Name	A	B	IA	IB	AT	BT	DIT	NOC
Shapes	1	2	0	0	1	2	0	13
Node	2	0	3	4	5	4	1	9
Window	7	0	1	0	8	0	1	8
KnowledgeWindow	0	2	8	0	8	2	2	8
Arcs	2	1	3	4	5	5	1	7
Knowledge	6	4	0	0	6	4	0	6
PullDownMenu	8	9	4	0	12	9	1	5
GUIresoruces	1	0	0	0	1	0	0	5
SDEFramework	0	1	1	0	1	1	2	5
InternalTool	1	1	0	0	1	1	0	4
KnowledgeREpresentation	0	0	8	0	8	0	2	4
GeneralWindows	6	2	15	10	21	12	1	4
Editor	1	0	0	0	1	0	0	3
ToolWindow	0	0	1	1	1	1	3	3
GeneralMenus	0	0	8	0	8	0	2	3
SDEWindow	0	0	21	12	21	12	2	3

Medição de Software OO



Visualização da Métrica