REPUBLIQUE DU CAMEROUN

Paix – Travail – Patrie

-. - . - . –

UNIVERSITE DE YAOUNDE I

Faculté des Sciences

Département d'Informatique

B.P. 812 Yaoundé

REPUBLIC OF CAMEROON

Peace – Work – Fatherland

-. - . - . –

UNIVERSITY OF YAOUNDE I

Faculty of Sciences

Department of Computer Science

P.O. Box 812 Yaoundé

EXPOSE DE INF352

TEST LOGICIEL

LISTE DES MEMBRES DU GROUPE

NOMS ET PRENOMS	MATRICULE
NTABET SALOMON PIERRE	21T2582
LIOMO SIMEU NAOMI	21T2268
TIAGOU AZAMBOU YOLLANDE MATHILDE	21T2523

LE MUTATION TESTING - SOFTWARE TESTING

I. INTRODUCTION

Traditionnellement, les tests de programmes étaient une technique improviste faites par tous les programmeurs : le programmeur crée des données de test qui, selon lui, capturent intuitivement les principales caractéristiques du programme, observe les programmes en exécution sur les données, et si le programme fonctionne sur les données (c'est-à-dire qu'il réussit son test), il conclut alors que le programme est correct. Tout comme la plupart des programmeurs ont testé les programmes de cette manière, la plupart des programmeurs ont également considéré comme corrects les programmes qui étaient effectivement incorrect.

Les techniques de test modernes tentent d'augmenter les capacités instinctives du programmeur en fournissant des informations quantitatives sur la façon dont un programme est testé par les données de test fournies. Il est certain que le grand nombre des cas de test ne suffisent pas à augmenter significativement notre confiance dans le bon fonctionnement d'un programme. Si tous les cas de tests exercent le programme à peu près de la même manière, alors rien n'a été gagné sur un plus petit nombre d'exécutions. L'idée clé des techniques de test modernes est d'exercer le programme dans diverses circonstances, donnant ainsi au programmeur une plus grande confiance dans le bon fonctionnement du composant logiciel.

Le développement logiciel moderne implique des processus rigoureux pour garantir la qualité et la fiabilité du code produit. Parmi les différentes approches de test logiciel, les tests de mutation occupent une place importante pour leur capacité à identifier des erreurs subtiles et à améliorer la robustesse du code. L'histoire des tests de mutation remonte à 1971 dans un article étudiant de Richard Lipton. La naissance du domaine peut également être identifiés dans des articles publiés à la fin des années 1970 par DeMillo. Les tests de mutation peuvent être utilisés pour tester les logiciels au niveau du test unitaire, au niveau du test d'intégration et au niveau du test d'acceptance. Il a été appliqué à de nombreux langages de programmation en tant que test unitaire en white-box testing, par exemple les programmes Fortran, C, C#, code SQL. Les tests de mutations ont également été utilisés pour les tests d'intégration. Les tests de mutation

s'appuient sur le principe de l'injection de fautes contrôlées dans le code source. L'objectif est de vérifier si les tests existants sont suffisamment efficaces pour détecter ces erreurs introduites intentionnellement.

Cet exposé a pour but de:

- Présenter les concepts fondamentaux des tests de mutation
- Démontrer les avantages et les limites de cette approche de test
- Illustrer la mise en œuvre des tests de mutation à l'aide d'un exemple concret
- Discuter des outils et des ressources disponibles pour les tests de mutations

PLAN

II. DEVELOPPEMENT

a. Objectifs du mutation testing

Suivant le contexte et les problématiques énonces plus haut dans l'introduction, les objectifs des tests de mutation sont multiples et sont comme suit:

- Identifier les morceaux de code qui ne sont pas testés correctement.
- Identifier les défaillances qui ne peuvent être détectés par d'autres méthodes de test.
- Découvrir de nouveaux types d'erreurs ou de bugs susceptible de surgir.
- Calculer le score de mutation, le score de mutation est le nombre de mutants tués / nombre total de mutants.
- Étudier la propagation des erreurs et l'infection de l'état dans le programme.
- Évaluer la qualité des cas de tests.
- Amélioration de la robustesse et de la fiabilité du logiciel.

b. Définition et principe des tests de mutation

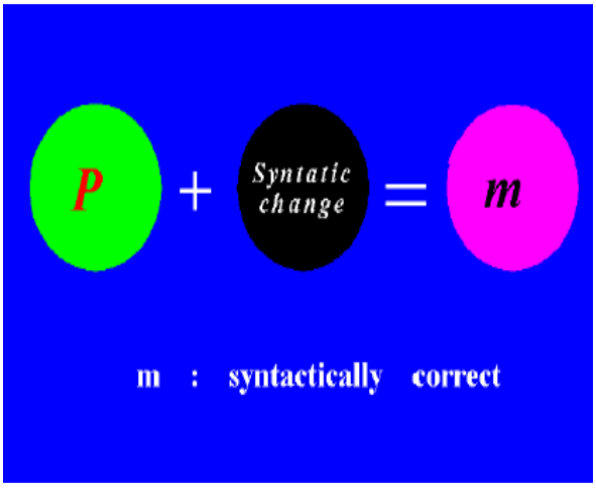
Les tests de mutation consistent à générer des versions modifiées, appelées mutants, du code source original en appliquant de petites modifications syntaxiques ou logiques. Ces modifications, appelées opérateurs de mutation, représentent des erreurs potentielles que l'on pourrait trouver dans le code réel.

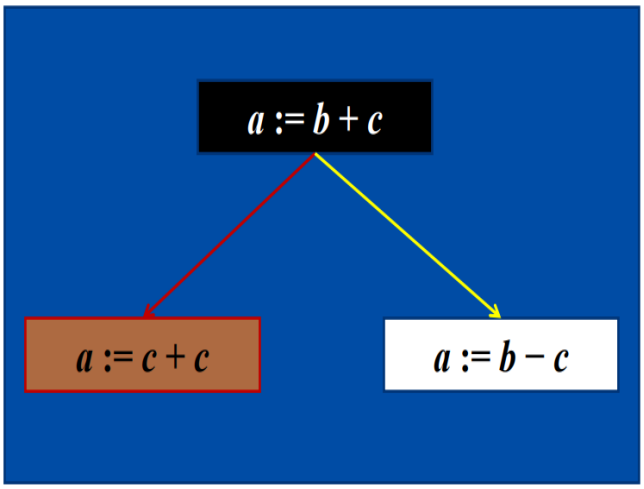
L'idée centrale est d'exécuter les tests existants avec ces mutants. Si un test échoue pour un mutant donné, cela signifie qu'il détecte effectivement l'erreur introduite. En revanche, si un test réussit pour un mutant, cela indique que le test est insuffisant pour détecter cette erreur particulière.

i. Les mutants

Étant donné un programme P, un mutant de P est obtenu en effectuant un simple changement dans P. C'est-à-dire, qu'un mutant est une version modifiée du code source original obtenue en

appliquant une opération de mutation. Ces opérations représentent des erreurs potentielles que l'on pourrait trouver dans le code réel.





Example (3)

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)
6.      z = z/x;
7.  else
8.      z = y;
```

ii. Le Score de mutation

Le score de mutation est une mesure utilisée pour évaluer l'efficacité de la suite de tests dans la détection de défauts ou d'erreurs dans le code.

- Comment le calculer?

Score de mutation = (Mutants tués / Nombre total de mutants) * 100

Les cas de test sont adéquats en matière de mutation si le score est de 100 %. Les résultats expérimentaux ont montré que les tests de mutation constituent une approche efficace pour mesurer l'adéquation des cas de test. Mais le principal inconvénient est le coût élevé de génération des mutants et d'exécution de chaque cas de test contre ce programme mutant.

- Comment améliorer le score de mutation ?
  Nous devons améliorer notre score de mutation en tuant le mutant survivant.

iii. Types de tests de mutations

Il existe 3 types de mutations qui sont :

- **Value Mutation (Mutations de valeur)** : Dans ce type de test, les valeurs sont modifiées pour détecter les erreurs dans le programme. Fondamentalement, une petite valeur est remplacée par une valeur plus grande ou une valeur plus grande est remplacée par une valeur plus petite. Dans ce test, les constantes sont essentiellement modifiées.

```
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;
```

- **Decision Mutation (Mutations décisionnelles)** : Dans les décisions, les mutations sont des opérateurs logiques ou arithmétiques modifiés pour détecter les erreurs dans le programme.

```
Initial Code:

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

Changed Code:

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

- **Statement Mutation (Mutations de déclaration)** : Dans les mutations d'instructions, une instruction est supprimée ou remplacée par une autre instruction.

```
Initial Code:

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

Changed Code:

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

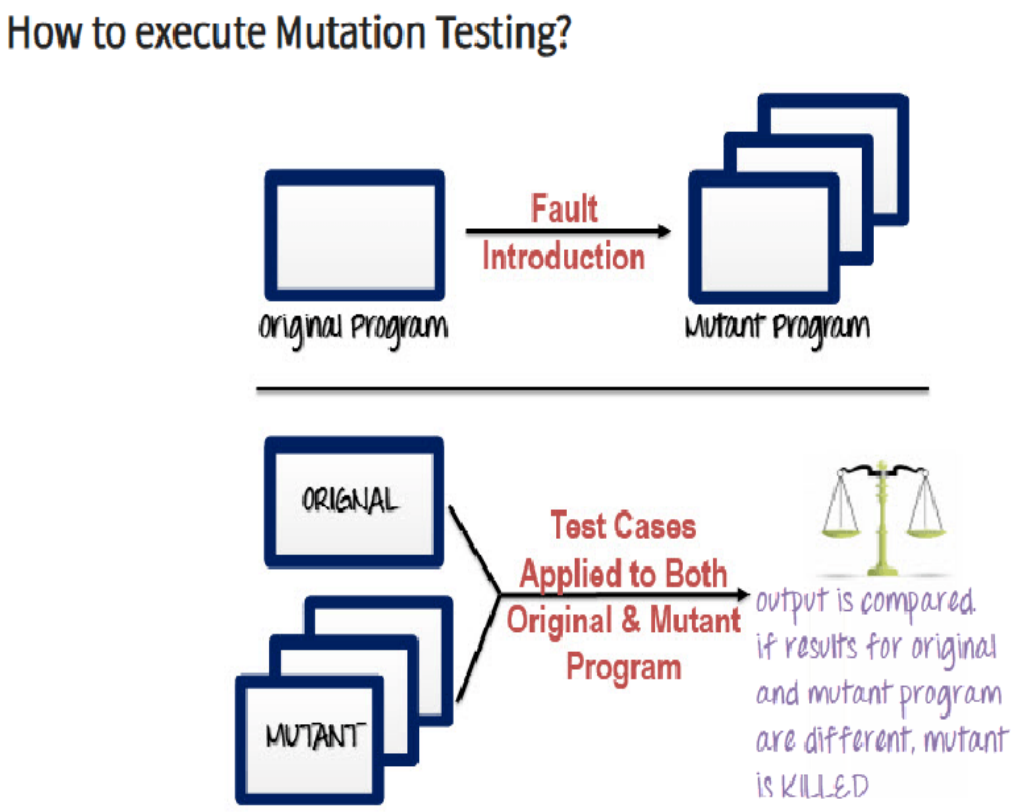
a. Cycle de vie des tests de mutations

Le cycle de vie habituel des tests de mutation est le suivant :

- **Analyse des besoins**: La première étape du cycle de vie des tests de mutation consiste à déterminer exactement ce qui doit être validé et quels éléments du code de l'application bénéficieraient le plus de ces tests. L'équipe peut s'entretenir avec les développeurs et les cadres pour déterminer leurs préoccupations et commencer à y répondre.

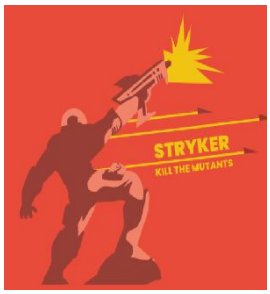
- **Planification des tests** : Les testeurs commencent alors à développer les contrôles exacts qu'ils ont l'intention de mettre en œuvre – dans ce cas, les mutations qui offriront le meilleur aperçu. Cette étape détermine la stratégie globale de test des mutations et la manière dont l'équipe va mettre en œuvre efficacement les mutations de code prévues.
- **Développement de cas de test** : Les tests de mutation impliquent une documentation de test distincte, comprenant des informations sur le code muté et sur la manière dont les testeurs doivent résoudre le problème. Une bonne tenue des registres permet de s'assurer que les tests se déroulent comme prévu et peut aider l'équipe à maintenir son engagement à respecter des normes de test élevées.
- **Configuration de l'environnement de test** : Les testeurs s'assurent que l'application est prête à être modifiée et qu'ils disposent d'une procédure pour résoudre ces problèmes si les autres membres de l'équipe ne sont pas en mesure de les détecter. Dans ce cadre, les testeurs de mutations établissent un serveur de test et l'utilisent comme canevas pour leurs mutations.
- **Exécution des tests** : Après avoir terminé leurs préparatifs, les testeurs modifient le code de plusieurs composants de l'application ; ils attendent ensuite que d'autres testeurs remarquent et corrigent les problèmes. Les testeurs de mutations et les testeurs d'applications doivent documenter ce processus de manière exhaustive afin de s'assurer que leurs enregistrements sont robustes.
- **Clôture du cycle d'essai** : Une fois les tests terminés, les testeurs de mutations vérifient à nouveau que toutes les modifications qu'ils ont apportées ont été corrigées par les testeurs d'applications ou par eux-mêmes.

  Ils clôturent ensuite le cycle de test et analysent les résultats, en discutant de la manière dont les testeurs ont réagi aux différentes erreurs et de leur capacité à les corriger.
- **Répétition des tests** : Après avoir clôturé le cycle de test, il pourrait être nécessaire de le réactiver après de futures mises à jour du logiciel. Chaque changement apporté à une application modifie sa fonctionnalité d'une manière ou d'une autre, ce qui entraîne de nouvelles possibilités dont l'équipe doit tenir compte afin de s'assurer que son processus de test est suffisamment méticuleux.



d. Outils utilisés pour le test de mutations

Ces différents outils sont utilisés sur différents langages de programmation. On a quelques outils suivant :

- 

Stryker : Stryker se spécialise dans la mutation JavaScript, typescript, c#, scala.

- PITest : PITest est un choix très populaire dans le monde entier en raison de sa capacité à modifier le code byte de Java et à effectuer des milliers de mutations par seconde.

- MutPy : MutPy prend en charge les tests de mutation pour les applications basées sur Python, offrant une prise en charge complète des mutations d'ordre élevé ainsi qu'une analyse complète de la couverture.

e. Avantages des tests de mutation

Les tests de mutation offrent plusieurs avantages significatifs pour le processus de test logiciel:

- **Identification d'erreurs non détectées:** Ils permettent de déceler des erreurs qui pourraient passer inaperçues avec d'autres approches de test, comme les tests unitaires ou les tests d'intégration.
- **Augmentation de la couverture de test:** En générant un grand nombre de mutants, les tests de mutation peuvent augmenter la couverture de test du code, en s'assurant que toutes les parties du code sont testées d'une manière ou d'une autre.
- **Amélioration de la qualité et de la robustesse du code:** En détectant et en corrigeant les erreurs identifiées par les tests de mutation, on contribue à améliorer la qualité et la robustesse du code source.

f. Limites des tests de mutation

Malgré leurs avantages, les tests de mutation présentent également certaines limites:

- **Génération d'un grand nombre de mutants:** Le processus de mutation peut générer un nombre important de mutants, dont certains peuvent être redondants ou triviaux. Cela peut s'avérer coûteux en termes de temps et de ressources de calcul.
- **Nécessité d'un outil de mutation efficace:** Un outil de mutation performant est indispensable pour automatiser la génération et l'exécution des mutants. Le choix de l'outil adapté dépend des besoins spécifiques du projet.
- **Difficulté d'interprétation des résultats:** L'analyse des résultats des tests de mutation peut être complexe, en particulier pour les mutants non tués. Il est important de comprendre les raisons pour lesquelles un test n'a pas tué un mutant donné.

III. CONCLUSION

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