Università della Svizzera italiana

Facoltà di scienze informatiche

DaTeC: Contextual Data Flow Testing of Java Classes

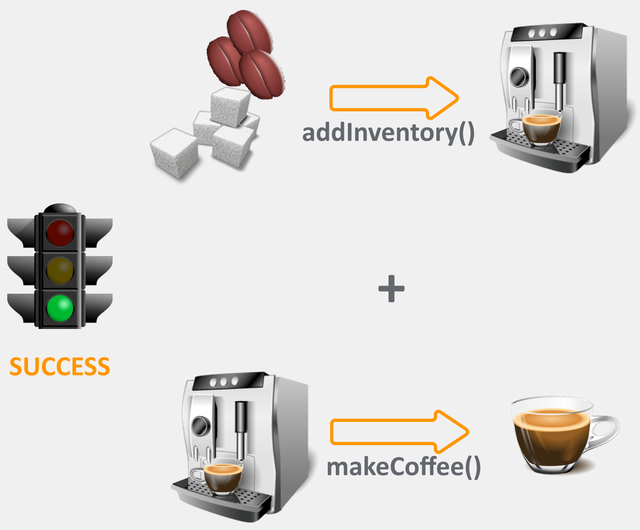
Giovanni Denaro — University of Milano-Bicocca

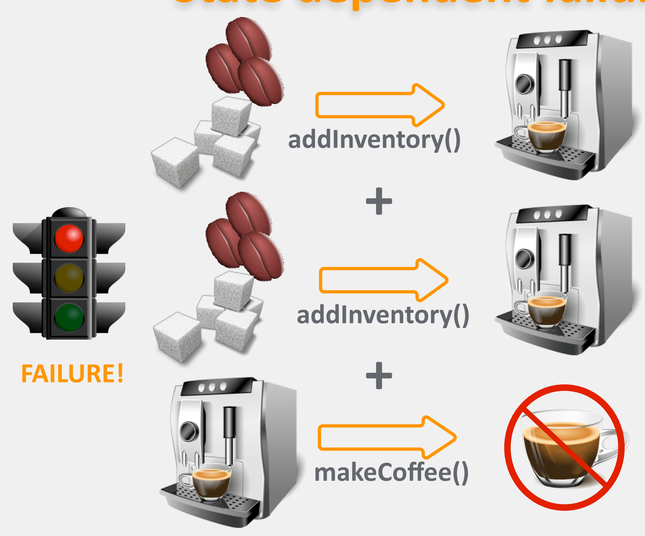
Alessandra Gorla — University of Milano-Bicocca

Mauro Pezzè — University of Lugano

UNIVERSITÀ DEGLI STUDI DI MILANO BICOCCA

State dependent failures





- **Control flow testing**
  - focuses on control flow
  - does not cope with state information
  - misses state-based faults
- **Traditional data flow testing**
  - focuses on primitive variables
  - does not cope with state encapsulation and aggregation
  - misses many state-based faults
- **Contextual data flow analysis**
  - **focuses on complex state information**
  - **copes with state encapsulation and aggregation**
  - **can find context-dependent state-based faults.**

Comparing coverage criteria

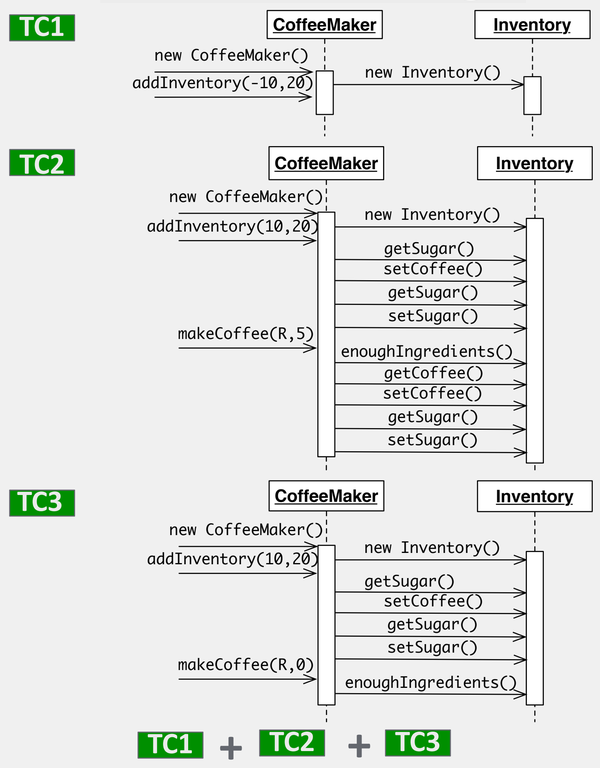
```
 1  public class CoffeeMaker {
 2     private Inventory inv;
 3     public CoffeeMaker() {
       ...
 4        inv = new Inventory();
 5     }
 6     public boolean addInventory(int amtCoffee,int amtSugar){
 7          if(amtCoffee < 0 || amtSugar < 0) return false;
 8          /*FAULT: Next statement should use inv.getCoffee()*/
 9          inv.setCoffee(inv.getSugar() + amtCoffee);
10          inv.setSugar(inv.getSugar() + amtSugar);
11          return true;
12     }
13     public int makeCoffee(Recipe r, int amtPaid) {
14          if(!inv.enoughIngredients(r) || amtPaid<r.getPrice())
15              return amtPaid;
16          inv.setCoffee(inv.getCoffee() - r.getAmtCoffee());
17          inv.setSugar(inv.getSugar() - r.getAmtSugar());
18          return amtPaid - r.getPrice();
19     }
       ...
   }

20  public class Inventory {
21     private int coffee;
22     private int sugar;
23     public int getCoffee() {
24         return coffee;
25     }
26     public void setCoffee(int coffee) {
27         this.coffee = coffee;
28     }
29     public int getSugar() {
30         return sugar;
31     }
32     public void setSugar(int sugar) {
33         this.sugar = sugar;
34     }
35     ...
36  }
```

Statement coverage

All-uses coverage

Contextual all-uses coverage

Statement coverage



TC1 + TC2 + TC3

cover all statements, but miss the fault

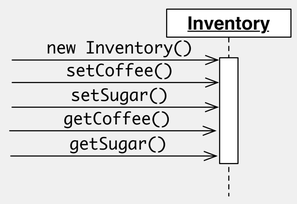
All-uses coverage

<Inventory: int coffee>
DEF: <Inventory: void setCoffee(int)><Line: 27>
USE: <Inventory: int getCoffee()><Line: 24>

<Inventory: int sugar>
DEF: <Inventory: void setSugar(int)><Line: 33>
USE: <Inventory: int getSugar()><Line: 30>

<CoffeeMaker: Inventory inv>
DEF: <CoffeeMaker: void <init>()><Line: 4>
USE: <CoffeeMaker: boolean addInventory(int,int)><Line: 9>
USE: <CoffeeMaker: boolean addInventory(int,int)><Line: 10>
USE: <CoffeeMaker: int makeCoffee(R,int)><Line: 14>
USE: <CoffeeMaker: int makeCoffee(R,int)><Line: 16>
USE: <CoffeeMaker: int makeCoffee(R,int)><Line: 17>

TC4



TC1 + TC2 + TC3 + TC4

cover all non contextual def-use pairs, but still miss the fault.

Contextual all-uses coverage

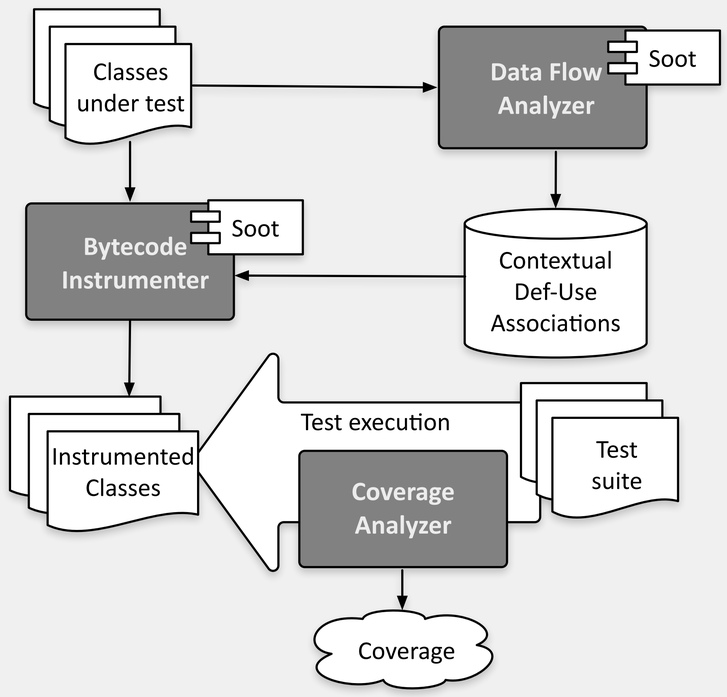
<CoffeeMaker: Inventory inv><Inventory: int coffee>
DEF: <CoffeeMaker: boolean addInventory(int,int)><Inventory: void setCoffee(int)><Line: 27>
USE: <CoffeeMaker: int makeCoffee(R,int)><Inventory: int getCoffee()><Line: 24>
DEF: <CoffeeMaker: int makeCoffee(R,int)><Inventory: void setCoffee(int)><Line: 27>
USE: <CoffeeMaker: int makeCoffee(R,int)><Inventory: int getCoffee()><Line: 24>

<CoffeeMaker: Inventory inv><Inventory: int sugar>
DEF: <CoffeeMaker: boolean addInventory(int,int)><Inventory: void setSugar(int)><Line: 33>
USE: <CoffeeMaker: boolean addInventory(int,int)><Inventory: int getSugar()><Line: 30>
USE: <CoffeeMaker: int makeCoffee(R,int)><Inventory: int getSugar()><Line: 30>
DEF: <CoffeeMaker: int makeCoffee(R,int)><Inventory: void setSugar(int)><Line: 33>
USE: <CoffeeMaker: boolean addInventory(int,int)><Inventory: int getSugar()><Line: 30>
USE: <CoffeeMaker: int makeCoffee(R,int)><Inventory: int getSugar()><Line: 30>

TC1 + TC2 + TC3 + TC4 + TC5 + TC6 + TC7 + TC8

cover all contextual def-use pairs and reveal the fault.

Framework details

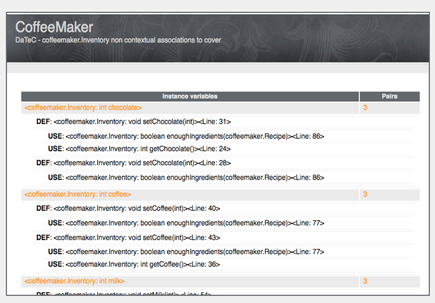


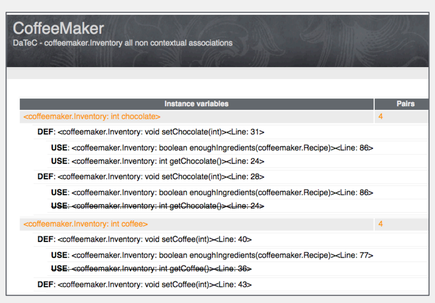
Summary information

CoffeeMaker
DaTeC Coverage Report

No.	Class	Ctx Pairs	% Ctx Covered	Non-Ctx Pairs	% Non-Ctx Covered
1	coffeemaker.CoffeeMaker	108	23.15% (83 to cover)	39	43.59% (22 to cover)
2	coffeemaker.Inventory	48	16.67% (40 to cover)	16	25% (12 to cover)
3	coffeemaker.Recipe	14	21.43% (11 to cover)	14	21.43% (11 to cover)
Total:		170	21.18% (134 to cover)	69	34.78% (45 to cover)

Detailed information





Preliminary evaluation

	No. classes	SLOC	No. state vars
JEdit	910	92,213	2,975
Ant	785	80,454	4,081
BCEL	383	23,631	929
Lucene	287	19,337	1,013
JTopas	63	5,359	196
NanoXML	25	3,279	39
Siena	27	2,162	66

	Max pairs for 95% classes	Time(sec)
JEdit	381	70
Ant	331	60
BCEL	792	24
Lucene	410	17
JTopas	49	8
NanoXML	4	5
Siena	35	5

Further information

For a demonstration and more information:

contact: alessandra.gorla@lu.unisi.ch **website**: www.inf.unisi.ch/phd/gorla

References

[1] Giovanni Denaro, Alessandra Gorla and Mauro Pezzè "**DaTeC: Dataflow Testing of Java Classes**" in ICSE 09: Proceedings of the International Conference on Software Engineering (Tool Demo) 2009

[2] Giovanni Denaro, Alessandra Gorla and Mauro Pezzè "**Contextual Integration Testing of Classes**" in FASE 08: Proceedings of the 11th International Conference on Fundamental Approaches to Software Engineering, pp. 246--260.