

# GERAÇÃO DE TESTES DE ACEITAÇÃO EM FIT A PARTIR DE ESPECIFICAÇÕES EM B

IV WDRA  
SBQS'2010

**Thiago C. de Sousa (EP-USP) , Claudia de O. Melo (IME-USP)**  
**[thiago.carvalho@poli.usp.br](mailto:thiago.carvalho@poli.usp.br), [claudia@ime.usp.br](mailto:claudia@ime.usp.br)**

11/06/2010

# Agenda

- Motivação
- Conceitos
  - Testes de aceitação e FIT
  - Métodos formais, B e ProB
- Proposta
- Trabalhos relacionados
- Solução B2Fit e seus mapeamentos
- Discussão e trabalhos futuros

# Motivação

3

- Uma das tarefas de um projeto ágil é definir um conjunto de testes de aceitação em conjunto com o cliente.
  - isso pode ser difícil em sistemas críticos - muitas restrições/muitos casos de teste.
- Como ajudar equipes ágeis a lidar com sistemas críticos?

# Conceitos::Testes de aceitação

4

- Testes de aceitação são testes formais conduzidos pelo cliente para determinar se o sistema satisfaz ou não seus critérios de aceitação, determinando também se o sistema deve ou não ser aceito [IEEE 1986].

# Conceitos::Testes de aceitação

5

- Automatizar testes de aceitação pode ser uma abordagem para especificar requisitos de forma executável e legível [Reppert 2004].
- A comunidade ágil incentiva a criação de testes de aceitação automatizados durante todo o desenvolvimento.

# Conceitos::Fit

6

- Framework for Integrated Tests.
- Ferramenta para especificação de testes de aceitação automatizados em forma tabular.
- Clientes podem escrever os testes em tabelas HTML, Excel etc.

# Conceitos::Fit – Tabelas Column

7

- **Column:** cada cenário de teste em uma linha (entradas e saídas esperadas).

eg. Division		
numerator	denominator	quotient()
1000	10	100.0000
-1000	10	-100.0000
1000	7	142.85715
1000	.00001	100000000
4195835	3145729	1.3338196

# Conceitos::Fit – Tabelas Row

8

- *Cada linha é, em geral, um dos objetos resultantes de uma consulta.*

eg. <code>music.Display</code>					
title	artist	album	year	time()	track()
Handy Man	James Taylor	JT	1977	3.30	7 of 12
Sailing To Philadelphia	James Taylor	October Rose	2001	5.47	3 of 3
Ananas	James Taylor	Hourglass	1997	5.73	5 of 13
Another Grey Morning	James Taylor	JT	1977	2.73	4 of 12
Copperline	James Taylor	New Moon Shine	1991	4.37	1 of 12

# Conceitos::Fit – Tabelas Action

9

- Cada linha é uma ação, em geral voltada para interfaces de usuário (ex: *ENTER*, *CHECK*).

fit.ActionFixture		
enter	select	1
check	title	Akila
check	artist	Toure Kunda
enter	select	2
check	title	American Tango
check	artist	Weather Report
check	album	Mysterious Traveller
check	year	1974
check	time	3.70
check	track	2 of 7

# Conceitos::Métodos formais

10

- Técnicas matemáticas para especificação, desenvolvimento e verificação de software.
- Componentes de um método formal:
  - Linguagem de especificação/modelagem formal;
  - Sistema de raciocínio formal (para verificação/análise formal).

# Conceitos::Método B

11

- Desenvolvido na década de 90 em Oxford para requisitos funcionais de sistemas críticos.
- Baseado em lógica de predicado, teoria dos conjuntos, aritmética de inteiros e substituições generalizadas.
- Forte aplicação industrial, especialmente sistemas metro-ferroviários.

# Conceitos::ProB

12

ProB 1.3.0-final.4: [RegistroNotas.mch]

File Edit Animate Verify Analyse Preferences Debug Files Help

```
MACHINE RegistroNotas
SETS ALUNO
VARIABLES banco
INVARIANT banco : ALUNO +-> 0..10
INITIALISATION banco := {}
OPERATIONS
registrar(cc,nn) =
  PRE cc : ALUNO & cc /: dom(banco) & nn : 0..10
  THEN banco(cc) := nn
  END;
nn <-- nota (cc) =
  BEGIN
  nn := banco(cc)
  END
END
```

OK max State Properties

EnabledOperations	History
registrar(ALUNO7,0) registrar(ALUNO7,1) registrar(ALUNO7,2) registrar(ALUNO7,3) registrar(ALUNO7,4) registrar(ALUNO7,5) registrar(ALUNO7,6) registrar(ALUNO7,7) registrar(ALUNO7,8) registrar(ALUNO7,9) nota(ALUNO1)->3 nota(ALUNO2)->5 nota(ALUNO3)->9 nota(ALUNO4)->7 nota(ALUNO5)->6	registrar(ALUNO6,2) nota(ALUNO4)->7 registrar(ALUNO5,6) registrar(ALUNO4,7) nota(ALUNO3)->9 nota(ALUNO2)->5 registrar(ALUNO3,9) registrar(ALUNO2,5) registrar(ALUNO1,3) initialise_machine()

# Proposta

13

- Utilizar especificações B como ferramenta auxiliar para a geração de testes de aceitação Fit.
- Aplicabilidade: apoiar o desenvolvimento de sistemas críticos por equipes ágeis.

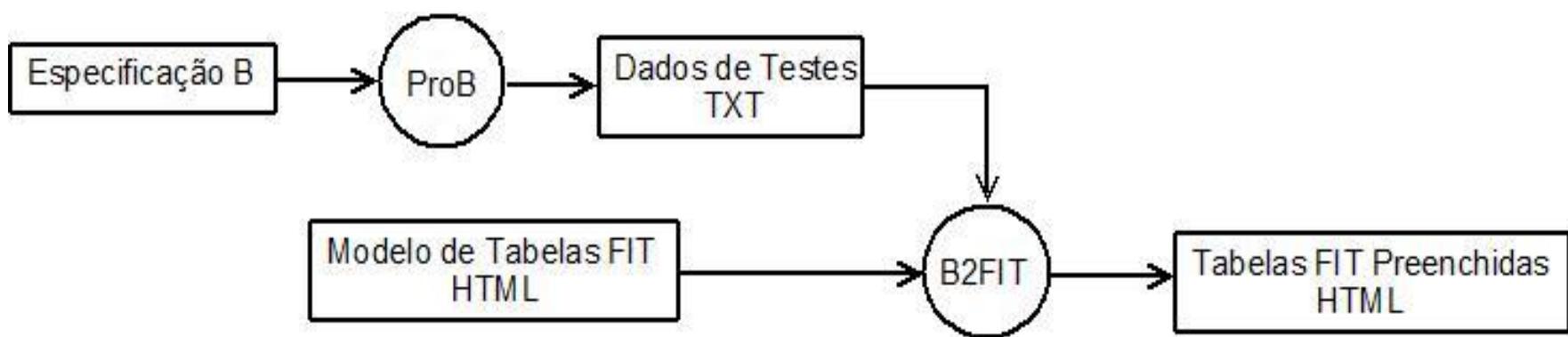
# Alguns trabalhos relacionados

14

- Métodos formais e testes são técnicas complementares para a redução de erros em sistemas [Bowen et al. 2002].
- Ferramentas de geração de testes podem melhorar trade-off entre restrições de tempo e qualidade em um projeto [Bacchelli et al. 2008].
- Método para geração de casos de testes a partir de especificações em B e a ferramenta ProB [Gupta e Bhatia 2010]. Não tem integração com ferramentas de testes de aceitação usadas por equipes ágeis.

# Solução proposta:: B2Fit

15



# Padrão para *ColumnFixture*

16

- Usada quando se deseja verificar se uma dada função produz corretamente uma saída de acordo com as entradas fornecidas.

MACHINE OPFUN1

OPERATIONS OPFUN1

END

# ColumnFixture:: Exemplo

17

ProB 1.3.0-final.4: [Lift.mch]

File Edit Animate Verify Analyse Preferences Debug Files Help

MACHINE Subtracao

OPERATIONS

```
sub <-- subtracao(a,b) =  
    PRE a:-3..3 & b: -3..3  
    THEN sub := a - b  
    END
```

END

eg. Subtracao

a	b	sub
-3	-3	0
-3	-2	-1
-3	-1	-2
-3	0	-3
-3	1	-4
-3	2	-5
-3	3	-6
-2	-3	1
-2	-2	0
-2	-1	-1
-2	0	-2
-2	1	-3
-2	2	-4
-2	3	-5
-1	-3	2

EnabledOperations

- subtracao(-3,-3)->0
- subtracao(-3,-2)->-1
- subtracao(-3,-1)->-2
- subtracao(-3,0)->-3
- subtracao(-3,1)->-4
- subtracao(-3,2)->-5
- subtracao(-3,3)->-6
- subtracao(-2,-3)->1
- subtracao(-2,-2)->0
- subtracao(-2,-1)->-1
- subtracao(-2,0)->-2
- subtracao(-2,1)->-3
- subtracao(-2,2)->-4
- subtracao(-2,3)->-5
- subtracao(-1,-3)->2

invariant\_ok

OK State Properties



# Padrão para RowFixture

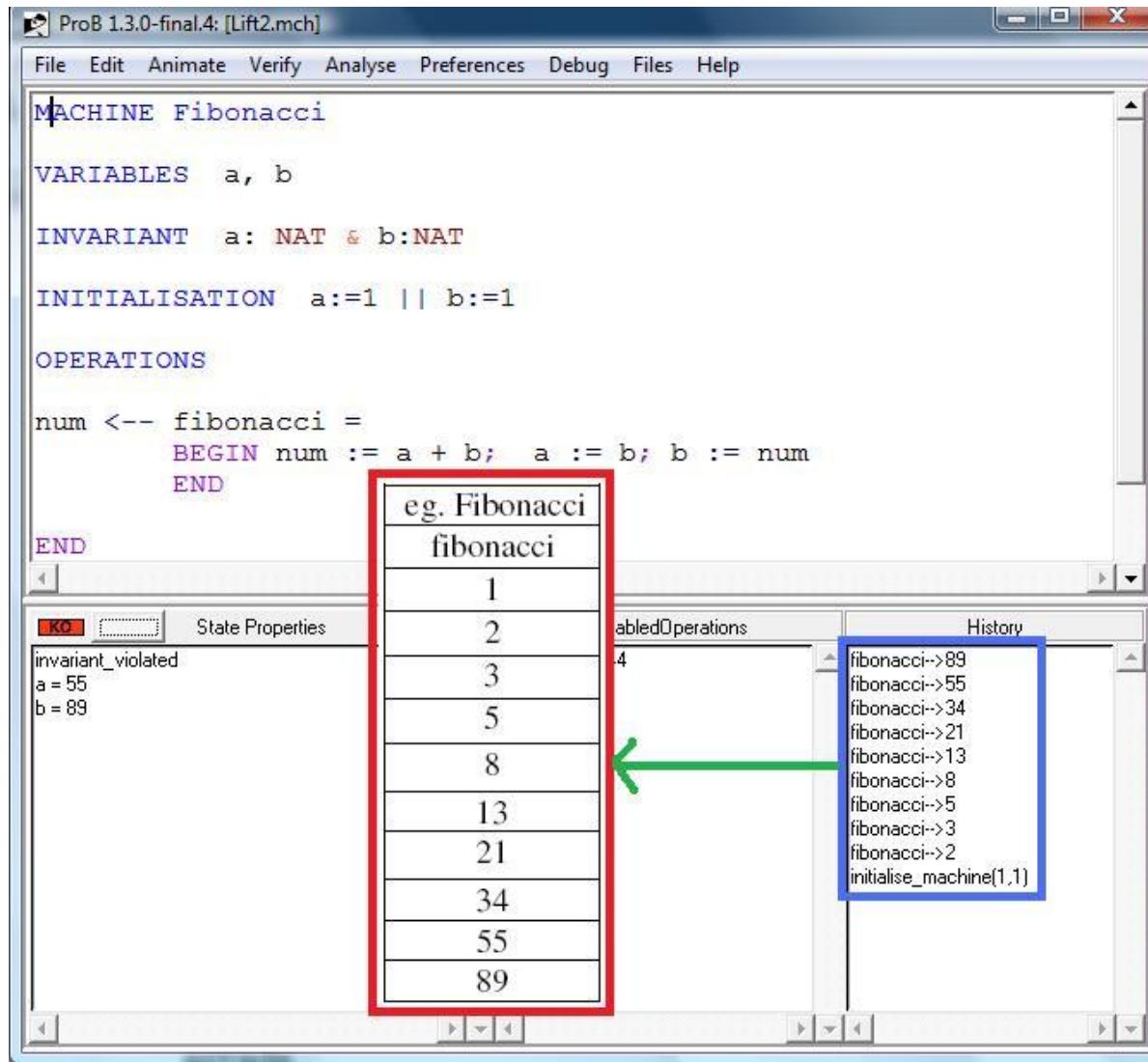
18

- Utilizado quando se deseja testar consultas que devolvem um conjunto com um número exato de elementos.

```
MACHINE OPQUERY1  
  
VARIABLES XX  
  
INVARIANT YY  
  
INITIALISATION ZZ  
  
OPERATIONS OPQUERY1  
  
END
```

# RowFixture:: Exemplo

19



The screenshot shows the ProB 1.3.0 interface with the title "ProB 1.3.0-final.4: [Lift2.mch]". The machine definition is as follows:

```
MACHINE Fibonacci
VARIABLES a, b
INVARIANT a: NAT & b:NAT
INITIALISATION a:=1 || b:=1
OPERATIONS
num <-- fibonacci =
  BEGIN num := a + b; a := b; b := num
  END
END
```

The "State Properties" table shows:

KO	State Properties
invariant_violated	
a = 55	
b = 89	

The "History" table shows a sequence of operations:

abledOperations	History
4	fibonacci->89 fibonacci->55 fibonacci->34 fibonacci->21 fibonacci->13 fibonacci->8 fibonacci->5 fibonacci->3 fibonacci->2 initialise_machine(1,1)

A red box highlights the "History" table, and a blue box highlights the "abledOperations" table. A green arrow points from the "History" table to the "abledOperations" table.

# Padrão para *ActionFixture*

20

- Usada quando se precisa simular eventos ocasionados pelos usuários na interface do sistema.

```
MACHINE OPQUERY1
VARIABLES   XX
INVARIANT   YY
INITIALISATION   ZZ
OPERATIONS OPEVENT1 END; OPEVENT2 END; .....OPEVENTN END
END
```

# ActionFixture:: Exemplo

21

The screenshot shows the ProB tool interface with the following components:

- Top Bar:** File, Edit, Animate, Verify, Analyse, Preferences, Debug, Files, Help.
- Middle Left:** Machine definition for "Contador".

```
MACHINE Contador
VARIABLES contador
INVARIANT contador: 0..10
INITIALISATION contador := 0
OPERATIONS
    pressiona =
        PRE contador < 10
        THEN contador := contador + 1 |END;
    at <-- atribui (x) =
        PRE x : 0..10
        THEN contador := x ; at := contador |END;
    num <-- verifica = BEGIN num := contador END
END
```
- Middle Right:** A red-bordered table titled "eg. Contador" showing a sequence of operations and their states. A green arrow points from the "History" tab in the bottom panel to this table.
- Bottom:** A "History" tab showing a list of operations and their states, with a blue box highlighting the list. The operations listed are: verifica->4, pressiona, atribui[3]->3, verifica->8, verifica->8, pressiona, atribui[7]->7, verifica->8, atribui[8]->8, verifica->3, pressiona, pressiona, verifica->1, atribui[1]->1, initialise\_machine[0].

# Discussão e trabalhos futuros

22

- Integrar métodos ágeis e métodos formais é uma ideia recente e promissora [Cohen e Money 2008][Black et al. 2009].
- Vantagens
  - Aumenta a possibilidade de aplicação de métodos ágeis em desenvolvimento de sistemas críticos;
  - Redução do tempo de especificação dos testes de aceitação;
  - Redução do tempo de aprendizado da notação B (mapeamento).

# Discussão e trabalhos futuros

23

- O projeto está em fase inicial
  - Protótipo em desenvolvimento.
- Trabalhos futuros
  - Realizar um estudo de caso na indústria – empresa de sistemas críticos;
  - Inclusão de particionamento de equivalência e análise do valor limite;
  - Integração com outras ferramentas (ex: Fitnesse).

# Referências

24

- Cohen, S. J. and Money, W. H. (2008) Bridge Methods: Complementary Steps Integrating Agile Development Tools and Methods with Formal Process Methodologies. In *Proc. of the 41st Annual Hawaii international Conference on System Science*. HICSS. IEEE Computer Society, Washington, DC, 460.
- Black, S., Boca, P. P., Bowen, J. P., Gorman, J., and Hinchey, M. (2009) Formal Versus Agile: Survival of the Fittest. *Computer* 42, 9, 37-45.
- IEEE Std 1012 (1986) IEEE Standard for Software Verification and Validation Plans.
- Reppert, T. (2004) “Don’t Just Break Software, Make Software: How Story-Test-Driven-Development is Changing the Way QA, Customers, and Developers Work”. *Better Software*, 6(6): 18–23.
- Stotts, P. D., Lindsey, M., and Antley, A. (2002) *An Informal Formal Method for Systematic JUnit Test Case Generation*. In *Proc of the Second XP Universe and First Agile Universe Conference on Extreme Programming and Agile Methods - Xp/Agile Universe 2002. Lecture Notes In Computer Science*, vol. 2418. Springer-Verlag, London, 131-143.
- Gupta, A. and Bhatia, R. (2010) *Testing functional requirements using B model specifications*. *SIGSOFT Software Engineering Notes* 35, 2 , 1-7.
- Bacchelli, A., Ciancarini, P., and Rossi, D. (2008) On the Effectiveness of Manual and Automatic Unit Test Generation. In *Proceedings of the 2008 the Third international Conference on Software Engineering Advances*. ICSEA. IEEE Computer Society, Washington, DC, 252-257.