CM7 – Analyse de Mutation

Jean-Marie Mottu, Gerson Sunyé, Yves Le Traon

Intuition

- Plus la qualité des tests est élevée plus on peut avoir confiance dans le programme
- L'analyse de mutation permet d'évaluer la qualité des tests vis à vis du logiciel :
  - Apparentable à une mesure de testabilité
- Si les cas de test peuvent détecter des fautes mises intentionnellement, ils peuvent détecter des fautes réelles

Hypothèses

- Le programmeur est compétent
  - En général, les programmeurs sont compétents et écrivent des programmes *presque* corrects.
  - Ces programmes sont seulement un peu différents de la version correcte.
- L'effet de couplage.
  - Une faute complexe commise par un programmeur compétent est la combinaison de fautes simples
  - Détecter les fautes simples => détecte les fautes complexes

Analyse de mutation

- Qualifie un ensemble de données de test
  - évalue la proportion de fautes que les test détectent
  - fondé sur l'injection de fautes
- L'évaluation de la qualité des données de test est importante pour évaluer la confiance dans le programme

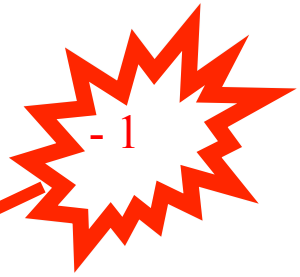
Analyse de mutation

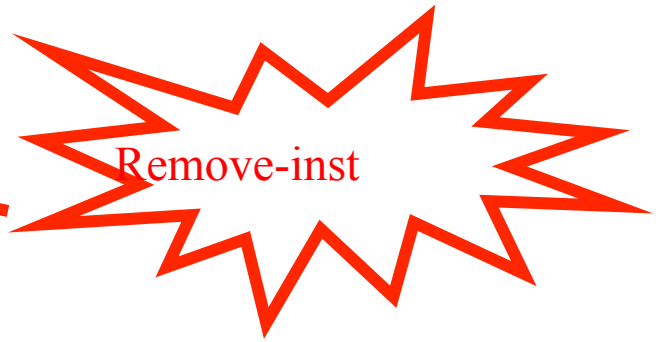
- Le choix des fautes injectées est très important
  - les fautes sont modélisées par des *opérateurs de mutation*
- Mutant = programme initial avec une seule faute injectée

- Deux fonctions d'oracle
  - Différence de traces entre le programme initial et le mutant
  - Contrats exécutables

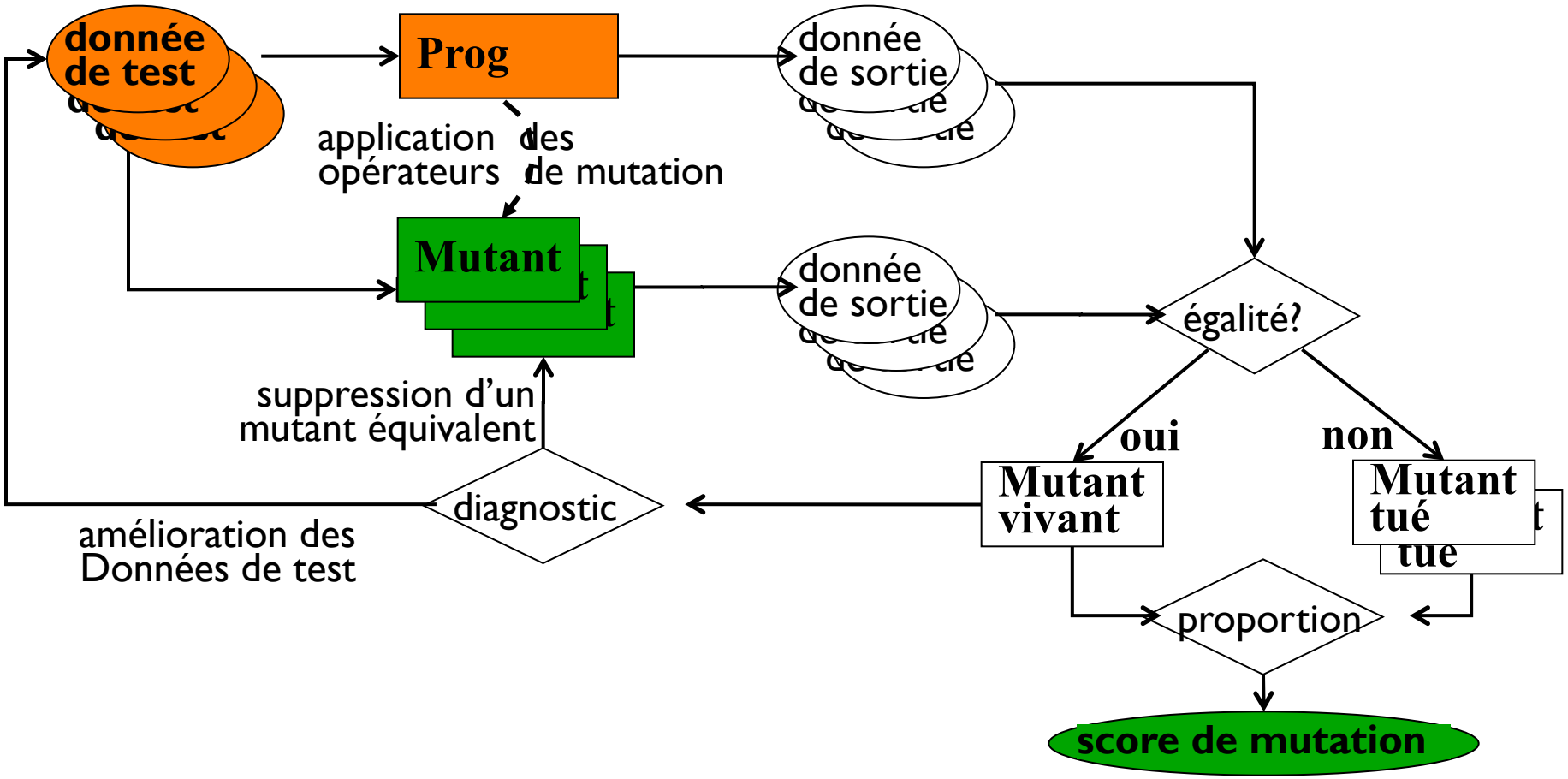
Analyse de mutation

```
put (x : INTEGER) is
      -- put x in the set
    require  not full: not full
    do
1     if not has (x) then
2       count := count + 1
3       structure.put (x, count)
      end -- if
    ensure
      has: has (x)
      not empty: not empty
end -- put
```





Processus de l'analyse de mutation



Mutants vivants et score de mutation

- Si un mutant n'est pas tué?
  - Données de test insuffisantes => ajouter des données de test
  - mutant équivalent => supprimer le mutant

- Q(Ci) = score de mutation de Ci = di/mi
  - di = nombre de mutants tués
  - mi = nombre de mutants non équivalents
- **Attention** Q(Ci)=100% *not*=> *bug free*
- Qualité d'un système S fait de composants di
  - Q(S) = Σ di / Σ mi

Modèles de fautes

- Quelles fautes doit-on **mettre à l'épreuve** avec des *modèles de test*?
- Quelles fautes doit-on **observer** avec des *oracles*?
- Quelles fautes **apparaissent** dans un *type de programme donné*?

Opérateurs de mutation

Opérateurs de mutation (1)

- Remplacement d'un opérateur arithmétique
  - Exemple: '+' devient '-' and vice-versa
- Remplacement d'un opérateur logique
  - les opérateurs logiques (and, or, nand, nor, xor) sont remplacés;
  - les expressions sont remplacées par TRUE et/ou FALSE

Opérateurs de mutation (2)

- Remplacement des opérateurs relationnels
  - les opérateurs relationnels (<, >, <=, >=, =, /=) sont remplacés.
- Suppression d'instruction
- Perturbation de variable et de constante
  - +1 sur une variable
  - chaque booléen est remplacé par son complément.

```
1   read(x);
2   read(y);
3   z :=0;
4   signe :=1;
5   if x < 0 then
    begin
6           signe = -1;
7           x:=-x;
    end
8   if y<0 then
    begin
9           signe := -signe;
10          y:=-y;
    end
11  while x >= y do
    begin
12          if x = y then
            begin
13                      print (x);
            end
14  x:=x-y;
15  z:=z+1;
    end
16  z: = signe*z;
17  print(z);
```

Exemple

- O1 : > est remplacé par >=
- O2 : >= est remplacé par >
- O3 : < est remplacé par <=
- O4 : <= est remplacé par <
- O5 : = est remplacé par <=
- O6 : l'opérateur binaire + est remplacé par -
- O7 : l'opérateur unaire - est remplacé par +

O2: >= est remplacé par >

```
1   read(x);
2   read(y);
3   z :=0;
4   signe :=1;
5   if x < 0 then
    begin
6       signe = -1;
7       x:=-x;
    end
    if y<0 then
    begin
9       signe := -signe;
10      y:=-y;
    end
    ~~while x >= y do~~                    while x > y do
    begin
12      if x = y then
        begin
13          print (x);
        end
14  x:=x-y;
15  z:=z+1;
    end
16  z: = signe*z;
17  print(z);
```

O3: < est remplacé par <= une première fois

```
1   read(x);
2   read(y);
3   z :=0;
4   signe :=1;
5   if x < 0 then                                   if x<=0 then
    begin
6           signe = -1;
7           x:=-x;
    end
    if y<0 then
    begin
9           signe := -signe;
10          y:=-y;
    end
    while x >= y do
    begin
12          if x = y then
            begin
13                      print (x);
            end
14  x:=x-y;
15  z:=z+1;
    end
16  z: = signe*z;
17  print(z);
```

O3: < est remplacé par <= une seconde fois

```
1   read(x);
2   read(y);
3   z :=0;
4   signe :=1;
5   if x < 0 then
    begin
6       signe = -1;
7       x:=-x;
    end
    if y<0 then                                   if y<=0 then
    begin
9       signe := -signe;
10      y:=-y;
    end
    while x >= y do
    begin
12      if x = y then
        begin
13              print (x);
        end
14  x:=x-y;
15  z:=z+1;
    end
16  z: = signe*z;
17  print(z);
```

Method-level Mutation Operators Ma, Offutt

Operator	Description
AOR	Arithmetic Operator Replacement
AOI	Arithmetic Operator Insertion
AOD	Arithmetic Operator Deletion
ROR	Relational Operator Replacement
COR	Conditional Operator Replacement
COI	Conditional Operator Insertion
COD	Conditional Operator Deletion
SOR	Shift Operator Replacement
LOR	Logical Operator Replacement
LOI	Logical Operator Insertion
LOD	Logical Operator Deletion
ASR	Assignment Operator Replacement

Test par mutation

- Génération de test dirigée par:
  - la qualité: choisir une qualité souhaitée Q(Ci)
  - l'effort: choisir un nombre minimum de données de test possibles

Test par mutation

- Améliorer la qualité d'un ensemble de cas de test
  - tant que Q(Ci) < Q(Ci) et nTC <= MaxTC
    - ajouter des cas de test (nTC++)
    - relancer l'exécution des mutants
      - éliminer les mutants équivalents
    - recalculer Q(Ci)

- Diminuer la taille d'un ensemble de cas de test
  - supprimer les cas de test qui tuent les même mutants

Exemple

	p_date.e	p_time.e	p_date_time.e
Nombre total de mutants	673	275	199
Nb équivalents	49	18	15
Score de mutation	100%	100%	100%
Taille init ensemble de test	106	93	78
Taille ensemble de test réduit	72	33	44

Rapport de test

id_mut	EQ	METHODE	SOURCE	MUTANT	COMMENTAIRE
2	1	empty	count = lower_bound - 1	count <= lower_bound - 1	jamais <
6	2	full	count = upper_bound	count >= upper_bound	jamais >
16	3	index_of - loop variant	count + 2	count * 2	même
24	4	index_of - loop until	count or else structure	count or structure	court test
30	5	make	count := 0	(nul)	valeur défaut
45	6	make	lower_bound, upper_bound	(lower_bound - 1), upper_bound	redondance
46	7	make	lower_bound, upper_bound	lower_bound, (upper_bound + 1)	redondance
60	I	full	count =	count + 1 =	test insuf.
63	II	full	upper_bound;	(upper_bound - 1);	test insuf.
72	III	put	if not has (x) then	if true then	Spec inc.
75	IV	put	if not has (x) then	if not false then	Spec inc.
98	8	index_of - loop variant	- Result)	- Result + 1)	même
99	9	index_of - loop variant	- Result)	- Result - 1)	même
100	10	index_of - loop variant	count + 2 -	(count + 2 + 1) -	même
101	11	index_of - loop variant	count + 2 -	(count + 2 - 1) -	même
102	12	index_of - loop variant	count + 2 -	(count + 1) + 2 -	même
103	13	index_of - loop variant	count + 2 -	(count - 1) + 2 -	même
104	14	index_of - loop variant	count + 2 -	(count + 3 -	même
105	15	index_of - loop variant	count + 2 -	(count + 1 -	même
110	V	index_of - loop until	> count or	> (count + 1) or	test insuf.

NON EQUIVALENT

EQUIVALENT

119 mutants, 99 dead, 15 equivalents MS= 99/104=95%

Opérateurs pour Orienté Objet

- Pour évaluer des données de test pour des programmes OO, il est important d'avoir des opérateurs spécifiques qui modélisent des fautes de conception OO

- Des idées de fautes OO?

Opérateurs OO(1)

- Exception Handling Fault
  - force une exception
- Visibilité
  - passe un élément privé en public et vive-versa
- Faute de référence (Alias/Copy)
  - passer un objet à null après sa création.
  - supprimer une instruction de clone ou copie.
  - ajouter un clone.

Opérateurs OO(2)

- Inversion de paramètres dans la déclaration d'une méthode
- Polymorphisme
  - affecter une variable avec un objet de type « frère »
  - appeler une méthode sur un objet « frère »
  - supprimer l'appel à super
  - suppression de la surcharge d'une méthode

Opérateurs OO(3)

- En Java
  - erreurs sur *static*
  - mettre des fautes dans les librairies

Class Mutation Mutation Operators for Java Ma, Offutt

Language Feature	Operator	Description
Encapsulation	AMC	Access modifier change
Inheritance	IHD	Hiding variable deletion
	IHI	Hiding variable insertion
	IOD	Overriding method deletion
	IOP	Overriding method calling position change
	IOR	Overriding method rename
	ISI	super keyword insertion
	ISD	super keyword deletion
	IPC	Explicit call to a parent's constructor deletion
Polymorphism	PNC	new method call with child class type
	PMD	Member variable declaration with parent class type
	PPD	Parameter variable declaration with child class type
	PCI	Type cast operator insertion
	PCC	Cast type change
	PCD	Type cast operator deletion
	PRV	Reference assignment with other comparable variable
	OMR	Overloading method contents replace
	OMD	Overloading method deletion
	OAC	Arguments of overloading method call change
Java-Specific Features	JTI	this keyword insertion
	JTD	this keyword deletion
	JSI	static modifier insertion
	JSD	static modifier deletion
	JID	Member variable initialization deletion
	JDC	Java-supported default constructor creation
	EOA	Reference assignment and content assignment replacement
	EOC	Reference comparison and content comparison replacement
	EAM	Acessor method change
	EMM	Modifier method change

Opérateurs Finite State Machine

- Fabbri et al.
  - 9 mutation operators
    - related to the states, events and outputs of an FSM.
  - implemented as an extension of the C mutation tool Proteum -> Proteum/FSM

Opérateurs Finite State Machine

- Fabbri et al.
  - 9 mutation operators
    - arc-missing
    - wrong-starting-state (default state)
    - event-missing
    - event-exchanged
    - event-extra
    - state-extra
    - output-exchanged
    - output-missing
    - output-extra

Opérateurs Statecharts

- hierarchy of states,
- ability to specify parallelism and communication mechanism via broadcasting, and a special notation set augmenting the representational power in relation to FSM
- state history

Statecharts : Fabbri

- FSM operator set

1. wrong-start-state
2. arc-missing
3. event-missing
4. event-extra
5. event-exchanged
6. destination-exchanged
7. output-missing
8. output- exchanged
9. state-missing

- EFSM operator set

1. expression deletion
2. boolean expression negation
3. term associativity shift
4. arithmetic operator by arithmetic operator
5. relational operator by relational operator
6. logical operator by logical operator
7. logical negation
8. variable by variable replacement
9. variable by constant replacement
10. constant by required constant replacement
11. constant by scalar variable replacement

- Statecharts-feature-based operator set

1. transition's history deletion
2. transition with history by transition replacement
3. history-missing
4. h by h* replacement
5. h* by h replacement
6. h-extra
7. h*-extra
8. in(s) condition-missing
9. in(s) condition state replacement
10. not-yet(e) condition-missing
11. not-yet(e) condition event replacement
12. exit(s) event-missing
13. exit(s) event state replacement
14. entered(s) event-missing
15. entered(s) event state replacement
16. broadcasting origin transition replacement
17. broadcasting destination transition replacement

Algorithme génétique

- Exploitation de l'analyse de mutation pour sélectionner les individus d'une population
  - Rejette les mauvais cas de test d'un ensemble de cas de test
  - Génération de nouveaux cas de test
    - Dérivé des meilleurs cas de test sélectionnés précédemment
  - Reprise du cycle

Conclusion

Outils

- MuJava
  - http://cs.gmu.edu/~offutt/mujava/
- Jester
  - http://jester.sourceforge.net/
- Jumble
  - http://jumble.sourceforge.net/
- Javalanche
  - http://javalanche.org
- PIT
  - http://pitest.org

Limites

- Il est difficile d'identifier les mutants équivalents.
- Le coût de la création de mutants est très (trop) important.
- Un peu de travail manuel est nécessaire.

Conclusion

- L'analyse de mutation est efficace
  - pour évaluer la qualité des cas de test
  - pour associer un niveau de confiance à une classe ou un composant
- Les opérateurs de mutation
  - bons exemples de fautes à rechercher