ソフトウェアテストシンポジウム 2004
JaSST'04: Japan Symposium on Software Testing 2004

ミューテーションスコアを利用した テストセット評価システム

斉藤　孝志† 　粕谷　英人†† 　大久保弘崇†† 　山本晋一郎††

† 愛知県立大学大学院 情報科学研究科　〒 480-1198 愛知県愛知郡長久手町大字熊張茨ヶ廻間 1522-3
E-mail: †saito@yamamoto.ist.aichi-pu.ac.jp, ††{kasuya,ohkubo,yamamoto}@ist.aichi-pu.ac.jp

あらまし　ソフトウェアの信頼性を向上させるために，テストセットの品質を高めることは重要である．本稿では，テストセットの品質を測定するために，ミューテーションスコアを用いた方法を提案する．ミューテーションスコアを測定するシステムを実装して，gawk-3.0.1 と gzip-1.2.4 に含まれるテストセットに適用した．gawk-3.0.1 のミューテーションスコアは 36%であり，gzip-1.2.4 のミューテーションスコアは 21%であった．
キーワード　ミュータント，ミューテーションスコア，テストセット

The test set evaluation system which used mutation score

Takashi SAITO†, Hideto KASUYA††, Hirotaka OHKUBO††, and Shinichiro YAMAMOTO††

† Graduate School of Information Science and Technology, Aichi Prefectural University　1522-3, Ibaragabasama, Kumabari, Nagakute-cho, Aichi-gun, Aichi, 480-1198, Japan
E-mail: †saito@yamamoto.ist.aichi-pu.ac.jp, ††{kasuya,ohkubo,yamamoto}@ist.aichi-pu.ac.jp

Abstract　To make software reliable, it is important to raise the quality of a test set. In this paper, we propose a method uses the mutation score in order to measure the quality of a test set. We applied our experimental implimentation to the test sets included in gawk-3.0.1 and gzip-1.2.4 distributions. Their mutation scores are 36% and 21%, respectively.
Key words　mutant, mutation score, test set

1. はじめに

ソフトウェアのライフサイクルは，開発段階と製品段階の 2 つの段階に大きく分けることができる．開発段階には要求分析，仕様書の作成，設計，コーディング，テスト等の作業があり，製品段階では保守・運営の作業がある．このソフトウェアのライフサイクルで最も多くのコストを占めているのは，製品段階での保守・運営である．次いで，開発段階で大部分を占めている製品のテストである．テストの役割は，安定したソフトウェアをユーザに提供するためにプログラムのバグを発見することである．しかし，プログラム中に存在する全てのバグを発見することは不可能である．そこで，できるだけ多くのバグを発見するためには，入力パターンを網羅したテストセットを用いたテストが有効である．すなわち，このテストセットのバグ発見能力がテストセットの品質と考えられる．テストセットの品質を測定するために DeMillo, Offutt らによってミューテーションスコアを用いる方法が提唱されている [1] [2]．文献 [1] [2] では，Fortran-77 を対象としている．

本稿では，今日のソフトウェア開発で広く用いられている C 言語のプログラムを対象としたテストセットの品質を測定するシステムを提案する．このシステムは，プログラムとそのプログラムに対するテストセットを入力とし，テストセットの品質を測定する．

本システムは 2 つの部分から成り立っている．1 つは，文献 [2] で提案されているミューテーション解析の考えを基に，元のプログラムに混入させる誤りを生成するシステム (以下ミュータント生成システム) である．もう 1 つは，ミュータント生成システムで生成したミュータントをオリジナルプログラムに混入させて，オリジナルプログラムが持っているテストセットでテストするシステムである．この 2 つのシステムを用いて，テストセットの品質を評価するためのミューテーションスコアを測定する．

2. ミューテーション解析

ミューテーション解析は，ソフトウェアの全体テストに使用されるテストセットの品質を評価・改良するための技術である．また，ミューテーション解析は，ある考えに準じたホワイトボックステストの技術である．その考えとは，テストセットの品質

が，オリジナルプログラムとわずかに異なった誤りを含んでいる代替プログラムとオリジナルプログラムとを区別する能力と関連しているということである．

2.1 ミューテーション解析のプロセス

テスト対象のプログラムを P，P についてのテストセットを T とし，以下にそのプロセスを示す．

（1） P を T でテストする．

このテストで T 中の全てのテストケースに対して 1 つでも意図しない結果を出力するなら，P あるいは T は即座に修正する必要がある．この場合，T はプログラム中のバグを検出するという義務を果たしたことになる．

（2） プログラム P の代替プログラム P_i $(i = 1, 2, \cdots)$ を生成する．

P_i は，P のステートメントの一ヶ所を変更して生成する．例えば，$a = b + c$ の式を $a = b + d$ という具合に変更する．P_i の数は変更個数に応じて決まる．元のプログラム P の一ヶ所を変更して作られた代替プログラム P_i のことを P の**ミュータントプログラム**と呼び，特に変更したステートメントを**ミュータント**と呼ぶ．また，オリジナルステートメントを変更することを**ミューテーション**と呼ぶ．

（3） 全てのミュータントプログラム P_i を T でテストする．

T のあるテストケースでテストしたとき，ミュータントプログラム P_i がオリジナルプログラム P と異なる結果を出すならば，このテストケースは P_i のミュータントを**摘出**したという．

（4） ミューテーションスコアを測定する．

(2) で生成したミュータントの個数 M と，(3) で摘出されたミュータントの個数 K からミューテーションスコア MS を以下のように定義する．

$$MS = \frac{K}{M} \times 100 \quad [\%] \tag{1}$$

2.1.1 ミューテーションタイプ

ソースプログラムのステートメントの変更，すなわちミューテーションには様々な方法がある．表 1 に Fortran-77 を対象としたミューテーションタイプの一覧 [3] を示す．

この様々なタイプのミューテーションを利用して，P から複数のミュータントプログラム P_i を作成する．20 行からなる単純なプログラムからでも約 400 個ものミュータントプログラムを生成することができる．ミュータントプログラムの個数はミューテーションを行う場所が多いほど増えていく．

2.1.2 ミュータントの摘出

あるテストケースがミュータントを摘出するということは，そのテストケースがミュータントが表すバグを発見できることを意味している．

図 1 の関数 `Max` を使ってミュータントを摘出する例を示す．関数 `Max` は 2 つの `int` 型の入力からその最大値を返すものである．

関数 `Max` から生成されたミュータントの一例を表 2 に示す．ミュータント `max = n` は，5 行目の式 `max = m` の右辺 `m` を `n` に変更したものである．また，他のミュータントも式の一ヶ所を変更して生成されたものである．

表 1 ミューテーションタイプ

Type
array for array replacement
scalar for array replacement
absolute value insertion
scalar for constant replacement
array constant replacement
scalar variable replacement
arithmetic operator replacement
constant replacement
array for variable replacement
data statement alterations
constant for array replacement
goto label replacement
comparable array replacement
return statement replacement
constant for scalar replacement
statement analysis
DO statement end replacement
statement deletion
logical connector replacement
source constant replacement
relational operator replacement
unary operator insertion

```
1    int Max(int m, int n)
2    {
3        int max;
4
5        max = m;
6        if (n > m)
7            max = n;
8        return max;
9    }
```

図 1 関 数 `Max`

ここで，関数 `Max` をテストケース：`(m = 2, n = 1)` でテストする．この時，関数 `Max` の結果は `m` と `n` の最大値である `2` を返す．これに対して，ミュータント：`max = n` を含むミュータントプログラムを $\texttt{Max}_1$ とし，$\texttt{Max}_1$ をテストケース：`(m = 2, n = 1)` でテストする．この結果として $\texttt{Max}_1$ は 1 を返すが，これはオリジナルプログラムの結果とは異なる．よって，テストケース：`(m = 2, n = 1)` はミュータント：`max = n` を摘出したということになる．もしプログラマが 5 行目で `max = m` と書くつもりが `max = n` と誤って書いても，テストセットに `(m = 2, n = 1)` のテストケースを含んでいれば，テストの段階でこのバグを摘出できる．

表 2 関数 `Max` から生成されたミュータント

line	original	mutant
5	`max = m`	`max = n` `max = abs(m)`
7	`n > m`	`n < m` `n >= m`

また，ミュータント：`n >= m` を含むミュータントプログラムについて考えてみる．このミュータントプログラムを $\texttt{Max}_2$ とする．$\texttt{Max}_2$ にどのようなテストケースを入力しても，`Max` と異なる結果を出力しない．これは，$\texttt{Max}_2$ が `max` と関数的に同一の効果を持っているためである．このようなミュータントのことを**等価ミュータント**と呼ぶ．等価ミュータントを含むミュータントプログラムからミュータントを摘出するテストケースは存在しない．そのため，等価ミュータントは人手によって摘出する．

2.1.3 ミューテーションスコア

ミューテーションスコアはテストセットの品質の指標である．プログラム P から生成されたミュータントの数と，テストセット T で摘出することができたミュータントの数からミューテーションスコアを測定する．今，P と T に対して，生成したミュータントの総数を M，摘出されたミュータントの数を K，等価ミュータントの数を E として，ミューテーションスコア $MS(P,T)$ を，以下に定義する．

$$MS(P,T) = \frac{K}{M-E} \times 100 \quad [\%] \qquad (2)$$

等価ミュータントを摘出するテストケースは存在しないので，ミューテーションスコアを測定する際，等価ミュータントの数は考えないものとする．

3. ミュータント生成システム

本稿では，オリジナルプログラムに含ませる誤りとなるミュータントを生成する．

ミュータントの生成は，プログラム作成者が間違えやすいと思われる部分を想定する．本システムで対象とする言語は C 言語である．そこで本システムでは，C 言語のプログラムで使用頻度の高い算術演算子，関係演算子，等値演算子を含んだ算術式からミュータントを生成する．

ミュータント生成システムはオリジナルプログラムを入力として，ミューテーションの対象となる式を取得して，ミュータントを生成する．全体の流れを，以下に示す．

（1） ソースプログラムの解析

（2） ソースプログラム中に出現するミューテーションの対象となる式の取得

（3） ミュータントの生成

ソースプログラムの解析には，CASE ツール・プラットフォーム Sapid [4] を利用する．

3.1 Sapid

ソースプログラムの解析に利用する Sapid (Sophisticated APIs for CASE tool Development) [4] [5] は，細粒度ソフトウェア・リポジトリに基づいた CASE ツール・プラットフォームである．従来，CASE ツールや研究システムの作成者は，個別に解析器やリポジトリを実現しなければならなかった．しかし，Sapid によって CASE ツールの基礎的な機能が提供されるため，開発者は本質的な機能の実現に専念できるようになる．Sapid は C 言語と Java を対象としている．

Sapid は，ソフトウェアデータベース (SDB : Software Database)，アクセスルーチン (AR : Access Routines) から構成される．

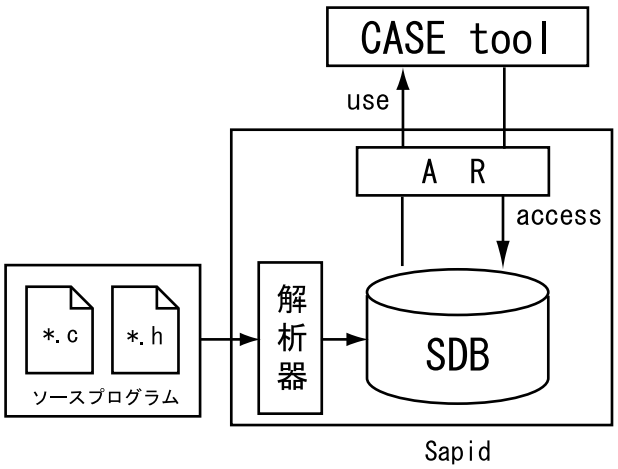

図 2 Sapid の概略

- SDB は Sapid の基盤であり，要求されたソフトウェアをソフトウェアモデルに基づいて解析する解析器と解析の結果得られた実体・関連情報を格納するデータベースからなる．
- AR は，SDB のデータにアクセスするための C 言語の API である．

図 2 に Sapid の概略を示す．本システムでは，SDB に格納されている前処理された C 言語ソースプログラムに対するソフトウェアモデルからミュータントを生成する．

3.2 本システムで扱うミューテーションタイプ

本システムでミューテーションの対象となるのは，代入式の右辺の算術式と，if 文の条件式の算術式である．while 文，do-while 文，for 文などの繰り返し文の条件式からミュータントを生成すると，無限ループに陥ってしまう可能性が高くなってしまうので，現段階の本システムではこれらの条件式を対象としていない．

以下に本システムで扱うミューテーションタイプを示す．

- 変数を他の変数に置き換える．
- 定数を他の定数に置き換える．
- 算術演算子を他の算術演算子に置き換える．
- 関係演算子を他の関係演算子に置き換える．

式 `a < b + 4` に対する各ミューテーションの例を表 3 に示す．この式は変数 `a, b, c` のスコープ内にあるものとする．それぞれのミューテーションタイプの規則，具体的方法を以下に示す．

a） 変数置き換え VR (Variable Replacement)

変数置き換えでは，算術式に現れる変数を他の変数に置き換える．置き換えられる変数の型は，算術式に多用される int 型，float 型，double 型とする．置き換える変数は，その式からのスコープ内にある同じ型の変数とする．

b） 定数置き換え CR (Constant Replacement)

定数置き換えでは，置き換えられる定数は整数型とする．置き換えられる定数を C とすると，置き換える定数の候補は 0,

表 3　ミューテーション例

式	タイプ	ミュータント
a < b + 4	VR	b < b + 4,　c < b + 4,　a < a + 4, a < c + 4
	CR	a < b + 0,　a < b + 5,　a < b + 3
	AOR	a < b - 4,　a < b * 4,　a < b / 4, a < b % 4
	ROR	a == b + 4,　a != b + 4,　a <= b + 4, a > b + 4,　a >= b + 4

$C+1$, $C-1$ とする．例えば，元の定数が 3 であれば，置き換える候補は 0, 4, 2 である．

c） 算術演算子置き換え AOR (Arithmetic Operator Replacement)

算術演算子置き換えでは，算術演算子を他の算術演算子に置き換える．ここでの算術演算子は+, -, *, /, %とする．例えば，置き換えられる演算子が+であれば，-, *, /, %に置き換える．

d） 関係演算子置き換え ROR (Relational Operator Replacement)

関係演算子の置き換えでは，関係演算子を他の関係演算子で置き換える．ここでの関係演算子は，>, >=, <, <=, ==, !=とする．例えば，置き換えられる演算子が>であれば，>=, <, <=, ==, !=に置き換える．

3.3　ミュータント生成システムの実現

本システムで生成されるミュータントは，オリジナルプログラムの一ヶ所を変更したものである．複数ヶ所を同時に変更したミュータントは生成しない．これは複数の変更を含んだミュータントから生成されるミュータントプログラムのテスト時に，そのミュータントが摘出される可能性が高くなるからである．例えば，テストセットがミュータント M_1 を摘出できるが，ミュータント M_2 は摘出できない場合，ミュータント M_1 とミュータント M_2 を同時に含むミュータントプログラムからそのミュータントを摘出できる可能性がある．しかし，そのテストセットではミュータント M_2 だけを含むミュータントプログラムは摘出できない．プログラマが複数の間違いを同時にする場合があっても，そのうちのそれぞれの間違いを発見できるテストセットがあれば，プログラマに間違いを気付かせることができると考えられる．

本システムは C 言語で作成しており，トータルで 1417 行である．対象となるオリジナルプログラムからミュータント生成システムを使用して，ミュータントを生成する手順を以下に示す．

1. 解析対象プログラムを SDB へ格納.
2. SDB に格納されたプログラムからミューテーションの対象となる式の取得.
3. 取得した式に変数置き換え，定数置き換え，算術演算子置き換え，関係演算子置き換えのうち可能なミューテーションを行う.
4. 2, 3 を繰り返す.

図 3 にミュータント生成システムの概略図を提示する．図中

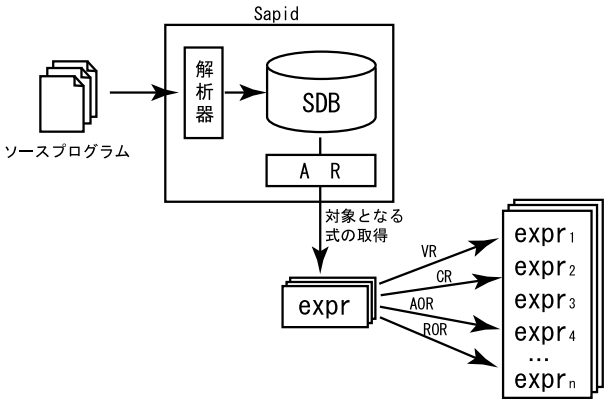

図 3　ミュータント生成システム

の **expr** は本システムで対象となる式を表し，$\mathbf{expr}_i$ は **expr** から各ミューテーションを行った結果生成されるミュータントを表す.

4.　ミュータントプログラムテストシステム

4.1　概　　要

ミュータント生成システムで生成したミュータントからミュータントプログラムを生成して，対象プログラムのテストセットでテストを行う．そして，テストセットの品質の評価するためのミューテーションスコアを測定する．

4.2　システムの実現

本システムは，ミューテーションスコアを測定するために 2 つのフェーズから成り立つ．また，本システムはシェルスクリプトで実装されており，200 行程度のものである．

a） ミュータントプログラムの生成

3 節のミュータント生成システムで生成されたミュータントを spdSed を使用してミュータントを含んだソースプログラムを生成する．spdSed は Sapid アプリケーションで，ファイルを編集コマンド群に従って編集するストリームエディタである．そして，ミュータントを含んだソースプログラムをコンパイルすることによりミュータントプログラムを生成する．

b） ミュータントプログラムのテスト

ミュータントプログラムを対象プログラムのテストセットでテストする．ミュータントプログラムと元のプログラムの各テストケースでのテスト結果を比較して，ミュータントが摘出されたかどうかを調べる．ミュータントプログラムのテスト時に余計に時間がかかってしまうことがある．この場合，正常値のテスト時にかかる時間の 2 倍以上の時間がかかったものは，強制的にそのテストを打ち切るようにした．このことについては，4.4 の考察で詳しく述べる．

本システムではミュータントの摘出を以下の場合とする．

- オリジナルプログラムのテスト結果と異なる.
- テスト時間がオリジナルプログラムのテスト時間に比べて 2 倍以上かかる.

以上のフェーズを生成されたミュータントの数だけ繰り返す．全てのミュータントプログラムに対するテストが終了した時

表 4　gawk-3.0.1, gzip-1.2.4 のプログラムの概要

	gawk-3.0.1	gzip-1.2.4
ファイル数 (.c)	19	14
ファイル数 (.h)	10	6
行数	23,638	8,163
サイズ [KB]	896	304
テストセットの数	40	2

表 5　gawk-3.0.1, gzip-1.2.4 のミューテーションスコア

	gawk-3.0.1	gzip-1.2.4
生成されたミュータント	8,569	3,036
コンパイル失敗	928	33
摘出したミュータント	2,718	627
結果が異なる	2,490	615
強制終了	228	12
ミューテーションスコア	35.6%	20.9%
SDB 生成時間	5m12s	56s
ミュータント生成時間	56s	10s
正常値のテスト時間	2.5s	0.15s
テスト時間	686m2s	91m38s

に，生成されたミュータントの総数，摘出されたミュータントの数からミューテーションスコアを計算する．ミュータントプログラムがコンパイル時にエラーを出力する場合がある．このとき，このミュータントはミューテーションスコアを測定する際には考えない．また，正確なミューテーションスコアを測定するためには等価ミュータントの数も必要となる．しかし，ミュータントが等価か否かは人手でなければ判断できないので現段階の本システムでは考えないものとする．すなわち，プログラム P のテストセット T のミューテーションスコア $MS(P,T)$ の計算方法を以下に示す．

$$MS(P,T) = \frac{K}{M-C} \times 100 \quad [\%] \qquad (3)$$

生成したミュータントの数を M，摘出されたミュータントの数 K，コンパイル時にエラーとなるミュータントの数を C とする．

4.3　ミュータントプログラムテストシステムの実行結果

gawk-3.0.1, gzip-1.2.4 に本システムを適用した．それぞれのプログラムの概要を表 4 に示す．

それぞれのテストは，Makefile に記載されたテストの規則に従い行ったものである．以下の動作環境で gawk-3.0.1, gzip-1.2.4 のテストセットのミューテーションスコアを測定した．

OS　：　Vine Linux 2.5
CPU　：　Pentium Ⅲ 800MHz
メモリ　：　128 MB

結果を表 5 に示す．表中の正常値のテスト時間については，オリジナルプログラムを 50 回テストしたうちの最大時間である．また，テスト時間は全てのミュータントプログラムのテストに要した時間である．

4.4　考　　察

今回，gawk-3.0.1 と gzip-1.2.4 のテストセットについてミューテーションスコアを測定した．それぞれのミューテーションスコアは 36%，21%であった．

本システムではミュータントプログラムのテスト時間がオリジナルプログラムのテスト時間の 2 倍以上かかるものについては強制的にそのテストを打ち切っている．gzip-1.2.4 の正常値のテスト時間は約 0.15s であった．そこで，ミュータントプログラムのテスト時間を 0.15s から 0.15s の 3 倍となる 0.45s まで 0.03s ずつ増やして，摘出できるミュータントについて調べてみた．その結果，打ち切り時間を 0.15s とした場合に摘出されるミュータントの数は打ち切り時間を増やしても変化することはなかった．

ミューテーションスコアは，摘出できるミュータントの数が多ければ高くなる．明らかに間違っているようなミュータントが多く生成された場合，摘出できるミュータントの数が多くなる可能性が高くなり，ミューテーションスコアが高くなる．この場合，テストケースの品質がいいものだとは一概に言えない．

ミュータントは，多くのプログラマが間違えやすい形になってるのが理想的である．よって，ミューテーションもそのようなミュータントを生成できる規則になっているものが望ましいと考えられる．

5.　おわりに

本稿では，オープンソースプログラムのテストセットの品質を評価する手助けとなるミューテーションスコアを測定するシステムを提案した．ミューテーションスコアを測定するために，次の 2 つのシステムを作成した．

- ミュータント生成システム
- ミュータントプログラムテストシステム

これらによって，gawk-3.0.1 と gzip-1.2.4 のテストセットのミューテーションスコアを測定した．この測定値から，ユーザがテストセットに改良を加える必要があるかどうかを判断する手助けになる．

しかし，現在はミューテーションスコアを測定するだけである．今後，ミューテーションタイプを拡張して，多くのプログラマが犯しやすい間違いを想定したミュータントを生成できるようにしたい．また，gawk-3.0.1, gzip-1.2.4 以外のオープンソースプログラムのテストセットのミューテーションスコアを測定して，ミューテーションスコアの基準値を求めていきたい．

謝　　辞

御指導頂く，名古屋大学大学院情報科学研究科 阿草清滋教授，愛知県立大学情報科学部 稲垣康善教授に感謝します．

文　　献

[1] Richard A.DeMillo and A.Jefferson Offutt. “Experimental Results from an Automatic Test Case Generator.” ACM Transaction on Software Engineering Methodology, Vol2, pp.109-175, April 1993.

[2] Roland Untch, A.Jefferson Offutt, and Mary Jean Harrold. “Mutation Analysis Using Mutant Schemata.” ACM International Symposium on Software Testing and Analysis, pp.139-148, June 1993.

[3] Richard A.DeMillo and A.Jefferson Offutt. “Constraint-based automatic test data generation.” IEEE Transactions on Software Engineering, 17(9), pp.900-910, September,

1991.
[4] “Sapid” URL: http://www.sapid.org/
[5] 福安 直樹, 山本 晋一郎, 阿草 清滋. “細粒度ソフトウェア・リポジトリに基づいた CASE ツール・プラットフォーム Sapid.” 情報処理学会論文誌, Vol.39, No.6, pp.1990–1998 (1998/6)