VALIDATION & VERIFICATION

MUTATION ANALYSIS

MASTER 1 ICE, 2017-2018

BENOIT COMBEMALE
PROFESSOR, UNIV. TOULOUSE, FRANCE

HTTP://COMBEMALE.FR
BENOIT.COMBEMALE@IRIT.FR
@BCOMBEMALE

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

- Si les cas de test peuvent détecter des fautes mises intentionnellement, ils peuvent détecter des fautes réelles

Analyse de mutation

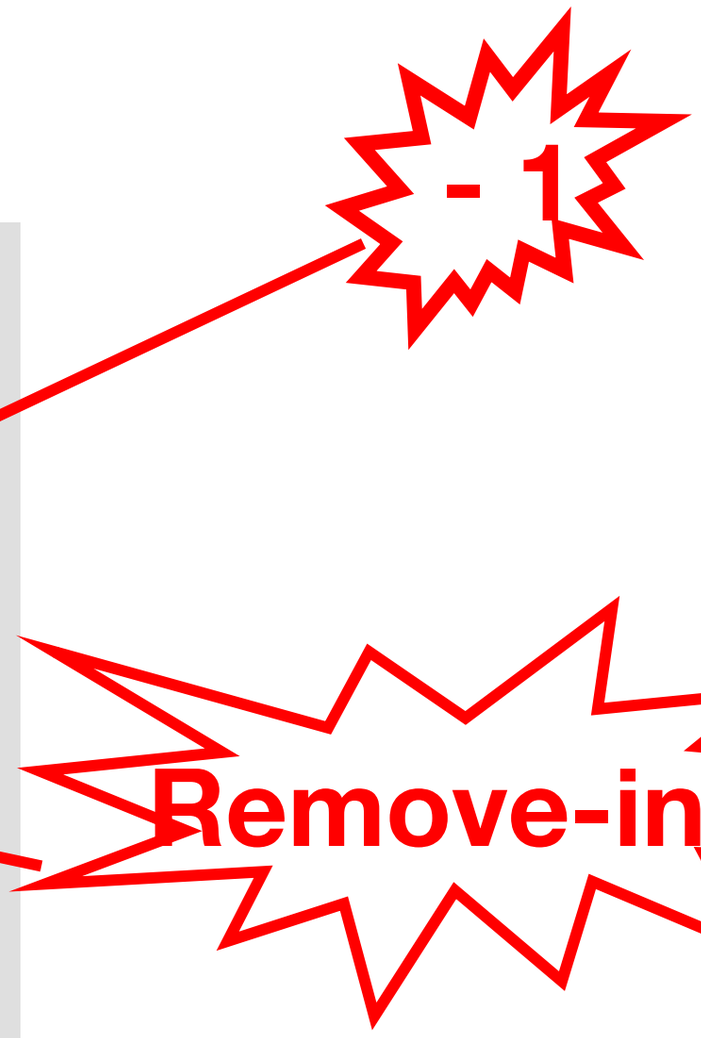
- Qualifie un ensemble de cas de test
  - évalue la proportion de fautes que les tests détectent
  - fondé sur l'injection de fautes
- L'évaluation de la qualité des cas de test est importante pour évaluer la confiance dans le programme

Analyse de mutation

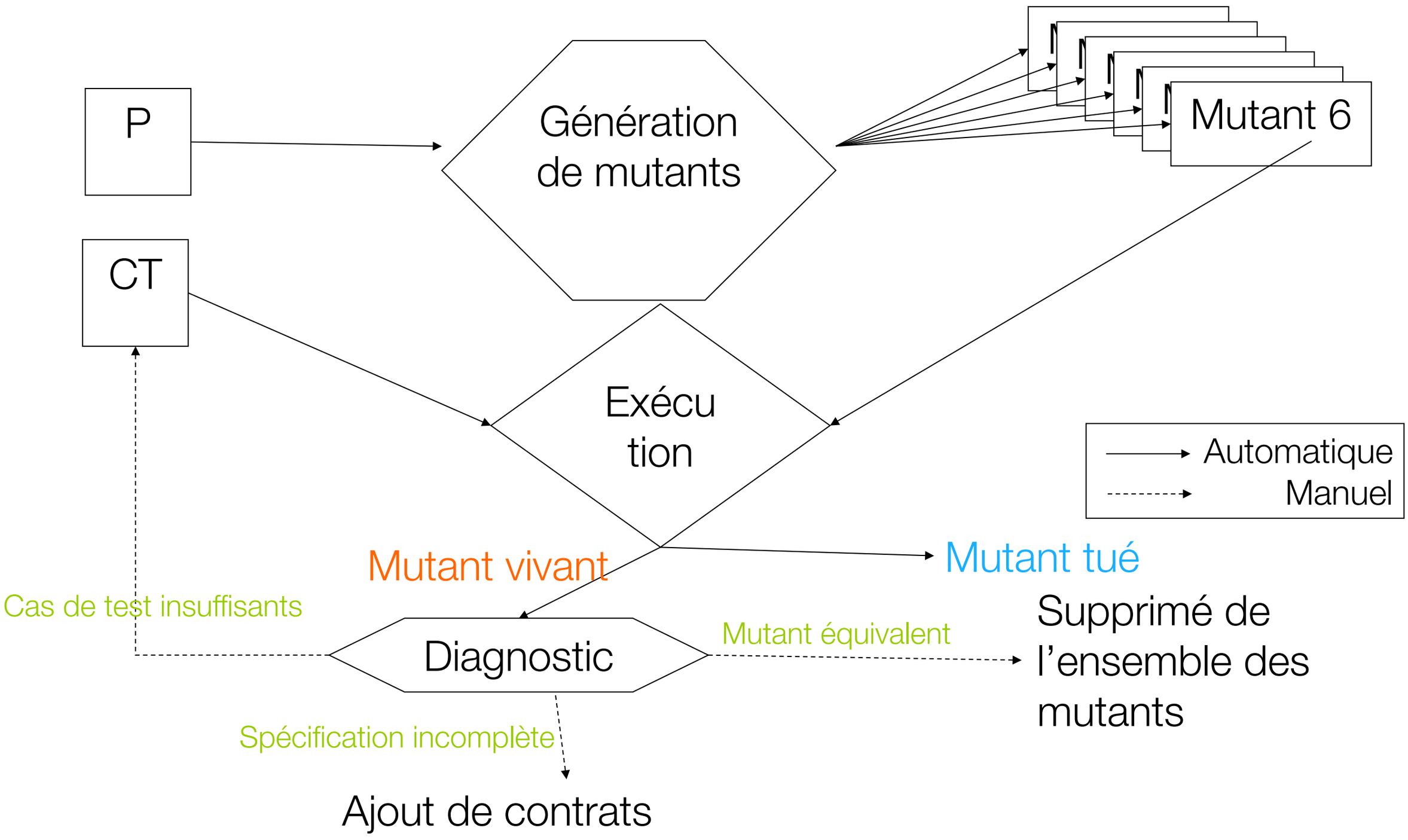
- Le choix des fautes injectées est très important
  - les fautes sont modélisées par des *opérateurs de mutation*
- Mutant = programme initial avec une faute injectée
- Deux fonctions d'oracle
  - Différence de traces entre le programme initial et le mutant
  - Contrats exécutables

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.

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

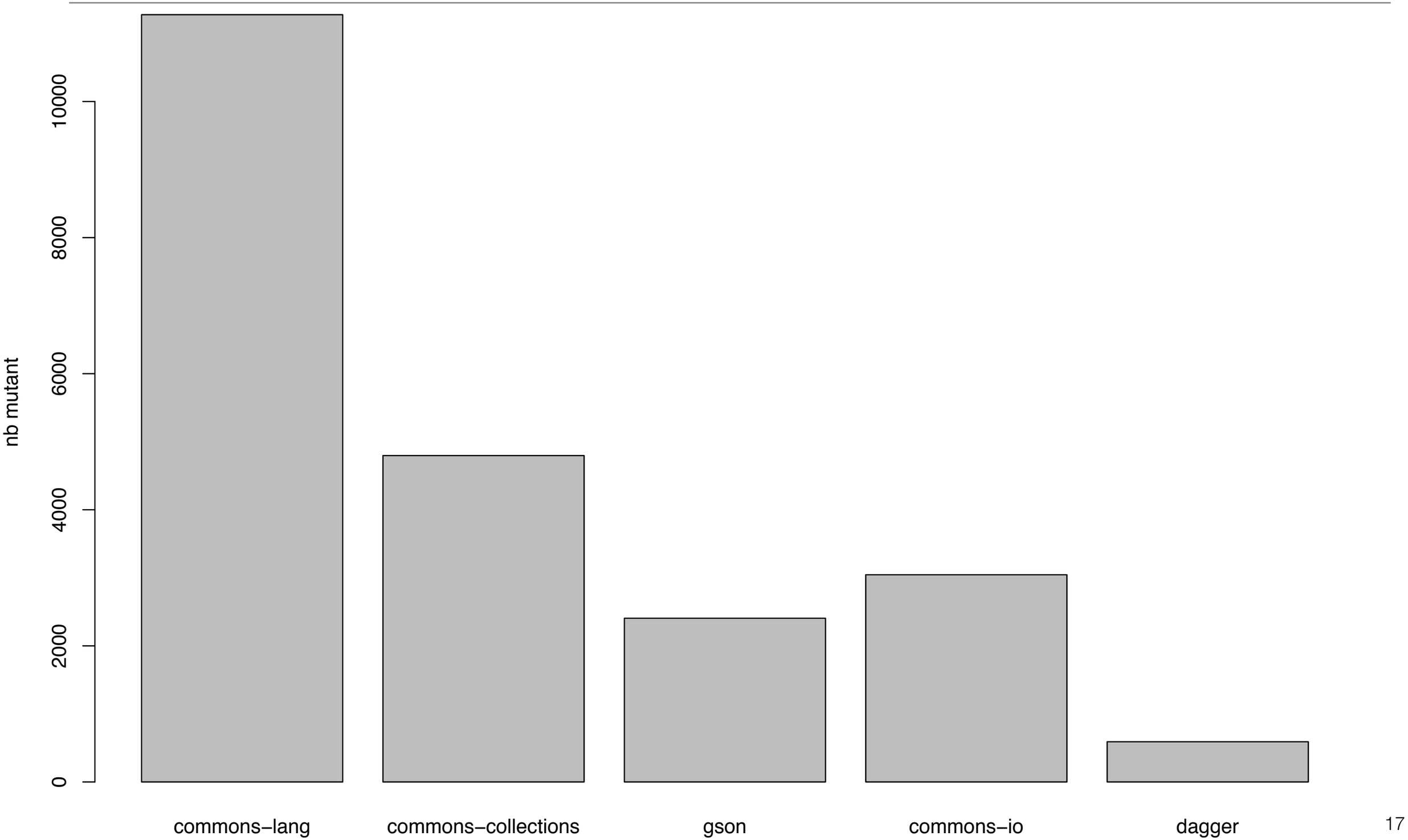
Opérateurs OO(3)

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

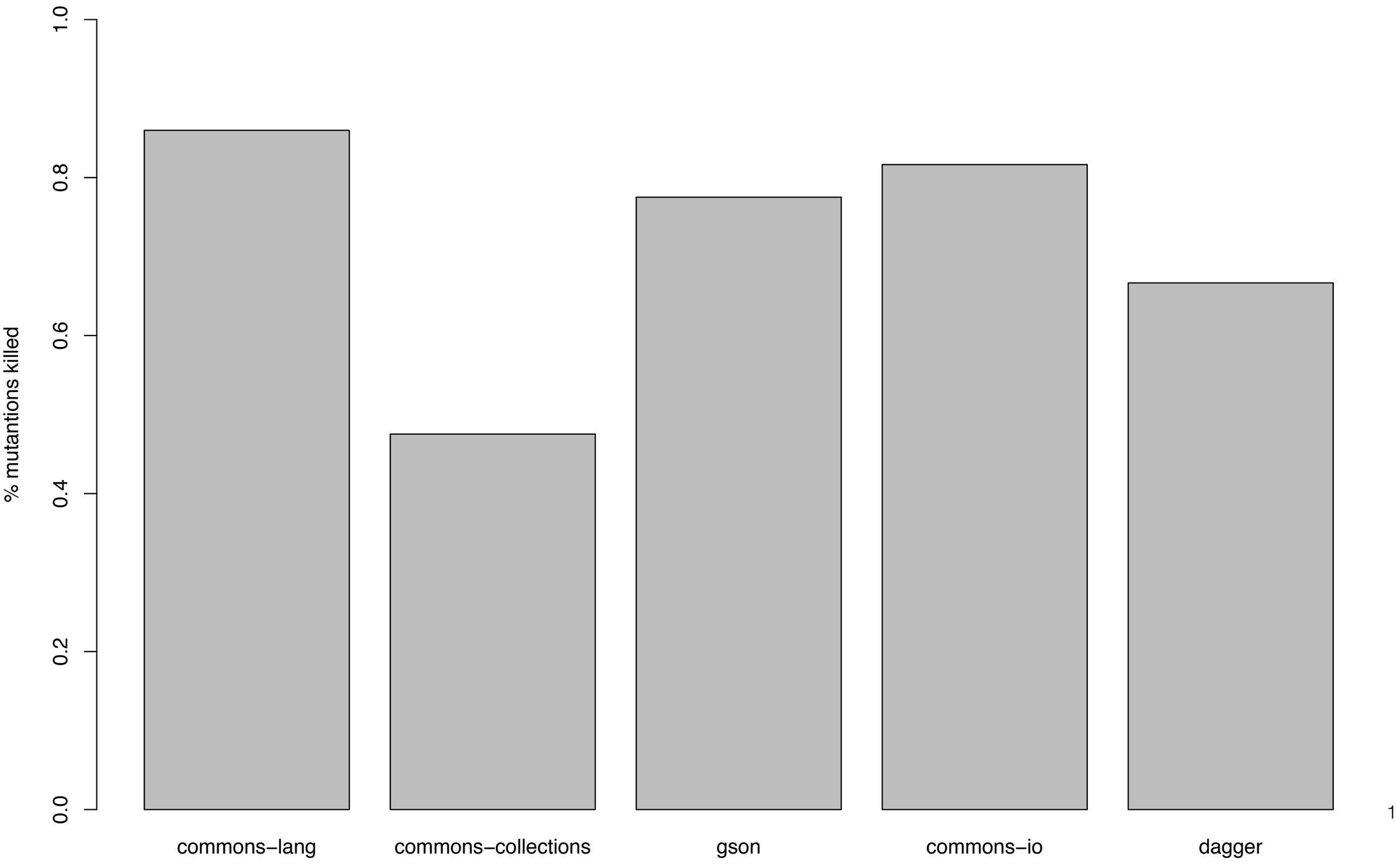
Five examples

	#classes	#statements	#test cases	coverage
lang	132	8442	2352	94%
collection	286	6780	13677	84%
gson	66	2377	951	79%
io	103	2573	962	87%
dagger	23	4984	128	89%

Number of mutants with PIT



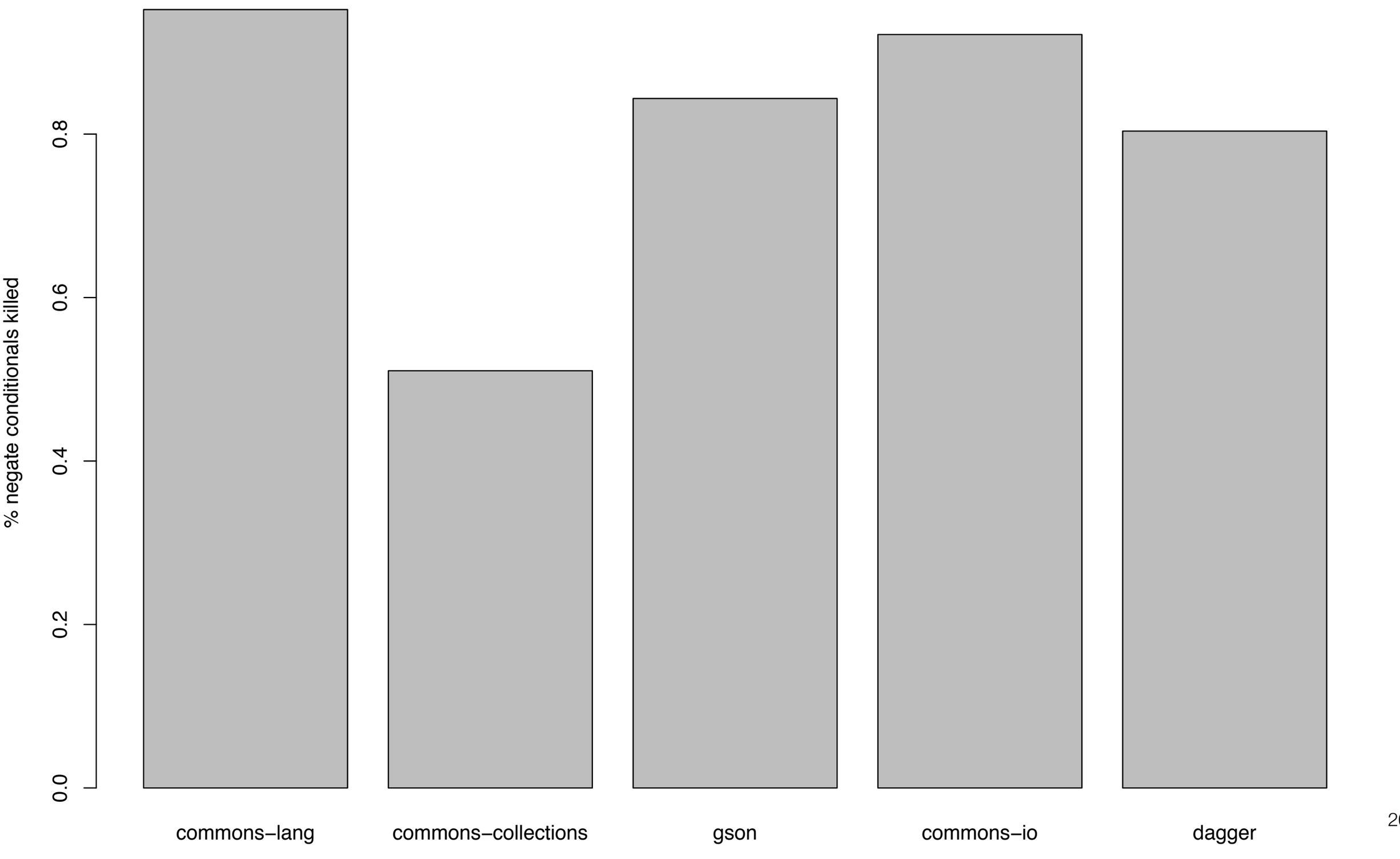
Mutation score



Negate condition

original	**mutant**
==	!=
!=	==
<=	>
>=	<
<	>=
>	<=

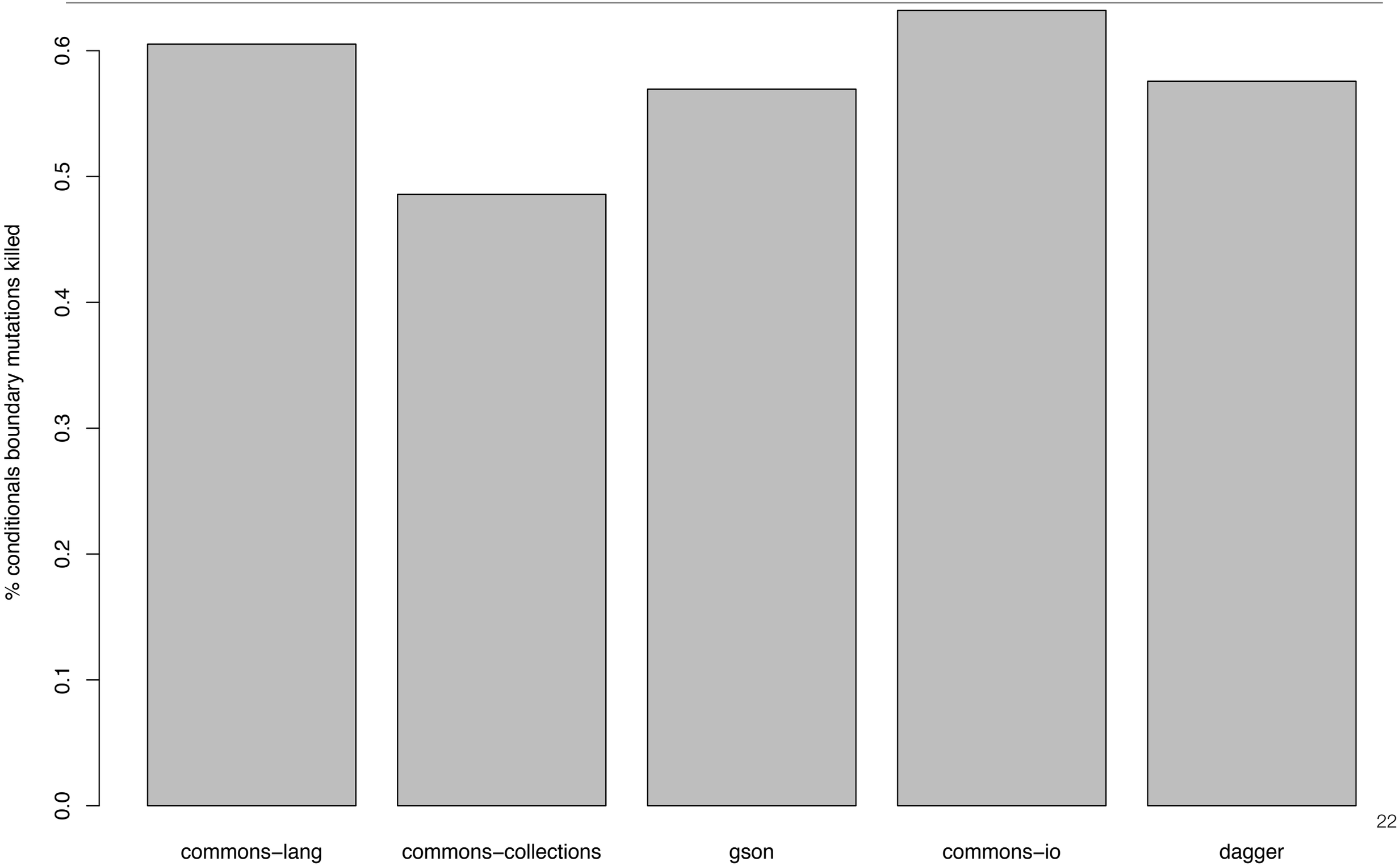
Mutation score (neg. cond)



Conditionals Boundary Mutator

original	mutant
<	<=
<=	<
>	>=
>=	>

Mutation score (neg. cond. boundaries)



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

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