# ミューテーション解析における非利用アサーションの実証評価

徳本 晋[1,a)]　石井 康嗣[2,b)]

**概要**：テストコードにおいてアサーションは結果確認のために必要な記述であるが，過剰な数のアサーションはテストコードの保守性やテスト実行時間を悪化させる恐れがある．本稿では，実際のテストコードにおいてアサーションの適正な記述量を考察すべく，7 件の Java 資産に対してミューテーション解析を適用し，バグの検出に貢献していないアサーションがどれだけ存在するかについて調査を行った．結果として，平均で 45.7%のアサーションがミューテーション解析におけるバグ検出に貢献していないことが分かった．

# An Empirical Evaluation of Inactive Assertions in Mutation Analysis

## 1. はじめに

テストコードはテスト対象プログラムの呼出を行い，その結果をアサーション (Assertion) によって確認することにより，自動テスト実行と自動結果確認を可能とする．

アサーションは期待する結果の確認のため必要であるが，多すぎるアサーションはテスト失敗時にどのアサーションで失敗したのか分かりにくくする．このことはテストコード中に深刻な問題が発生する兆候を示す”テストコードの臭い (Test code smell)”の 1 つ”アサーションルーレット (Assertion Roulette)”として知られている [1][2].

また Bavota ら [3] は 25 件の OSS と 2 件のプロプライエタリソフトウェアを調査し，アサーションルーレットが 55%のテストコードにおいて発生していることを報告した．さらに学生と企業開発者を対象にテストコード理解に関する対照実験を行うことで，アサーションルーレットの存在が保守時のテストコード理解の正確性に統計的有意に影響することを示した．

このように過多のアサーションはテストコードの保守性に影響するため，アサーションの数はバグ検出能力に影響のない限り少なくすることが求められる．そこで本稿ではバグ検出能力に影響のないアサーションの実態を知るべく，ミューテーション解析を用いて人為的にバグを生成し，それらの検出に貢献しないアサーションがどれだけあるかについて調査した．まず 2 章でバグの検出に貢献しないアサーション（非利用アサーション）の具体例を紹介し，3 章で実験方法，4 章で実験結果を示し，5 章ではその考察を述べる．

1　株式会社富士通研究所
Fujitsu Laboratories Limited, Kawasaki, Kanagawa 211–8588, Japan
2　東京工業大学
Tokyo Institute of Technology, Meguro, Tokyo 152–8550, Japan
a)　tokumoto.susumu@jp.fujitsu.com
b)　ishii.k.al@m.titech.ac.jp

## 2. ミューテーション解析における非利用アサーション

例として以下のようなメソッド isEmpty とそのテスト testIsEmpty に対してミューテーション解析を行う場合を考える．

リスト 1: ソースコード例

```
public static boolean isEmpty(CharSequence cs) {
  return cs == null || cs.length() == 0;
}
```

リスト 2: テストコード例

```
@Test public void testIsEmpty() {
  assertTrue(StringUtils.isEmpty(null));
  assertTrue(StringUtils.isEmpty(""));
  assertFalse(StringUtils.isEmpty("␣"));
  assertFalse(StringUtils.isEmpty("foo"));
  assertFalse(StringUtils.isEmpty("␣␣foo␣␣"));
}
```

表 1: 実験対象

| プロジェクト名 | LOC | テストケース数 | アサーション数 | ミュータント数 |
|---|---|---|---|---|
| JCodings | 4,324 | 24 | 30 | 4,717 |
| JSoup | 7,062 | 541 | 1,880 | 4,763 |
| jOOL | 7,572 | 310 | 1,933 | 8,320 |
| commons-lang | 77,588 | 3,975 | 15,385 | 39,303 |
| commons-math | 223,987 | 6,518 | 10,657 | 140,109 |
| commons-io | 28,160 | 1,157 | 2,948 | 11,012 |
| commons-collections | 67,798 | 2,721 | 4,954 | 16,715 |
| 合計 | 416,491 | 15,246 | 37,787 | 224,939 |

生成されるミュータント（バグが注入されたプログラム）は比較演算子を反転するものが 2 つと，論理和を論理積に変えるものが 1 つの，計 3 つとする．これらのミュータントは全て testIsEmpty の実行が失敗することにより，全てのミュータントを殺すことができるが，そのときにテスト失敗を引き起こしたアサーションはテストコード中の 2-4 行目だけで，5,6 行目のアサーションは使われていない．このようもミュータントを殺すことに使われていないアサーションを本稿では未使用アサーションと呼び，冗長な記述である可能性があると考える．

## 3. 実験手法

未使用アサーション検出のため，ミューテーション解析ツール PIT[4] に以下のような手順の処理を追加した．

(1) テストコードを静的解析し，アサーション個所を記録
(2) ミューテーション解析を実行し，例外発生時にスタックトレースからアサーション個所を記録
(3) (1) で記録したアサーション箇所のから，(2) で記録したアサーション個所を除外したものを未使用アサーションとする

実行するミューテーションオペレータは PIT のバージョン 1.2.2 で実装されているもの全てとした．実験対象は github.com 内の java のプロジェクトでビルドツールが maven のものから，中規模（1〜10KLOC）のもの 3 件と，大規模（10KLOC〜）のもの 4 件を選択した．表 1 に実験対象とした資産の規模，テストケース数，アサーション数，ミュータント数を示す．

## 4. 実験結果

実験結果を図 1 に示す．図中の数値はそれぞれ算定されたアサーションの数になる．全資産の合計では，45.7%のアサーションがミュータントを殺すのに貢献していない非利用アサーションであることが分かった．最も非利用アサーションの割合が高い資産は，commons-collections であり，67.6%が非利用アサーションであった．反対に最も非利用アサーションの割合が低い資産は，JCodings で，その割合は 23.3%だった．

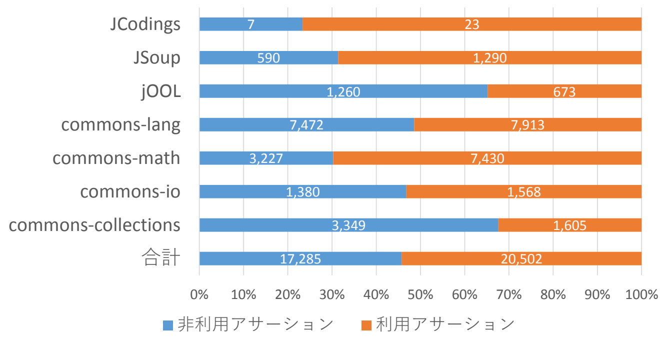


図 1: 非利用アサーションの割合

## 5. 考察

冗長なアサーション記述の検出を目的に非利用アサーションについて調査をしてきたが，非利用アサーションが多い理由については大きく分けて 2 通りの考え方がある．1 つはテストコードに問題を抱えている場合で，もう 1 つはミューテーションオペレータに問題を抱えている場合である．テストコードの問題としては，非利用アサーションの他に，テストケースがバグを感染（Infection）・伝搬（Propagation）しにくい場合も考えられる．感染・伝搬しない場合，アサーションの個所までバグによる状態変化が伝わらず，結果アサーションは使われないことになる．一方，ミューテーションオペレータの問題としては，開発者が想定しているミスをミューテーションオペレータが表現できていない場合や，PIT のミューテーションオペレータが十分でない場合 [5] が考えられる．非利用の原因がどちらの問題によるものかの判別方法は今後の課題とする．

## 6. まとめ

ミューテーション解析における非利用アサーションについて 7 件の OSS を用いて調査した．今後はより多くの資産における調査と，アサーション記述のリファクタリング方法についての検討を進めていきたい．

### 参考文献

[1] Deursen, A. et al.: Refactoring Test Code, Technical report, Amsterdam, The Netherlands, The Netherlands (2001).
[2] Meszaros, G.: *XUnit Test Patterns: Refactoring Test Code*, Prentice Hall PTR, Upper Saddle River, NJ, USA (2006).
[3] Bavota, G. et al.: Are test smells really harmful? An empirical study, *Empirical Software Engineering*, Vol. 20, No. 4, pp. 1052–1094 (2015).
[4] Coles, H.: PIT, `http://pitest.org`.
[5] Laurent, T. et al.: Assessing and Improving the Mutation Testing Practice of PIT, *2017 IEEE International Conference on Software Testing, Verification and Validation (ICST)*, pp. 430–435 (2017).