

# BIENVENUE

A N o t r e P r e s e n t a t i o n

# INTRODUCTION

L'évolution des techniques de tests de programmes passent des méthodes intuitives a des approches plus rigoureuses comme les tests de mutation.

Les tests de mutation consistent a introduire délibérément des erreurs dans le code pour évaluer l'efficacité des tests existants. Cette approche permet d'identifier des erreurs subtiles et d'améliorer la robustesse du code. Les tests de mutation sont utilisés a différents niveaux de test logiciel et dans divers langages de programmation.



## Objectifs du mutation testing

- Identifier les morceaux de code qui ne sont pas testés correctement
- Identifier les défaillances qui ne peuvent être détectées par d'autres méthodes de test
- Découvrir de nouveaux types d'erreurs susceptible de sugar
- Calculer le score de mutation **i.e.** nombre de mutants tués/nombre total de mutants
- Evaluer la qualité de cas de tests
- Améliorer la robustesse et la fiabilité du logiciel

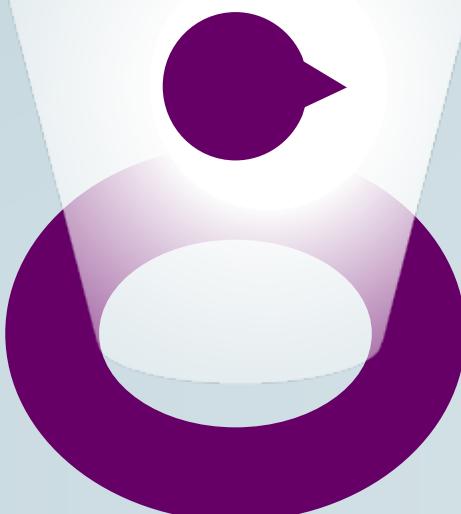
# PLAN



Définition et  
Principes



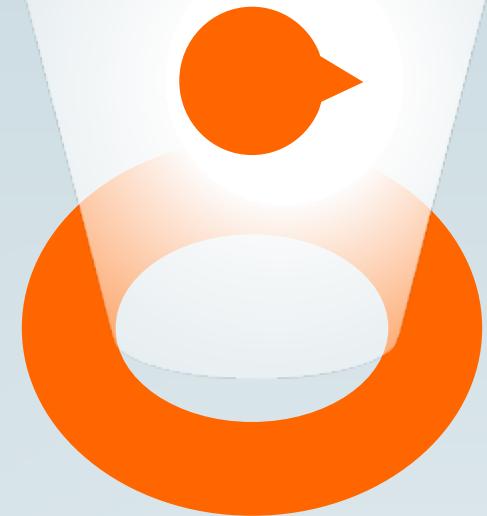
Types de tests  
de mutation



Cycle de vie des  
tests de mutation



Conclusion

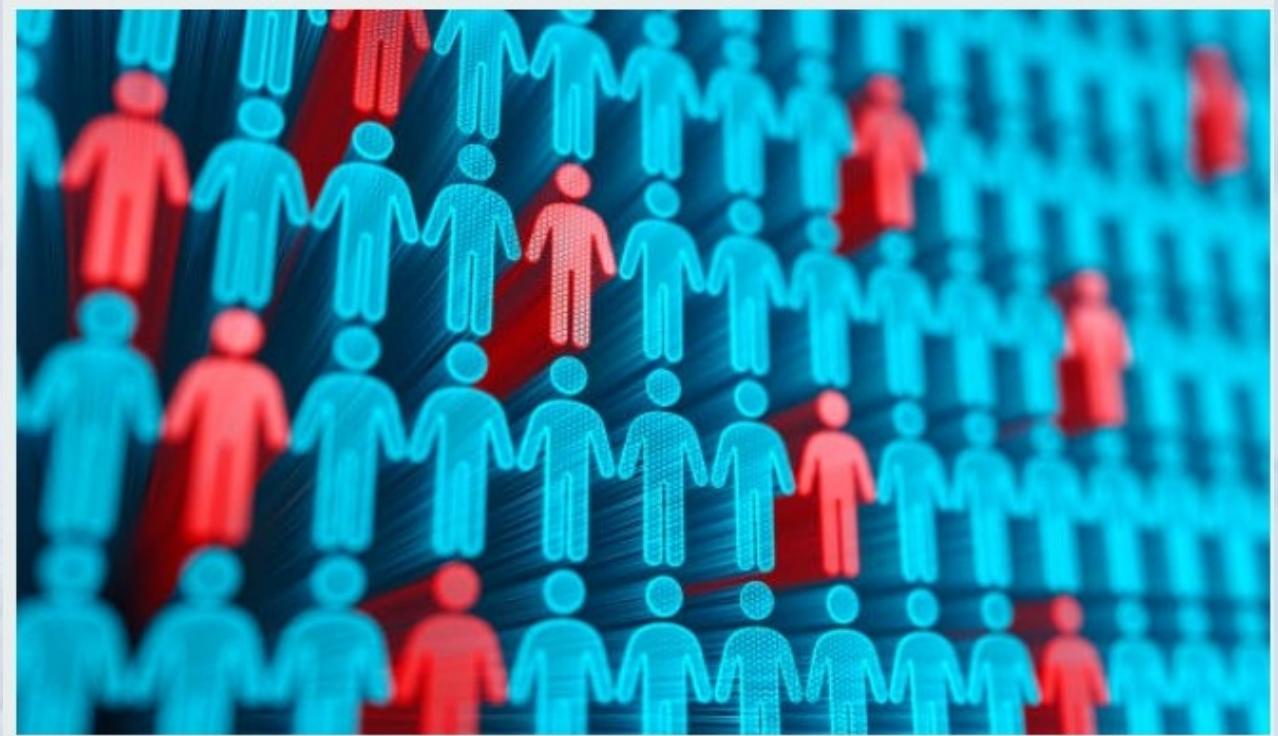


Cas Pratique

## Définition et principe des tests de mutation

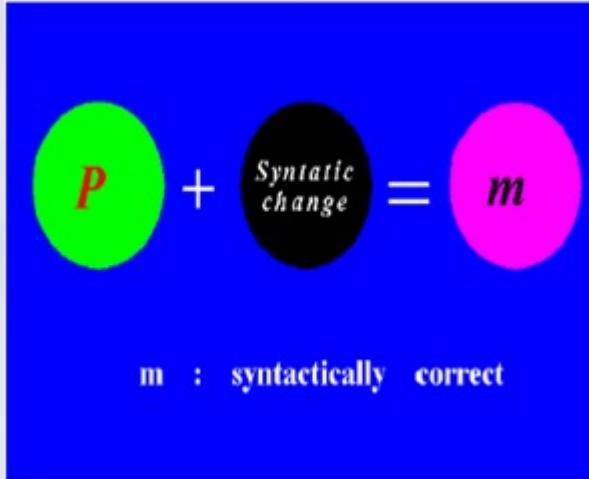
Les tests de mutation consistent à générer des versions modifiées, appelées mutants, du code source original en appliquant des petites modifications syntaxiques ou logiques.

L'idée centrale est d'exécuter les tests existants avec ces mutants. Si donc un test échoue pour un mutant donné, cela signifie qu'il détecte effectivement l'erreur introduite mais si le test réussit pour un mutant, ceci implique que le test est insuffisant pour détecter cette erreur particulière.



## Qu'est-ce qu'un mutant?

Etant donné un programme P, un mutant de P est obtenu en effectuant un simple changement dans P. C'est une version modifiée du code source original obtenue en appliquant une opération de mutation.



### Program

1. int x, y;
2. if (x! = 0)
3.     y = 5;
4. else z = z - x;
5. if (z > 1)
6.     z = z/x;
7. else
8.     z = y;

### Mutant

1. int x, y;
2. if (x! = 0)
3.     y = 5;
4. else z = z - x;
5. if (z < 1) ←  
      z = z/x;
6. else
8.     z = y;



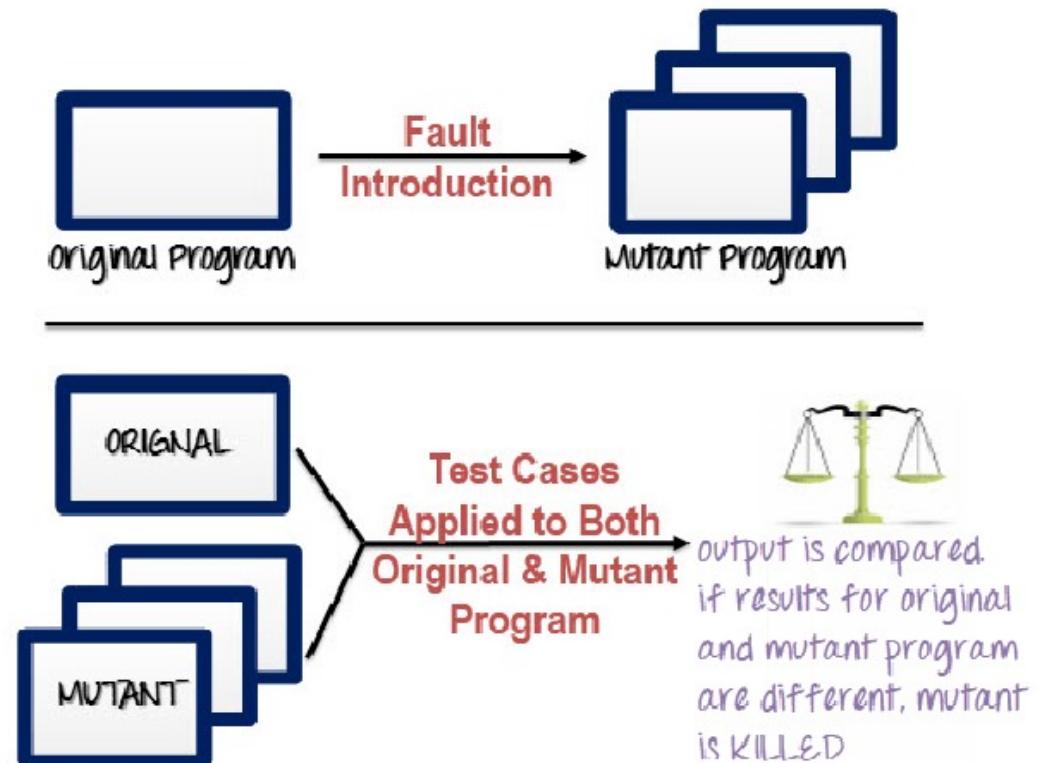
## Comment réaliser un test de mutation?

Les tests de mutation sont généralement effectués en suivant les étapes suivantes:

1. Ecrire un test unitaire
2. Exécuter le code avec le test
3. Générer des mutants
4. Exécuter les mutants avec le test
5. Calculer le score de mutation

# Comment exécuter un test de mutation?

## How to execute Mutation Testing?



## Types de tests de mutation

### 1. Value mutation(mutations de valeur):

**Initial Code:**

```
int mod = 1000000007;  
int a = 12345678;  
int b = 98765432;  
int c = (a + b) % mod;
```

**Changed Code:**

```
int mod = 1007;  
int a = 12345678;  
int b = 98765432;  
int c = (a + b) % mod;
```

### 2. Decision mutation(mutations décisionnelles):

**Initial Code:**

```
if(a < b)  
    c = 10;  
else  
    c = 20;
```

**Changed Code:**

```
if(a > b)  
    c = 10;  
else  
    c = 20;
```

## Types de tests de mutation

### 3. Statement mutation(mutations de déclaration):

**Initial Code:**

```
if(a < b)
    c = 10;
else
    c = 20;
```

**Changed Code:**

```
if(a < b)
    d = 10;
else
    d = 20;
```

## Cycle de vie des tests de mutation

1. Analyse des besoins
2. Planification des tests
3. Développement de cas de test
4. Configuration de l'environnement de test
5. Exécution des tests
6. Clôture du cycle d'essai
7. Répétition des tests

## Outils utilisés pour le test de mutation

- Stryker
- Pitest
- Insure++
- Le bric-à-brac
- Mutpy

## Avantages des test de Mutation

1. Identification d'erreurs non détectées.
2. Augmentation de la couverture de test.
3. Amélioration de la qualité et la robustesse du code.

## Limites des tests de Mutation

1. Génération d'un grand nombre de mutants.
2. Nécessité d'un outil de mutation efficace
3. Difficulté d'interprétation des résultats.

# CONCLUSION

Les tests de mutation se révèlent être un outil précieux pour le processus de test logiciel, offrant ainsi plusieurs avantages significatifs tels que l'identification d'erreurs subtiles, amélioration de la robustesse du code et bien d'autres. Il est généralement utilisé préalablement pour tester avant que d'autres tests soient faits.



MERCI.

Avez-vous des questions ?