筑波大学大学院博士課程

博士論文概要

入出力データの順序情報に基づくブラックボックステスト手法に関する研究

湯本剛

2018年1月

本研究はブラックボックステストのテストケース抽出に抜け漏れやケース数が増 大する課題に対して，テスト対象に対する入出力データの順序情報に基づいてテストケースを抽出する 手法を提案し，合理的にテストケースを抽出することを成果とするものである

目 次

図 目 次

表 目 次

0.1 諸論

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る．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，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．

0.2 ブラックボックステストにおけるテスト分析とその課題

0.2.1 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

テストケースの開発方法

テストケースを開発する方法は，ソフトウェアの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 [5] ．本研究では，ブラックボックステストを対象とする．ブラックボックステストは，テスト対象そのものではなく，テスト対象の動作条件や振る舞いについて記述した仕様をベースにしてテストケースを開発する．ブラックボックステストは，テストベースが AUT の物理的な構造ではなく論理的なふるまいの記述であるがゆえに，テストを作るための詳細化が複数の解釈で行われることが多い [6].

テストレベル

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．

0.2.2　テスト分析

V モデルであらわす各レベルにて行われるテストはそれぞれ，開発プロセスと類似したプロセスを持っている [4]. 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 つの活動はテスト開発プロセスと呼ばれている [5]．テスト設計の前には，テスト設計技法が適用できるサイズにテスト対象を詳細化することが必要となる．この活動はテスト分析と呼ばれている．テスト分析は，図 1 で示すとおり，テスト開発プロセス中の最初の活動である．

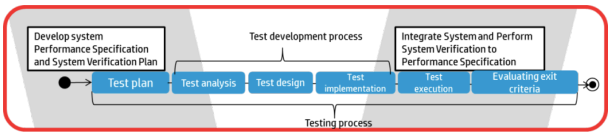

図 1: テストプロセスとテスト開発プロセス

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ため，各側面の関係を整理するための構造化が必要である．しかし，テスト分析におけるテスト条件群の構造化は，研究や実務においても，経験則や個人の考え方に基づく構造化に留まっている [6]．そのため，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．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，重複に大きく影響を及ぼす．

ブラックボックステストにおけるテスト分析の先行研究

ISTQB では，テスト分析を実行するための要求事項や必要性は述べているけれども，テストベースを分析していくためのアプローチは定義されていない．G.J.Myers や B.Beizer といったテストケースを作る技術の先行研究は，テスト分析にてテスト条件が特定された後に適用されるテストパラメータの設計に焦点を当てている．そのため，テスト条件は，すでに全て準備されたと言う前提になっている. テスト分析手法に関する研究は，Nishi [12]， Akiyama[13] があるが，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，手法を適用すると実際はどの程度効果的であるかについては，大きく着目していない．

0.3　論理的機能構造を使ったテストケースの特定方法

0.3.1　テストカテゴリベースドテストのアプローチ

本章で提案する分析手法は，図 2 の論理的機能構造基づいてフィーチャを MECE(互いに相容れなくて完全に徹底的) な方法で分解して，テスト条件を特定する [16]．図 2 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．

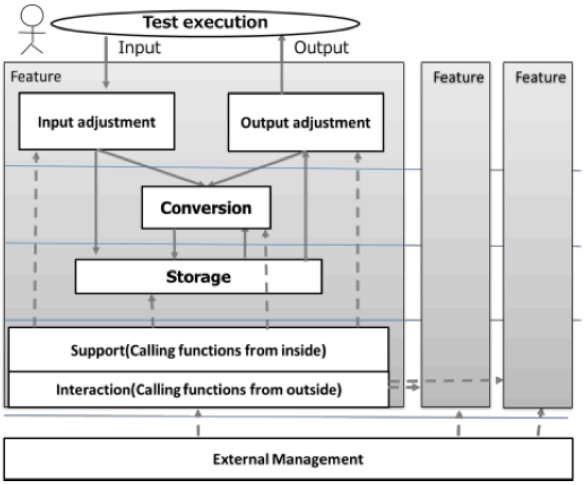

図 2: MECE にフィーチャからテスト条件を識別する方法

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 AUT 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．

0.3.2　検証実験からの考察

検証実験の手順

テストカテゴリを使った分析手法の効果を証明するために，複数のグループに対して本手法の習得前と習得後で選択した仕様項目の数を比較する検証実験を行った．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すが，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を説明し、その手順に沿って再度テスト分析を参加者が各自で実施し，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．ワークショップの一連のアウトプットを検証実験のデータとして利用した．

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，テストカテゴリを用いた分析手法の手順にそった演習結果とで比較をした．実験結果の評価のために，A,A-,-,B,N/A という 5 段階の評価レベルを利用すします。表 **??**に示した評価レベルを利用した．検証実験での評価結果は，表 1 に示したとおりである．

これらの評価結果を確認すると分かる通り、テストカテゴリを使った手法を実装した時の結果では、8 つのグループ中 7 つにてテストカテゴリによる効果が確認できた。

表 1: 2 つの検証実験結果の評価

Logical Structure	Team							
	TM1	TM2	TM3	TM4	TM5	TM6	TM7	TM8
Conv	B	A	B	B	B	B	A	A
Input	n/a	n/a	n/a	n/a	n/a		A	B
Output	-	-	-	-	-	A+	A	A
Storage	-	A+	-	A+	A+	-	A	A
Support Mngt	B B	B A	B A	B A+	B A	B A+	B B	A A

0.3.3　3 回目の検証実験の評価

3 回目の検証実験では，各参加者の実施結果を収集し．57 名分の実験結果を収集した．ワークショップのを通して，テスト分析手法を知らない状態での演習実施、一部分だけ説明した状態で演習実施，すべて説明した状態で演習実施、すべて理解した状態で題材を変更して演習実施といったように 4 回行った．

各参加者が特定できたテスト条件数 (回答数) を図 3 の箱ひげ図を用いて比較した．図 3 の Y 軸は参加者が回答したテスト条件数を示し，X 軸は，各演習における回答数の分布を箱ひげ図で示している．1a(テスト分析手法を知らない状態での演習実施) では，最高点は 5 であり，中央値は 1 と非常に低い値であった．その後、1b では中央値が 3，1c では 4，2 では 7 と，演習が進むごとに中央値が増えているので， テストカテゴリを使ったテスト分析手法を使うことによって，テスト条件数を特定するスキルが向上したと考えられる．

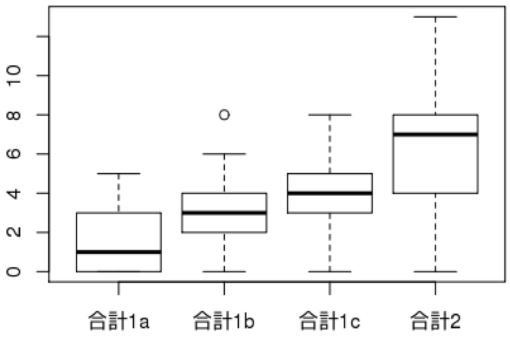

図 3: 参加者あたりのテスト条件特定数

0.4 終わりに

３回の検証実験を通じて，本手法の説明を参加者にすることによるテスト条件を特定できる数の向上が観察できた．以降では，テスト条件を網羅的に特定する方法を提案し，このテスト分析手法の更に洗練させる．

0.5 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

３章では，テスト実行時のデータの I/O に着目し，テストベースを分析する際にテスト実行時のデータ I/O の要素で分解し網羅性を確認する方法を，既存の手法に追加することを提案する．

0.5.1 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入出力（以降 I/O と呼ぶ）は，パターンとして分類できる．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．

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 4 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～ref fig：D-4-Fig7 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．

	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 4: I/O データパターンの説明

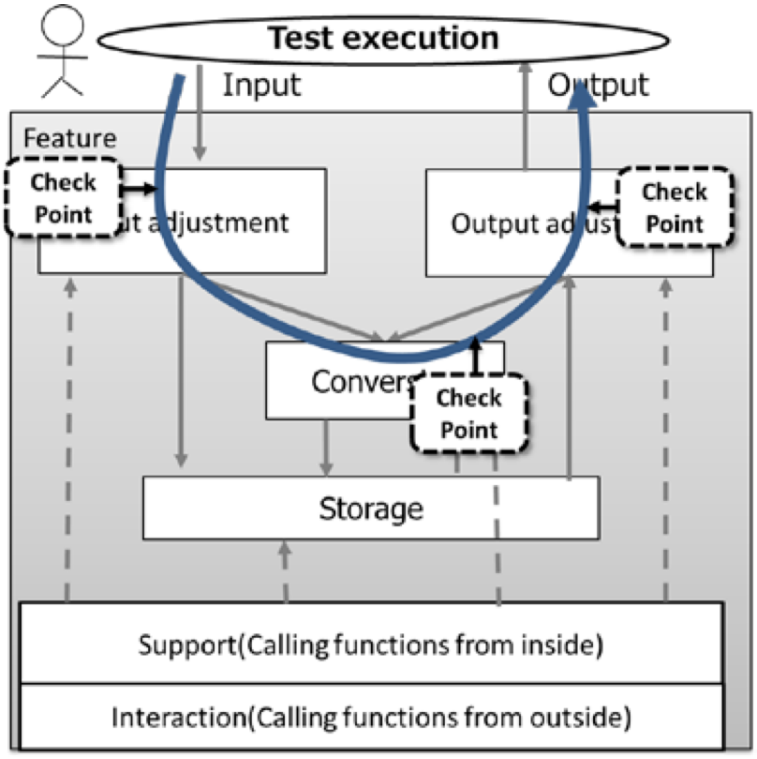

図 5: Fig. 7. An explanation of the I/O data patterns.

しかし，実際に期待結果を確認するチェックポイントは 3 箇所とは限らない．なぜなら，入力調整に該当する入力の際に適切な値だけ受け入れることと，変換に該当する計算が適切に行われていることの 2 つだけであり，出力が適切にされることはチェックポイントとしないといったケースが考えられるためである．

0.5.2　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 I/O テストデータパターンを使うとどこに分類されるかを調査し，結果を考察した．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の中で期待結果を確認するチェックポイントが限られていることが確認できた．

0.5.3　サポートと相互作用に関する考察

論理的機能構造の要素であるサポートと相互作用は，単一の入出力だけではなく，関係する他の処理の呼び出しに着目してテスト条件を特定するための分類である．サポートは，テスト対象フィーチャでのテスト実行時のアクションによって内部的に呼び出される別の処理の結果確認のことを指している．

一方，相互作用は，テスト実行時のアクションによる副作用を，他フィーチャに対するアクションにて呼び出して確認することを指している．サポートに該当する仕様項目の特定に使う呼び出しのきっかけと相互作用に該当する仕様項目の特定に使う呼び出しのきっかけを整理して，I/O テストデータパターンとの対応がわかるようにした．

0.5.4　I/O テストデータパターンを使ったテスト分析の実験

本章の実験では，現実のプロジェクトで作られたテストケースと，今回提案する I/O テストデータパターンを使ったテスト分析結果を比較し，手法の効果を分析する．

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．

実験の題材について

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．

実験の実施手順

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．実験での実施手順を次に説明する．

実際のプロジェクトでのテストケースをテスト条件ごとに整理する

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，テストケースを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．

2) テスト対象フィーチャで使われる入力データ，出力データを明らかにする

テスト対象フィーチャであるアップロードとグリッドビュー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．

3)I/O テストデータパターンを付与する

特定した入力データと出力データは，2) で明らかにしたテスト条件に対して，入力データ , 出力データというカラムに追記していった．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．

4) 論理的機能構造と I/O テストデータパターンを使ってテストベースを分析する

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．

0.5.5　実験結果の評価

1)IO テストデータパターンの効果実証の結果

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〜ref fig : D-4-Fig10 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，テストカテゴリと 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．

2)I/O テストデータパターン毎の出現傾向の評価

現実のプロジェクトで作られたテストケースとテストカテゴリと 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結果を P1 から P9 の分類で出現割合を比較した結果が，図〜ref fig：D-4-Fig11 である．それぞれの I/O テストデータパターンの選択数の差異を図〜ref fig：D-4-Fig11 にて確認すると，P1 が特にテストカテゴリと実プロジェクトの差異が大きいことがわかる．P1 は外部からの入力を

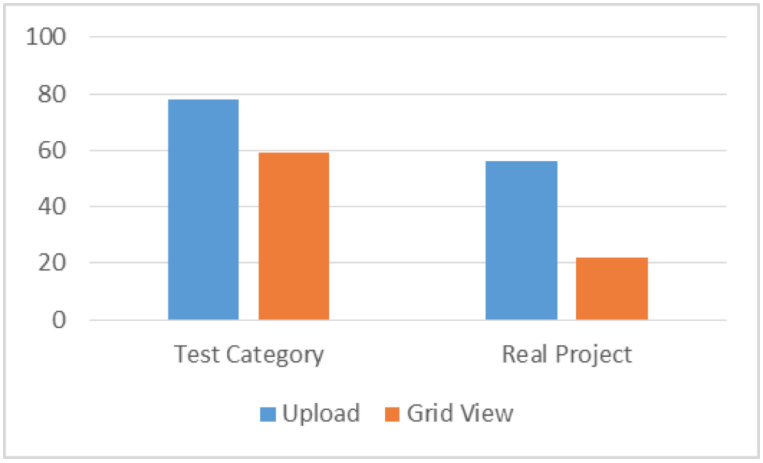

図 6: Finding the remainder of each data pattebrn between Test-Category and the real project.

行い，外部に出力する最も単純なパターンであり，単純なパターンの仕様項目がより網羅的に特定できていたことが確認できる．

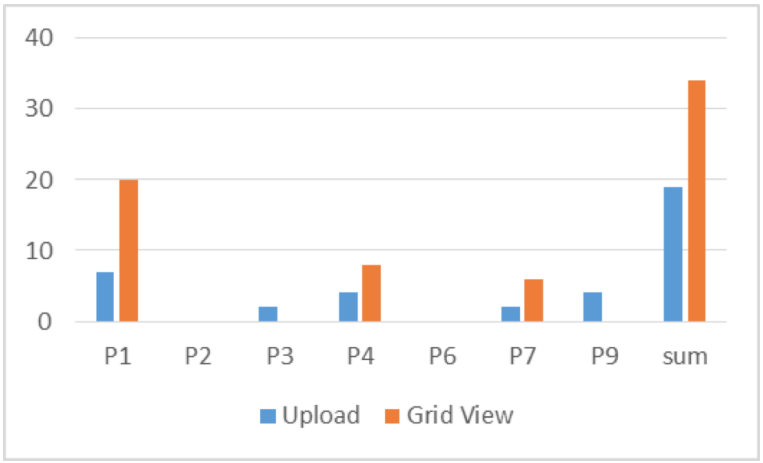

図 7: IO テストデータパターンごとの違い

3) 現実のプロジェクトにて不足していた仕様項目

不足していた仕様項目には，論理的機能構造の要素別に見ても Input Adjustment, Output Adjustment 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仕様項目でも漏れているものもあることが確認できる．一般的に，仕様項目のリストを作成せずにテストケースを作った場合，テストケースのままでは数量の多さから網羅すべき仕様項目の見易さが低下するため，仕様項目の数が不足することが多い．実験結果も同様の傾向となった．

0.5.6 終わりに

本章では，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．I/O テストデータパターンを活用したテスト分析をするためには，まだ多くのサンプルを入手し，全ての I/O テストデータパターンをナレッジにする必要がある．ナレッジにすることで，テスト分析にて仕様項目を効率よく特定するルールを確立させていきたい．

0.6 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

0.6.1 はじめに

これまでの研究にて、テストカテゴリを使ったブラックボックステストのためのテスト分析手法を提案してきた。テストカテゴリを使ったテスト分析では、論理的機能構造という参照モデルを使う。モデルの要素である入力調整，出力調整，変換，貯蔵、サポート、相互作用でテスト対象をカテゴライズしてテスト条件を特定する。入力調整，出力調整，変換，貯蔵は、外部観察可能な単一の入出力のみを考慮している分類であり，既出の研究て提案した I/O テストデータパターンで網羅することができる。しかし、サポートと相互作用は，単一の入出力だけではテスト条件の抽出が出来ない。なぜなら，サポートと相互作用は，単一の入出力だけではなく，関係する他の処理の呼び出しに着目してテスト条件を抽出するための分類であり，統合して確認すべきテスト条件の抽出が必要だからである。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。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。本論文では，機能間の統合における状態遷移間の組合せに着目する。

0.6.2 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, テストケース数を削減する研究と, 自動化により工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。この方法は, テスト対象となる IT システム の動作を正確に記述したモデルを定義し, そのモデルから 特定の長さの連続した遷移を抽出する方法である (14) 。対象システムが運動方程式などに従う一般的なモデルベー ステストと異なり, 状態遷

移にて生ずるシステムの動的な 振舞いを形式仕様化する必要があり, それが困難であるこ とから一般的な IT システムで適用された例は見当たらない。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 (15) (16) 。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。本論文では, 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。

0.7 順序組合せによるテストケース抽出法

本論文では, 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。

0.7.1 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．

0.7.2 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．

0.7.3 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．

表 2: 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 3: 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	$Ta_3C \rightarrow Ta_2U$

完成させた拡張 CRUD 図の例を表 2 に示す.

0.7.4 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 3 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．

0.8 順序組合せテストの適用評価

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．

0.8.1 考察

次に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 AP の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．適用範囲を合わせるために，4 章の適用のためのサブセットである新規フライト予約を行うために必要な画面と隣接する画面遷移に該当する範囲の図となっている．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．例えば，フライト検索をした後にキャンセルをするとフライト予約画面に遷移するパスは，前の処理の結果によって影響を及ぼさない．

S1 網羅基準では抽出できないが，本手法によって抽出できたテストケースは，

別の機能セットに含まれるタスクであるため，画面遷移図の網羅では現れない．

0.8.2 おわりに

本論文では，状態遷移を持つソフトウェアにおいて，変更による変更波及がデータベースや外部変数などの保持データを介して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として IDAU 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．

提案手法に対する今後の取り組みは 2 つある．1 つは, 適用範囲の明確化である. 変更のパターン（タスク内の制御ロジックの変更, 新しい要素の追加など）に対して, どこまで適用でき, どこからは適用できないかを明らかにする．

もう 1 つは，今回の提案手法のツール化である．実際の開発プロジェクトで扱う規模の大きいデータ設計文書に対して本手法を適用する際には，本手法のルールをツール化するといった方法での適用が必要になる．これらの準備を行い，実践の場に本手法を適用していく．

0.9 結論

参考文献

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

[2] D. Longstreet, “Productivity of software from 1970 to present,” 2000. http://www.softwaremetrics.com/Articles/history.htm

[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] A. vanEwijk, B. Linker, M. vanOosterwijk, and B. Visser, TPI next: business driven test process improvement, Uitgeverij kleine Uil, 2013.

[5] G.J. Myers, C. Sandler, and T. Badgett, The art of software testing, John Wiley & Sons, 2011.

[6] T. Yumoto, T. Matsuodani, and K. Tsuda, “A test analysis method for black box testing using aut and fault knowledge,” Procedia Computer Science, vol.22, pp.551–560, 2013.