博士 (工学) 論文概要

テスト対象の機能構造とデータの入出力情報を使ったブラックボックステスト手法の研究

システム情報工学研究科 リスク工学専攻

湯本剛

2017 年 10 月

目 次

1 緒論

ソフトウェア開発の中の品質を確保する主要な技術として, ソフトウェアテストがある．求められるテストケースの数は，昨今のソフトウェアの複雑性と規模の急激な増加に伴い増加の一途をたどっている．ブラックボックステストの場合，ソフトウェアの規模とテストケース数の関係は，ファンクションポイント総計値の 1.15 乗から 1.3 乗となる [1]．また，開発プロジェクトのファンクションポイント総計値は 1970 年から 2000 年までの 30 年間で約 10 倍の増加を示している [2]．

日本におけるテスト工数の割合は開発工数全体の 28 パーセントから 35 パーセントを占めるケースが多いが，90 パーセントを超えるケースもあるという調査結果が出ている [3] また，テストケースを作成する工数は，平均的にテスト工数全体の 40 パーセントだと言われている [4]．これは，テストケースの開発に多くの人員が必要となることを示している．多数の人員がテストケースを作成する工程に必要とされているにもかかわらず， テストケースを作成するための明確に定義されたルールがないために，投入された人員は個々の考え方に基づいてテストを開発することが多い．これはテストケースの重複や漏れの原因となり，テストの活動がソフトウェアの品質を確保する役割を果たせないばかりか，コスト増や納期遅延の原因となってしまう．本研究では，テストケースを作成する工程に投入される人員が、必要なテストケースを網羅的に抽出し，抜け漏れを防止できるようにすることを目的とし，適切な数のテストケースを開発するための手法を提案し，その適用評価を行う．

本論文は 6 章で構成される. 2 章では，研究対象とシステムテストにおけるブラックボックステストにおける課題と，関連する先行研究について述べる．また，本研究で使用するテスト分析手法について解説する．3 章では，前章で述べた課題を更に分析するために，実験を行い実験結果で確認を行った．4 章では，テストデータの入出力に着目し、テスト対象の分析を網羅的に行う手法の提案と適用評価を行った．5 章では，テストデータの入出力を順番に組み合わせる必要がある際に，重要な順序組み合わせを抽出してテストケースにする方法の提案と適用評価を行なった．最後の 6 章では，結論を述べる．

2 複数人でテストケースを作成する課題

2.1 本研究の対象となるテストケースの開発方法とテストレベル

2.1.1 テスト開発プロセス

テストのプロセスは, テスト計画が V モデルの左側の活動と並行に行われ，その後時系列にテスト分析，テスト設計，テスト実装が行われた後，V モデルの右側の活動の中で，テスト実行と終了基準の評価が行われる．テストのプロセスの中でテスト分析，テスト設計，テスト実装の 3 つのテストケースを作成するための活動はテスト開発プロセスと呼ばれている [5]．本研究では，テスト開発プロセスの中のテスト分析とテスト設計を対象とする．テスト分析では，テスト対象をテスト設計ができるサイズに詳細化する．このアウトプットはテスト条件と呼ばれている．このテスト条件を合理的にある基準で網羅する方法を考える行為がテスト設計であり，そのための技法をテスト設計技法と呼ぶ．テスト設計のアウトプットはテストケースである．

2.1.2 テストケースを開発する方法

テストケースを開発する方法は，ソフトウェアの物理的な構造を基にテスト設計するホワイトボックステストと，ソフトウェアの仕様を基にテスト設計するブラックボックステストに大別できる [6] ．本研究では，ブラックボックステストを対象とする．テスト対象そのものではなく, テスト対象の動作条件や論理的な振る舞いの記述がベースあるがゆえに, ベースとなる情報からテストケースにしたてるまでの詳細化が複数の解釈で行われることが多くなるためである．

2.1.3 テストレベル

テストのプロセスは，V モデルであらわす各レベルごとに行われる．ソフトウェアテストは，開発ライフサイクルの中で複数のテストレベルに分けて行われる．各テストレベルはソフトウェア開発の段階的詳細化のレベルと対応している．本研究は，複数のレベルの中で，システムレベルで行われるブラックボックステストに焦点を当てている．システムレベルのテストは，開発した単体のソフトウェアがすべて統合されるため，規模の増大と複雑性の増加の影響を直接的に受けるからである．

2.2 システムレベルでのブラックボックステストの課題

2.2.1 システムレベルでのテスト分析の課題

テスト分析の活動の出力となるテスト条件は，機能，トランザクション，品質特性，構造的要素といった多くの側面の総称である．テスト対象の詳細化をするときの起点や中間分類が混乱しないように各側面の関係を整理する必要がある．しかし，テスト分析におけるテスト条件群は，研究や実務においても，経験則や個人の考え方に基づいている．一般的には，テストベースを大項目，中項目，小項目と詳細化していくことが多い．この方法は，詳細化する際の各分類項目に明確なルールが定義されていないため，個人毎の何かしらの考え方で詳細化するための分類を決めていくことになる．そのため，複数人で作業を行うと分類にばらつきが発生し，同じテスト条件が複数の階層に現れてしまったり，同じ意味のテスト条件が別の名称で選択されるといった混乱が起きてしまう．したがって，テスト分析が複数人で行われるときには，テスト条件の重複，もしくは完全に抜け落ちるといった可能性が高くなる．テスト開発の最初の活動であるテスト分析にて特定するテスト条件にこのような問題があると，その後の活動で作られるテストケースの抜け漏れ，重複に影響を及ぼす．テスト分析手法に関する研究は，Nishi[7]， Akiyama[8], Yumoto[9] がある．しかし，複数の人数でテスト分析を行う際のテスト条件の重複と欠落について手法がどの程度有効であるかは，ほとんど研究されていない.

2.2.2 システムレベルでのテスト設計の課題

テストケース数の増加は, 単一機能のテストより機能間の統合において問題となる. この場合のテストケース数は, 単一の機能や制御構造の和で求めるのではなく, 積となるためである. それに加え, このテストでは, 状態遷移に伴う時系列の組合せのテストも求められることから, テストケース数の爆発問題が生じる. しかし，必要なテストケースの抽出方法とその網羅性に関する研究は多々あるが, 多くは機能や制御構造を基にした方法である.

状態遷移間の組合せについては，N スイッチカバレージに従ってテストケースを抽出する方法がある．N スイッチカバレージとは, 状態の遷移をパスとし,N+1 個の遷移パスを網羅する基準に従て組合せテストケースを作成する．N=0 では遷移パスの組合せをテストできないため N=1, すなわち S1 網羅基準 (1 スイッチカバレージ) が必要とされている．しかし,S1 網羅基準を満たすテストケース数は, 2 つの状態遷移間における遷移数の積となり, 膨大なテスト工数を必要とする課題になる．

2.2.3 論理的機能構造を使ったテスト分析手法の利用

本研究では，論理的機能構造を使ったテスト分析手法を利用した検証実験を行い，テスト分析の課題の調査を行う [9]. この分析手法を採用する理由は，1 つは，前述するテスト分析の課題を解決するために提案し，現場にて適用している手法であるためである．もう 1 つは，検証実験のための題材となる仕様書，模範解答が揃っており，それらの題材を使って実験を行った先行研究結果があるためである．

先行研究では，同一組織内で本分析手法を導入したグループと導入していないグループでのテスト条件の選択数を比較する実験結果が得られている．グループの回答を実験データを使った理由は，この手法の効果が複数の人員でテスト分析をしたときのばらつきからくる欠損や重複を防ぐことを狙っているためである．この実験にて，分析手法を導入したグループが, 導入していないグループよりも抜け漏れが少なく重複も少ない結果となった．

この手法は，図 1 の論理的機能構造基づいてフィーチャを MECE に分解して，テスト条件を特定する．図 1 に示す各箱が，各フィーチャに要求されるテスト条件を特定する有用なガイドとなる．

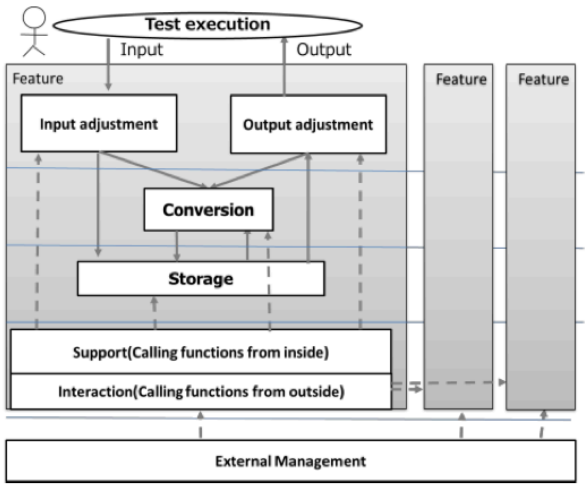

図 1: MECE にフィーチャからテスト条件を識別する方法

ただし，論理構造は抽象的な概念であるため，各個人の解釈に違いが出る可能性がある．テスト条件の特定に一貫性を持たせるため， 論理構造に対してテスト対象に特化した名前付けを行う．名前付けしたものをテストカテゴリと呼ぶ．更に，このテスト分析手法では，テストベースからテスト条件を特定するステップも定義した．多くのテスト開発に従事するテスト担当者がそのステップに従うと，その全てのテスト担当者は，同じルールに沿って各自の仕事を実行できる．結果として，特定したテスト条件の重複や漏れの防止につながる．これは本手法の主たる効果となる．

3 テスト分析でのテスト条件の抜け漏れの傾向

3.1 検証実験の背景

先行研究の実験にて，分析手法を導入したグループが, 導入していないグループよりも抜け漏れが少なく重複も少ない結果となったが，実験データは 1 組のみであり, 傾向を見るためにで不十分である.

テスト分析手法を適用する前のテスト分析の傾向，及びその傾向と適用後の結果との相関をすることを目的に，複数のグループに対して検証実験を行った．検証実験は 3 回行った．1 回目，2 回目の検証実験と 3 回目の検証実験は実施方法が異なるため,2 つは分けて考察を行う.

3.2 1 回目，2 回目の検証実験からの考察

3.2.1 検証実験の手順

検証実験は，ワークショップを通じて実施した．ワークショップでは，最初に，演習に使うテストベースを示し，参加者が各自の考えに基づいたテスト分析を実行してもらう．その後，4〜5 名の参加者をランダムにグルーピングし，グループ内で各自のテスト分析の結果をグループの回答としてまとめる.

最初の分析結果のまとめが終わった後，テストカテゴリを用いたテスト分析手法と実施手順を説明して手順に沿って再度テスト分析を参加者が各自で実施する．その後は最初の分析結果同様にグループの回答をまとめる．テスト分析手法の知識を与える前と後のグループの回答を検証実験のデータとして利用した.

3.2.2 テスト分析手法適用前のテスト分析の結果

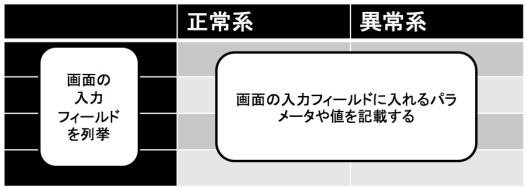

図 2: テスト分析の事例：CS1

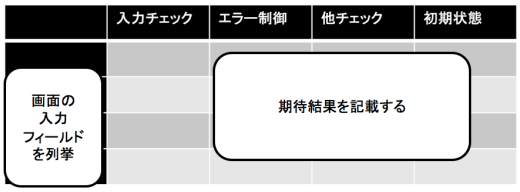

図 3: テスト分析の事例：CS2

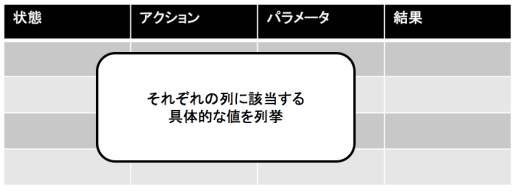

図 4: テスト分析の事例：CS3

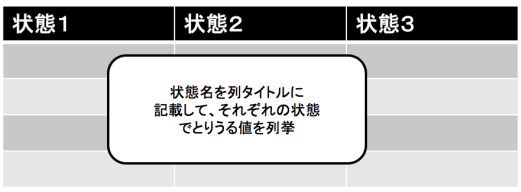

図 5: テスト分析の事例：CS4

テスト分析結果のばらつきは図 2 から図 5 にて確認できる．検証実験による典型的な 4 パターンのテスト分析結果を図 2 から図 5 で示す．

1. CS1 と CS2:両方とも AUT の入力フィールドを行タイトルに列挙している．しかしながら列名と表の内容が異なっている．
2. CS3: 表の中に記載されている各列，アクションとパラメータと期待結果などが独立しているため，それぞれの間の関連を特定できない．
3. CS4: AUT の状態が列名として列挙されている．各状態で取りうるパラメータと値が表の中に記載されている．

この結果は複数の個人がそれぞれ複数の結果に到達することを示している．複数の結果とは，テスト分析を通して特定したテスト条件にばらつきがあることを意味している．したがって，テスト分析が多くの人たちで行われるときには，テスト条件の重複，もしくは完全に抜け落ちるといった可能性が非常に高くなる．

3.2.3 1 回目と 2 回目の検証実験結果の評価

1 回目，2 回目の実験では，グループでの回答を実験結果として利用し，8 つの実験結果が収集できた．テストカテゴリを使った手法を知らないで行った演習結果と，テストカテゴリの知識を与えた後の演習結果とで比較をした．実験結果の評価のために，表 1 に示した評価レベルを設定した．検証実験での評価結果は，表 2 に示したとおりである．

表 1: 評価レベルの定義

評価レベル	比較結果
B	リストした仕様項目数は増加していない, かつ実験の期待結果よりも少ない.
-	リストした仕様項目数は増加していない, しかしすでに期待結果と同数である.
A	リストした仕様項目数は増加している, しかし実験の期待結果よりも少ない.
A+	リストした仕様項目数は増加している, かつ実験の期待結果に達している. 仕様項目の数は増加していない,.

表 2: 2 つの検証実験結果の評価

論理的機能構造	チーム							
	TM1	TM2	TM3	TM4	TM5	TM6	TM7	TM8
変換	B	A	B	B	B	B	A	A
入力	-	-	-	-	-	-	A	B
出力	-	-	-	-	-	A+	A	A
貯蔵	-	A+	-	A+	A+	-	A	A
サポート	B	B	B	B	B	B	B	A
相互作用	B	A	A	A+	A	A+	B	A

TM1 から TM6 までは最初の実験の結果である．音楽生成機器がテスト対象でありテスト対象フィーチャはボリュームコントロールであった．全チームにてテストカテゴリ内に列挙したテスト条件の数が増えており，論理構造の項目ごとの比較では，相互作用にて 5 チームの列挙数が増加しているが, サポートでは全チームにて増加していない．TM7 と TM8 は 2 回目の実験であり，フライト予約システムがテスト対象で，新規飛行機予約がテスト対象フィーチャであった．全チームにてテストカテゴリ内に列挙したテスト条件の数が増えており，論理構造の項目ごとの比較では，変換と出力と貯蔵にて両チームともに仕様項目の列挙数が増えた．

全体的に定量的な向上が見られたが，特定のグループが著しく成長した，もしくは論理的機能構造の項目に特徴的な傾向がるということは見出せなかった．そのため，３回目の検証実験では，実験データ取得の方法を変更して実験を行った．

3.3 3 回目の検証実験からの考察

3.3.1 3 回目の検証実験結果の評価

3 回目の検証実験では，グループ単位ではなく，各参加者の実施結果を収集した．ワークショップを通して，テスト分析手法を知らない状態での演習実施（1a），一部分だけ説明した状態で演習実施（1b），全てを説明した状態で演習実施を（1c）行い，各参加者の演習結果を実験データとして収集した. このワークショップでは，57 名分の実験結果を収集した．

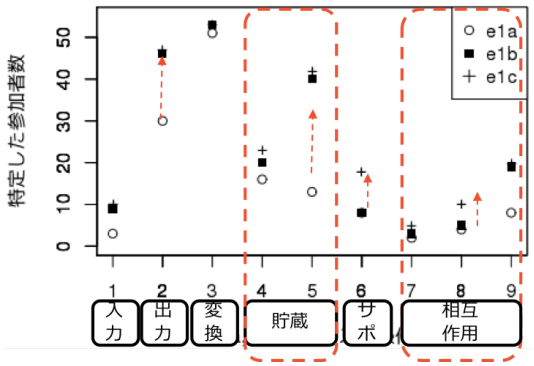

図 6: 参加者あたりのテスト条件特定数

各参加者が特定できたテスト条件数 (回答数) を図 6 のように論理的機能構造で分類して比較した．図 6 の Y 軸はテスト条件毎に解答できた参加者数を示し，X 軸は，テスト条件を示している．1a，1b，1c と進むにつれて，分析で特定できるテスト条件が増えていることがわかる．テスト分析手法の知識を与えることで特に伸びたのは, 出力と貯蔵に属するテスト条件であった．

3.3.2 テスト分析手法適用前のテスト分析方法の分類

テスト分析手法の知識を与える前のときのテスト分析結果から，テスト分析結果の記載は，図 7 に示す通り、大きく 4 パターンに分類できた. この 4 パターンとテスト条件の特定数に相関があるかをスピアマンの順序相関分析を使って調べたが，相関があると結論付けられる値にはならなかった.

	パターン	記載内容	
1	仕様項目	「○○な場合に××なること」といったテスト対象の仕様	分析的
2	テストケース	入力値，アクション，期待結果	実装的
3	P-V（パラメータ/値）	パラメータと値	分析的
4	シナリオ	操作手順として記載	実装的

図 7: テスト分析パターン

これらの実験結果から，それぞれの分析方法のばらつきから起きる重複や欠落の課題は現象として確認できたが，特定の要因は見出せなかった．

4 I/O テストデータパターンを使った仕様項目特定の方法

4.1 研究の概要

4.1.1 研究の目的

本章では，テスト分析において，テスト条件を網羅的に特定する方法として，テスト実行時のデータの入出力（以降 I/O と呼ぶ）に着目し，テストベースを分析する際にテスト実行時の I/O の要素で分解し網羅性を確認する方法を提案する．

4.1.2 I/O テストデータパターン

テストを実行するためには，データを AUT にインプットし，AUT のアウトプットを期待結果と実際の結果で比較する．テスト実行をするときのデータの I/O は，AUT へのデータの入力の方法は，外部からの入力，内部に保持したデータの入力，外部と内部からの入力の 3 パターンに分類できる．同じように AUT からのアウトプット方法は 3 パターンに集約でき，入出力の組み合わせ数はすべてで 9 パターンとなる．

たとえば，シンプルな機能の四則演算の計算結果が正しいことを検証するときには，AUT の外部から複数の値を入力し，AUT がそれらの値を計算し，計算結果を AUT の外部にアウトプットする．これは「外部からのデータ入力，外部へのデータ出力」というパターンになる．

	入力	出力
P1	外部	外部
P2	外部	内部
P3	外部	外部 内部
P4	内部	外部

	入力	出力
P5	内部	内部
P6	内部	外部 内部
P7	外部 内部	外部
P8	外部 内部	内部
P9	外部 内部	外部 内部

図 8: I/O データパターンの説明

AUT に対するテスト実行時のデータの入出力をまとめたパターンは図 8 のように 9 パターンに集約できる．これを I/O テストデータパターンと呼ぶ．I/O テストデータパターンがテスト実行時のデータの入出力から見た全体集合となる．

P1 から P9 の I/O データパターンは単一の入出力の全体像となる．単一の入出力では，論理的機能構造の入力調整，出力調整，変換，貯蔵を通過する．P1 に分類できるシンプルな四則演算の場合，外部からの入力に対して外部に出力する間に，図 9 のように論理的機能構造の入力調整，変換，出力調整を通過する．この 3 箇所がテスト分析で特定すべきテスト条件の候補となる．

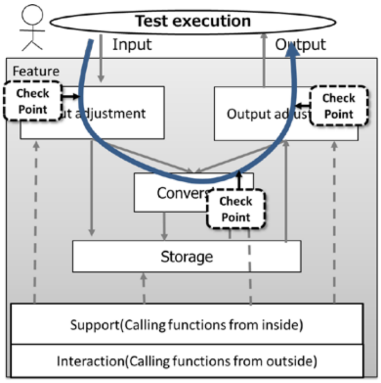

図 9: I/O テストデータパターンのデータの流れ

4.1.3 これまでの実験データを使った調査

２章で行った検証実験の結果を基に I/O テストデータパターンと，テスト条件として特定した論理的機能構造を確認することで，各 I/O データパターンのデータのフローで全てのテスト条件が特定可能であることが確認できた．

4.2 I/O テストデータパターンを使ったテスト分析の実験

4.2.1 実験の目的

この実験は今回提案しているテストカテゴリに I/O テストデータパターンを使う手法が，現実のテスト設計と比較して網羅的に仕様項目を特定できることを目的とする．そのために現実のテスト設計の結果と，I/O テストデータパターンを使った分析結果を比較する．

4.2.2 実験の概要

実験対象の AUT は，実在するオンラインのモバイル写真共有アプリケーションを使った．モバイル写真共有アプリケーションの開発にて実際に使われたテストケースと、提案する手法で分析した結果を比較する．

今回の実験で使う成果物には，テストケースのみであり，テスト分析のアウトプットとなるテスト条件の一覧はなかった．そのため，比較元のテストケースは，今回の実験のためにテスト分析でのアウトプットであるテスト条件になるようまとめなおした．

今回の提案手法では，まず最初に，テスト対象フィーチャのテストベースを分析し，画像データ，画像の情報、設定データ，外部コマンドを入力データ，出力データとして扱うこととした．そして，テスト実行時の追記したデータの流れをシミュレーションし，該当する I/O テストデータパターン

を明らかにした．その後，I/O テストデータパターンと既存の分析手法であるテストカテゴリを併用してテスト分析を行った．

4.2.3 実験結果

テストカテゴリと I/O テストデータパターンを使ったテストベースの分析を行った結果を図〜10 に示す．

実プロジェクトの仕様項目との比較をした結果を両者を比較すると，I/O テストデータパターンを利用したテスト分析の結果が実プロジェクトより多くの仕様項目を選択できたことが確認できている．

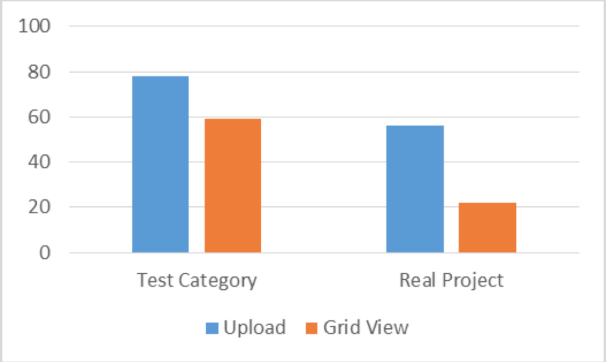

図 10: 実際のプロジェクトと I/O テストデータパターンを使った場合の比較

現実のプロジェクトにて不足していたテスト条件には，論理的機能構造の要素別に見ても入力調整, 出力調整といったメッセージが現れることや入力制御といった単純なことを確認するテスト条件でも漏れているものもあることが確認できる．

5 データ共有タスク間の順序組合せテストケース抽出手法

5.1 研究の概要

5.1.1 研究の目的

前章では，I/O テストデータパターンで単一のデータの入出力からテスト条件を網羅的に特定する方法を提案した．本章では，単一の入出力だけではなく，統合して確認すべきテスト条件に着目する. 機能間の統合における状態遷移間の組合せに着目する．

5.1.2 テストケース抽出方法と課題

S1 網羅基準の課題に対するアプローチとしては, 自動化により工数を削減する研究とテストケース数を削減する先行研究がある. 自動化による工数削減の研究は,N-スイッチカバレージ を満たすテストケースを形式仕様から自動生成する方法が 知られているが生成されるテストケース数は N-

スイッチカバレージと 同じであり削減されないので, テストケースが自動抽出さ れても, 実行のための操作は人手に頼る部分が残り, 作業 工数を合理化できない課題がある.

テストケース数を削減する研究としては, 状態遷移の組合せに対して直交表を応用し 2 因子間の組合せを中心に, 一部 3 因子の組合せも抽出する研究がある . この方法は, 決定表を用いて機械的に組合せを抽出でき,2 因子間 の組合せ即ち S0 網羅基準は完全に網羅できるが,S1 網羅 基準の網羅は不完全であり, かつその選択基準が用いた直 交表に左右されるため重要なテストケースが漏れる課題がある. 本研究は, テストケース数を削減するアプローチに属する. テストケースを機械的に削減するのでは無く, 実践の場における経験から生じるノウハウを用いて削減する. 状態遷移に係る不具合は, 定義された状態変数や画面とは別に, 内部に保存されたデータが影響していることが知られ ている. 具体的には, 状態の制御が状態変数や画面によっ て一意に動作するように設計されていたとしても, 内部変 数で保持されたデータが存在すると, これが隠れたサブ状態となり, 設計とは異なる振舞いが生じ不具合となる. 本研究では, このような不具合を見つけ出すために必要なテストケースを,DFD,ER 図,CRUD 図といったデータ設計文書を入力情報として使うことで合理的に抽出する方法を提案する.

5.2 順序組合せによるテストケース抽出法

本章では，状態遷移テスト設計における S1 網羅基準 ではテストケース数が爆発するが,S0 網羅基準では漏れが 生じるという課題を解決する手法として順序組合せテストを提案する.

5.2.1 順序組合せテストの概要

提案する手法は，2 タスク間の順序組合せを対象とする．2 タスク間の順序組合せの抽出は以下のルールを適用する.

5.2.2 ルール 1：変更タスクの特定

ルール 1 を用いて変更タスクとそのデータストアを特定し，拡張 CRUD 図の変更タスク部分を作成する.

拡張 CRUD 図とは，テストベースとして与えられた DFD，ER 図，CRUD 図から $P\{Ta\}$ の各 Ta_i と関連する So_k，そして $P\{Ds\}$ となる Ds_j の関係を追加して作成したものである.

5.2.3 ルール 2：波及タスクの特定

ルール 2 を用いて波及タスク群を特定し，拡張 CRUD 図へ波及タスク部分を追加し図を完成させる.

先に作成した中間の拡張図から変更タスクの操作が C か U か D であるデータストアに着目する. 着目したデータストアに対してエッジを持つタスクが波及タスクのうち，源泉に出力エッジを持つタスクを波及タスクとして特定する. 特定した波及タスクを拡張 CRUD 図に追記し完成させる.

完成させた拡張 CRUD 図の例を表 3 に示す.

表 3: 完成した拡張 CRUD 図の例

タスク	データストア			源泉		
	Ds_1	Ds_2	Ds_3	So_1	So_2	So_3
$Ta_1[St_1]$	CU			In		
$Ta_3[St_1]$		C		In		
$Ta_2[St_1]$	R				Out	
$Ta_5[St_1]$		RU				Out

表 4: 順序組合せテストによる論理的テストケースの例

No	論理的テストケース	順序組合せ
1	概要	$Ta_1C \rightarrow Ta_2R$
2	概要	$Ta_1U \rightarrow Ta_2R$
3	概要	$Ta_3C \rightarrow Ta_2R$
4	概要	$Ta_3C \rightarrow Ta_2U$

5.2.4 ルール 3：順序組合せテストケースの抽出

拡張 CRUD 図を基に順序組合せのテストケースを抽出し，テストケース表を作成する．表 4 にその例を示す．概要の部分は，当該組合せが持つ入力の条件や出力の特性を仕様から抜き出して記載する．

以上の手順で，順序組合せテストに必要なテストケースを抽出する．

5.3 評価実験

5.3.1 実験の概要

本節では，旅行代理店向けフライト予約システムの仕様を用いて，3 章で述べた実施手順を適用し，順序組合せが抽出できることを確認する．新規フライト予約を，実験の対象となる機能セットとする．

5.3.2 順序組合せテストの適用評価

提案手法で抽出したタスク間の順序組合せと既出の状態を含むテストケースを設計する手法である状態遷移テストで, 抽出されるテストケースの比較を行う. 状態遷移テストのテストベースとなるフライト予約システムの画面遷移図を使って, 順序組合せが確認できる網羅基準である S1 網羅基準を適用する．適用範囲と仕様の詳細度合いは，DFD，ER 図，CRUD 図と画面遷移図では同等にしている．S1 網羅基準を適用すると 28 の状態遷移パスとなる．28 の状態遷移パスのうち，対応する提案手法で抽出した順序組合せは，4 つであった．

これらは, 本状態遷移図のフライト予約状態での登録イベントを起点にするもののみであった．順序組合せに該当しない状態遷移パスは，互いのタスクで同一のデータを介して処理をするといったことがないため，本提案手法では選択されない．

6 結論

本研究では，テストケースを作成する工程に投入される人員が、必要なテストケースを網羅的に抽出し，抜け漏れを防止できるようにすることを目的とし，適切な数のテストケースを開発するための手法を提案し，その適用評価を行った．

3 章では, 検証実験にて，本手法の説明を参加者にすることによる仕様項目の一貫性と特定する量の向上が観察できた．更に I/O データパターンを使った実験結果の分析によって，実験結果の一部が本手法で提唱している仮説と一致することを観察できた．更に高精度に傾向を分析するため，更なる検証実験は必要である．以降のこの手法の効果と関連する要因とその傾向に対する深い理解とそのための更なる実験をすることで，AUT とフォールトの知識をベースにしたテストカテゴリを作るためのルールをより洗練できると考えている．

4 章では，テスト実行時のデータ I/O の要素で分類し網羅的に分析する方法を提案した．そして，現場のテストプロジェクトのテストケースを使い，提案した方法の実証を試みた．結果的に，提案した方法で特定した仕様項目と実プロジェクトで作られるテストケースと比較して，不足している仕様項目の発見が可能であることが確認できた．

5 章では，状態遷移を持つソフトウェアにおいて，データベースや外部変数などの保持データを介して影響が生ずる場合のテストに関して，その網羅基準と, 順序組合せテストケースを抽出する手法を提案した．DFD，ER 図、CRUD 図をテストベースとして，3 つのルールを適用することでテストが必要な順序組合せを抽出できることを説明した．提案した手法で組合せが抽出できることを確認するため，フライト予約システムの仕様を具体例にして適用を行った．最後に従来手法である画面遷移図から S1 網羅基準にて抽出した状態遷移パスと提案手法を比較して，テストケース数の削減ができる効果と，S1 レベルの画面遷移の網羅では抽出できないテストケースが抽出できる効果を示した．

参考文献

[1] T Capers Jones. *Estimating software costs*. McGraw-Hill, Inc., 1998.

[2] David Longstreet. Productivity of software from 1970 to present, 2000.

[3] 独立行政法人情報処理推進機構 技術本部ソフトウェア高信頼化センター（編）. ソフトウェア開発データ白書 2014-2015. 独立行政法人情報処理推進機構, 2015.

[4] Alexander van Ewijk, Bert Linker, Marcel van Oosterwijk, and Ben Visser. *TPI next: business driven test process improvement*. Uitgeverij kleine Uil, 2013.

[5] ISTQB/FLWG. *Foundation Level Syllabus*. International Software Testing Qualifications Board, 2011.

[6] Glenford J Myers, Corey Sandler, and Tom Badgett. *The art of software testing*. John Wiley & Sons, 2011.

[7] Yasuharu Nishi. Vewpoint-based test architecture design. In *Software Security and Reliability Companion (SERE-C), 2012 IEEE Sixth International Conference on*, pp. 194–197. IEEE, 2012.

[8] K.Akiyama, T.Takagi, and Z.Furukawa. Development and evaluation of hayst method tool (software testing). *SoMeT*, pp. 398–414, 2010.

[9] Tsuyoshi Yumoto, Toru Matsuodani, and Kazuhiko Tsuda. A test analysis method for black box testing using aut and fault knowledge. *Procedia Computer Science*, Vol. 22, pp. 551–560, 2013.

[10] Robert S Arnold. *Software change impact analysis*. IEEE Computer Society Press, 1996.

[11] Bixin Li, Xiaobing Sun, Hareton Leung, and Sai Zhang. A survey of code-based change impact analysis techniques. *Software Testing, Verification and Reliability*, Vol. 23, No. 8, pp. 613–646, 2013.

[12] Tom DeMarco. Structure analysis and system specification. In *Pioneers and Their Contributions to Software Engineering*, pp. 255–288. Springer, 1979.

[13] Lionel C Briand, Yvan Labiche, and L O'sullivan. Impact analysis and change management of UML models. In *Software Maintenance, 2003. ICSM 2003. Proceedings. International Conference on*, pp. 256–265. IEEE, 2003.

[14] Hassan Gomaa. Designing software product lines with uml. In *SEW Tutorial Notes*, pp. 160–216, 2005.

[15] Andy Maule, Wolfgang Emmerich, and David S Rosenblum. Impact analysis of database schema changes. In *Proceedings of the 30th international conference on Software engineering*, pp. 451–460. ACM, 2008.

[16] Boris Beizer. *Software Testing Techniques*. Van Nostrand Reinhold, 1990. (翻訳 :小野間 彰， 山浦恒央 :ソフトウェアテスト技法, 日経 BP 出版センター (1994)).

[17] James N Campbell. Data-flow analysis of software change. *Oregon Health & Science University*, pp. 1–118, 1990.

[18] 小谷正行, 落水浩一郎. UML 記述の変更波及解析に利用可能な依存関係の自動生成. 情報処理学会論文誌, Vol. 49, No. 7, pp. 2265–2291, 2008.

[19] 加藤正恭, 小川秀人. CRUD マトリクスを用いたソフトウェア設計影響分析手法. 情報処理学会第 73 回全国大会講演論文集, Vol. 73, pp. 249–250, 2011.

[20] Anthony L Politano. Salvaging information engineering techniques in the data warehouse environment. *Informing Science*, Vol. 4, No. 2, pp. 35–44, 2001.

[21] T.Yumoto, K.Uetsuki, T.Matsuodani, and K.Tsuda. A study on the efficiency of a test analysis method utilizing test-categories based on aut and fault knowledge. *ICACTCM '2014*, pp. 70–75, 2014.

[22] ISO/SC7/WG26. *Software and Systems Engineering-Software-Testing Part 2:Test Processes*. ISO/IEC/IEEE29119 2013(E), 2013.

[23] 湯本剛, 松尾谷徹, 津田和彦ほか. 効率的な機能テスト設計のための欠陥情報の活用方法. 第 75 回全国大会講演論文集, Vol. 2013, No. 1, pp. 321–322, 2013.