

SOFIA GATTUCCI E PRISCILLA RAUCCI

HIGHER ORDER MUTATION TESTING IN PIT-HOM

Verifica e Convalida del Software AA 2019/20

COS'È IL MUTATION TESTING?



MUTATION TESTING: INTRODUZIONE

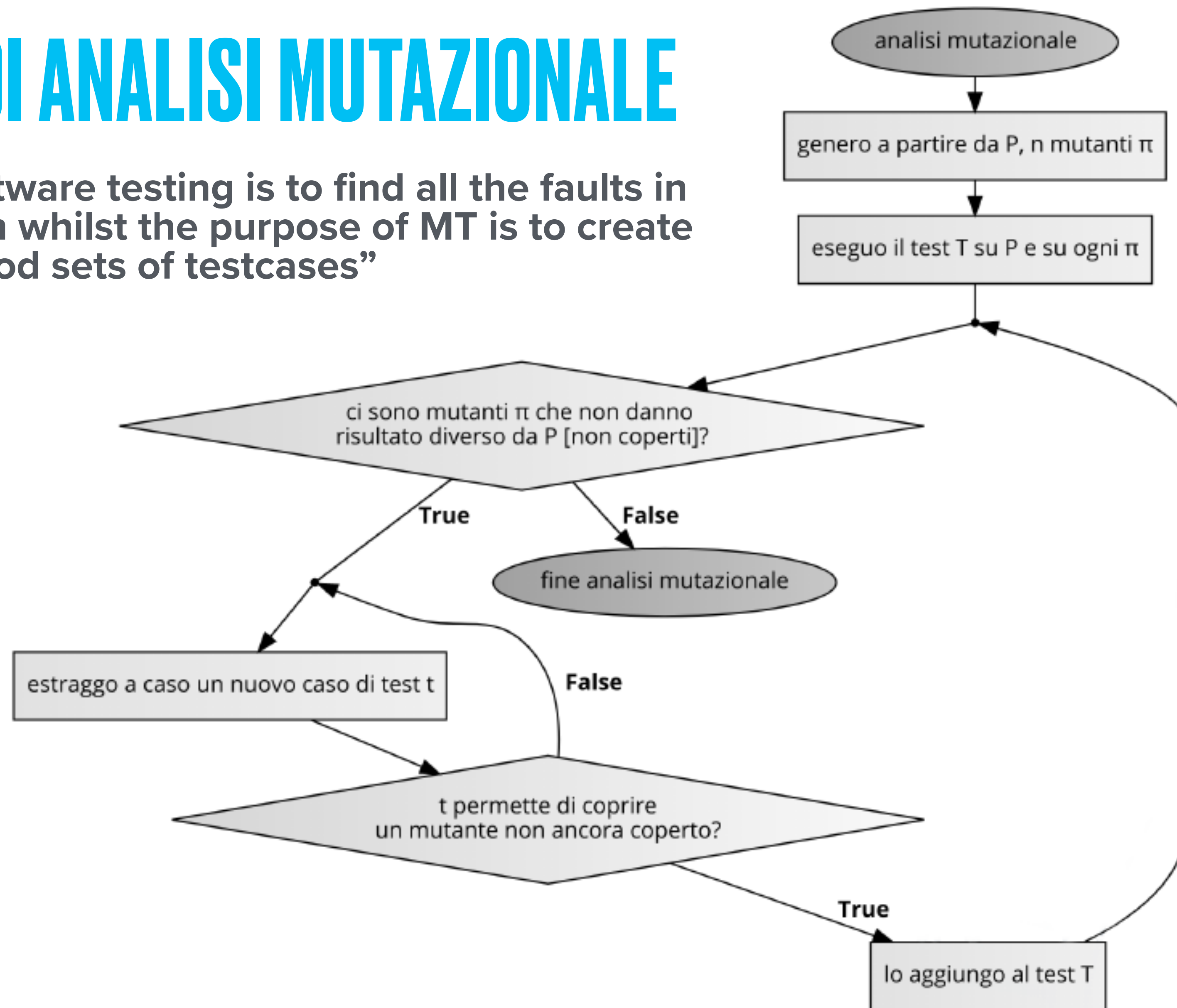
“Mutation testing is the process whereby a fault is deliberately inserted into a software system, in order to assess the quality of test data, in terms of its ability to find this fault.” (ii)

- **MUTANT**: versione del programma originale a cui è stato inserito inserito un fault
- **MUTATION OPERATOR**: cambiamento sintattico che simula un errore nella scrittura del programma
- **MUTATION SCORE**: percentuale di mutanti uccisi

$$MS = \frac{\text{Number of killed mutants}}{\text{Total mutants} - \text{Equivalent mutants}}$$

PROCESSO DI ANALISI MUTAZIONALE

“the purpose of software testing is to find all the faults in a particular program whilst the purpose of MT is to create good sets of testcases”



FOMT VS HOMT

- First Order Mutation Testing
 - I mutanti sono caratterizzati dall'inserimento di un unico *mutation operator*
- Higher Order Mutation Testing
 - I mutanti sono caratterizzati da più cambiamenti sintattici



FALSI MITI DI HOMT

- **Competent programmer hypothesis:** un programmatore, se ritenuto competente, non farà grandi errori nel suo codice.

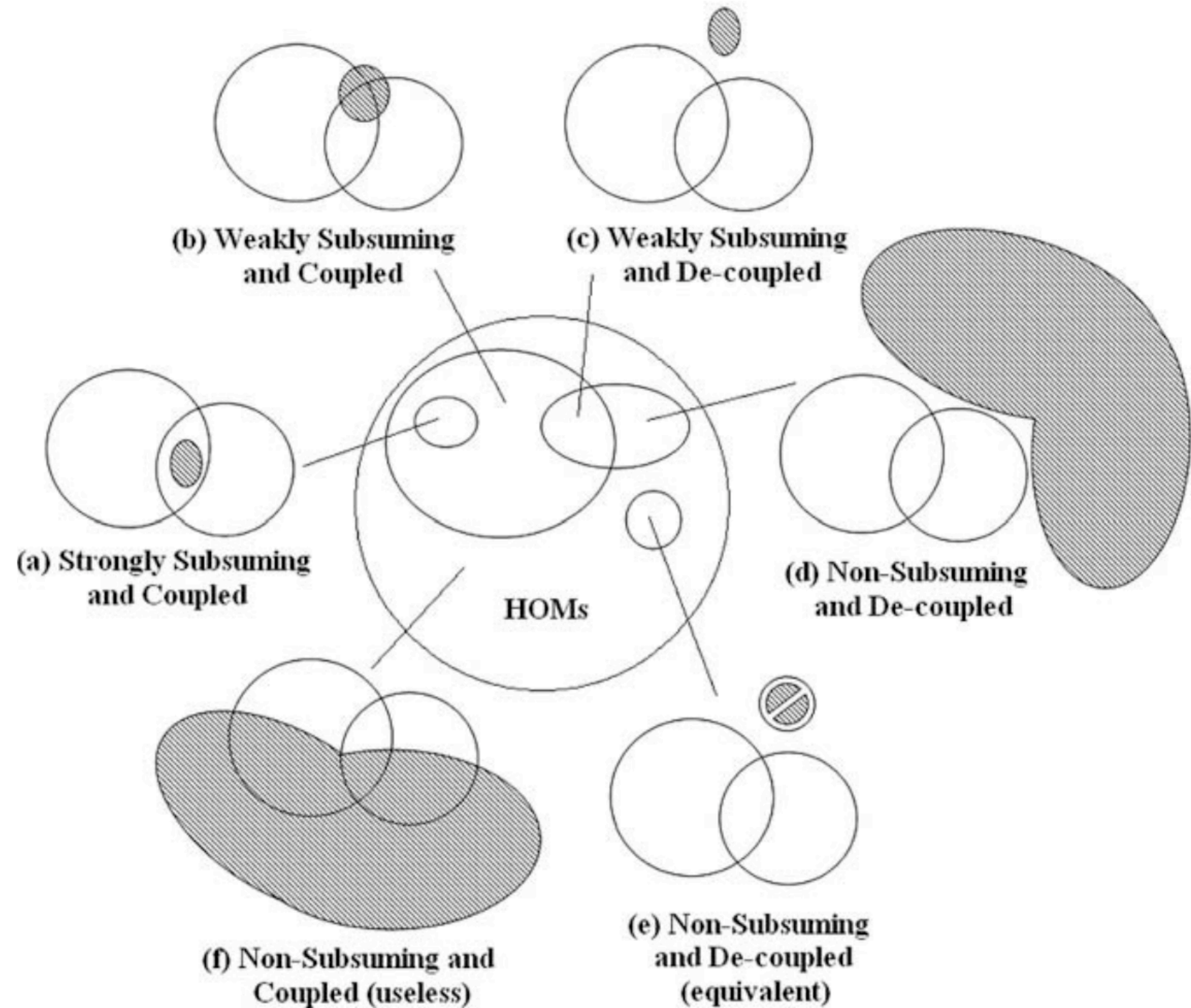
“states that programmers are generally within a few keystrokes of being correct “ (iv)

- **Coupling effect:**

“Complex faults are coupled to simple ones in such a way that test data which find all simple ones will detect a high percentage of complex faults.” (iv)

INTERESTING HOMs

- **DECOUPLED HOM:**
HOM che viene ucciso da casi di test differenti da quelli che uccidono i FOM che lo compongono.
- **STRONG SUBSUMING HOM:**
HOM che viene ucciso da un sottoinsieme del test set che uccide tutti i FOMs che lo compongono.
- **SUBTLE HOM:**
HOM che non viene ucciso da nessun test presente nella suite di test (che uccide tutti i FOMs che lo compongono).



PRO DI HOMT

- **Equivalent mutant problem:**

gli *equivalent mutant* sono quelli che sono semanticamente identici al programma originale e per cui non possono essere uccisi da nessun test.

Codice originale	mutante equivalente
<code>customerAccount -= payment; customerPayments += payment;</code>	<code>customerAccount -= payment++; customerPayments += --payment;</code>

- **Realism problem:**

un mutante simula un fault realmente verificabile?

Se riusciamo ad uccidere tutti i mutanti, vuol dire che possiamo individuare una grande percentuale di faults “reali”?

- **Difficulty of killing**

CONS OF HOMT

- **Costo:**

Il problema del numero enorme di mutazioni generate è un problema che esiste già nei FOM. Questo problema è esasperato nella generazione degli HOM.

ma è proprio vero che tutti i mutanti possibili debbano essere generati ed eseguiti?

SEARCH TECHNIQUES

L'uso di algoritmi di ricerca guidata permette di focalizzare l'indagine su HOM più interessanti e ridurre l'insieme degli HOM trovati.

le tipologie di algoritmi più utilizzati sono:

- Greedy
- Genetic Algorithm
- Hill-climbing
- Linear search
- ...

funzione di Fitness per subsuming HOMs:

$$fragility(\{M_1, \dots, M_n\}) = \frac{|\bigcup_{i=1}^n kill(M_i)|}{|T|}$$

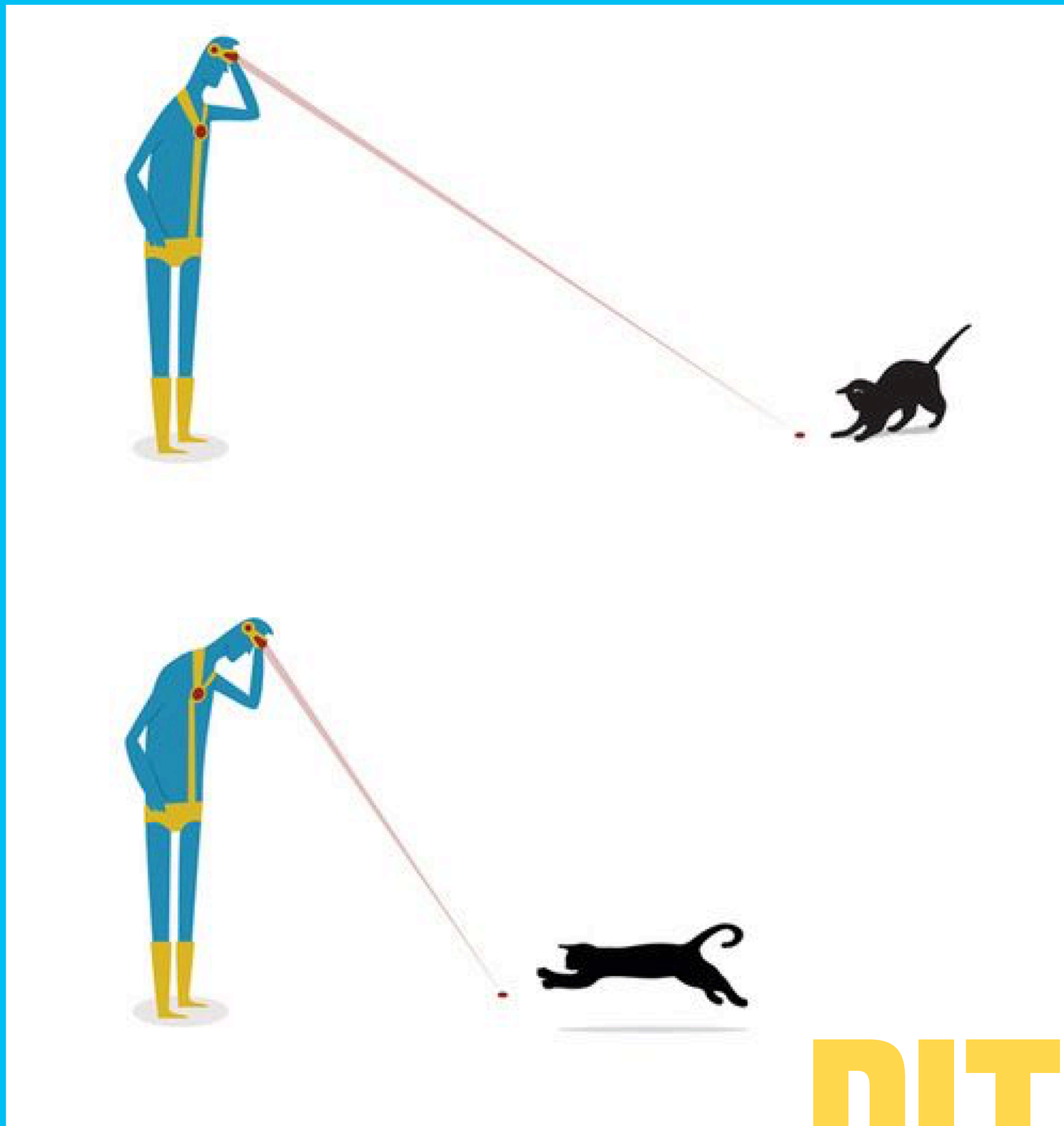
$$fitness(M_{1\dots n}) = \frac{fragility(\{M_{1\dots n}\})}{fragility(\{F_1, \dots, F_n\})}$$

Dove T è l'insieme di test cases, $\{M_1, \dots, M_n\}$ l'insieme di mutanti, la funzione $kill(\{M_1, \dots, M_n\})$ restituisce l'insieme di test cases che uccidono i mutanti M_1, \dots, M_n e l'insieme di FOMs che costituiscono l'HOM analizzato sono indicati con $F_1 \dots F_n$

TOOLS FOR MUTATION TESTING

FOMT AND HOMT

- Mujava (Java)
- Major (Java)
- Stryker (JS, C#, Scala)
- Judy (Java)
- PitTest (Java)
- Milu (C)
- HomaJ (Java)
- LittleDarwin (Java)
- Pit-HOM (Java)



PIT-HOM DEMO _

IMPORTING PIT-HOM

```
<plugin>
  <groupId>org.pitest</groupId>
  <artifactId>pitest-maven</artifactId>
  <version>1.5.1-HOM</version>
  <configuration>
    <targetClasses>
      <param>prova.pit.hom.*</param>
    </targetClasses>
    <targetTests>
      <param>prova.pit.hom.*</param>
    </targetTests>
    <hom>3</hom>
    <mutantProcessingMethod>stream-batch</mutantProcessingMethod>
  </configuration>
</plugin>
```

- il tag **<hom>** indica il grado degli HOM
- il tag **<mutantProcessingMethod>** indica l'algoritmo con cui vengono eseguiti gli HOM (stream o stream-batch)

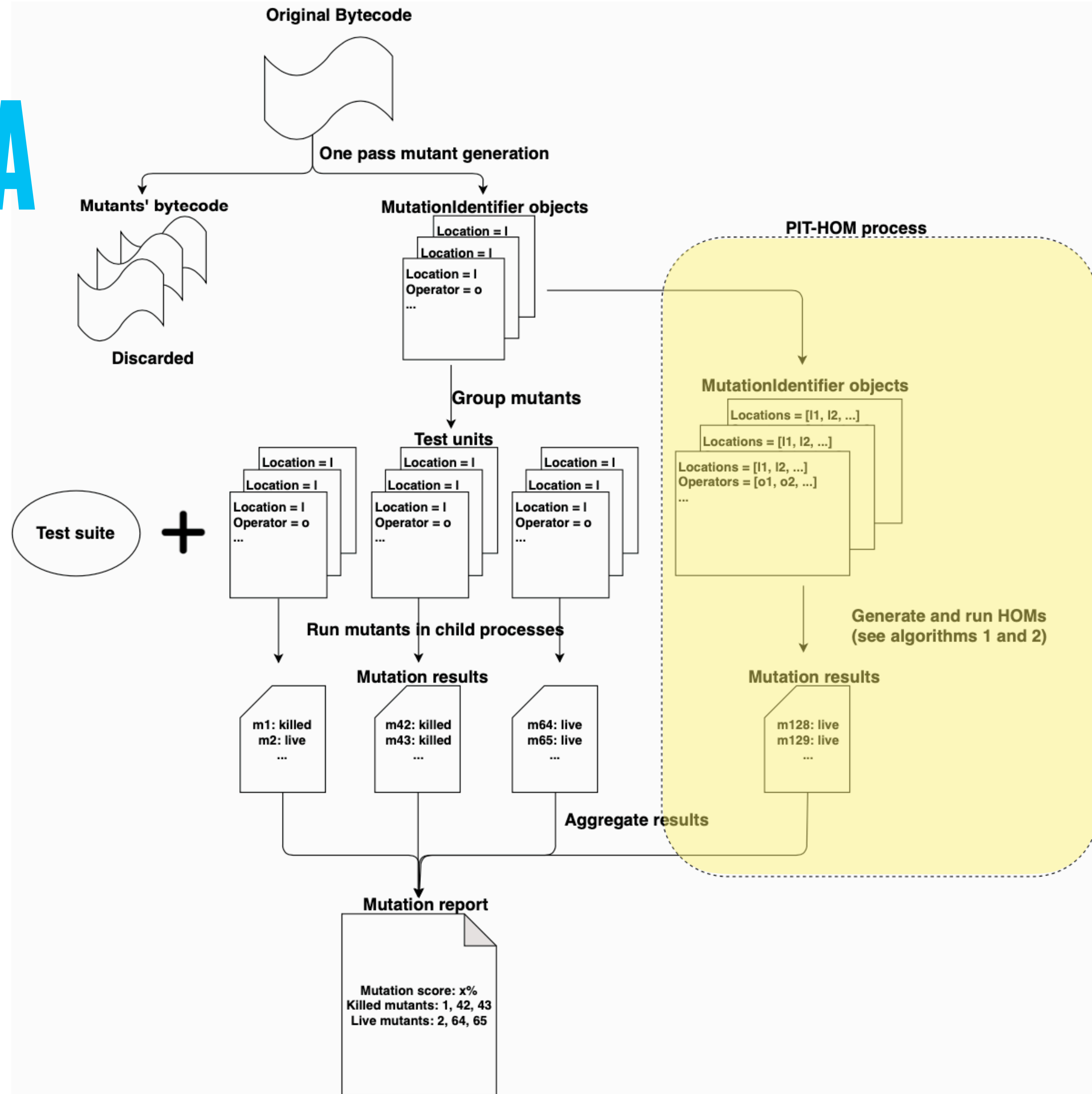
PIT-HOM: STRUTTURA

Struttura comune

- Mutant generation
- Mutant evaluation
- Reporting

Differenze

- Mutation identifier
- Algoritmi di valutazione



PIT-HOM: ALGORITMI

Algorithm 1 Mutation analysis process with streaming method

Input:

classesToAnalyse: List of classes that should be mutated
ordersToRun: List of mutation orders that should be

run

```
1: for  $class \in classesToAnalyse$  do
2:    $foms \leftarrow MutationSource.findMutants(class)$ 
3:   if  $1 \in ordersToRun$  then
4:     for  $mutant \in foms$  do
5:        $run(new\ TestUnit(mutant))$ 
6:     end for
7:   end if
8:   for  $order \in ordersToRun$  do
9:      $hom \leftarrow findNextCombination(foms, order)$ 
10:    while  $hom \neq null$  do
11:       $run(new\ TestUnit(mutant))$ 
12:       $hom \leftarrow findNextCombination(foms)$ 
13:    end while
14:  end for
15: end for
```

Algorithm 2 Mutation analysis process with batch-streaming method

Input:

classesToAnalyse: List of classes that should be mutated
ordersToRun: List of mutation orders that should be

run

```
1: for  $class \in classesToAnalyse$  do
2:    $foms \leftarrow MutationSource.findMutants(class)$ 
3:   if  $1 \in ordersToRun$  then
4:      $run(makeTestUnits(mutants, maxTestUnitSize))$ 
5:   end if
6:   for  $order \in ordersToRun$  do
7:      $hom \leftarrow findNextCombination(foms, order)$ 
8:      $homsToRun \leftarrow \{\}$ 
9:     while  $hom \neq null$  do
10:       $homsToRun.add(hom)$ 
11:      if  $homsToRun.size() == 10000$  then
12:         $run(makeTestUnit(homsToRun, maxTestU-$ 
13:           $nitSize))$ 
14:         $homsToRun \leftarrow \{\}$ 
15:      end if
16:       $hom \leftarrow findNextCombination(foms)$ 
17:    end while
18:    if  $homsToRun.size() > 0$  then
19:       $run(makeTestUnits(homsToRun, maxTestUnit-$ 
20:         $Size))$ 
21:    end if
22:  end for
23: end for
```

GRAZIE PER L'ATTENZIONE

BIBLIOGRAFIA

- (i) *Thomas Laurent and Anthony Ventresque, PIT-HOM: an Extension of Pitest for Higher Order Mutation Analysis, 2019*
- (ii) *Ahmed S. Ghiduk, Moheb R. Girgis, Marwa H. Shehata, Higher order mutation testing: A Systematic Literature Review, 2016*
- (iii) *Elmahdi Omar, Sudipto Ghoshâ, Darrell Whitley, Subtle higher order mutants, 2016*
- (iv) *Mark Harman, Yue Jia and William B. Langdon, A Manifesto for Higher Order Mutation Testing, 2010*
- (v) *Yue Jia *, Mark Harman, Higher Order Mutation Testing, 2009*
- (vi) *R.A. DeMillo, R.J. Lipton, F.G. Sayward, Hints on test data selection: Help for the practicing programmer, 1978*

