日本ソフトウェア科学会第 27 回大会 (2010 年度) 講演論文集

シーケンススライシング技術を用いたオブジェクト指向テストのバグ原因推測手法の提案

森 淳郎　上田 賀一

現在のソフトウェア開発において，テストは品質に対し非常に深く関わっており，テスト工程の効率化は重要な問題である．特にオブジェクト指向で開発されたソフトウェアはメッセージ通信で動作しているため，エラーの発生原因であるバグ内在メソッドを発見することが難しい．そこで，本研究では実行履歴を分析することによりバグ内在メソッドを推定する手法を提案する．複数のテストケースから，正常系 (仕様と合致しているテストケース) の実行履歴と，異常系 (仕様と矛盾しているテストケース) の実行履歴をアスペクト指向によって取得する．シーケンススライシング技術を用い，これらの実行履歴からメソッドシーケンス及び呼び出し関係を比較し，バグ内在メソッドを推定する手法である．

1 はじめに

近年のソフトウェア開発は複雑化，大規模化が進んでいるが，それに反して短納期化，低コスト化，高品質化が望まれている．また，ソフトウェア開発において，テスト工程は全工程の約半分を占める場合もあり [1]，テスト工程の効率化は重要な問題である．そして，ソフトウェア開発は手続き型言語からオブジェクト指向言語へと移行しており [2]，実用的なソフトウェアをオブジェクト指向言語で開発するには，ソフトウェアの信頼性を確保するためのテストが必要である [3]．オブジェクト指向で開発されたソフトウェアはメッセージ通信によって動作することが特徴である [4]．メッセージ通信が頻繁に行われることで，エラーが発生した場合，どのオブジェクト間での通信からエラーが発生しているかを特定することは難しい．現行の解決法として，ステップ実行が挙げられる [5]．命令を順番に実行し，変数の値を確認する方法であり，確実ではあるが，多くの時間と労力を要する．

A Technique of Bugs Speculation for Object-Oriented Software Testing Using Sequence Slicing.

Junro Mori　Yoshikazu Ueda, 茨城大学大学院理工学研究科情報工学専攻, Major in Computer and Information Sciences,Graduate School of Science and Engineering,Ibaraki University.

そこで，本研究では実行履歴を分析し，バグの原因箇所を推測する手法を提案する．メソッド及びメソッドシーケンススライスの出現回数という定量的なバグ原因推測手法により，分析の自動化が可能なことから効率的にテストすることができる．これにより，バグ原因を推測するための作業人員や時間といったコストを削減できると考える．

本研究は先行研究である実行履歴情報を用いたオブジェクト指向ソフトウェアテスト手法 [6] の改良である．先行研究は，まずアスペクト指向 [7] を用いて実行履歴を取得する．取得された実行履歴は仕様と合致しているテストケースである正常系と，仕様と合致していないテストケースである異常系に分けられる．それから，異常系全てのテストケースと正常系全てのテストケースにおいて差分を取り，最も全体的な差分が少ない一対の正常系のテストケースと異常系のテストケースにおいて，差分を分析する．そして，差分個所に関連するメソッドにバグがあるのではないかと推測する手法である．以前の手法には問題点がいくつか存在し，本手法において以下のように改善している．以前の手法は正常系テストケースと異常系テストケースにおいて，最も近似した一対のテストケースのみを用いるのに対し，本研究における手法では全てのテストケースを用いる．これにより，異常系に近似

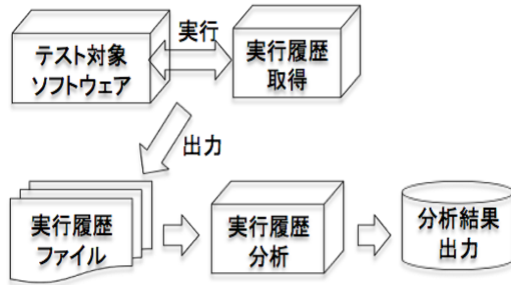

図 1　手法概要

した正常系テストケースが存在しない場合でも分析が可能となっている．また，以前の手法はユーザ操作単位で分析していた．つまり，前のユーザ操作によって次のユーザ操作でエラーが発生した場合，ユーザ操作を超えたエラー原因の推測は不可能であった．しかし，本手法ではテストケースにおけるメソッドシーケンス全体を分析しているため，ユーザ操作間を超えたエラー原因推測も可能である．以前の手法は動的優先度数と静的優先度数を決定する必要があった．しかし，これらの決定方法は多くの実験を通して導出する必要のあるものであったため，多くのテスト対象ソフトウェアが必要であった．本手法ではメソッド及びメソッドシーケンススライスの出現回数という一意的な数値を用いるだけなので，そのような必要もない．

2 節で本手法を説明し，3 節で実際に本手法を適用した例を示す．

2　提案手法

本手法はテスト対象ソフトウェアをテストケースに従って実行し，実行によって得られた実行履歴ファイルを分析し，分析した結果を開発者に提示する．本手法の簡単な流れを図 1 に示す．

2. 1　実行履歴取得

本研究において，実行履歴を解析対象として扱うので，実行履歴を取得する．実行履歴を取得する際に，アスペクト指向を用いる．アスペクト指向とは，オブジェクト指向と異なる技術であり，オブジェクト指向ではうまくモジュール分割できない場合に利用される．本研究で以下の 4 つのポイントカットを用いる．図 2 に実行履歴を取得するためのアスペクトクラスの一部を示す．allExeLogMethod() は全ての実行の際に，allCallLogMethod() は全ての呼び出しの際に，getAllVaryLog() は全ての参照の際に，setAllVaryLog() は全ての代入の際に呼び出される．within はアスペクト対象クラスを表し，“!” を指定することで否定形を表す．これによって Aspect クラス及びログ出力に関係するクラスに対して Aspect を織り込まないことにより，検査対象ソフトウェアの純粋な実行履歴を取得することが可能である．よって，本研究で用いたアスペクトクラスである GetLog クラスと，実行履歴をテキストファイルで加工して出力する FileIO クラスにはアスペクトを用いないことを示している．

```
pointcut allExeLogMethod():execution(* *.*(..))
    &&!within(GetLog)&&!within(FileIO);
pointcut allCallLogMethod():call(* *.*(..))
    &&!within(GetLog)&&!within(FileIO);
pointcut getAllVaryLog():get(* *.*)
    &&!within(GetLog)&&!within(FileIO);
pointcut setAllVaryLog():set(* *.*)
    &&!within(GetLog)&&!within(FileIO);
```

図 2　AspectJ による実行履歴出力クラス (一部)

前述した allExeLogMethod() を利用し，実行履歴のみを抽出した例を図 3 に示す．各行は “パッケージ名．クラス名，メソッド名” を示している．

```
controll.ActionNum,actionPerformed
view.Screen,getField
view.Screen,getNumSetFlg
view.Screen,getField
controll.ActionFunc,actionPerformed
view.Screen,getModel
view.Screen,getField
```

図 3　実行履歴情報 (一部)

図 3 のようなメソッドシーケンスから構成メソッドをスライシングしていき，分析する．詳細は 2.2 節

に記述する．

2.2 メソッドシーケンススライシング

テストケースごとに実行履歴におけるメソッドの流れを切り出し，正常系および異常系における共通部分を導き，比較する．テストケースから共通部分を抽出する概要図を図 4 に，共通部分における比較の概要図を図 5 に示す．

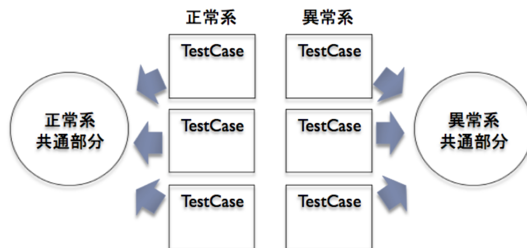

図 4 共通部分抽出概要図

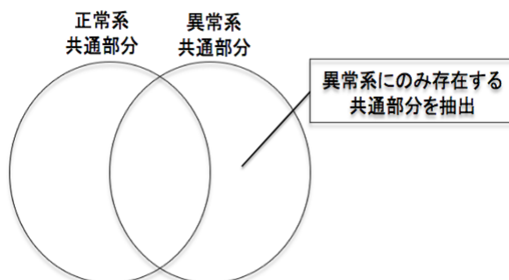

図 5 比較概要図

メソッド A，メソッド B というメソッドがあった場合，以後それぞれ M_AM_B と表記する．例として $M_AM_BM_CM_D$ というメソッドシーケンスを定義する．例から抽出されるメソッドの流れの一部分，つまりメソッドシーケンススライスは $M_AM_BM_C$，$M_BM_CM_D$，M_AM_B，M_BM_C，M_CM_D，M_A，M_B，M_C，M_D である．以下，メソッドシーケンスを MS(Method Sequence)，メソッドシーケンススライスを MSS(Method Sequence Slice) と記述する．MSS の数は，MS が n メソッドならば，MSS は $\frac{n(n+1)}{2}-1$ 存在する．また，メソッドの流れから切り出すにあたり，メソッドの呼び出し関係も抽出する．呼び出し関係は直接呼び出しでなく，間接的な呼び出し関係も抽出する．これは，データベースや通信モジュールにおいて接続確立と接続解除の呼び出し関係を，直接呼び出しのみを対象としてしまうと抽出できない可能性が高いからである．接続を確立した後にデータベースにおける内部処理を行う場合や，通信モジュールの接続確立後に，内部処理を行い接続解除した場合，直接呼び出しだけを抽出してしまうと，接続確立から内部処理，内部処理から接続解除の二系統しか抽出することができない．よって，接続を確立したが解除していない，または接続を確立していないのに解除しているといった呼び出し関係が抽出できない．ゆえに，本研究では直接呼び出し関係だけでなく，間接的な呼び出し関係も抽出する．$M_AM_BM_CM_D$ という MS の場合，呼び出し関係は M_AM_B，M_AM_C，M_AM_D，M_BM_C，M_BM_D，M_CM_D の 6 つ存在する．MS が n メソッドならばメソッド呼び出し関係の数は $\frac{n(n-1)}{2}$ 存在する．以下，メソッドの呼び出し関係を MCR(Method Call Relation) と記述する．また，表においてテストケースを TC(TestCase) と記述し，TC は正常系を，TC' は異常系を示している．

以下に例における MS の各メソッドにおける条件を示す．

1. M_C が実行される場合にはその前に M_B が呼び出されていなければならない．
2. M_B は単独でも実行できるが，M_A が実行される場合には M_B の前に実行されなければならない．
3. M_F はバグを内包している．

以下に算出手順を示す．

1. それぞれのテストケースにおける MS から MSS 及び MCR を抽出する．
2. 正常系の MSS 及び MCR の出現回数を正常値とし，異常系の出現回数を異常値として求め，それぞれを合計した値を信頼値とする．この際，正常値を正，異常値を負とする．
3. 信頼値ごとに MSS 及び MCR の数を集計する．
4. 最小信頼値における MSS と MCR，及び異常系にのみ存在する MSS と MCR を提示する．以

降の MSS 及び MCR に付記している括弧内の値は信頼値である．

最初に，例における MS から MSS 及び MCR を抽出する．抽出した結果を表 2.2 に示す．

表 1 信頼値計算例

TestCase	MS	MSS	MCR
TC_1	$M_AM_BM_CM_D$	$M_AM_BM_C$	M_AM_B
		$M_BM_CM_D$	M_AM_C
		M_AM_B	M_AM_D
		M_BM_C	M_BM_C
		M_CM_D	M_BM_D
		M_A,M_B,M_C,M_D	M_CM_D
TC_2	$M_AM_BM_CM_E$	$M_AM_BM_C$	M_AM_B
		$M_BM_CM_E$	M_AM_C
		M_AM_B	M_AM_E
		M_BM_C	M_BM_C
		M_CM_E	M_BM_E
		M_A,M_B,M_C,M_D	M_CM_E
TC_3	$M_BM_CM_DM_E$	$M_BM_CM_D$	M_BM_C
		$M_CM_DM_E$	M_BM_D
		M_BM_C	M_BM_E
		M_CM_D	M_CM_D
		M_DM_E	M_CM_E
		M_B,M_C,M_D,M_E	M_DM_E
$TC_1^{'}$	$M_AM_CM_DM_E$	$M_AM_CM_D$	M_AM_C
		$M_CM_DM_E$	M_AM_D
		M_AM_C	M_AM_E
		M_CM_D	M_CM_D
		M_DM_E	M_CM_E
		M_A,M_C,M_D,M_E	M_DM_E
$TC_2^{'}$	$M_BM_AM_CM_D$	$M_BM_AM_C$	M_BM_A
		$M_AM_CM_D$	M_BM_C
		M_BM_A	M_BM_D
		M_AM_C	M_AM_C
		M_CM_D	M_AM_D
		M_B,M_A,M_C,M_D	M_CM_D
$TC_3^{'}$	$M_AM_BM_CM_F$	$M_AM_BM_C$	M_AM_B
		$M_BM_CM_F$	M_AM_C
		M_AM_B	M_AM_F
		M_BM_C	M_BM_C
		M_CM_F	M_BM_F
		M_A,M_B,M_C,M_F	M_CM_F

次に，正常系の MSS 及び MCR は出現回数ごとに正常値を加算し，異常系には出現回数ごとに異常値を加算し，信頼値を求める．MSS 信頼値を表 2.2 に，また，MCR 信頼値を表 2.2 に示す．そして，全ての MSS 及び MCR の信頼値ごとの出現数をそれぞれ表 2.2 と表 2.2 に示す．MSS 及び MCR はシーケンススライシングで生成された順に示す．

表 2 MSS 信頼値

MSS	信頼値
$M_AM_BM_C$	2+(-1)=1
$M_BM_CM_D$	2+0=2
M_AM_B	2+(-1)=1
M_BM_C	3+(-1)=2
M_CM_D	2+(-2)=0
$M_BM_CM_E$	1+(0)=1
M_CM_E	1+(0)=1
$M_CM_DM_E$	1+(-1)=0
M_DM_E	1+(-1)=0
$M_AM_CM_D$	0+(-2)=-2
M_AM_C	0+(-2)=-2
$M_BM_AM_C$	0+(-1)=-1
M_BM_A	0+(-1)=-1
$M_BM_CM_F$	0+(-1)=-1
M_CM_F	0+(-1)=-1

以上から $TC_1^{'}$，$TC_2^{'}$，$TC_3^{'}$ それぞれのバグ原因を推測する．

$TC_1^{'}$ の MSS において，最小信頼値における MSS は

- $M_AM_CM_D$(-2)
- M_AM_C(-2)

そしてそれぞれ正常系にはなく，異常系においてのみ存在する MSS である．

MCR においては

- M_AM_C(-1)
- M_AM_D(-1)

M_AM_C は異常系においてのみ存在する MCR である．これは条件 1 に該当しないことを導いている．

$TC_2^{'}$ の MSS において，異常値を示しているのは

- $M_BM_AM_C$(-1)
- $M_AM_CM_D$(-2)
- M_BM_A(-1)
- M_AM_C(-2)

MCR において，異常値を示している MCR は

表 3　MCR 信頼値

MCR	信頼値
M_AM_B	2+(-1)=1
M_AM_C	2+(-3)=-1
M_AM_D	1+(-2)=1
M_BM_C	3+(-2)=1
M_BM_D	2+(-1)=1
M_CM_D	2+(-2)=0
M_AM_E	1+(-1)=0
M_BM_E	2+0=2
M_CM_E	2+(-1)=1
M_DM_E	1+(-1)=0
M_BM_A	0+(-1)=-1
M_AM_F	0+(-1)=-1
M_BM_F	0+(-1)=-1
M_CM_F	0+(-1)=-1

表 4　MSS における値ごとの出現数

信頼値	全 MSS	正常系 MSS	異常系 MSS
-3	0	0	0
-2	2	0	3
-1	4	0	9
0	3	0	0
1	4	4	0
2	2	4	0
3	0	1	0

- M_BM_A(-1)

この MCR は正常系にはない MCR である．条件 2 に該当していないことを導いている．

TC_3' の MSS において異常値を示しているのは

- $M_BM_CM_F$(-1)
- M_CM_F(-1)

そして上記 2 つの MSS は正常系にはなく異常系にのみ存在する MSS である．

MCR においては

- M_AM_C(-1)
- M_AM_F(-1)

表 5　MCR における値ごとの出現数

信頼値	全 MCR	正常系 MCR	異常系 MCR
-3	0	0	1
-2	0	0	3
-1	5	0	9
0	3	0	0
1	5	3	0
2	2	6	0
3	0	1	0

- M_BM_F(-1)
- M_CM_F(-1)

それぞれ異常系にのみ存在する MCR である．異常系 MSS 及び MCR における構成メソッドに M_F が多く存在していることから，条件 3 を導いている．

以上のように MSS，MCR によって計算された信頼値が低い MSS 及び MSR，または構成メソッドをユーザに示す．

3　適用実験

3.1　電卓

実験例として小規模アプリケーションである電卓システムを取り上げる．以下の規模をもつ簡単な四則演算を行う GUI アプリケーションである．

- コード総数:359
- パッケージ数:4
- クラス数:5
- メソッド数:30

3.1.1　成功履歴と失敗履歴

電卓アプリケーションで用いたテストケースを表 3.1.1 に示す．

3.1.2　エラー原因箇所

適用対象アプリケーション Calc は swing を用いて GUI が作成されている．JButton において乗算を表す “x” を用いて乗算記号を登録しているが，ActionCommand においては乗算記号として “*” を登録している．これにより，乗算を行う際に ActionCommand が呼び出されていないというバグが発生している．Calc は MVC モデルで設計されており，GUI を提供

表 6 電卓におけるテストケース

履歴名	入力	判定
SuccessLog-01	500+500=	OK
SuccessLog-02	500C500+500=	OK
SuccessLog-03	50-50=	OK
ErrorLog-01	500*500=	NG
ErrorLog-02	1+500*500=	NG
ErrorLog-03	10*10*10*10=	NG

する View に対応するように Controller である ActionFunc クラスを設計している．つまり，エラー原因であるバグは ActionFunc クラスの ActionPerformed メソッドに存在している．原因箇所の一部を図 6 に示す．

```
ActionFunc クラス actionPerformed メソッド
if(e.getActionCommand().equals("+"))
if(e.getActionCommand().equals("*"))
```

図 6 エラー原因箇所 (一部)

3.1.3 実行結果

MSS，MCR の信頼値ごとの出現数を表 3.1.3 及び表 3.1.3 に示す．また，異常系にのみ存在する MSS 構成メソッド出現数を表 3.1.3 に，全 MSS，MCR における最小値構成メソッド出現数を表 3.1.3，表 3.1.3 に示す．異常系にのみ存在する MCR は抽出されなかった．

- 全 MSS 数:4811
- 全 MCR 数:212

3.1.4 考察

異常系のみに存在する MCR がなかったのは，メソッド数が 30 しかなく，また電卓というソフトの性質上動作が限られており，同じような呼び出し関係しか導けなかったのが原因だと思われる．異常系にのみ存在する MSS にて，バグ内在メソッドである ActionFunc クラスの actionPerformed メソッドが第 6 位に出力されている．アクセサメソッド以外だと第 3 位に出力されている ActionNum クラスの actionPerformed メソッドがある．ActionFunc クラスの actionPerformed メソッドは電卓における四則演算を入力した際に呼び出され，ActionNum クラスの actionPerformed メソッドは数字キーを入力した際に呼び出されるメソッドであり，今回のテストケースのように 500+500 といった四則演算キーよりも数字キーの入力が大幅に多いため，ActionNum クラスにおける actionPerformed メソッドが上位に検出さ

表 7 信頼値ごとの MSS 出現数

信頼値	全 MSS	正常系 MSS	異常系 MSS
-3	117	0	285
-2	143	0	248
-1	1902	0	2328
0	310	0	0
1	2003	1815	0
2	292	458	0
3	44	376	0

表 8 信頼値ごとの MCR 出現数

信頼値	全 MCR	正常系 MCR	異常系 MCR
-3	0	0	119
-2	0	0	0
-1	0	0	56
0	119	0	0
1	53	37	0
2	40	16	0
3	0	159	0

表 9 異常系にのみ存在する MSS 構成メソッド出現数 (上位 6)

出現数	メソッド
23206	view.CalcScreen,getField
13007	view.CalcScreen,getNumSetFlg
9431	controll.ActionNum,actionPerformed
7068	view.CalcScreen,setNumSetFlg
4124	view.CalcScreen,getModel
4117	controll.ActionFunc,actionPerformed

表 10　全 MSS における最小値 (-3) 構成メソッド出現数

出現数	メソッド
544	view.CalcScreen,getField
296	view.CalcScreen,getNumSetFlg
217	controll.ActionNum,actionPerformed
172	view.CalcScreen,setNumSetFlg
110	view.CalcScreen,getModel
99	controll.ActionFunc,actionPerformed

表 11　全 MCR における最小値 (0) 構成メソッド出現数 (上位 6)

出現数	メソッド
26	view.CalcScreen,setNumSetFlg
20	view.CalcScreen,getField
20	view.CalcScreen,getNumSetFlg
20	controll.ActionFunc,actionPerformed
20	view.CalcScreen,getModel
20	controll.ActionNum,actionPerformed

れたと考えられる．最も精度が高いと考えられる結果は，全ての MSS における異常値が最も高い MSS を構成しているメソッドにおいて 6 つのメソッドが検出され，バグ内在メソッドも含まれている．これによって全てで 30 のメソッドから 6 つに絞られ，また，この 6 つの中でアクセサメソッドが 4 つを占めているため，実質 2 つに絞られていると言える．

3.2　ドキュメント管理システム

実験対象としてドキュメント管理システムに本ツールを適用した．この適用例については適用対象の概要と考察のみ述べる．このシステムは授業で学生が作成したものであり，サーバクライアント方式のドキュメント管理システムで，ブラウザ上で動作する．データベースとして MySQL を用いており，言語は jsp/Servlet を用いてサーバクライアント方式を実現させている．ログインの際に，ユーザ ID とパスワード，もしくはフェリカ ID とパスワードでのログインが可能であり，管理者と利用者でフィルタを掛けているので，システム利用範囲を制限している．主な機能として，管理者側機能はユーザの登録，閲覧，修正，削除，検索があり，利用者側機能はドキュメントの登録，閲覧，修正，削除，検索が可能である．

- コード総数:4284
- パッケージ数:15
- クラス数:75
- メソッド数:234

3.2.1　成功履歴と失敗履歴

用いたテストケースを表 3.2.1 に示す．成功履歴か失敗履歴かは判定を参照する．

表 12　ドキュメント管理システムにおけるログイン機能テストケース

履歴名	入力	判定
SuccessLog-01	登録済 ID&Pass	OK
ErrorLog-01	登録済 FelicaID&Pass	NG
SuccessLog-02	未登録 ID&Pass	OK
SuccessLog-03	未登録 FelicaID&Pass	OK
SuccessLog-04	登録済 ID&誤った Pass	OK
SuccessLog-05	登録済 FelicaID&誤った Pass	OK

3.2.2　エラー原因箇所

ログインの際に，ユーザ ID とパスワードが一致するか，もしくはフェリカ ID とパスワードが一致するユーザがデータベース上に存在するかどうか検索を行い，合致した場合ログインできるよう実装されている．ユーザを検索するメソッド内での sql を作成しており，フェリカ ID を検索するはずの sql が，間違ってユーザ ID を検索してしまっているためログインできないバグが発生している．よって，バグは user．DAO パッケージの SearchDAO クラスの findUserByFelicaId メソッドに存在している．図 7 に原因箇所の一部を示す．

3.2.3　実行結果

MSS，MCR の信頼値ごとの出現数を表 3.2.3 及び表 3.2.3 に示す．また，異常系にのみ存在する MSS，MCR 構成メソッド出現数を表 3.2.3 及び表 3.2.3 に示す．全 MSS，MCR における最小値構成メソッド

```
SearchDAO クラス findUserByFelicaId メソッド
String sql = "select * from user
    where user_id = ’" + setUserId + "’";
```

図 7 エラー原因箇所 (一部)

出現数はそれぞれ表 3.2.3 と表 3.2.3 と同じ結果となったので省略する．

- 全 MSS 数:1143
- 全 MCR 数:383

表 13 信頼値ごとの MSS 出現数

信頼値	全 MSS	正常系 MSS	異常系 MSS
-1	292	0	568
0	178	0	0
1	469	592	0
2	103	158	0
3	58	58	0
4	43	0	0
5	0	43	0

表 14 信頼値ごとの MCR 出現数

信頼値	全 MCR	正常系 MCR	異常系 MCR
-1	29	0	309
0	172	0	0
1	68	215	0
2	16	31	0
3	26	36	0
4	72	0	0
5	0	72	0

3.2.4 考察

異常系にのみ存在する MSS 構成要素出現数において，バグ内在メソッドである findUserByFelicaId が 3 位に示されており，MCR においては 1 位に示され，出現回数 20 と 2 位の setFelica_id メソッドの 9 回に対して 2 倍の数値という顕著な結果が出力された．全 MSS，MCR においても同様である．今回，MCR に

表 15 異常系にのみ存在する MSS 構成メソッド出現数 (上位 5)

出現数	メソッド
695	user.Bean.User.isManager
586	common.ConnectionDAO.createConnection
512	user.DAO.SearchDAO.findUserByFelicaId
364	user.Bean.User.getPassword
336	user.Bean.User.getFelica_id

表 16 異常系にのみ存在する MCR 構成メソッド出現数 (上位 5)

出現数	メソッド
20	user.DAO.SearchDAO.findUserByFelicaId
9	login.AuthBean.setFelica_id
3	common.Controller.init
3	common.Domain.getDomain
3	common.Controller.doGet

おいて抜きん出た結果を示したのは，異常系が 1 つに対し，正常系が 5 つ，そしてログイン系のテストケース全体の誤差が低いため，異常系特有のメソッドが抽出できたと考える．また，異常系，正常系のみに存在する構成メソッドと，最小値を構成しているメソッドのどちらも同じ結果となった．これは，正常系テストケースの MSS 及び MCR が，異常系テストケースにおける MSS 及び MCR を完全に網羅できず，異常系における特有の MSS，MCR が存在したのが原因であると考える．このように，異常系にのみ存在する MSS，MCR は非常にバグの原因に近い結果を有していると考える．

3.3 二つの適用例からの考察

電卓における全 MSS 数は 4811 であり，全 MCR 数は 212 である．ドキュメント管理システムにおいては全 MSS 数は 1143 であり，全 MCR 数は 383 である．電卓に対してドキュメント管理システムはコード総数，パッケージ数，クラス数，メソッド数全てにおいて多いが，全 MSS 数は電卓のほうが高い．これ

は，電卓が多くのユーザ操作を受けて動作していることが原因と考えられる．ドキュメント管理システムにおけるログイン機能は ID とパスワードを入力し，ログインボタンを押すだけであり，あとはシステム側の処理である．これにより，全 MSS 数において電卓の方が多くなったと考えられる．また，全 MCR 数においてドキュメント管理システムが高いのは，電卓のメソッド数が 30 に対し，ドキュメント管理システムにおけるメソッド数が 234 であることが原因と考えられる．前述したとおり，電卓のテストケースにおいて全 MSS 数は高いが，その中で同じメソッドを何度も実行しており，ドキュメント管理システムのメソッドはメソッドが多く，多くの MCR が取得されたと考える．

バグ内在メソッド検出率において，電卓よりもドキュメント管理システムのほうが上位に検出された．これは，電卓における動作メソッドがドキュメント管理システムよりも非常に少ない点にあると考える．電卓では数字キーを押下し，四則演算キーを押下し，結果出力する，という主に 3 つのユーザ操作に伴ってメソッドが実行される．これに対し，ドキュメント管理システムは ID，パスワードを入力した後にログインボタンを押すと，サーバ側とのセッションを確立し，サーバ側のデータベースにアクセスし，同一の ID があるか，そしてその ID とパスワードの二つが正しいか，そして正しければページを遷移させると言った多くの処理が伴う．また，通常の ID とフェリカ ID をデータベースで問い合わせる際のメソッドは異なる．

以上の結果から，本手法はオブジェクト指向で開発され，ある程度の規模があり，適切にクラス分けされたソフトウェアに対してより効果的であると考える．つまり，大規模で複雑なソフトウェアほど効果的である．規模が小さい，もしくは適切にクラス分けされておらず，全ての機能が全てのクラス，メソッドを実行する場合，本手法は異常系に関連の強いメソッドを抽出することはできない．しかし，規模が大きい，もしくは適切にクラス分けされているソフトウェアならば，機能ごとに利用するクラス，メソッドの遷移が変われば，異常系に関連の強いメソッドを容易に抽出し，異常系にのみ存在する MSS，MCR を示すことが可能である．

4 関連研究

オブジェクト指向ソフトウェアに対するテスト手法として複数の UML ダイアグラムを用いてテストする研究が挙げられる [8]．この研究では，テストの際に UML を利用することにより，Java で記述されたプログラムの信頼性を向上させることを目的としている．UML のクラス図，シーケンス図，ステートチャート図の 3 つのダイアグラムと Java ソースコードとの対応関係を抽出し，テスト設計に利用することによってプログラムの構造，システムの処理の流れと状態間の遷移の流れといったソフトウェアの仕様を効果的にテストすることが可能と締め括っている．本研究では UML は取り扱っておらず，テスト仕様書から要求仕様と矛盾が生じていないかを判断しており，開発に UML を用いていない場合でも適用が可能という点で異なっている．

また，別の研究としてオブジェクト指向ソフトウェアテストのためのカラーペトリネットスライシング技術 [9] という研究が挙げられる．この研究では，オブジェクト指向ソフトウェアは実行時に決定される性質を持つため，状態爆発問題を起こし，ソフトウェアの挙動に関するテストや解析を困難にしている問題に対して，カラーペトリネットを用いたテスト方法を提案している．状態図とクラス図で記述した仕様をカラーペトリネットへ変換し，カラーペトリネットのためのスタティックスライシング技術を提案することで，機能ごとにテストすることを可能にしている．機能ごとにテスト可能にすることで，従来よりも効率的に，開発初期段階でのテストを可能にする．本研究ではメソッドごとに信頼値を示しており，機能ごとにテストするという観点から目的が異なっている．

実行履歴情報を利用した研究として，オブジェクト指向プログラムの実行履歴に対する機能単位での自動分割手法 [10] という研究がある．この研究では，オブジェクト指向は動的に決定される要素が多いため，システムの振舞いを理解するためにシーケンス図を可視化する方法を提案している．そして，シーケンス図の可読性を向上するためにキャッシュアルゴリズム

とメソッド呼び出し情報を用いてフェイズ単位で分割する手法を提案している．本研究はバグ原因を推測することを目的としており，システムの振舞いの理解を目的としていない点で異なっている．

5 おわりに

5.1 まとめ

本研究では，オブジェクト指向ソフトウェアに関して実行履歴情報を元にバグ内在メソッドを推定する方法を提案した．この手法を用いることで，バグ内在メソッドを効率的に特定，検査しテスト効率を向上することができると考える．ツールを用いてテストを行うことで，経験や知恵によるエラー発生箇所を特定する必要が無く，定量的にテストを行うことができる．そして，テストを効率的に行うことにより，テストにかかる時間，及びコストを削減することが可能となる．それによって，納期内に十分なテストが可能となり，出荷後に発見されるバグを防ぐ可能性が高くなり，品質をより高いものとすることが可能であると考えている．2 つの適用例においてバグ内在メソッドを上位に検出し，多くのメソッドから候補を絞り込むことに成功した．本手法は成功実行履歴と失敗実行履歴が存在することが前提となっているが，テスト工程でこれらを取得することは当然可能である．

5.2 今後の課題

本研究の改良点，今後の方向性を以下のように考えている．

5.2.1 リファクタリング候補の提示

異常系において，ある呼び出し関係の出現が顕著であれば，呼び出し関係に基づくバグであると考えられる．そのような場合，その呼び出し関係と類似している正常値の高い呼び出し関係を提示すれば，リファクタリング候補に成り得るのではないかと考える．今回の二つの適用例ではバグ内在メソッドが問題であり，呼び出し関係におけるバグではなかったため，提示することができなかった．今後，適用事例を増やし，その中で呼び出し関係に関するバグを取り扱いたいと考えている．

5.2.2 潜在バグの提示

MSS 及び MCR はそれぞれ正常値，異常値情報を保持しており，これをテストケースに合計することにより，正常系であるのに異常値が高い場合，潜在バグがある可能性を示唆できると考えている．

5.2.3 テスト結果の自動判別

全てのテストケースを入力し，その出力が仕様通りか，もしくは異なっているかの判断は利用者に任せている．膨大なテストケースの量を考えると，これはかなりの労力を必要とする．そのため，テスト駆動開発などで用いられているアサーションを用いて成功か失敗かを自動判別できれば，開発者の負担を減らすことができると考えている．

参 考 文 献

[1] 紀本眞，土屋達弘，菊池亨:テスト実行コストを考慮したペアワイズテストセット生成法の提案，電子情報通信学会技術研究報告，Vol.107,No.254,pp.47-50(2007).
[2] 小倉崇，藁谷大輔，櫻井孝平，古宮誠一:オブジェクト指向プログラムのためのデバッグ/テスト支援システム-プログラムトレーサー機能について-，電子情報通信学会技術研究報告，Vol.104,No.588,pp.7-12(2005).
[3] 古川善吾，梅田修一，片山徹郎，伊藤栄典，牛島和夫:オブジェクト指向プログラムのテスト法に関する一考察，情報処理学会研究報告，Vol.94,No.6,pp.123-130(1994).
[4] 小宮誠一，廣田豊彦監訳:オブジェクト指向ソフトウェアテスト技法，共立出版株式会社 (2000).
[5] 町田欣史，高橋和也，小堀一雄:現場で使えるソフトウェアテスト JAVA 編，翔泳社 (2008).
[6] 森淳郎，上田賀一:実行履歴情報を用いたオブジェクト指向テスト手法の提案，電子情報通信学会技術研究報告，Vol.109,No.456,pp.151-156(2010).
[7] 天野まさひろ，鷲崎弘宜，立堀道昭:AspectJ によるアスペクト指向プログラミング入門，ソフトバンク パブリッシング株式会社 (2004).
[8] 数谷悠介，下村希世人，片山徹郎:UML の複数のダイアグラムを用いた Java プログラムのテスト手法に関する一考察，電子情報通信学会技術研究報告，Vol.104, No.466, pp.7-12(2004).
[9] 渡辺晴美:オブジェクト指向ソフトウェアのためのカラーペトリネットスライシング技術，情報処理学会論文誌， Vol.44,No.6,pp.1461-1472(2003).
[10] 渡邊結，石尾隆，井上克郎:オブジェクト指向プログラムの実行履歴に対する機能単位での自動分割手法，電子情報通信学会技術研究報告，Vol.107,No.392,pp.103-108(2007).