派生開発におけるテスト漏れを防止する Difference Statement Coverage 分析法の提案

Proposal of “Difference Statement Coverage Analysis Method” preventing lack of test cases in the derivation development

株式会社デンソークリエイト
DENSO CREATE INC.

○柏原 一雄　白井 正人[1)]　小嶋 秀和[1)]　都築 功[1)]
○Kazuo Kashiwabara　Masato Shirai[1)]　Hidekazu Kojima[1)]　Isao Tsuzuki[1)]
新留 光治[1)]　村上 雅哉[1)]
Mitsuharu Shintome[1)]　Masaya murakami[1)]

Abstract

In derivation development, in order to prevent the release of defect, it is important that we test all changes. It is difficult to detect lack of test cases only by reviewing the test design. In order to detect the lack of test cases, we devised a method to extract the difference of source code, and to measure and analyze the statement coverage for the changed part. The proposed method “Difference Statement Coverage Analysis Method” was applied to actual development, and the effect of preventing test omissions at the changed part was confirmed.

1. はじめに

ソフトウェアの派生開発において，ソースコードの欠陥は「変更漏れ」と「変更誤り」に分類できる.「変更漏れ」に分類される欠陥については，各種影響分析手法や回帰テストを適用することで，欠陥の流出を防止する効果が期待できる.「変更誤り」に分類される欠陥については，変更箇所に対して漏れなくテストを実行することで，欠陥の流出を防止する効果が期待できる.

我々の組織では，テスト設計のレビューを実施し，変更仕様に対して漏れなくテストケースが作成されていることを確認しているが，テストケースの漏れは検出し切れていなかった．ソースコードの変更箇所に関連する仕様が明確に定義されていない場合は，テスト設計のレビューでテストケースの漏れを検出することが難しい.

本研究では，派生開発において,「変更誤り」に分類される欠陥の流出を防止することを目的とし，テスト設計のレビューだけでは検出が困難である「変更箇所に対するテスト漏れ」を検出することを課題とした．課題解決のために,「ソフトウェアの検算」と「テストデバッグ」という理論を参考に，テストカバレッジを利用する方針とした．また，短期間での開発が求められる派生開発において，現実的に実行可能な手法とするため，ソースコードの差分を抽出し，変更箇所に対してテストカバレッジを計測する方針とした.

本稿では,「変更箇所に対するテスト漏れ」を検出するために考案した手法「Difference Statement Coverage 分析法」を説明する．考案した手法は， Difference Statement Coverage の定義とその分析手順を技術要素とする.

考案した手法の有効性を確認するために，実開発において統合テストで，Difference Statement Coverage 分析法を適用し，以下の評価を行った.

- テストが漏れている変更箇所を特定できるか？
- テスト設計のレビューでは検出できなかったテスト漏れを検出できるか？
- 変更箇所に対して不足しているテストケースを追加できるか？

評価の結果，Difference Statement Coverage 分析法により，テスト設計のレビューでは検出

株式会社デンソークリエイト　DENSO CREATE INC.
愛知県名古屋市中区栄 3-1-1　Tel: 052-238-0460　e-mail:kashiwabara@dcinc.co.jp
3-1-1, Sakae, Naka-ku, Nagoya-shi, Aichi, Japan
1) 株式会社デンソークリエイト　DENSO CREATE INC.

【キーワード：】派生開発，変更仕様，テスト設計，レビュー，テストカバレッジ，Statement Coverage

できなかった「変更箇所に対するテスト漏れ」を検出できることが確認できた．提案手法は，派生開発において，「変更誤り」に分類される欠陥の流出を防止する効果が期待できる．

これ以降の本稿の構成は次のとおりである．2 章で現状分析の結果と研究の課題を示す．3 章では，課題解決の参考とした先行研究を示す．4 章では，考案した解決策を提案する．5 章では，提案手法の評価結果と考察を示す．6 章では，まとめと今後の進め方を示す．

本稿で使用する用語を，ソフトウェアテスト標準用語集[1]等を参考に，表 1 のように定義する．

表 1 用語と意味

用語	意味
既存仕様	変更要求により変化しないベースソフトの仕様．
変更仕様	変更要求によりベースソフトから変化する仕様．変更仕様は，仕様の要素（トリガ，入力，出力，事前条件，事後条件，処理等）の変更部分だけを差分で示す．追加する仕様・削除する仕様も変更仕様とする．
変更箇所	ソースコードの追加行・修正行・削除行．コメント・空行は除く．
テストケース	入力値，実行事前条件，期待結果，そして，実行事後条件のセット．
テスト漏れ	テストケースの漏れ．変更仕様に対するテストケースの漏れとソースコードの変更箇所に対するテストケースの漏れの両方を指す．
テスト設計	テスト目的を具体的なテスト条件とテストケースに変換するプロセス．同値分割，境界値分析，CFD，デシジョンテーブル等のテスト技法を活用する．
テスト実装	テストケースを入力に，テストデータを考え出し，テスト手順の開発を行うプロセス．
テスト実行	テスト対象のコンポーネントやシステムでテストを実行し，実行結果を出力するプロセス．

2. 課題設定

2.1 現状分析

(1) 派生開発における欠陥の分類とその対策手法

派生開発[2]において，ソースコードの欠陥は「変更漏れ（追加漏れ，削除漏れも含む）」と「変更誤り（誤って実施された不要な追加，変更，削除も含む）」に分類できる．「変更漏れ」に分類される欠陥については，XDDP[2]，DRBFM[3]，響波及パス分析法[4]といった手法を適用することで，欠陥の流出を防止する効果が期待できる．「変更誤り」に分類される欠陥については，変更箇所に対して漏れなくテストを実行することで，欠陥の流出を防止する効果が期待できる．そのために，不足しているテストケースを検出する手法が必要となる．

(2) テスト設計のレビューで検出が難しいテストケースの漏れ

我々の組織では，全変更仕様に対して漏れなくテストケースが作成されていることを確認するため，テスト設計のレビューを実施している．しかし，ソースコードの変更箇所に関連する変更仕様が明確に定義されていない場合は，テストケースの漏れは検出し切れなかった．

テスト設計のレビューでは，定義されている仕様に対するテスト漏れを検出することはできても，明確に定義されていない仕様に対するテスト漏れを検出することは難しい．定義されている仕様に対しては，テスト設計に段階的詳細化手法[5]を適用することにより，テストケースの全体像を把握しやすくし，レビューでのテストケースの漏れの見逃しを防止できる．

明確に定義されていない仕様に対しては，レビュー結果がレビューアの知識に依存し，テストケースの漏れを確実に検出することは難しい．

(3) テストケースの漏れを誘発する変更仕様

ベースソフトから変化しない既存仕様が，抽象的に表現されるまたは省略されることが，テストケースの漏れを誘発する[6]．テストケースの漏れを誘発する変更仕様の例を次に示す．

・抽象的に表現される仕様：ベースソフトから変化しない条件
例えば，「異常時は，処理中止ではなくリトライするように変更する」というような変更仕様があった場合，「異常時」という条件は変化しない．変更仕様では，「異常時」には具体的にどのような種類があるのかは示されないことがある．
・省略される仕様：全機能・処理で共通の仕様
例えば，新たな機能が追加されても，「キャンセル操作によりすべての処理を中止する」というような全機能で共通の振る舞いは変化しない．変更仕様では，全機能で共通のキャンセルの仕様が示されないことがある．

2.2 課題提起

本研究では，派生開発において，「変更誤り」に分類される欠陥の流出を防止することを目的として，テスト設計のレビューだけでは検出が困難な「変更箇所に対するテスト漏れ」を検出する手法を考案することを課題とした．

考案手法は，以下の 3 つの要求を満たすものとする．

・テストが漏れている変更箇所を特定できる．
・テスト設計のレビューでは検出できなかったテスト漏れを検出できる．
・変更箇所に対して不足しているテストケースを追加できる

また，短期間での開発が求められる派生開発において，以下のような条件下でも，考案手法が現実的に実行可能であることを確認する．

・一部の変更箇所に関連する仕様が，変更仕様または既存仕様として定義されていない．
・ベースのテストケースが存在せず，変更仕様に対して新規にテストケースを作成する．
・テストカバレッジ分析担当者は，ベースソフトの既存仕様について知識が不足している．

3. 先行研究

本研究で，参考とした「ソフトウェアの検算」と「テストデバッグ」という理論，「プログラム変更を考慮したテストカバレッジ率」という手法を示す．

3.1 ソフトウェアの検算とテストデバッグ

松尾谷[7]は，ソフトウェアの検算という次のような理論を提唱している．

・ソフトウェアテストは，仕様，実装されたプログラム，テストケースの 3 つの要素について比較を行う行為である．
・実装されたプログラムにテストデータを与えて，動作させた結果とテストの予測正解値を比較することが具体的な検算である．しかし，この検算では，仕様からテストケースへの変換をするテスト設計の不完全性により検算されない要素が生じる可能性がある．
・ブラックボックステストで作成された（仕様から作成された）テストケースの漏れを，プログラム側から検算するときに，ホワイトボックステストが有効である．

また，松尾谷[8]は，テストデバッグという理論を提唱している．テストデバッグは，テストのデバッグ環境を与えると飛躍的にテスト漏れを少なくすることが可能であり，テストの網羅計測ができる環境を用意すべきという理論である．

3.2 プログラム変更を考慮したテストカバレッジ率

浜田等[9]は，テストカバレッジ率を，プログラムを変更した場合でも累積し，システムテストを通した値として算出する方法を提案している．カバレッジ率としては，Statement Coverage を適用している．テストカバレッジは以下のように分析する．

1. 前版とのソースプログラムの差分行を管理し，変更・追加したプログラム行に関してテストカバレッジを確認する．
2. 試験項目は機能仕様から抽出した項目から実施する．テストカバレッジを分析し未通過行を通るような試験項目の作成を行う．

4. 解決策の提案

4.1 課題の解決方針

本研究では,「ソフトウェアの検算」と「テストデバッグ」の理論を参考に，図 1 に示すように，テスト設計のレビューだけでなく,プログラムの変更箇所に対して,テストカバレッジを用いてテストの網羅計測を行い,テストケースの漏れを検出する方針とした.

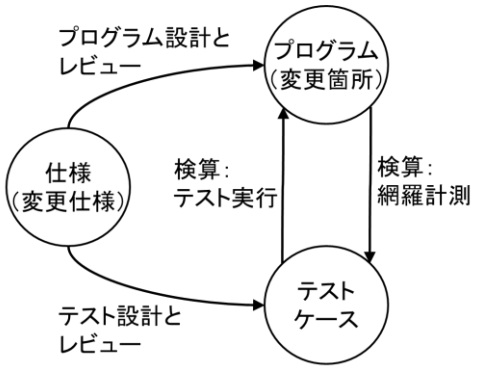

図 1 ソフトウェアの検算

短期間での開発が求められる派生開発において,現実的に実行可能な手法とするため,テストの網羅性計測は,「プログラム変更を考慮したテストカバレッジ率」を参考に，ソースコードの差分を抽出し，変更箇所に対するテストカバレッジを算出する方針とした.

テスト設計のレビューで変更仕様に対する論理網羅[7]を確認することを前提に,テストの網羅性計測では変更箇所に対する機能網羅[7]を確認する方針とし，Statement Coverage を利用することにした.

考案した手法は,「Difference Statement Coverage 分析法 (DSC 分析法)」と呼び，Difference Statement Coverage の定義とその分析手順を技術要素とする.

4.2 Difference Statement Coverage の定義

Difference Statement Coverage (DSC) は，ソースコードの変更箇所の中からテスト未実行の箇所を特定するための指標である．DSC は，関数毎に算出する．DSC は，変更関数（追加関数も含む）内の変更行と位置付けられた行のうち，テストで実行された行の割合である．分母となる変更行の数を算出するときに，テスト実行不可能な行は含めない．また，条件付きコンパイルの命令の条件を網羅してテストが実行されていることを確認できるようにする．図 2 に，条件付きコンパイル箇所に対する DSC の算出例を示す.

```
void funcA(void){
#if ( AAA == NOT_SUPPORT )
   funcB();
#else
   funcC();
#endif
}
```

算出例：
funcB(),funcC()がどちらも変更行である場合
- funcB(),funcC()どちらもテスト実行されている状態で，Difference Statement Coverageは100%とする.
- funcB()のみテスト実行されている状態で，Difference Statement Coverageは50%とする.

図 2 条件付きコンパイル箇所に対する DSC の算出例

次に，DSC における変更行を定義する．ソースコードの差分比較により特定された関数内の実行可能な追加行と修正行は，変更行と位置付ける．関数内に追加行と修正行が存在せず，削除行のみが存在する場合は，該当関数内の実行可能な行をすべて変更行と位置付ける．更に，関数内の未修正の行であっても，変数・定数・条件付きコンパイルの変更の影響を直接受ける行も変更行と位置付ける．具体的には，以下の行を，変更行と位置付ける.

- 変数の定義（型やサイズ）が変更された場合，その変数が使用されている行
- 定数の定義が変更された場合，その定数が使用されている行
- 条件付きコンパイルが追加・修正された場合，そのプリプロセッサ命令の対象範囲内にあり条件付きコンパイルにより実行可能となる行

4.3 Difference Statement Coverage 分析法の定義

DSC 分析法は，テスト未実行の変更行を特定し，特定した変更行に対してテストケースの追加要否を判断する手法である．DSC 分析の結果は，変更仕様・テストケースの漏れ・誤りを修正し，テストを再度実行するために利用される.

DSC 分析法では，カバレッジが 100%にならない場合に，カバレッジが 100%になるようソースコ

ードのみを入力にテストケースを追加するということはしない．ソフトウェアの検算の理論を参考に，テスト未実行の変更行に関連する仕様を特定し，仕様を再確認したうえで，テスト設計をやり直す．また，テストが実行されない原因が，テストケースの不足にあるか，またはソースコードの不必要な変更にあるかを判断する．原因がソースコードの不必要な変更にあると判断した場合には，テストケースの追加は行わない．

図 3 に手法の全体像を示し，図 4 にカバレッジ分析結果のイメージを示す．

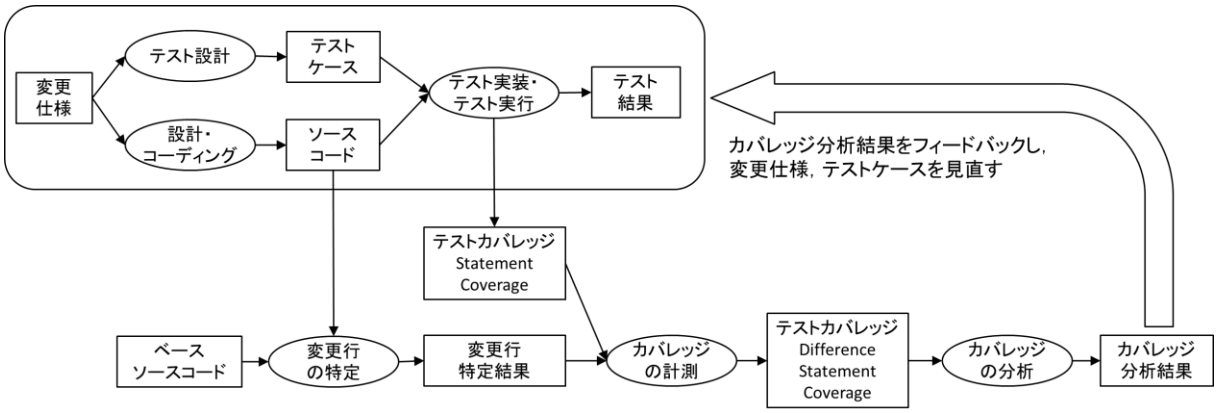

図 3 Difference Statement Coverage 分析法の全体像

【Difference Statement Coverage計測結果】

変更行	テスト実行行	テスト未実行の原因	ソースコード
			void funcA(void){
			int i;
○	○		if((state != IDEL) && (state != DOING)){
○	○		return;
○			}
	○		for(i=0; i<MAX; i++){
○	○		if(state == DOING){
○	○		array1[i]=array2[i];
○	×	テスト	}else if(state == IDEL){
○	×	テスト	array1[i]=FALSE;
○			}else{
○	×	ソースコード	return;
○			}
			}
○			#if(AAA == NOT_SUPPORT)
○	○		funcB();
○			#else
○	×	変更仕様	funcC();
○			#endif
			return;
			}

$$\text{DSC} = \frac{\text{変更行かつテスト実行行の数}}{\text{テスト実行可能な変更行の数}} = \frac{5}{9} \fallingdotseq 0.55 = 55(\%)$$

【テストケース追加要否の判断結果】

関数名	DSC(%)	テストケース追加要否	テストケース追加不要理由	テスト未実行の変更行に関連する仕様
funcA	50%	要	–	・異常時の定義
funcB	100%	–	–	–
funcC	100%	–	–	–
funcD	70%	不要	・他のテストレベルで確認	–
funcE	90%	不要	・ソースコードの問題（デッドコード）	–
funcF	85%	要	–	・異常時の定義 ・キャンセル処理の仕様

図 4 カバレッジ分析結果のイメージ

DSC 分析法は，変更行の特定，カバレッジの計測，カバレッジの分析の 3 ステップで，テスト未実行の変更行に対してテストケースの追加要否を判断する．

(1) 変更行の特定

ベースソースコードと変更されたソースコードの差分比較結果を入力に，変更行を特定する．変更行の定義は，4.2 に示す．ソースコードの差分比較ツールを利用し自動で変更行を判定することが難しい条件については，手動で変更行を判定する．ソースコードの実行可能行毎に変更行か否かの情報を付加する．図 4 の Difference Statement Coverage 計測結果の表に，変更行の特定結果の記入例を示す．

(2) カバレッジの計測

通常の Statement Coverage 計測によって得られたテスト実行行の情報をもとに，ソースコードにテスト実行行か否かの情報を付加する．同一関数に対する複数の Statement Coverage 計測結果がある場合，すべての Statement Coverage 計測結果をマージして反映する．複数の Statement Coverage 計測結果の中に一つでもテスト実行行と判定されている結果があれば，テスト実行行と判定する． 図 4 の Difference Statement Coverage 計測結果の表に，テスト実行行の記入例を

示す.

ソースコードに付加された変更行とテスト実行行の情報をもとに，DSC を関数毎に算出する．図 4 のテストケース追加要否判定結果の表に，DSC の記入例を示す.

(3) カバレッジの分析

関数毎に算出した DSC が 100%未満の場合に，該当関数のテスト未実行行に対して，原因を特定する．原因は，表 2 に示した，変更仕様の問題，テストの問題，ソースコードの問題に分類する．図 4 の Difference Statement Coverage 計測結果の表に,テスト未実行行の原因の記入例を示す.

表 2 テスト未実行の原因の分類

テスト未実行の原因	説明
変更仕様の問題	変更仕様のみでは仕様が把握できず，テストケースが漏れた.
テストの問題	仕様は把握していたが，テスト設計／テスト実装／テスト実行の問題で，テストが漏れた.
ソースコードの問題	不必要な変更がされた箇所のため，テストが実行されなかった.

特定したテスト未実行行の原因をもとに，テストケースの追加要否判断を行い，関数毎に「テストケースの追加要否」「テストケース追加不要理由」「テスト未実行の変更行に関連する仕様」を示す．テストケースの追加が不要と判断した場合には，「テストケース追加不要理由」を示し，テストケースの追加が必要と判断した場合には,「テスト未実行の変更行に関連する仕様」を特定する.「テストケース追加不要理由」と「テスト未実行の変更行に関連する仕様」は，一つの関数に対して複数存在する場合がある．図 4 のテストケース追加要否判定結果の表に，カバレッジの分析結果の記入例を示す.

5. 解決策の評価

5.1 評価方法

変更行の特定とカバレッジ計測を可能とするツールを開発したうえで，実開発において統合テスト[1]で，DSC 分析法を適用し，以下の観点で評価を行った.

(A) テストが漏れている変更箇所を特定できるか？

(B) テスト設計のレビューでは検出できなかったテスト漏れを検出できるか？

(C) 変更箇所に対して不足しているテストケースを追加できるか？

評価のために，案件毎に，表 3 に示すデータを計測した.

表 3 計測データの種類

データ			説明
関数の数	変更		変更された関数の数.
	テスト未実行箇所あり		変更関数の内，DSC が 100%未満の関数の数.
テストケースの数	ベース		DSC 分析法によりテストケースを追加する前に，実行していたテストケース数.
	追加	レビュー	レビューにより，追加したテストケースの数.
		DSC 分析法	DSC 分析法により，追加したテストケースの数.
テスト未実行の原因	変更仕様		変更仕様のみでは仕様が把握できなかったことで,テストが漏れた変更箇所の数.
	テスト		仕様は把握していたが，テスト設計・テスト実装・テスト実行の問題で，テストが漏れた変更箇所の数.
	ソースコード		不必要な変更がされた変更箇所の数.

DSC 分析法で検出したテストが漏れた変更箇所の数は，テスト未実行の原因毎に集計した．ただし，他のテストレベル[1]（関数単体テスト等）でテストを実行するべきと判断した箇所は除いて集計を行った.

提案手法の適用対象の案件は，構造化手法で設計を行い C 言語でコーディングしているミドル

ウェアを対象に派生開発をする 4 案件とした．評価対象案件では，統合テストでは変更仕様を入力に新規にテスト設計を実施した．また，テスト設計のレビューを実施した後に，ベースソフトの開発に携わっていない担当者が提案手法を実施した．

5.2 評価結果

表 4 に，案件毎に表 3 に示すデータを計測した結果を示す．

表 4 評価結果

案件ID	関数の数(個)		テストケースの数(項目)			テスト未実行の原因(件)		
	変更	テスト未実行個所あり	ベース	追加		変更仕様	テスト	ソースコード
				レビュー	DSC分析法			
1	276	52	1174	8	1	0	1	3
2	16	4	105	2	2	2	0	0
3	58	15	705	2	6	5	1	2
4	21	6	195	2	4	4	0	0

DSC 分析法では，以下のような仕様に対するテストケースの漏れが検出できた．

・曖昧な用語（「異常時」「すべての」等）が使用されている変更仕様

・変更仕様としては定義されていない既存仕様（全機能・処理で共通の仕様）

評価観点毎に，評価結果を示す．

(A) テストが漏れている変更箇所を特定できるか？

全案件で，テスト未実行の変更行が存在している関数が検出できた．

(B) テスト設計のレビューでは検出できなかったテスト漏れを検出できるか？

全案件で，テスト設計のレビューでは検出できなかったテストケースの漏れを，BSC 分析法で検出できた．

(C) 変更箇所に対して不足しているテストケースを追加できるか？

テスト未実行の変更行に関連する仕様（変更仕様には明確に定義されていない仕様）を特定し，特定した仕様を入力に漏れているテストケースを追加できた．テストケース追加により，変更箇所に対してテストが実行されるようになったことが確認できた．

5.3 結果の考察

提案手法では，変更行に対する Statement Coverage を分析する．この方法で，テスト設計のレビューでは検出できなかった変更箇所に対するテストケースの漏れを検出できることが確認できた．また，派生開発で陥りやすい以下の条件下でも，テストケースの漏れを検出し，テストケースを追加するアクションができることが確認できた．

・一部の変更箇所に関連する仕様が，変更仕様または既存仕様として定義されていない．

・ベースのテストケースが存在せず，変更仕様に対して新規にテストケースを作成する．

・テストカバレッジ分析担当者は，ベースソフトの既存仕様について知識が不足している．

提案手法では，カバレッジの分析の対象はソースコードの変更行のみとなる．また，変更行の特定（ソースコードの差分比較で抽出不可能な変更行の特定を除く）とカバレッジの計測については，ツールで実行可能である．これにより，カバレッジの分析の対象となる関数が 50 個程度までであれば，手法適用による工数増加は 8H から 24H 程度であり，大きな工数増加なく適用可能な手法であることが確認できた．ただし，検出したテストケースの漏れに対して，テスト設計・テスト実装・テスト実行をやり直すための工数・期間が必要となることを，計画時に考慮しておく必要がある．

提案手法により，テストケースの漏れを検出するだけでなく，ソースコードのリファクタリング対象箇所を特定することが可能であることも確認できた．テスト未実行の変更行に対して，テストケースの追加要否を判断すると共に，次のようなソースコードの改善点も特定できた．

・コードクローン（類似した処理が複数個所に存在している）
・デッドコード（無駄なフェールセーフ処理等が存在している）

6. おわりに

ソースコードの変更箇所に関連する仕様が明確に定義されていない場合は，テスト設計のレビューを実施しても，テストケースの漏れを検出し切ることが難しい．本研究では，この問題解決のために Difference Statement Coverage の定義とその分析手順を技術要素とする「Difference Statement Coverage 分析法（DSC 分析法)」を考案した．

DSC 分析法を実開発に適用し評価した結果，手法が以下の 3 つの要求を満たし，現実的に実行可能な手法であることが確認できた．

・テストが漏れている変更箇所を特定できる．
・テスト設計のレビューでは検出できなかったテスト漏れを検出できる．
・変更箇所に対して不足しているテストケースを追加できる．

テスト設計のレビューに加えて，DSC 分析法を実行することで，派生開発において変更箇所に対するテスト漏れを防止する効果が期待できる．また，ソースコードのリファクタリングを検討すべき箇所の特定も可能であり，テストケースを減らす効果も期待できる[10]．

今後は以下の取り組みを行い，DSC 分析法を適用する効果を高め，効果を広げる．

・実開発への DSC 分析法の適用を継続し，手法を適用しても流出を防止できない欠陥を明らかにしたうえで，手法を改善する．必要に応じて，Statement Coverage 以外の基準を使用することも検討する．
・本研究において評価を行ったプロジェクト以外でも，手間なく DSC 分析法を導入できるようにするため，ツールを改善する．
・手戻り工数を削減するために，テストケースの漏れを誘発する未定義の変更仕様と曖昧な変更仕様を，テスト設計の前に特定するための方法を検討する．

謝辞

本研究に対して有益なご助言をいただいた 有限会社デバッグ工学研究所 松尾谷徹氏，株式会社デンソークリエイト 山路厚氏，竹下千晶氏，松原潤弥氏，周廣有氏に感謝の意を表する．

参考文献

[1]JSTQB,「ソフトウェアテスト標準用語集 日本語版 Version 2.3.J02」, 2014
[2]清水吉男,「「派生開発」を成功させるプロセス改善の技術と極意」, 技術評論社, 2007
[3]柏原一雄,「欠陥混入メカニズムの知識を活用した DRBFM の提案」, ソフトウェア品質シンポジウム 2018, 2018
[4]柏原一雄,「統合テストにおいて影響範囲に対するテスト漏れを防止する「影響波及パス分析法」の提案」, ソフトウェア品質シンポジウム 2017, 2017
[5]吉岡克浩, 水野昇幸, 西康晴「見通しのよいテストの段階的詳細化の手法」, ソフトウェアテストシンポジウム 2013 東京, 2013
[6]2014 年度 SQiP 研究会第 7 分科会,「ソフトウェア欠陥予測アルゴリズム」, ソフトウェア品質シンポジウム 2015, 2015
[7]松尾谷徹,「テスト／デバッグ技法の効果と効率」, 情報処理 Vol.49 No.2 Feb.2008, 2008
[8]松尾谷徹,「テストから観た実体のモデリングと実装構造の評価」, JaSST'15Tokai, 2015
[9]浜田信,山田哲靖,岡本明,「プログラム変更を考慮したテストカバレッジ率に関する一考察」, 情報処理学会第 59 回全国大会, 1999
[10]坂本一憲, 土肥拓生「テストカバレッジを用いた新しいテスト駆動開発プロセスの提案」, 情報処理学会研究報告ソフトウェア工学（SE）2015-SE-187(34), 1-5, 2015