Analyse de mutation

Intuition

- Plus la qualité des test est élevée plus on peut avoir confiance dans le programme
- L’analyse de mutation permet d’évaluer la qualité des tests
- Si les cas de test peuvent détecter des fautes mises intentionnellement, ils peuvent détecter des fautes réelles

Analyse de mutation

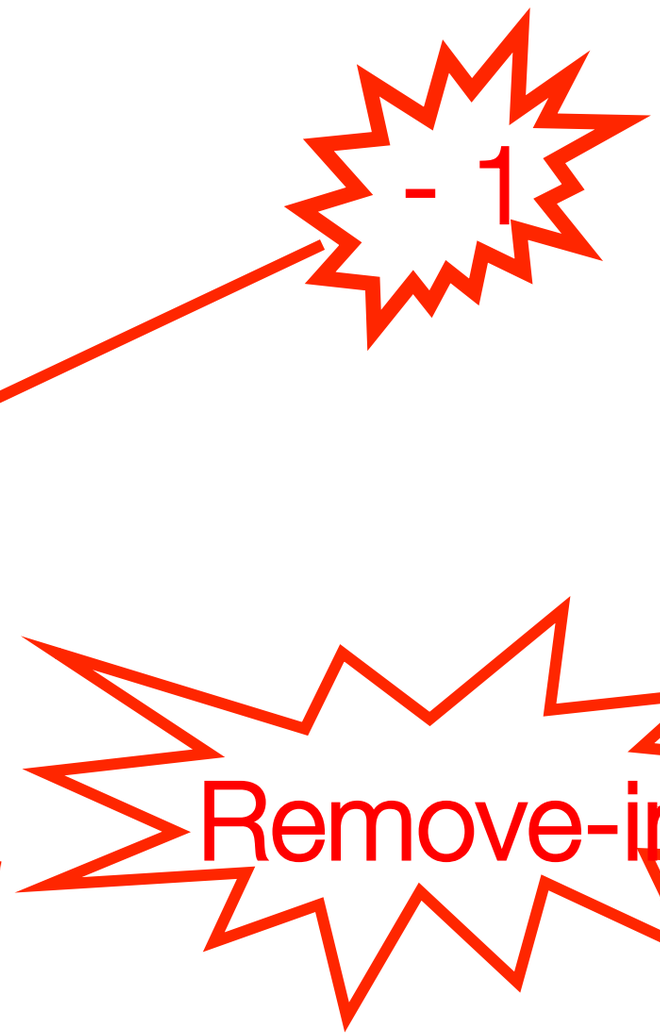
- Qualifie un ensemble de cas de test
  - évalue la proportion de fautes que les test détectent
  - fondé sur l'injection de fautes
- L'évaluation de la qualité des cas de test est importante pour évaluer la confiance dans le programme

Analyse de mutation

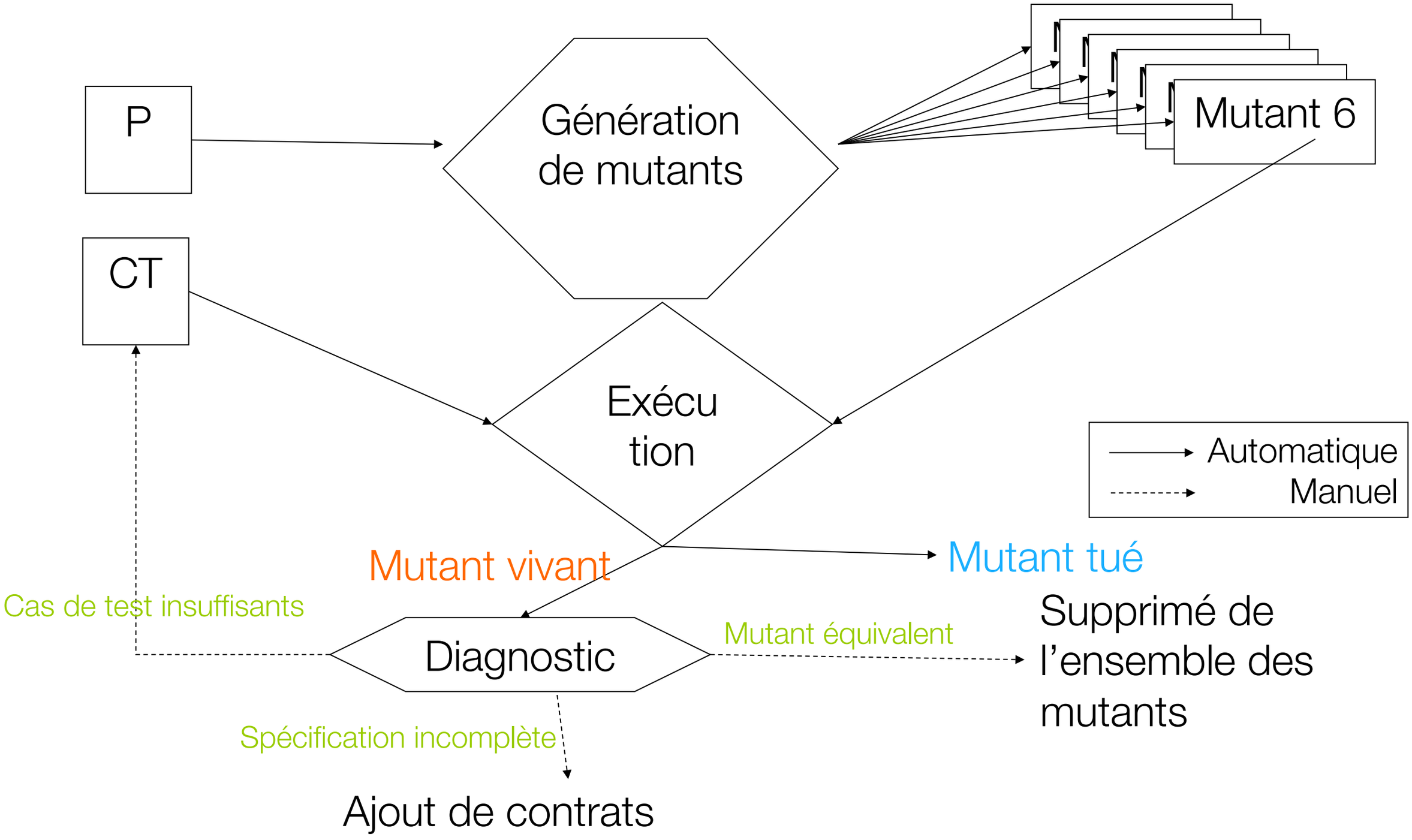
- Le choix des fautes injectées est très important
  - les fautes sont modélisées par des *opérateurs de mutation*
- Mutant = programme initial avec une faute injectée
- Deux fonctions d'oracle
  - Différence de traces entre le programme initial et le mutant
  - Contrats exécutables

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


P
CT
Génération de mutants
Mutant 6
Exécu tion
Automatique
Manuel
Mutant vivant
Mutant tué
Cas de test insuffisants
Diagnostic
Mutant équivalent
Supprimé de l'ensemble des mutants
Spécification incomplète
Ajout de contrats

Mutants équivalents

```
int Min (int i, intj){
  int minval = i;
  if (j<i) then minval = j;
  return minval
}
```

```
int Min (int i, intj){
  int minval = i;
  if (j<minval) then minval = j;
  return minval
}
```

- Mutant équivalent est fonctionnellement équivalent à l'original
  - aucun cas de test ne permet de le tuer

Mutants vivants

- Si un mutant n'est pas tué?
  - cas de test insuffisants => ajouter des cas de test
  - mutant équivalent => supprimer le mutant

Score de mutation

- Q(Ci) = score de mutation de Ci = d_i/ m_i
  - d_i = nombre de mutants tués
  - m_i = nombre de mutants non équivalents
- *Attention* Q(Ci)=100% not=> *bug free*
- Qualité d'un système S fait de composants d_i
  - $Q(S) = \Sigma d_i / \Sigma m_i$

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.

Exemple

```
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
      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);
```

- 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 +

O1: > 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
   begin
```

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
      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 <=

```
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;                    if x<=0 then
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);
```

O4: <= 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
      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);
```

O5: = 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
      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);
```

O6: l'opérateur binaire + 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
    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);
```

O7: l'opérateur unaire - 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
      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);
```

signe = 1

x:=x

Exemple

```
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
     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);
```

if x<=0 then

signe = 1
x:=x

if y<=0 then

signe=-signe
y:=-y

while x>y

if x<=y

z=z-1

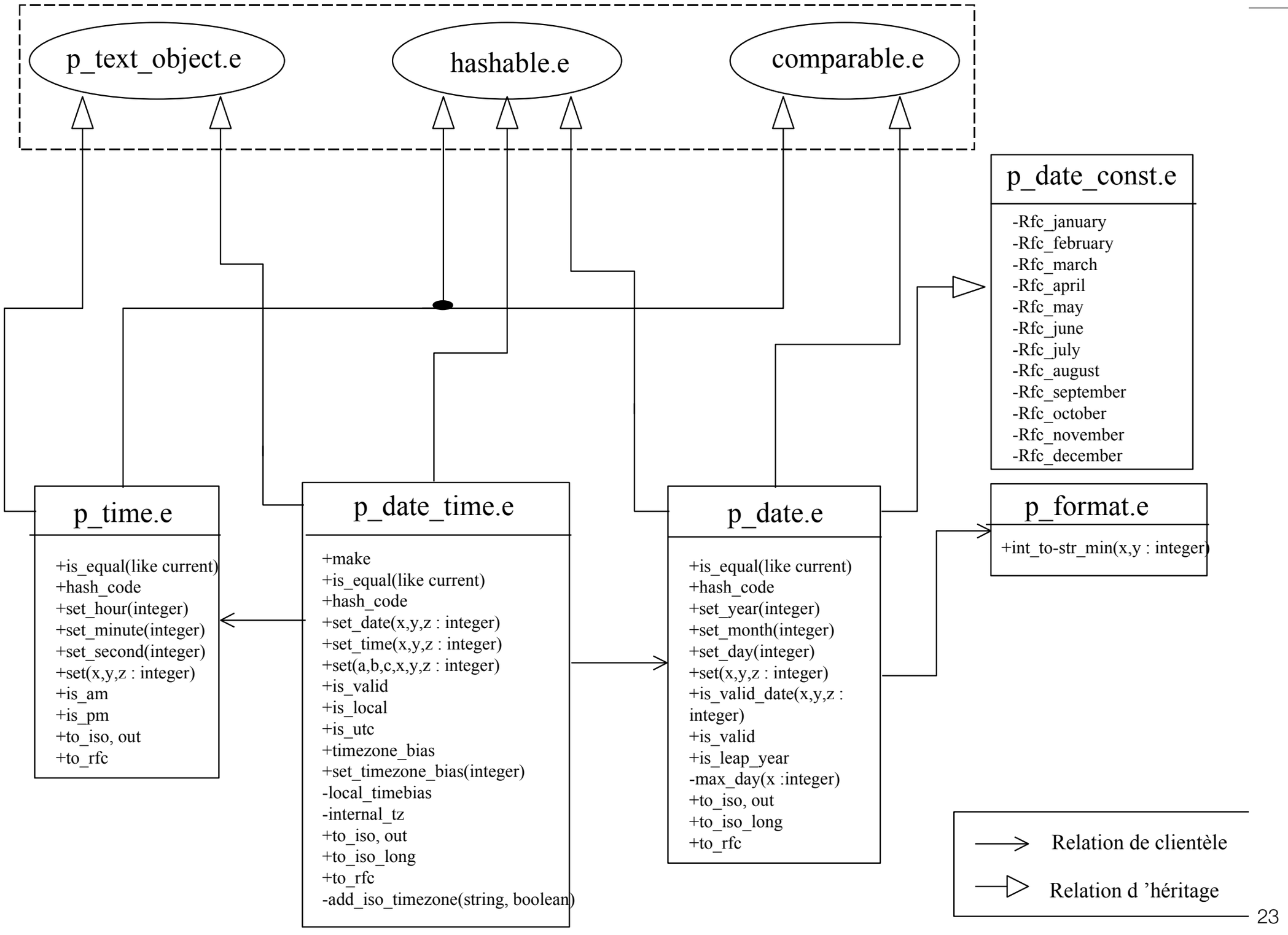
Test par mutation

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

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 mêmes mutants

Exemple


p_text_object.e
hashable.e
comparable.e
p_date_const.e
-Rfc_january
-Rfc_february
-Rfc_march
-Rfc_april
-Rfc_may
-Rfc_june
-Rfc_july
-Rfc_august
-Rfc_september
-Rfc_october
-Rfc_november
-Rfc_december
p_time.e
+is_equal(like current)
+hash_code
+set_hour(integer)
+set_minute(integer)
+set_second(integer)
+set(x,y,z : integer)
+is_am
+is_pm
+to_iso, out
+to_rfc
p_date_time.e
+make
+is_equal(like current)
+hash_code
+set_date(x,y,z : integer)
+set_time(x,y,z : integer)
+set(a,b,c,x,y,z : integer)
+is_valid
+is_local
+is_utc
+timezone_bias
+set_timezone_bias(integer)
-local_timebias
-internal_tz
+to_iso, out
+to_iso_long
+to_rfc
-add_iso_timezone(string, boolean)
p_date.e
+is_equal(like current)
+hash_code
+set_year(integer)
+set_month(integer)
+set_day(integer)
+set(x,y,z : integer)
+is_valid_date(x,y,z : integer)
+is_valid
+is_leap_year
-max_day(x :integer)
+to_iso, out
+to_iso_long
+to_rfc
p_format.e
+int_to-str_min(x,y : integer)
Relation de clientèle
Relation d ’héritage

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 reduit	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 OO

- Pour évaluer des cas 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

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
- Quelques outils
  - Exemple: Pitest, MuJava