Diferença estrutural entre versões de um programa

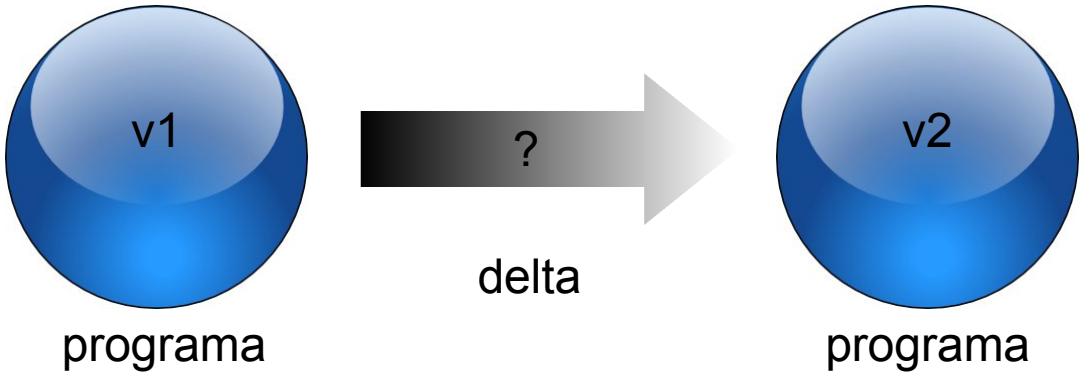
Evolução de Software @ UFCG

6 de agosto de 2008

Rodrigo Rocha <rodrigorgs@gmail.com>

Questão de Pesquisa

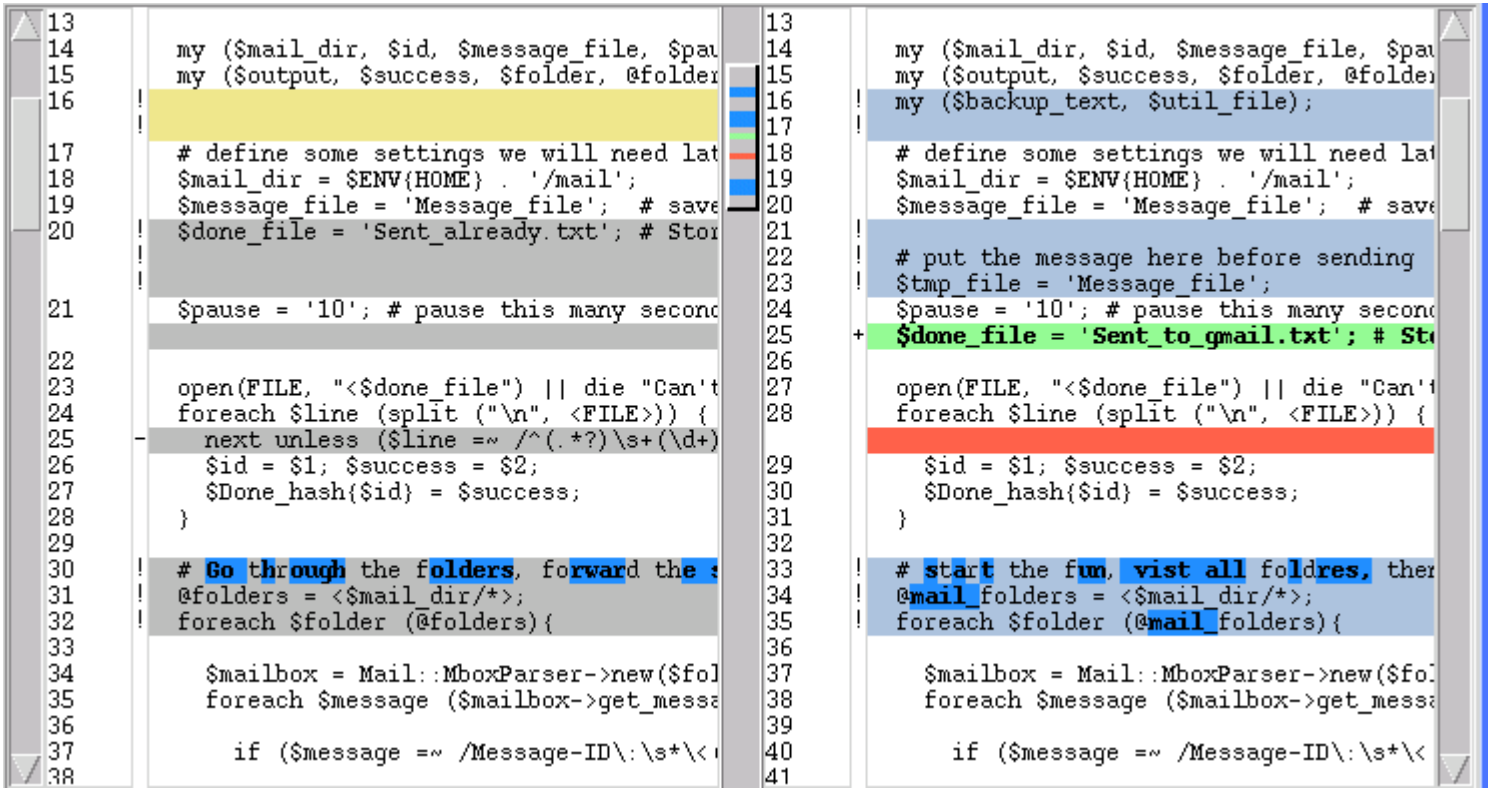
- A partir duas versões de um programa, como descobrir quais mudanças foram feitas?



- Importante ferramenta para entender a evolução de programas

Questão de Pesquisa

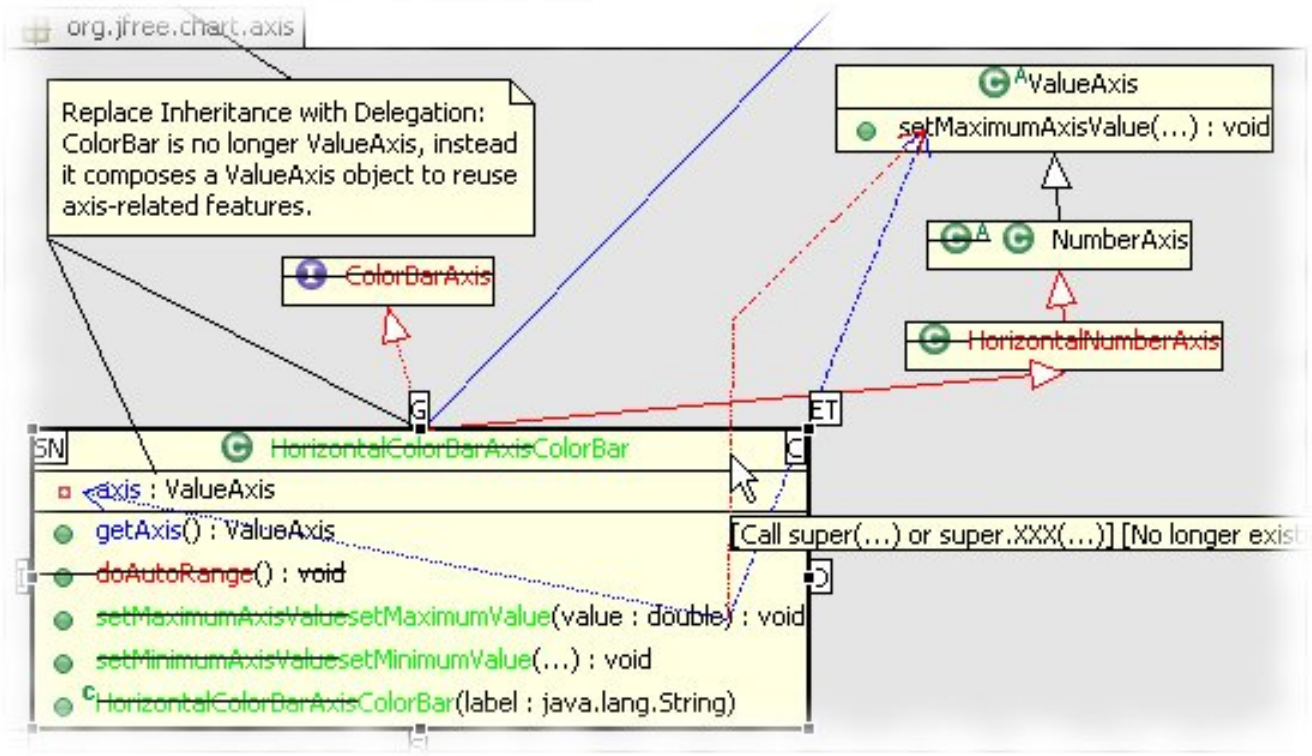
- GNU diff: arquivo, linha, caractere



http://en.wikipedia.org/wiki/Image:Tkdiff_screenshot.png

Questão de Pesquisa

- Estrutural: pacote, classe, método, atributo...



http://www.cs.ualberta.ca/~xing/jdevanviewer.html

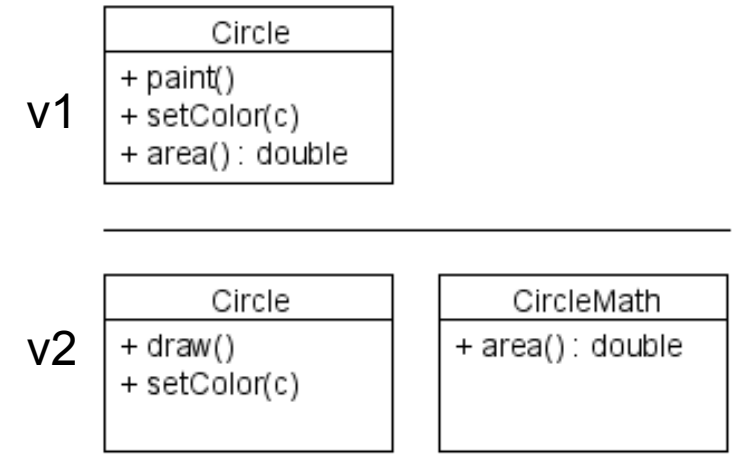
Sumário

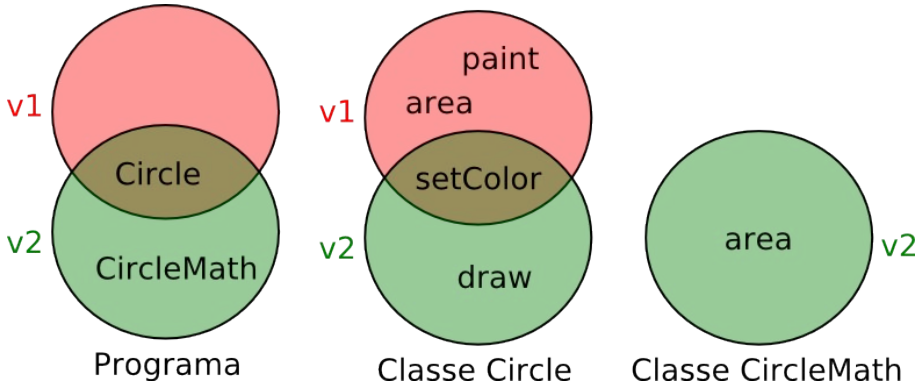
- Definições
- Exemplo
- Programação estruturada: BEAGLE
  - Análise de Bertillonage
  - Análise de dependência
- Programação OO: UMLDiff
  - Similaridade de nome
  - Similaridade estrutural
- Considerações finais

Definições

- Entidade (de um programa):
  - (normalmente possui um identificador)
  - classe, método, atributo... (OO)
  - função, variável, arquivo... (prog. estruturada)
- Mudança (de uma entidade):
  - adicionar, remover, manter
  - renomear, mover
- Identidade
  - O que torna uma entidade reconhecível e diferenciável das outras

Exemplo





- Ações:
  - Adicionou **CircleMath**
  - Manteve **Circle**
    - renomeou **paint** para **draw**
    - manteve **setColor**
    - moveu **area** para **CircleMath**

Problema: identificar a origem de entidades aparentemente novas

BEAGLE

[Tu e Godfrey, 2002]

BEAGLE

- Ferramenta para estudo de evolução
- Programação estruturada
- Entidades: funções
- Métricas de similaridade entre funções
  - Análise de Bertillonage
  - Análise de dependência

Análise de Bertillonage

- Inicialmente usada para detectar clones
- Seleciona rapidamente prováveis entidades originais (triagem inicial)
- Caracteriza um trecho de código através de 5 métricas (dimensões):
  - {S,D,Ciclomatic}-Complexity
  - Albrecht
  - Kafura
- Similaridade = distância euclidiana

Análise de Bertillonage

v1

```
build_binary_op_nodefault in cp-typeck.c
```

v2

```
1. combine in fold-const.c:
d=1005745.47

2. recog_4 in insn-recog.c:
d=2496769.23

3. insn-recog.c in recog_5.c:
d=7294066.05

4. fprop in hard-params.c:
d=8444858.78

5. build_binary_op_nodefault in c-typeck.c:
d=8928753.44
```

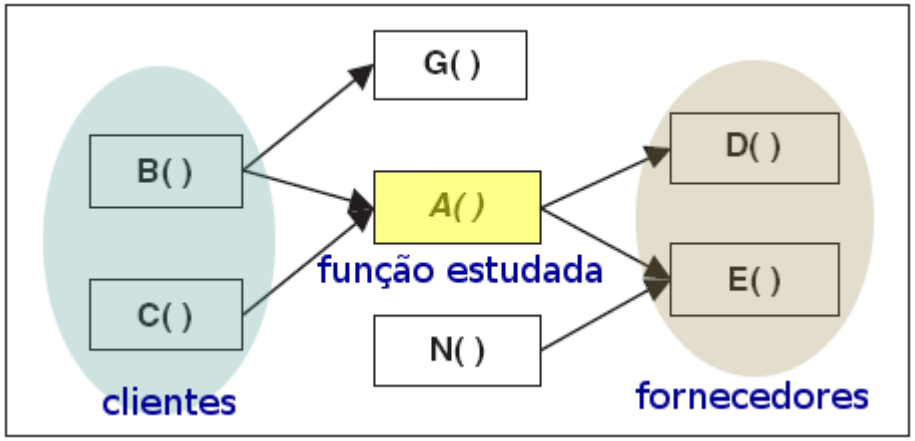
- Seleciona as 5 entidades mais prováveis
- Escolhe a que tiver o mesmo nome, se existir

Análise de Bertillonage

- Ponto forte
  - Bom tratamento de renomear e mover
- Pontos fracos
  - Se confunde quando há funções parecidas
  - Mudanças na implementação das funções prejudicam a análise

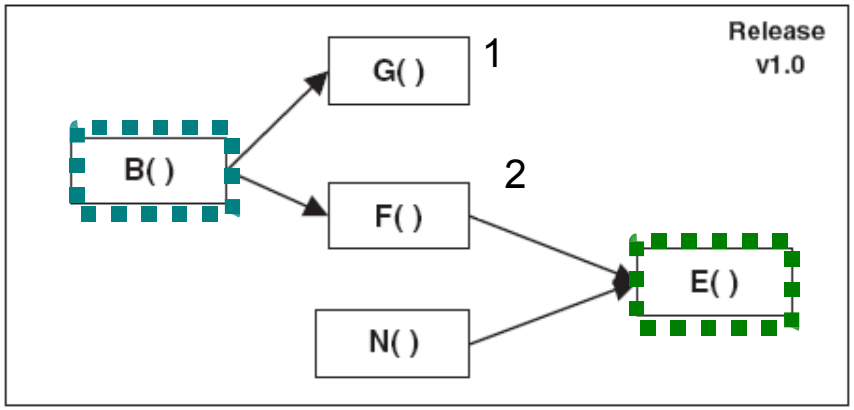
Análise de dependência

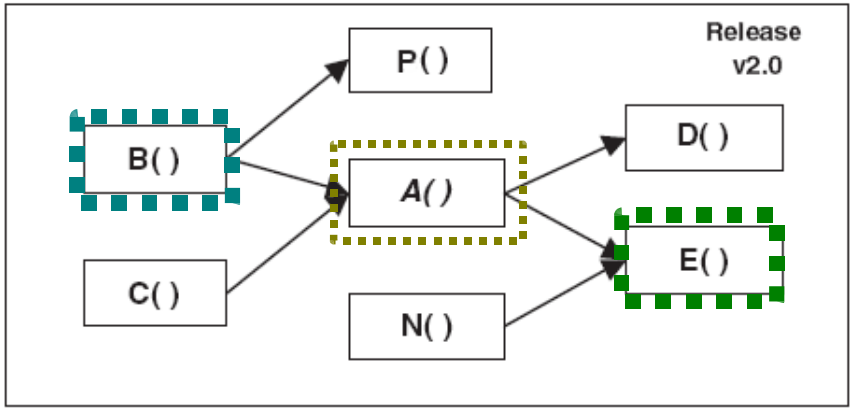
- Uma função...
  - ... chama outras funções (**fornecedores**) e
  - ... é chamada por outras funções (**clientes**)
- Uma função pode ser caracterizada pelo seu conjunto de fornecedores e clientes



Análise de dependência

- Funções "adicionadas" (v2): {**A**, C, D, P}
- Funções "removidas" (v1): {G, F}
- Clientes de A: {**B**, C} (v2)
  - Chama {**G**, **F**} (v1)
- Fornecedores de A: (v2) {D, **E**}
  - Chamada por {**F**, N} (v1)
- Qual a origem de A?
  - **F** (2 pontos) ou G (1)





Análise de dependência

- Ponto forte
  - Pouco afetada por mudanças na implementação das funções
- Ponto fraco
  - Assume que clientes e fornecedores são estáveis
    - Pior caso: todas as funções são renomeadas ou movidas

Algoritmo

- Como combinar Bertillonage e dependência?
- (O artigo não explica o algoritmo)

Avaliação

- Estudo de caso: GCC 2.7.2.3 => EGCS 1.0
- Resultados
  - Não avalia o algoritmo em termos de precisão e revocação!
  - Relatos específicos de sucesso
  - Avalia quanto do código do EGCS 1.0 é novo

File	Func	New	Old	Type
gcc/cplus-dem.c	36	36	0	Mostly New
gcc/crtstuff.c	5	5	0	Mostly New
gcc/insn-output.c	107	95	12	Mostly New
gcc/final.c	33	20	13	Half-Half
gcc/regclass.c	20	12	8	Half-Half

UMLDiff

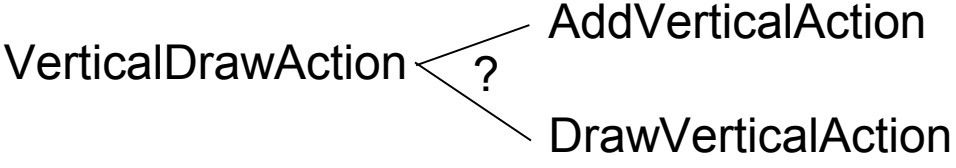
[Xing e Stroulia, 2005]

UMLDiff

- Parte da ferramenta JDEvAn (Java Design Evolution and Analysis)
- Programação OO
- Entidades: classes, métodos, atributos...
- Métricas de similaridade
  - Similaridade de nome
  - Similaridade estrutural

Similaridade de nome

- Aplicada a identificadores
- Métrica baseada em LCS (maior subseqüência comum)



$$\frac{2\left|LCS(s_1, s_2)\right|}{\left|s_1\right| + \left|s_2\right|}$$

VerticalDraw**Action** vs Add**Vertical** **Action**: $\frac{2 \times 14}{35} = 0.8$

Sensível a permutações de nomes!

VerticalDraw**Action** vs Draw**Vertical** **Action**: $\frac{2 \times 14}{36} = 0.77$

Similaridade de nome

- Métrica baseada em pares de caracteres adjacentes (*case-insensitive*)
  - P("VerticalDrawAction") = {(v,e), (e,r), (r,t), (t,i), (i,c), ..., (o,n)}

$$\frac{2\left|P(s_1) \cap P(s_2)\right|}{\left|P(s_1)\right| + \left|P(s_2)\right|}$$

  - DrawVerticalAction => 0.88
  - AddVerticalAction => 0.73

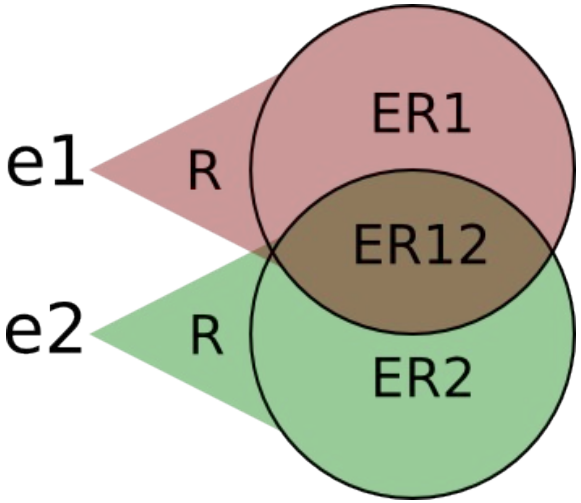
Similaridade estrutural

- relativa a um tipo de relacionamento R)

Table 2. Facts for computing structure similarity

Entity Type	Entity and relationship facts
Package	The top-level classes and interfaces it *contains*
Named class and interface	The fields, methods, constructors, inner classes and interfaces it *contains*; classes/interfaces it *uses* and is *used by*
Field	The methods/constructors that *read* and *write* it
Method and constructor	Type of parameters it *declares*; fields it *reads* and *writes*; methods/constructors it *calls* and is *called by*

Similaridade estrutural



$$\approx \frac{|ER_{12}|}{|ER_1 \cup ER_2|}$$

- ERn são multi-conjuntos
- Se ER12 é vazio, usa (similaridade de nome)^k
- Para calcular ER12, usa um critério arbitrário de comparação de entidades

Algoritmo UMLDiff

- Similaridade geral: média de todas as métricas calculadas (mais ou menos)
- Identifica entidades
  mantidas => renomeadas => movidas
- Parâmetros: limiar de *renomear*, limiar de *mover*
- Limitação: não identifica *renomear* + *mover*

Avaliação

- Estudo de caso: 31 releases do JFreeChart em 4 anos, analisadas par-a-par (até 800 classes)
- Performance
  - Intel Centrino 1.6GHz, 768M RAM, Windows XP, PostgreSQL 7.4.5
  - Tempo de um diff: entre 1 e 60 min (média: 10 a 12 min)

Avaliação

- Metodologia
  - Inspeção manual das correspondências encontradas para calcular a precisão
  - Para obter uma idéia da revocação, inspeção manual de entidades renomeadas e movidas encontrados pelo UMLDiff com limiar 1%
- Resultados (limiar de 30%)
  - Revocação de renomear: 96,4%
  - Revocação de mover: 97,1%

Avaliação

- Resultados (precisão, limiar de 30%)

Table 5. *UMLDiff* results at threshold 30%

Type of change	#Correct	#Reported	Precision
Renamed package	29	29	100%
Renamed class/interface	121	128	94.5%
Moved class/interface	306	306	100%
Renamed field/method	1927	2024	95.2%
Moved field/method	630	721	87.3%
Data type and return type	677	710	95.4%
Visibility modifier	845	855	98.8%
Non-visibility modifier	299	303	98.7%
Class inheritance	180	185	97.3%
Interface inheritance	970	1025	94.6%
Total	**5894**	**6286**	**95.2%**

Considerações finais

- Resultados melhores
  - Incrementos pequenos
  - Uso disciplinado do SCM: separar reengenharia de mudanças funcionais
- Entidades com poucos relacionamentos são de difícil análise (ex.: getters, métodos abstratos)
- As técnicas não consideram mudanças do tipo união ou separação

Referências

- Qiang Tu, Michael W. Godfrey, "**An Integrated Approach for Studying Architectural Evolution**," in Proc. 2002 Intl. Workshop on Program Comprehension (IWPC 2002), Paris, June 2002.
- Zhenchang Xing, Eleni Stroulia. **UMLDiff: An Algorithm for Object-Oriented Design Differencing**. Proceedings of the 20th IEEE/ACM international Conference on Automated software engineering, 2005.