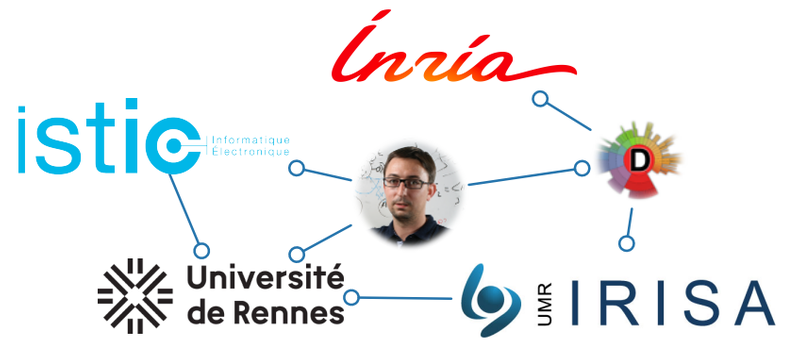
VALIDATION & VERIFICATION

MUTATION ANALYSIS

UNIVERSITY OF RENNES, ISTIC & ESIR, 2024-2025

BENOIT COMBEMALE
FULL PROFESSOR, UNIVERSITY OF RENNES, FRANCE

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



A typical test case

```
public class CharSetTest {
  ...
  @Test
  public void testConstructor () {
    CharSet set = CharSet.getInstance("a");
    CharRange[] array = set.getCharRanges();
    assertEquals("[a]", set.toString());
    assertEquals(1, array.length);
    assertEquals("a", array[0].toString());
  }
  ...
}
```

Initializes the program
Triggers specific behavior

Specifies the expected effects

Test cases are expected to:

- Cover main requirements
- Stress the application
- Prevent regressions
- Reveal bugs

Test case quality is impacted by:

- How well the input has been chosen
- How strong the assertions are

How can we be sure that a test suite is adequate enough to find bugs?

Code coverage

- Most used approach
- Relatively “cheap” to compute
- Multiple effective implementations
  - (JaCoCo,OpenClover,Cobertura)
- IDE integration out-of-the-box and via plugins
- Supported in Continuous Integration Servers and Github

Example

```
long fact(int n) {
  if(n==0) {
    return 1;
  }
  long result = 1;
  for(int i = 2; i <= n; i++) {
   result = result * i;
  }
  return result;
}
```

```
@Test
factorialWith5Test() {
  long obs = fact(5);
  assertTrue(5 < obs);
}
```

```
@Test
factorialWith0Test() {
  assertEqual(1, fact(0));
}
```

Code coverage

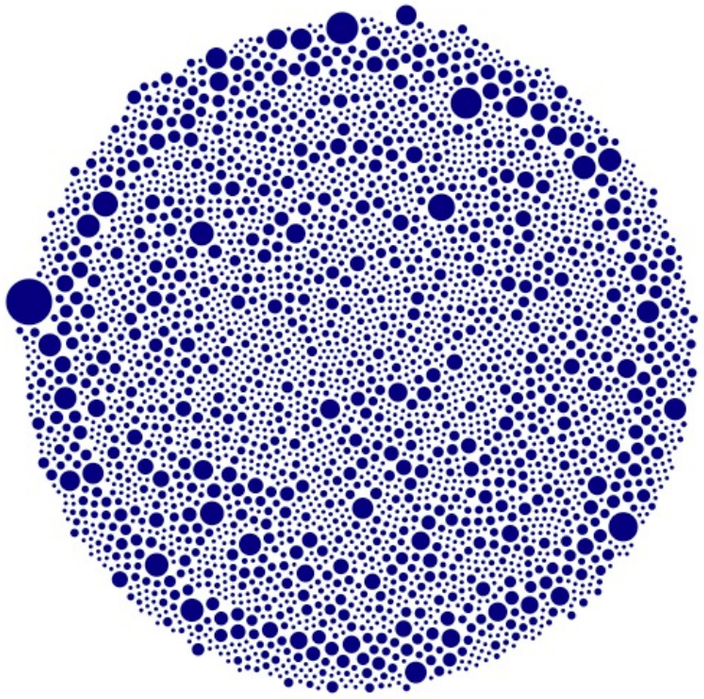
- Useful to spot not tested code
- High coverage ⇏ Effective test suite
- Well tested projects have high coverage
- The opposite may not be true! (e. g. No assertions)
- 100% coverage is hard to achieve and may be impractical

A real example…

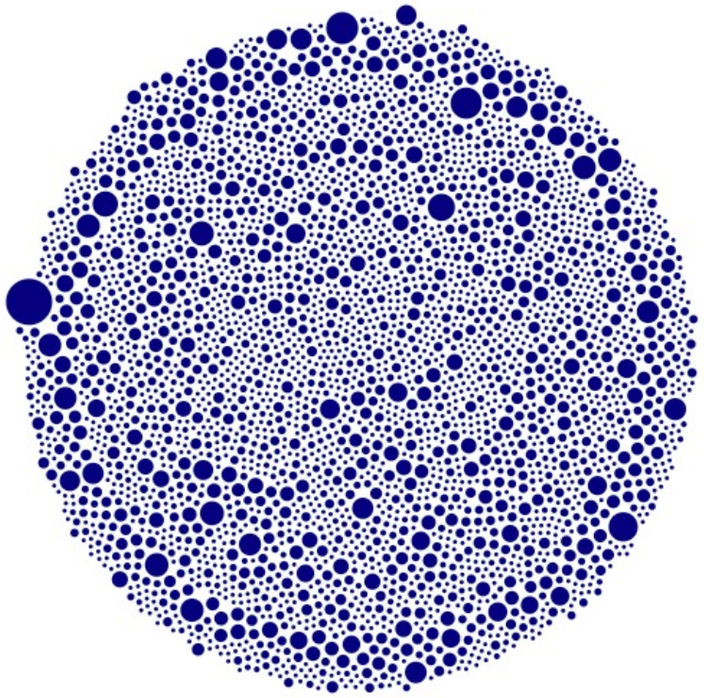
Apache Commons Collections

- Issue tracker 
- Continuous Integration  Jenkins
- Automated Static Analysis  `CheckStyles` 
- Code coverage monitoring `Cobertura`

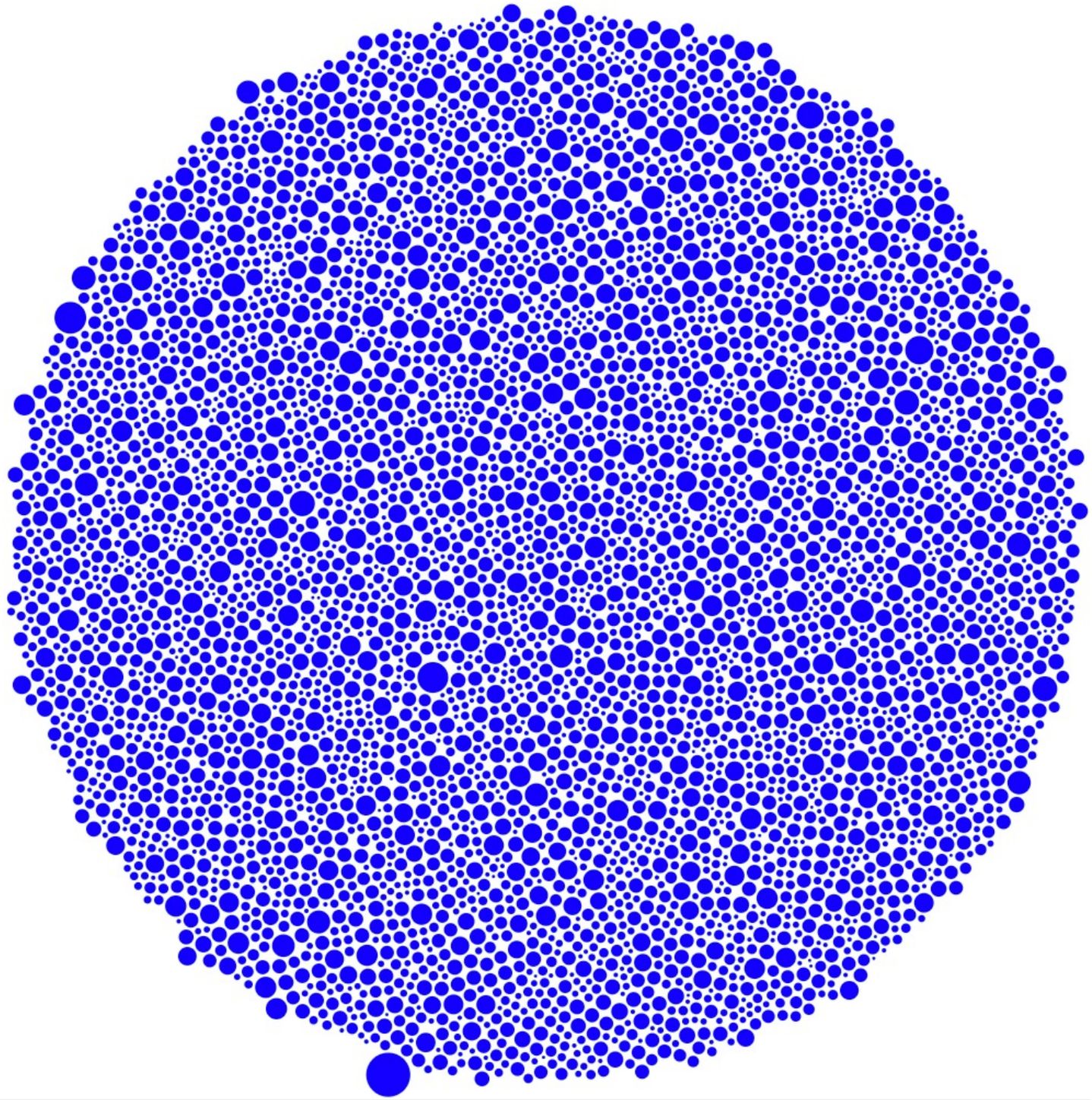
3,552 methods



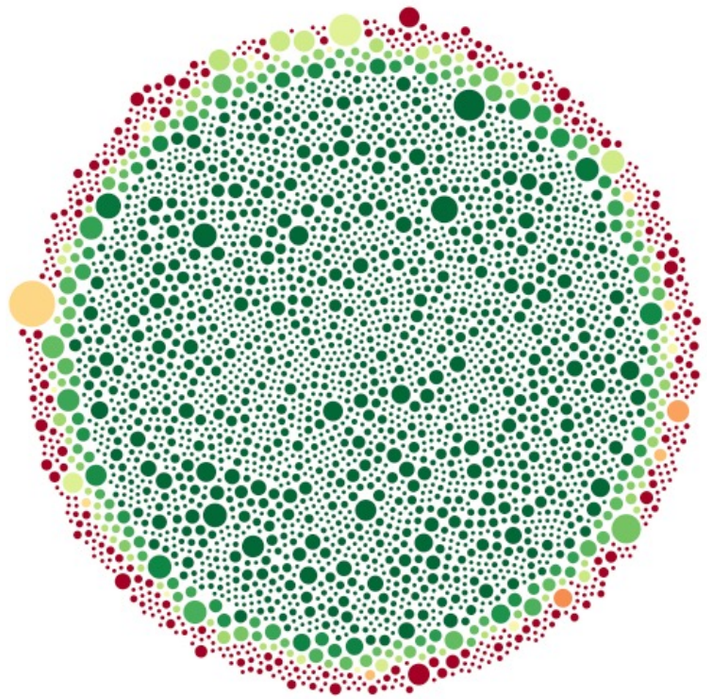
3,552 methods

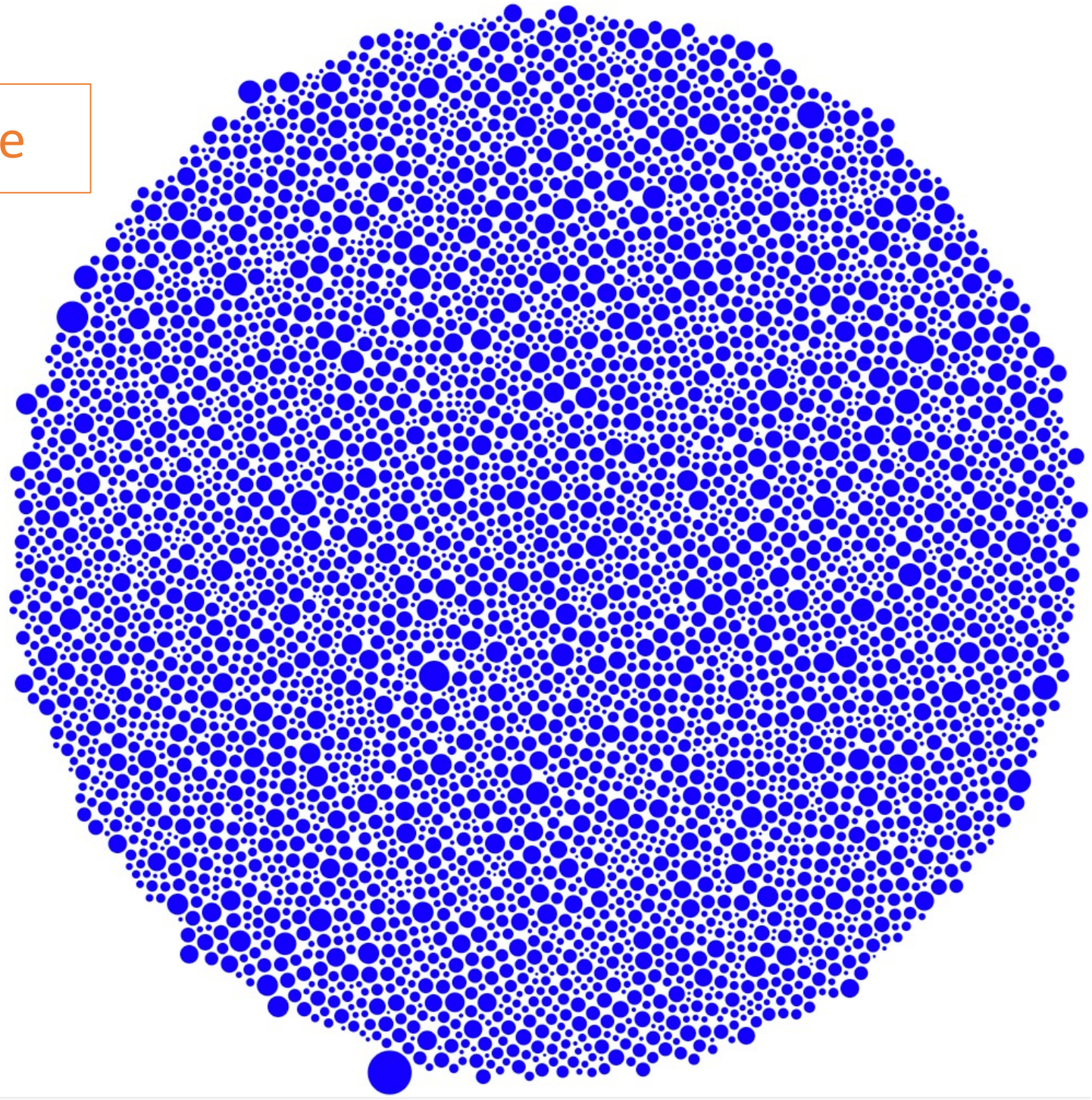


13,677 tests

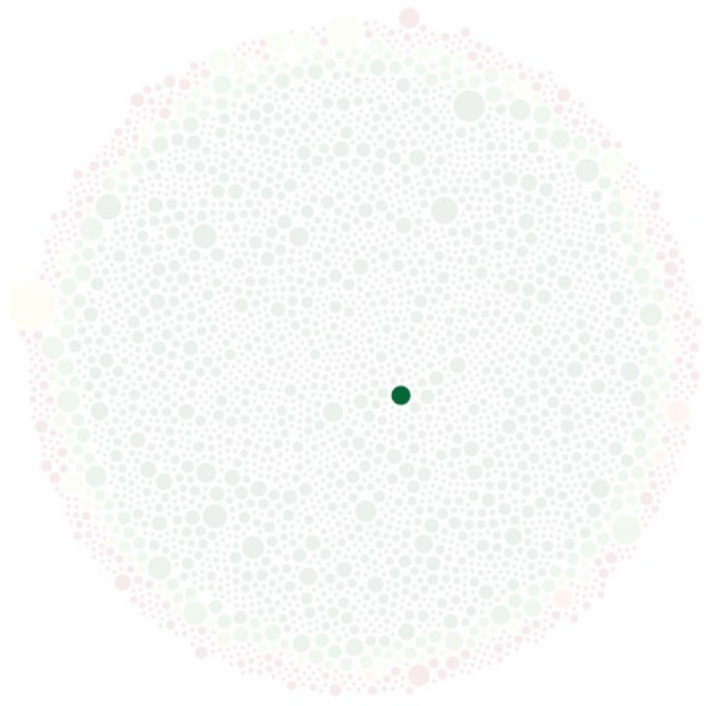


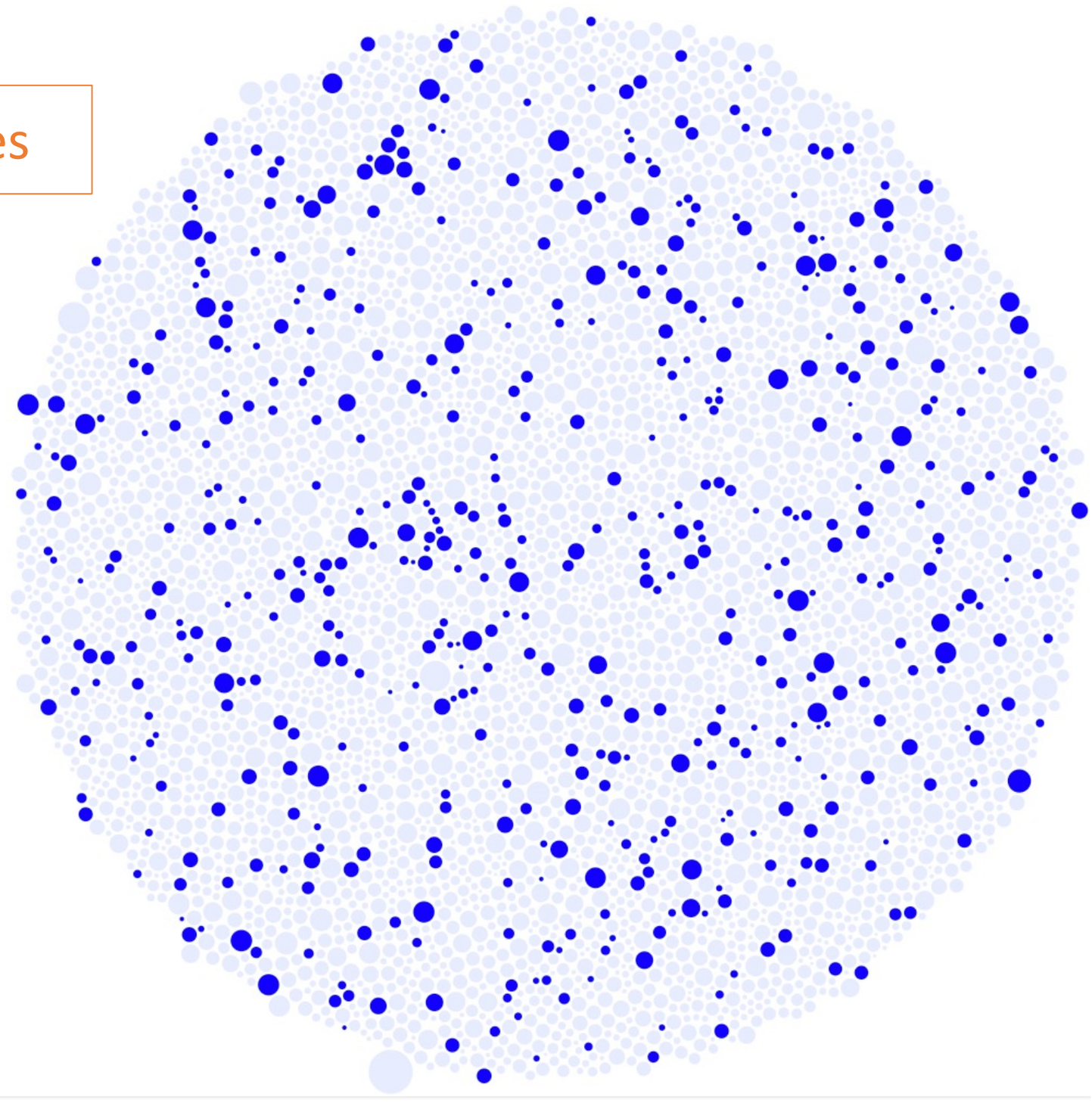
85% coverage





926 test cases





```java
public class AbstractHashedMap<K, V> extends AbstractMap<K, V> implements IterableMap<K, V> {
  protected void ensureCapacity(final int newCapacity) {
    final int oldCapacity = data.length;
    if (newCapacity <= oldCapacity) {
      return;
    }
    if (size == 0) {
      threshold = calculateThreshold(newCapacity, loadFactor);
      data = new HashEntry[newCapacity];
    } else {
      final HashEntry<K, V> oldEntries[] = data;
      final HashEntry<K, V> newEntries[] = new HashEntry[newCapacity];

      modCount++;
      for (int i = oldCapacity - 1; i >= 0; i--) {
        HashEntry<K, V> entry = oldEntries[i];
        if (entry != null) {
          oldEntries[i] = null;
          do {
            final HashEntry<K, V> next = entry.next;
            final int index = hashIndex(entry.hashCode, newCapacity);
            entry.next = newEntries[index];
            newEntries[index] = entry;
            entry = next;
          } while (entry != null);
        }
      }
      threshold = calculateThreshold(newCapacity, loadFactor);
      data = newEntries;
    }
  }
}
```

926 test cases

Executed 11,593 times

If the code is removed
no test fails

Mutation Analysis

- Proposed in 1970
- *Programmers create programs close to being correct*
- *A test suite that detects all simple faults also detects the complex ones*

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

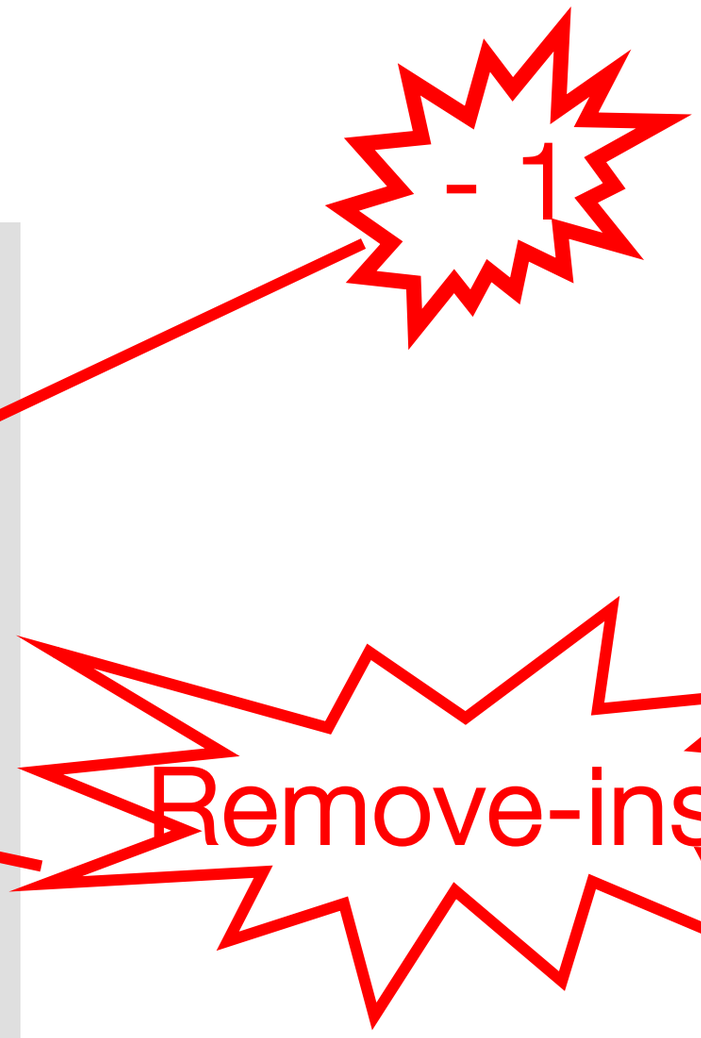
- Qualifie un ensemble de cas de test
  - évalue la proportion de fautes que les tests détectent
  - fondé sur l'injection de fautes
- L'évaluation de la qualité des cas de test est importante pour évaluer la confiance dans le programme

Analyse de mutation

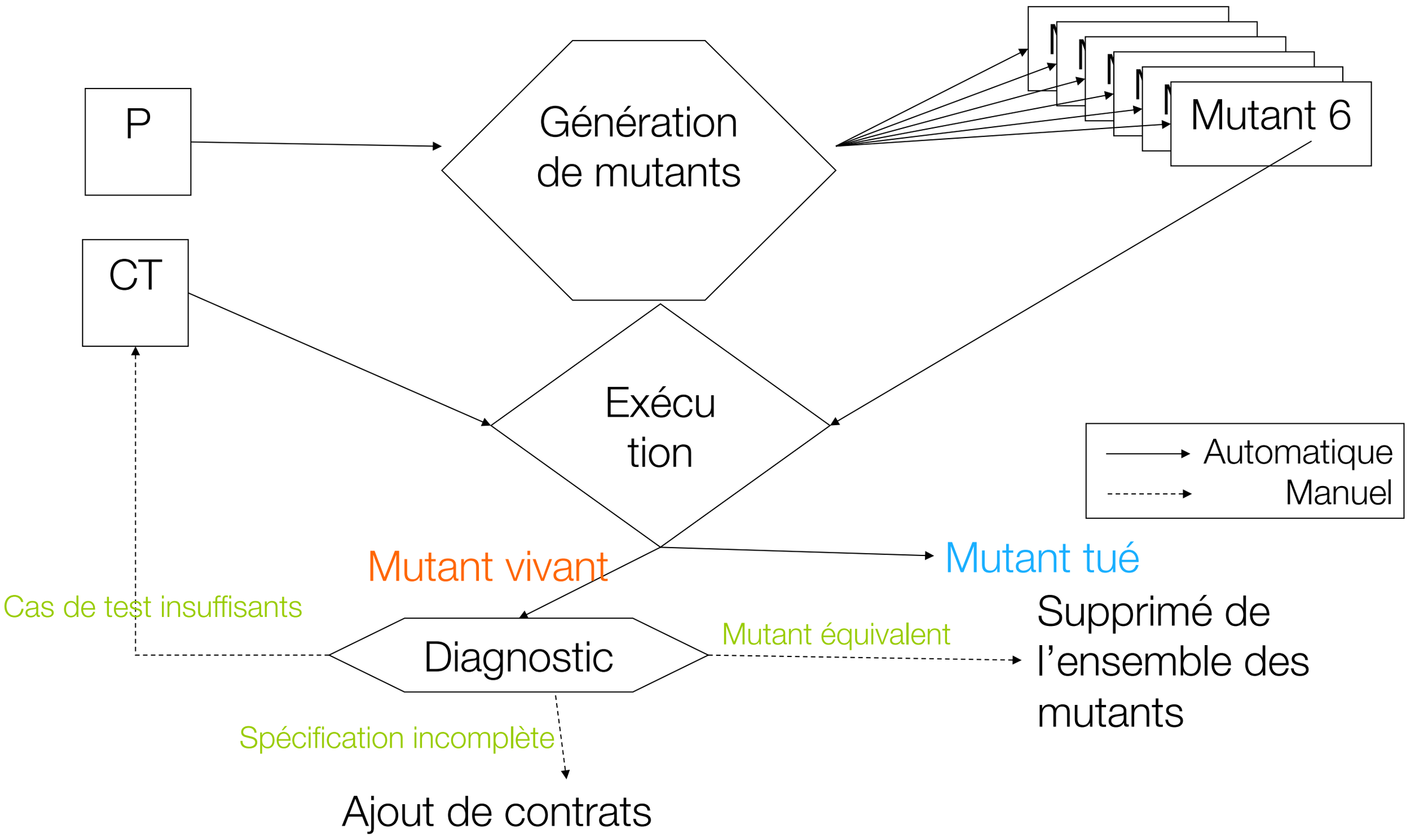
- Le choix des fautes injectées est très important
  - les fautes sont modélisées par des *opérateurs de mutation*
- Mutant = programme initial avec une faute injectée
- Deux fonctions d'oracle
  - Différence de traces entre le programme initial et le mutant
  - Contrats exécutables

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

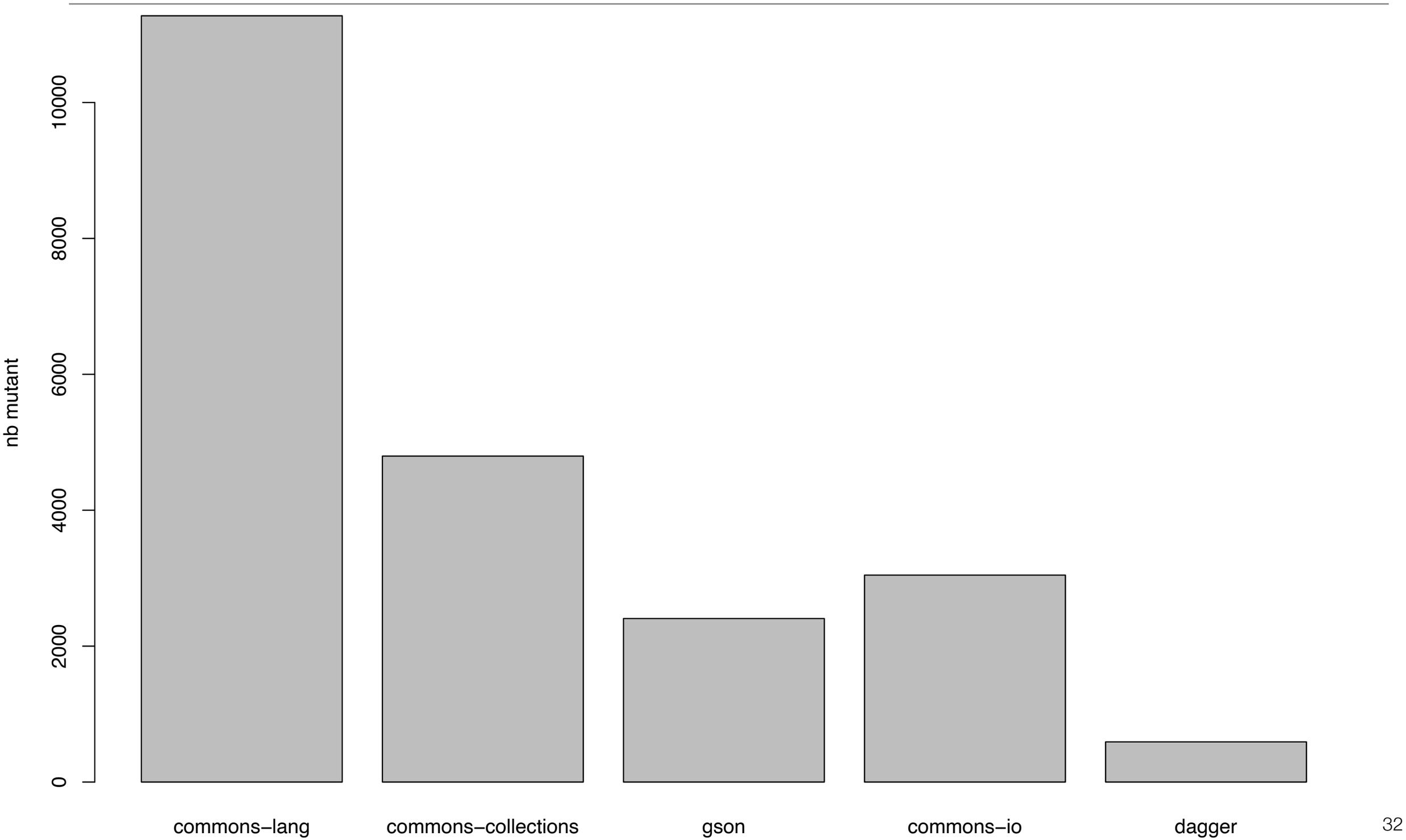
Opérateurs OO(3)

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

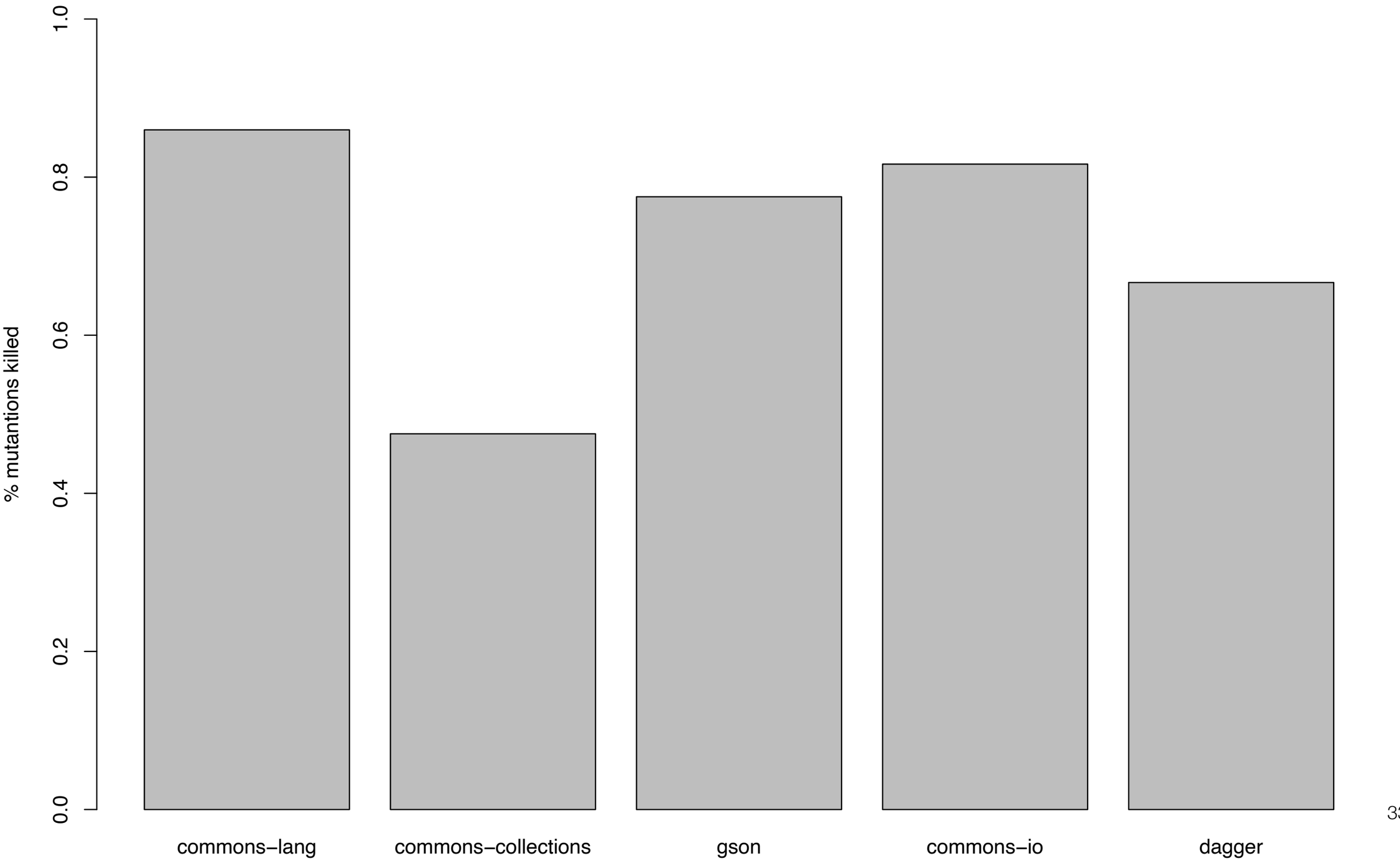
Five examples

	#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



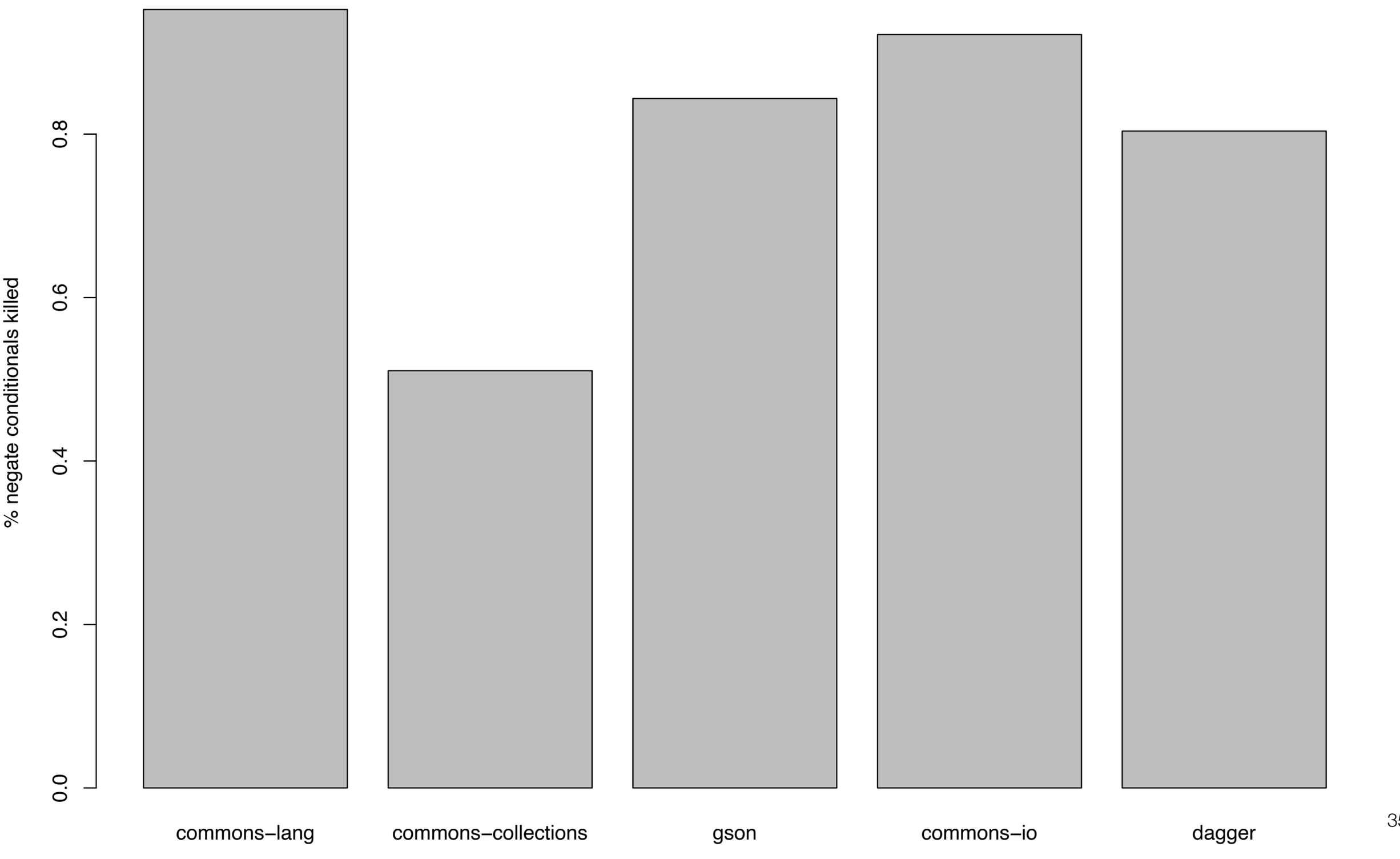
Mutation score



Negate condition

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

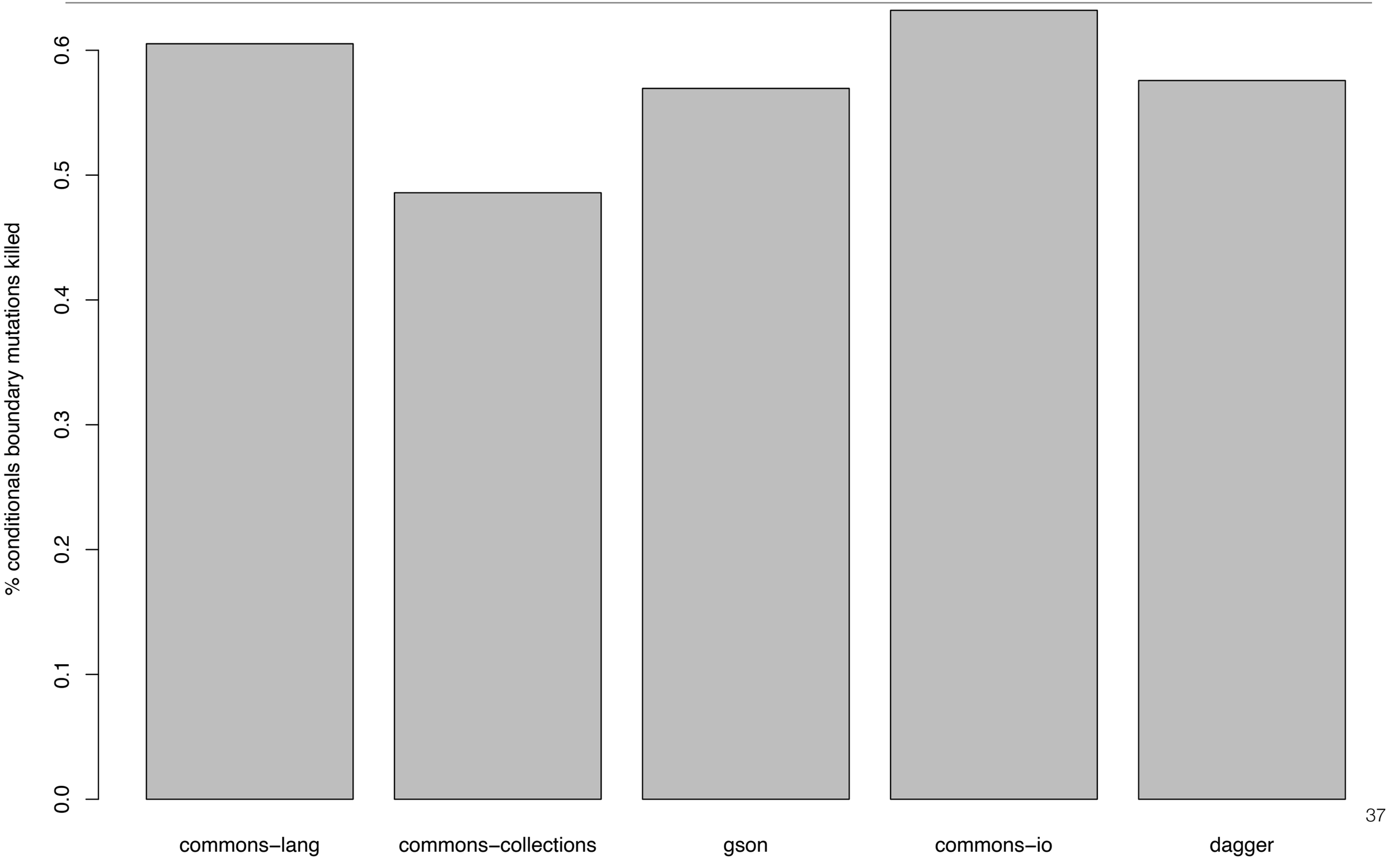
Mutation score (neg. cond)



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

Some available tools for Java

- Javalanche
  - https://www.st.cs.uni-saarland.de/mutation/
  - Bytecode manipulation
- Major
  - http://mutation-testing.org/
  - AST manipulation
  - Extensible
- PIT
  - http://pitest.org
  - Bytecode manipulation
  - Out favorite choice ☺

PIT or PITest

- Open source, in active development and production ready
- Integrates with major build systems
- State of the art mutation testing
- Extensible via plugins
- Concurrent execution
- Test selection

Current limitations of mutation testing

- Few integrated tools
- Expensive computation
- Huge number of mutants
- Presence of equivalent mutants

Equivalent mutants

```
class Foo {
  int min;
  public void bar(int i) {
    if (i < min) {
      min = i;
    }
   System.out.println("" + min);
  }
}
```

```
class Foo {
  int min;
  public void bar(int i) {
    if (i <= min) {
      min = i;
    }
   System.out.println("" + min);
  }
}
```

Henry Coles, *Making Mutants Work For You or how I learned to stop worrying and love equivalent mutants*, Paris JUG October 24th 2018

Equivalent mutants

- Harmful for quality metrics
- May provide helpful information for developers
  - May signal code duplication or redundancy → Refactor
  - May signal buddy tests → Fix

Henry Coles, *Making Mutants Work For You or how I learned to stop worrying and love equivalent mutants*, Paris JUG October 24th 2018

Equivalent mutants

```
class Foo {
  int min;
  public void bar(int i) {
    if (i < min) {
      min = i;
    }
   System.out.println("" + min);
  }
}
```

```
class Foo {
  int min;
  public void bar(int i) {
    if (i <= min) {
      min = i;
    }
   System.out.println("" + min);
  }
}
```

Henry Coles, *Making Mutants Work For You or how I learned to stop worrying and love equivalent mutants*, Paris JUG October 24th 2018

Equivalent mutants

```
class Foo {
  int min;
  public void bar(int i) {
    min = Math.min(i, min);
    System.out.println("" + min);
  }
}
```

The code is more expressive

Henry Coles, *Making Mutants Work For You or how I learned to stop worrying and love equivalent mutants*, Paris JUG October 24th 2018

Equivalent mutants

```
public void someLogic(int i) {
  if (i <= 100) {
    throw new IllegalArgumentException();
  }
  if (i > 100) {
    doSomething();
  }
}
```

```
public void someLogic(int i) {
  if (i <= 100) {
    throw new IllegalArgumentException();
  }
  if (i >= 100) {
    doSomething();
  }
}
```

Code is redundant

Henry Coles, *Making Mutants Work For You or how I learned to stop worrying and love equivalent mutants*, Paris JUG October 24th 2018

Equivalent mutants

```
public void someLogic(int i) {
  if (i <= 100) {
    throw new IllegalArgumentException();
  }
  doSomething();
}
```

Refactor the code

Henry Coles, *Making Mutants Work For You or how I learned to stop worrying and love equivalent mutants*, Paris JUG October 24th 2018

Overcoming the limitations

- Do fewer
  - Reduce the number of mutants
  - Sample the mutants
  - Use less operators
- Do smarter
  - Distribute the analysis
  - Cloud infrastructure
- Do faster
  - Improve efficiency

Extreme mutation

Proposed in 2016

R. Niedermayr, E. Juergens, and S. Wagner

Will my tests tell me if I break this code?

Proceedings of the International Workshop on Continuous Software Evolution and

Delivery, 2016, pp. 23–29.

Extreme mutation

```
public void setValue(int x) {
  // if(a + x < 10)
        a += x;
  }
```

```
public int fact(int x) {
  int result = 0;
  return 42;
} for(int i=1; i <= x; i++) {
    result *= i;
  }
  return result;
}
```

Example

```
long fact(int n) {
  if(n==0) {
    return 1;
  }
  long result = 1;
  for(int i = 2; i <= n; i++) {
   result = result * i;
  }
  return result;
}
```

```
long fact(int n) {
  return 1;
}
```

```
long fact(int n) {
  return 0;
}
```

Extreme mutation

- Creates less mutants → Faster analysis
- Works at the method level → Easier to understand
- Many equivalent mutants can be detected → More reliable

Pseudo-tested methods

- Methods executed by the test suite
- No extreme mutation is detected
- Found in well tested projects

A pseudo-tested method

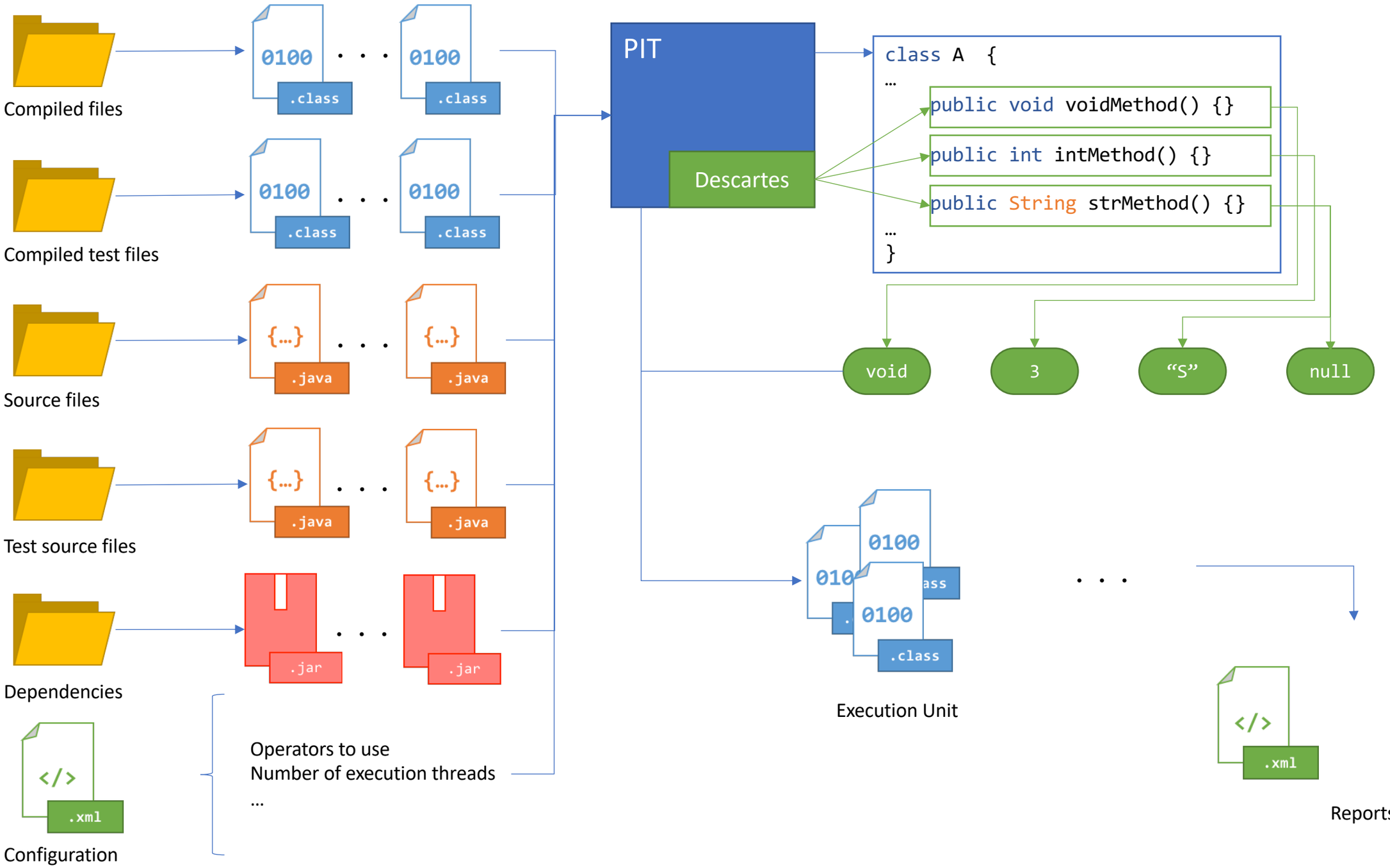
```
class VList {
  private List elements;
  private int version;
  public void add(Object item) {
    elements.add(item);
    incrementVersion();
  }
  public int size () {
    return elements.size();
  }

  private void incrementVersion () {
    version ++;
  }
}
```

```
class VListTest
  @Test
  public void testAdd (){
    VList l = new VList();
    l.add(1);
    assertEquals(l.size(), 1);
  }
}
```

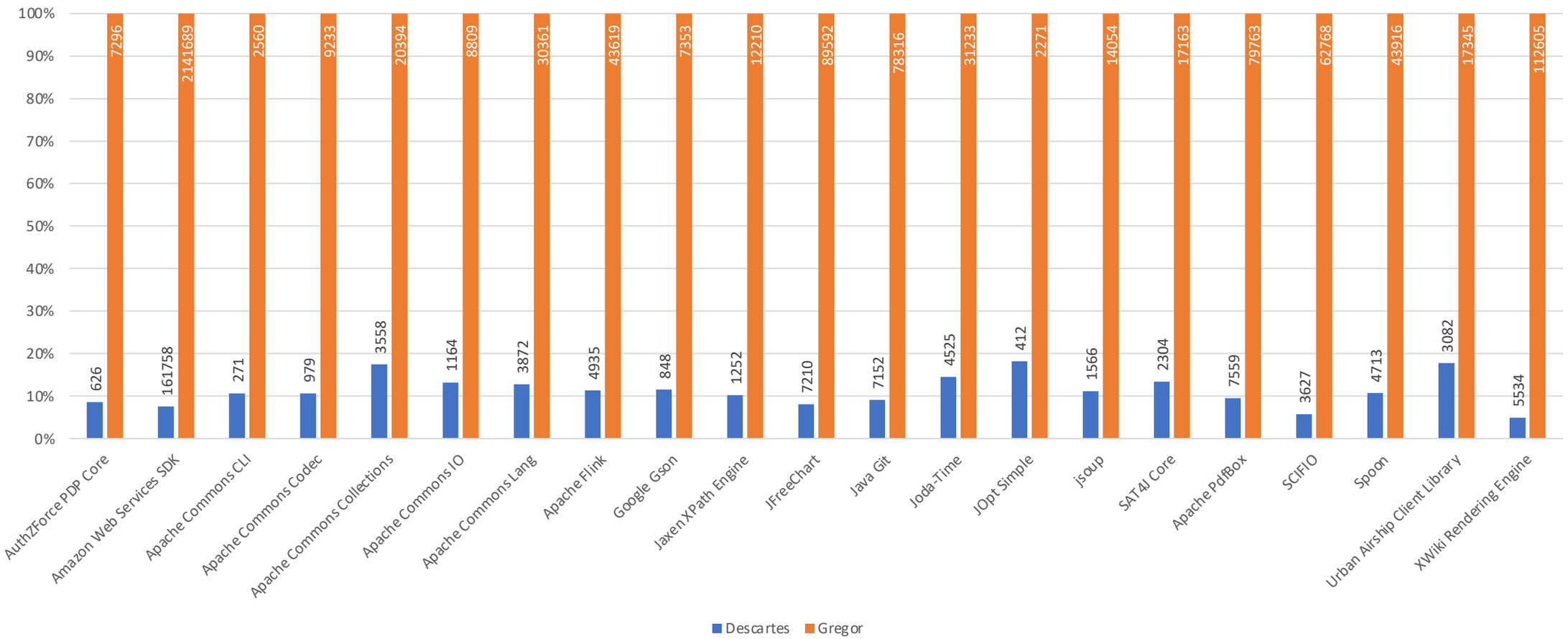
Pseudo-tested
Testability issues

Descartes *I mutate therefore I am*

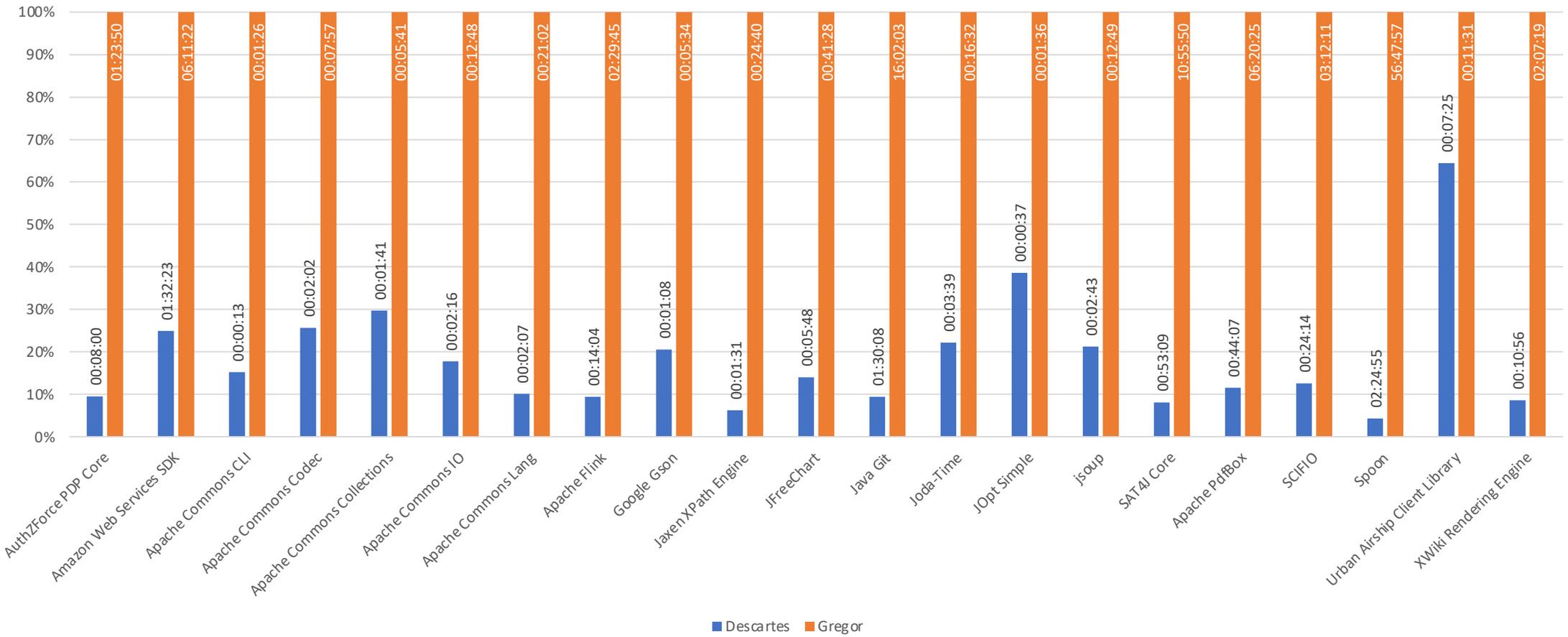
- A set of extensions for PIT
- Implements extreme mutation
- Finds pseudo-tested methods in Java projects


Compiled files
0100
.class
Compiled test files
Source files
{…}
.java
Test source files
Dependencies
.jar
Configuration
.xml
Operators to use
Number of execution threads
…
PIT
Descartes
class A {
…
public void voidMethod() {}
public int intMethod() {}
public String strMethod() {}
…
}
void
3
“S”
null
Execution Unit
Reports
.json

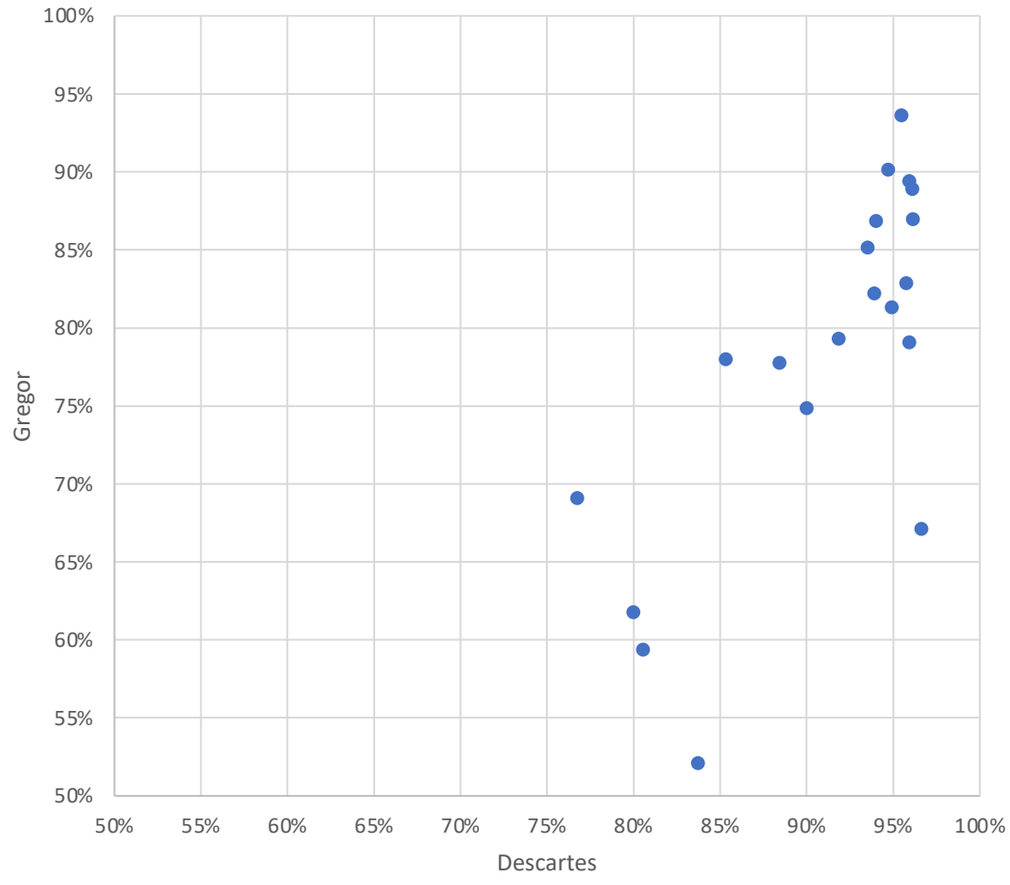
Practical results: number of mutants



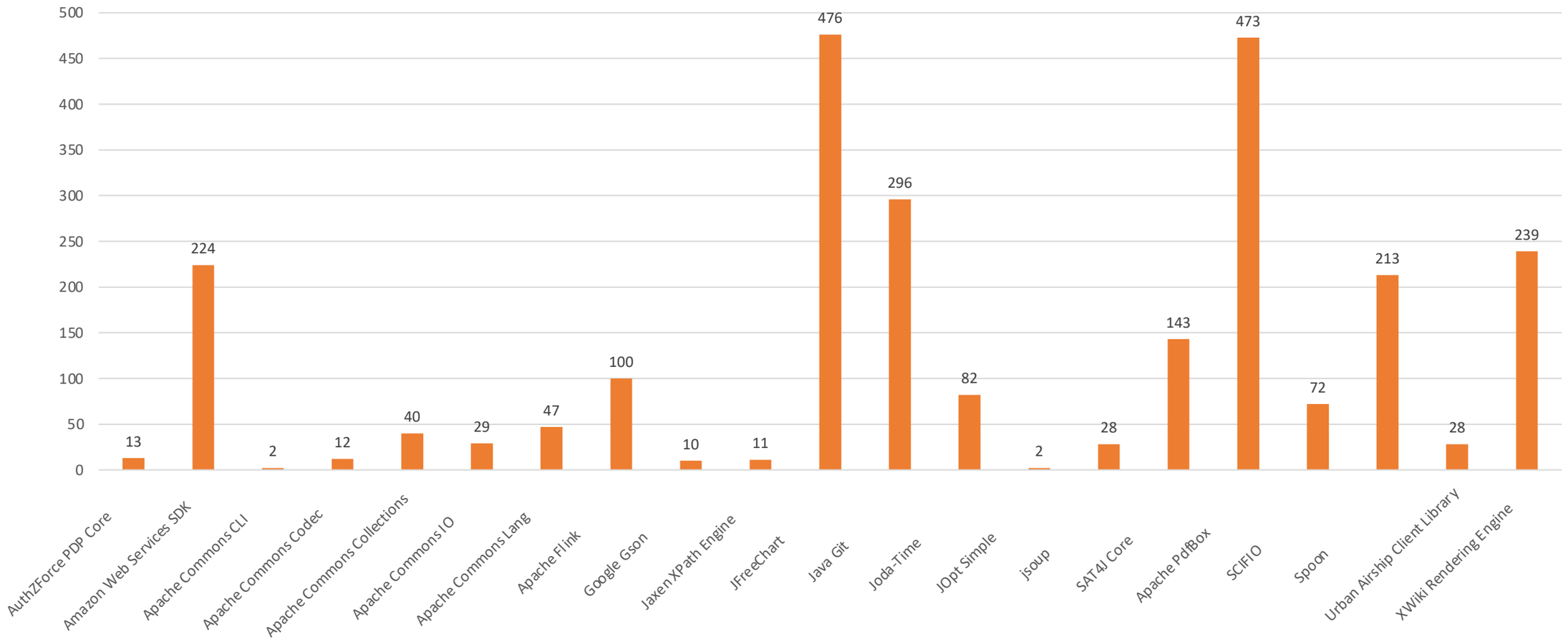
Practical results: time



Raw mutation score correlation



Pseudo-tested methods



Some examples

Apache Commons Codec

```
public void testIsEncodeEquals() {
  final String[][] data = {
    {"Meyer", "M\u00fcller"},
    {"Meyer", "Mayr"},
    ...
    {"Miyagi", "Miyako"}
  };
  for (final String[] element : data) {
    final boolean encodeEqual =
      this.getStringEncoder().isEncodeEqual(element[1], element[0]);
  }
}
```

No assertions

Pseudo-tested

Apache Commons IO

```
public void testTee() {
  ByteArrayOutputStream baos1 = new ByteArrayOutputStream();
  ByteArrayOutputStream baos2 = new ByteArrayOutputStream();
  TeeOutputStream tos = new TeeOutputStream(baos1, baos2);
  ...
  tos.write(array);
  assertByteArrayEquals(baos1.toByteArray(), baos2.toByteArray());
}
```

Pseudo-tested

Weak oracle
Result is the same if nothing is written

Apache Commons Collections

```
class SingletonListIterator
   implements Iterator<Node> {
   ...
   void add() {
     throw
         new UnsupportedOperationException();
   }
   ...
}
```

Pseudo-tested

```
class SingletonListIteratorTest {
  ...
  @Test
  void testAdd() {
    SingletonListIterator it = ...;
    try {
      it.add(value);
    }
    catch(Exception ex) {}
    ...
}
```

No exception is thrown
A fail is needed here

Amazon Web Services SDK

```
class SdkTLSSocketFactory {
  protected void prepareSocket(SSLSocket s){
    ...
    s.setEnabledProtocols(protocols);
    ...
  }
}
```

Pseudo-tested

```
@Test
void typical() {
  SdkTLSSocketFactory f = ...;
  f.prepareSocket(new TestSSLSocket() {
    @Override
    public void setEnabledProtocols
      (String[] protocols) {
      assertTrue(
        Arrays.equals(protocols, expected));
    }
    ...
  });
}
```

No assertion is verified if the method is emptied

Partially-tested methods

```
class AClass {
  private int aField = 0;
  public AClass(int field) {
    aField = field;
  }
  public boolean equals(object other) {
    return other instanceof AClass &&
      ((AClass) other).aField == aField;
  }
}
```

return false;

return true;

```
@Test
public void test()
  AClass a = new AClass(3);
  AClass b = new AClass(3);
  AClass c = new AClass(4);
  assertTrue(a.equals(b));
  assertFalse(a == c);
}
```

Is always false (in Java)

Mixed results may also reveal faults

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: PIT