Benchmarks para experimentos em localização de defeitos

Higor Amario de Souza
IME-USP
hamario@ime.usp.br

Depuração de programas

- Depuração: localização + correção
- Tarefa custosa
- Inicia-se a partir da identificação de falhas
- Processo manual e *ad-hoc*

Localização de defeitos

- Automatização
- Testes automatizados
- Cobertura de código
- Heurísticas
- Indicar trechos de código mais suspeitos
  - Blocos, comandos, predicados

Exemplo: max [1]

```
1   int max(int array[], int length)
2   {
3      int i = 0;
4      int max = array[++i]; //array[i++];
5      while(i < length){
6         if(array[i] > max)
7            max = array[i];
8         i++;
9      }
10     return max;
11  }
```

Conjunto de teste

Casos de teste	
t_1	$\langle([1,2,3],3),3\rangle$
t_2	$\langle([5,5,5],3),5\rangle$
t_3	$\langle([2,10,1],3),10\rangle$
t_4	$\langle([4,2,3],3),4\rangle$
t_5	$\langle([4],1),4\rangle$

Matriz de cobertura

L	t_1	t_2	t_3	t_4	t_5
3	●	●	●	●	●
4	●	●	●	●	●
5	●	●	●	●	
6	●	●	●	●	
7	●	●	●	●	
8	●	●	●	●	
10	●	●	●	●	
-	P	P	P	F	F

Matriz de cobertura

L	t_1	t_2	t_3	t_4	t_5
3	●	●	●	●	●
4	●	●	●	●	●
5	●	●	●	●	
6	●	●	●	●	
7	●	●	●	●	
8	●	●	●	●	
10	●	●	●	●	
-	P	P	P	F	F

Matriz de cobertura

L	t_1	t_2	t_3	t_4	t_5
3	●	●	●	●	●
4	●	●	●	●	●
5	●	●	●	●	
6	●	●	●	●	
7	●	●	●	●	
8	●	●	●	●	
10	●	●	●	●	
-	P	P	P	F	F

Matriz de cobertura

L	t_1	t_2	t_3	t_4	t_5
3	●	●	●	●	●
4	●	●	●	●	●
5	●	●	●	●	
6	●	●	●	●	
7	●	●	●	●	
8	●	●	●	●	
10	●	●	●	●	
-	P	P	P	F	F

Coeficientes de cobertura

$$c_{ef}, c_{nf}, c_{ep}, c_{np}$$

Caso de teste	Executado (e)	Não executado (n)
Falhou (f)	c_{ef}	c_{nf}
Passou (p)	c_{ep}	c_{np}

Matriz de cobertura

L	t_1	t_2	t_3	t_4	t_5	c_{ef}	c_{nf}	c_{ep}	c_{np}
3	●	●	●	●	●	2	0	3	0
4	●	●	●	●	●	2	0	3	0
5	●	●	●	●		1	1	3	0
6	●	●	●	●		1	1	3	0
7	●	●	●	●		1	1	3	0
8	●	●	●	●		1	1	3	0
10	●	●	●	●		1	1	3	0
-	P	P	P	F	F	-	-	-	-

NAPSOL
NÚCLEO DE APOIO À PESQUISA EM SOFTWARE LIVRE

Tarantula [2]

$$\frac{\frac{c_{ef}}{c_{ef}+c_{nf}}}{\frac{c_{ef}}{c_{ef}+c_{nf}}+\frac{c_{ep}}{c_{ep}+c_{np}}}$$

Matriz de cobertura

L	t_1	t_2	t_3	t_4	t_5	c_{ef}	c_{nf}	c_{ep}	c_{np}	$H_{(T)}$
3	•	•	•	•	•	2	0	3	0	0.5
4	•	•	•	•	•	2	0	3	0	0.5
5	•	•	•	•		1	1	3	0	0.33
6	•	•	•	•		1	1	3	0	0.33
7	•	•	•	•		1	1	3	0	0.33
8	•	•	•	•		1	1	3	0	0.33
10	•	•	•	•		1	1	3	0	0.33
-	P	P	P	F	F	-	-	-	-	-

NAPSOL
NÚCLEO DE APOIO À PESQUISA EM SOFTWARE LIVRE

Matriz de cobertura

L	t_1	t_2	t_3	t_4	t_5	c_{ef}	c_{nf}	c_{ep}	c_{np}	$H_{(T)}$
3	•	•	•	•	•	2	0	3	0	0.5
4	•	•	•	•	•	2	0	3	0	0.5
5	•	•	•	•		1	1	3	0	0.33
6	•	•	•	•		1	1	3	0	0.33
7	•	•	•	•		1	1	3	0	0.33
8	•	•	•	•		1	1	3	0	0.33
10	•	•	•	•		1	1	3	0	0.33
-	P	P	P	F	F	-	-	-	-	-

NÚCLEO DE APOIO À PESQUISA EM SOFTWARE LIVRE

Localização de defeitos

```
1   int max(int array[], int length)
2   {
3      int i = 0;
4      int max = array[++i]; //array[i++];
5      while(i < length){
6         if(array[i] > max)
7            max = array[i];
8         i++;
9      }
10     return max;
11  }
```

Benchmarks

- Software livre
- Conjunto de teste
- Poucas linhas de código
- Defeitos semeados
- Um defeito por versão

SIR

- Software-artifact Infrastructure Repository [3]
- http://sir.unl.edu

SIR

Software-artifact Infrastructure Repository

Get Access!

Home
SIR Software Objects

SIR Users and Publications
Frequently Asked Questio
Repository License

Acknowledgments

Search for Objects

Java
C
C#
C++

Display

Object name:

Test types: TSL tests Unit tests Other tests
Fault types: Real Seeded Mutation
Other: Concurrency Invariants SPL

Advanced Search

Matches:

1. account

Language	Java
Test Types	none
Fault Types	real
Concurrency Subject	yes
Sequential Versions	1
Size	66 LOC, 3 classes
Acknowledgements	

Updated: **2012-03-26**
Downloads: **507**
SIR Version: **1.1**

2. accountsubtype

Language	Java
Test Types	none
Fault Types	real
Concurrency Subject	yes
Sequential Versions	1
Size	89 LOC, 6 classes
Acknowledgements	

Updated: **2012-03-26**
Downloads: **236**
SIR Version: **1.1**

3. airline

NAPSOL
NÚCLEO DE APOIO À PESQUISA EM SOFTWARE LIVRE

Benchmarks mais usados

Siemens suite

Programa	LDC	Defeitos	Testes
print tokens	565	7	4.130
print tokens2	510	10	4.071
replace	412	32	5.542
schedule	307	9	2.650
schedule2	563	10	2.680
tcas	173	41	1.578
tot info	406	23	1.054

Unix suite

Programa	LDC	Defeitos	Testes
cal	202	20	162
checkeq	102	20	166
col	308	30	156
comm	167	12	186
crypt	134	14	156
look	170	14	193
sort	913	21	997
spline	338	13	700
tr	137	11	870
uniq	143	17	431

grep, gzip, make

Programa	LDC	Defeitos	Testes
gzip	1.670	28	211
grep	3.306	19	470
make	5.318	31	793

Benchmarks maiores

Programa	LDC	Defeitos	Testes
Ant	25.834 – 80.420	19	141 – 851
JBoss	180.437 – 324.176	369	916 – 2.693
JTopas	1.895 – 5.428	4	22 – 31
NanoXML	4.334 – 7.646	33	214 – 216
Space	9.564	38	13.585
Xml-Security	18.302 – 16.861	13	92 – 84

Criação de novos benchmarks

Objetivos:

- Diversidade
- Conjuntos de teste com qualidades diferentes
- Defeitos múltiplos
- Defeitos reais

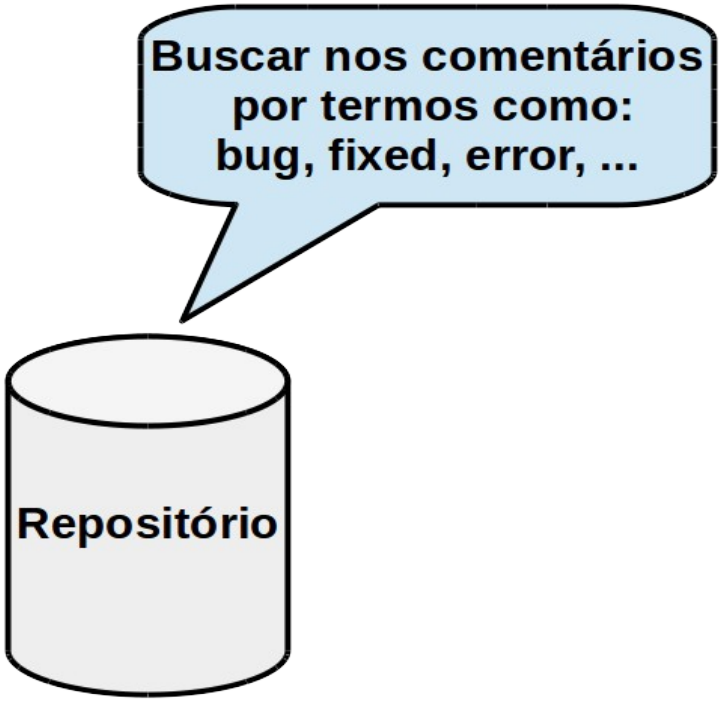
Novos benchmarks

- Identificados com o apoio do NAPSoL

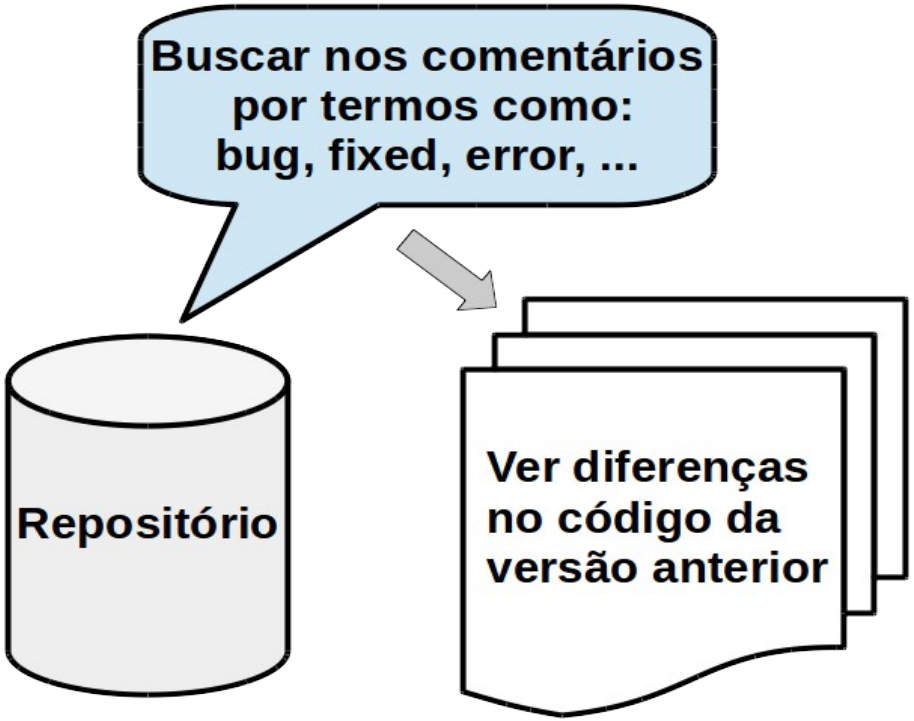
Programa	LDC	Defeitos	Testes
Commons-Math	16.485 – 39.991	20	1.167 – 2.114
HSQLDB	155.744 – 158.409	4	1.146 – 1.299
PMD	50.284	2	616

Criação de novos benchmarks

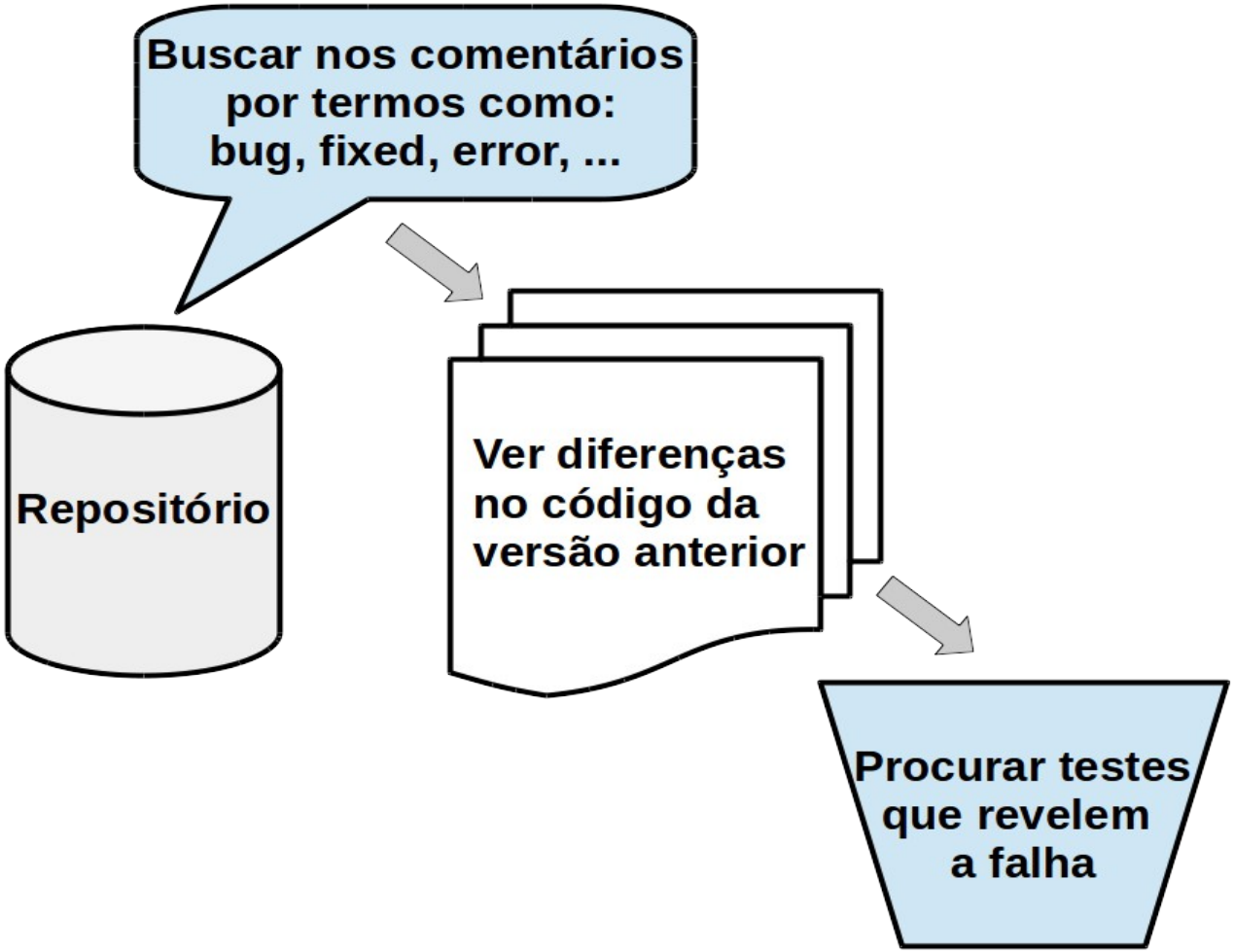
Criação de novos benchmarks



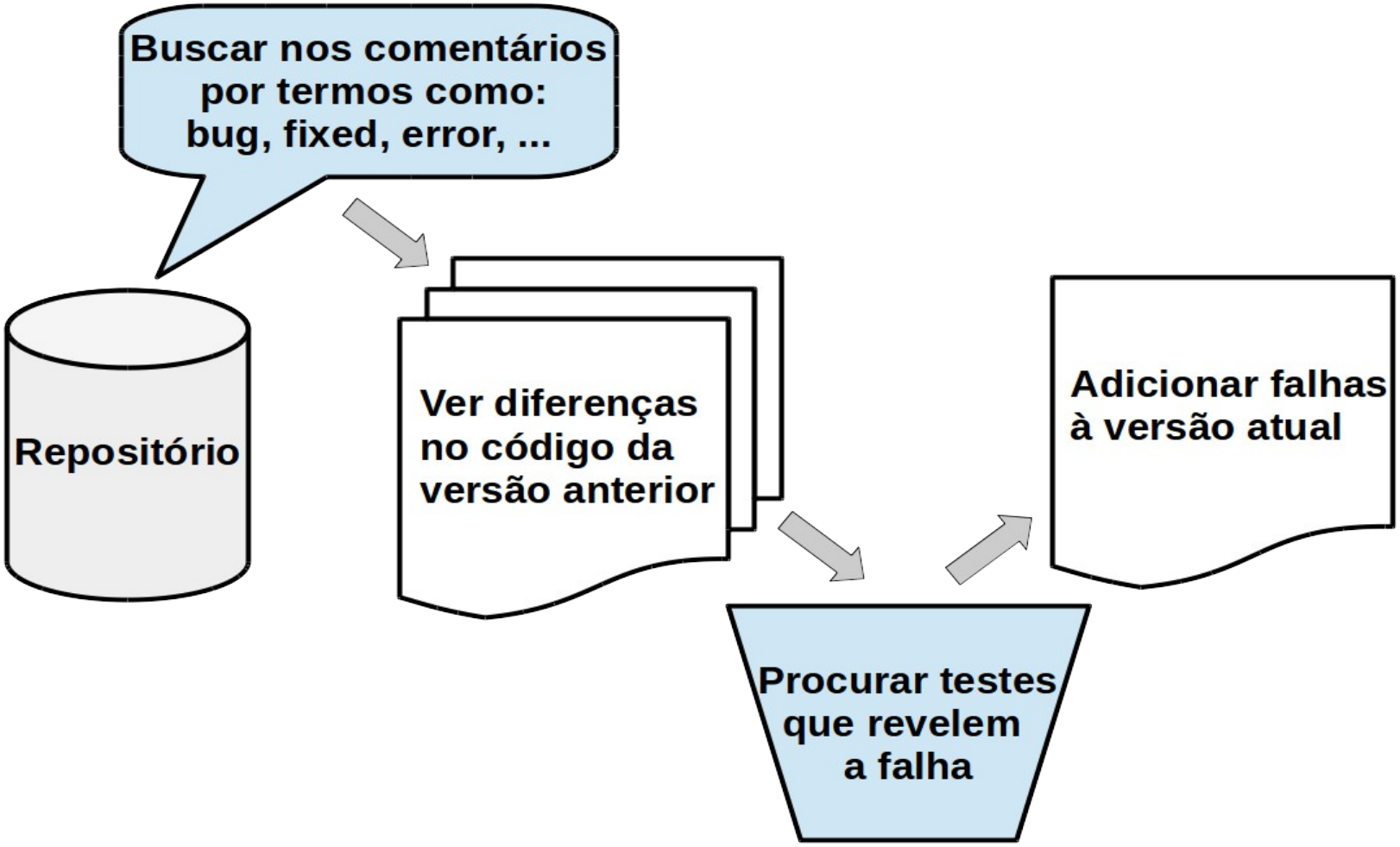
Criação de novos benchmarks



Criação de novos benchmarks



Criação de novos benchmarks



Desafios

- Melhorar a avaliação das técnicas de localização de defeitos:
  - Características dos defeitos
  - Qualidade dos conjuntos de teste
  - Presença de defeitos múltiplos

Agradecimentos

- Fapesp: 2013/24992-2
- NAPSoL

Obrigado!

Referências

[1] CHAIM, M.L.; ANDRIOLI, R.P. An efficient bitwise algorithm for intra-procedural data-flow testing coverage. Information Processing Letters Journal, v. 113, n. 8, p. 293-300, Abril 2013.

[2] JONES, J.A.; HARROLD, M.J.; STASKO, J. Visualization of test information to assist fault localization. In: Proceedings of the 24th international conference on software engineering. (ICSE '02). p. 467-477. 2002.

[3] DO, H.; ELBAUM, S.; ROTHERMEL, G. Supporting controlled experimentation with testing techniques: an infrastructure and its potential impact. Empirical Software Engineering, v. 10, n. 4, p. 405-435, 2005.