筑波大学大学院博士課程

博士論文

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3月

概要

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

第1章　序論

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．ソフトウェアテストでは，テスト結果の情報を分析してテスト対象の品質を可視化することができる．そのためには，可視化するために十分なサンプルデータを得られるだけのテストケースを実行しなければならない．テストを十分に行うためには，重複が無く抜け漏れの無いテストケースを開発することが重要になる．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]．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]．組み込みソフトウェア開発のソフトウェア規模の増加は，ドメインによって毎年 10 パーセントから 20 パーセントに及ぶという調査結果もある [3].

ソフトウェアテストの活動のうち，テスト実行工程がソフトウェア開発のクリティカルパス上にある唯一の工程となる．テスト実行の前にテストケースを開発し，実行するテストの全体像を示せると，効率のよいテスト実行を計画できる．効率のよいテストとは，テストの効果に対するリソースが少ないことである．しかし，ソフトウェアの規模の増加に伴ったテストケース数の増加に対応するために必要となるテスト工数は，ソフトウェア開発工数の多くを占めるようになってきている．

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 [4] 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 [5]．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．日本では，複数のシステムを統合するといった大規模な開発にて 8ヶ月の間に約 5000 人がテストに投入された

という事例もある [6]．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する，ソフトウェアテストの中でもシステムテストレベルでのブラックボックステストに着目し，そのレベルでのテスト対象の入出力データの順序情報から，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．

第2章　システムテストにおけるブラックボックステストの課題

2.1　テストケースの開発方法とテストのレベル

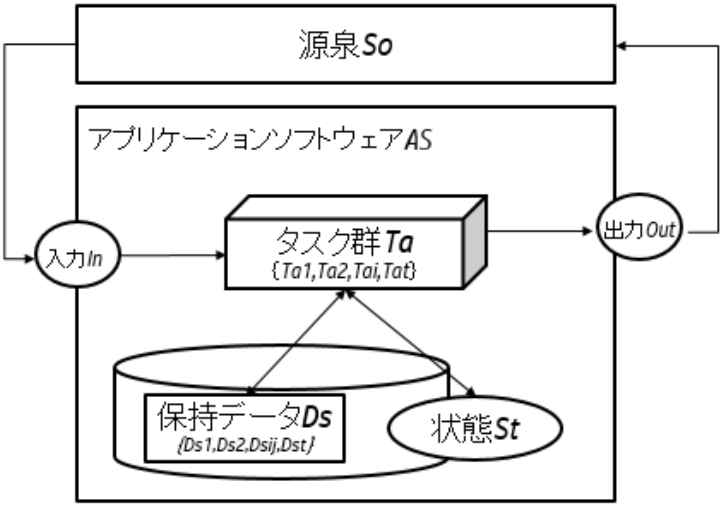


図 2.1: アプリケーションソフトウェアの構成

2.1.1　アプリケーションソフトウェアの構成

一般的なアプリケーションソフトウェアは，入力 In に対して，何らかの出力 Out を返す．ソフトウェアの機能は，何らかの入力を出力に変換する処理により実現されていると考えられる．この処理を本論文ではタスクと呼ぶ．タスクは，該当のテストレベルからみた入力を出力に変換している 1 処理である．そ

のため，タスクの粒度は，後述するテストレベルによって決まる．ソフトウェアの構成要素であるタスク Ta の出力について考えると，Ta への入力 In だけでなく状態 St と保持データ（データベースや内部メモリに保存されているデータ）Ds の影響を受けると考えられる．例えば，Web アプリケーションにて予約を行うタスクについて考えると，予約が可能か否かを示す状態と，予約オブジェクトの予約状況を示す保持データによって，予約の成否が決まる．本論文では，対象として図 2.1 に示すような状態と保持データを持つアプリケーションソフトウェア (AS) のタスクに対するテストを研究対象にする．AS の構成要素は，タスク群 Ta と状態 St と保持データ Ds とし，各タスクは外部の源泉 So からの入力 In と So への出力 Out があるとする．タスク群 Ta は，その要素を $Ta = \{Ta_1, Ta_2, \cdots, Ta_i, \cdots, Ta_t\}$ とし, 対応する入出力は In_i と Out_i とする．

2.1.2 テストケースの種類

テストケースの種類は，ソフトウェアの物理的な構造を基にテスト設計を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をするブラックボックステストに大別できる [7] .

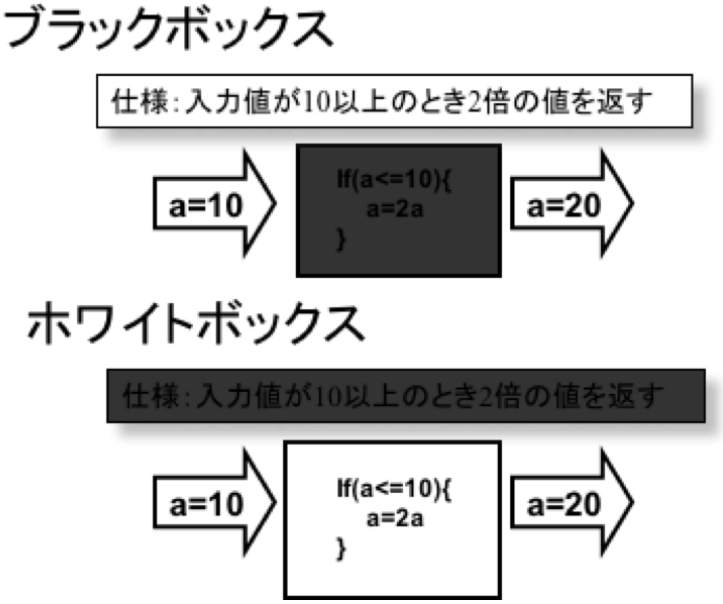


図 2.2: V モデル 1

ホワイトボックステストはテスト設計のベースがソースコードのようなテスト対象そのものとなるため，テスト対象プログラムの行を網羅，分岐を網羅といった具合にテストにて網羅すべきアイテムを明確に選択することが容易である．網羅基準はテスト設計技法として提唱されている [7, 8].

一方，ブラックボックステストでは，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のテスト設計技法の中で、仕様に対する網羅基準は，ホワイトボックステスト同様，数多く提唱されている [7, 8].

しかし，ブラックボックステストは，テストベースがテスト対象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．結果的にテストケースの重複や抜け漏れを引き起こす可能性も高くなる．

本研究では，設計されるテストケースの種類はブラックボックステストを対象とする．

2.1.3 テストレベル

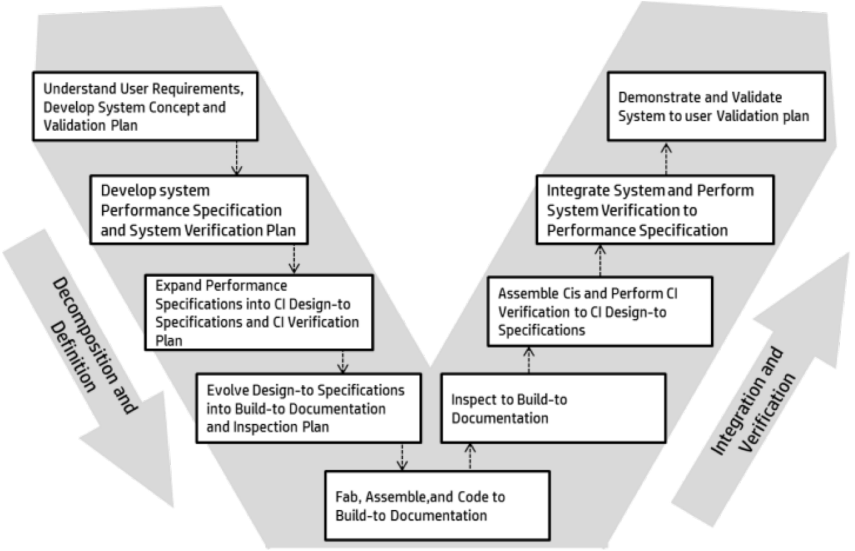


図 2.3: V モデル

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．複数のテストレベルは，図 2.3 で示す V モデルと呼ばれる技術面にフォーカスしたライフサイクルモデルにて表現することができる [?]．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．

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．本研究は，複数のレベルの中で，図 2.3 の上から 2 番目の箱となる，「Develop System performance specification and System verification plan」と「Integration system and Perform system verification to performance specificetion」のレベル，つまりシステムレベルのテスト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．

2.1.4 テスト開発プロセス

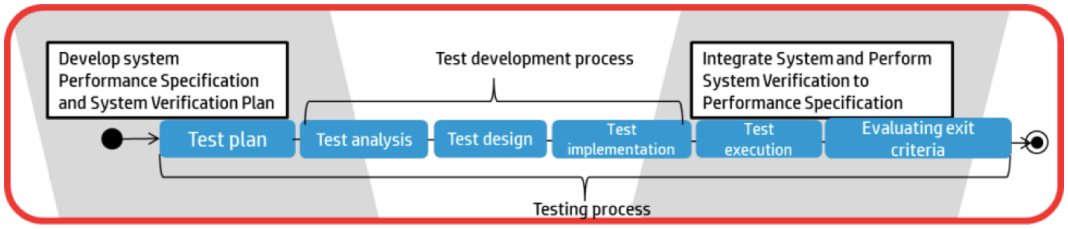


図 2.4: テスト開発プロセス

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 [9]. テストのプロセスは, 図 2.4 のようにテスト計画が 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 [9].

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．ブラックボックステストでのテストを開発するベースは, 対象とするアプリケーションソフトウェアの仕様である．仕様とは，図 2.3 で示した V モデルの左側の成果物のことである．各レベルでテスト設計のベースする仕様はテストベースと呼ば

れている．本研究の対象となるテストレベルでは,「Develop System performance specification and System verification plan」のテストベースにて，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項目及び仕様項目とそれに該当する事前条件や事前入力の取捨選択をする，整理する．テスト分析でのアウトプットはテスト条件と呼ばれている．つまり，テスト条件とは，図 2.5 のように仕様項目と仕様項目に該当する事前条件と事前入力のことを指している．

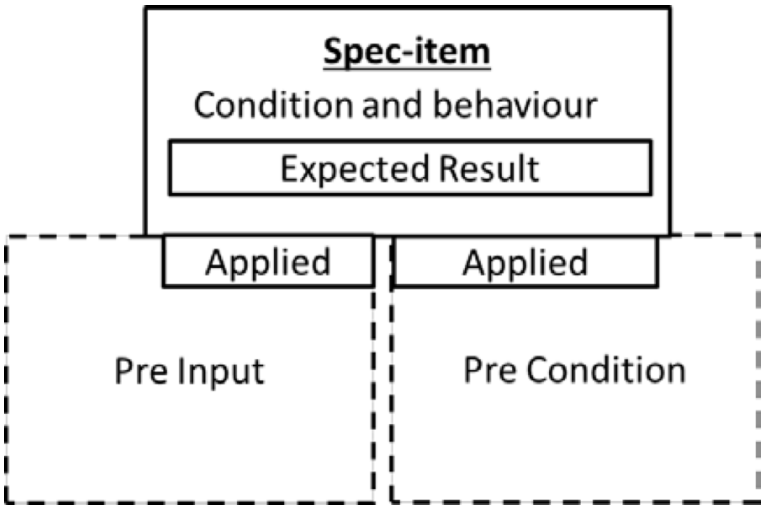


図 2.5: テスト条件の構成要素

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．

テストケースは，IEEE610 において，特定の目的のために開発されたテスト入力，実行条件，期待結果の 3 つで構成されると定義している．また，機能テストは，選択した入力と実行条件のレスポンスとして生成されたアウトプットを確認する，と定義している [ ]．すなわち，タスクに対するテストとは，テスト入力，実行条件，を入力して生成されたアウトプットが期待結果と一致するかを確認することである．このプロセスを図示すると，図 [ ] のようになる．テスト入力，実行条件には, 事前に設定されているものと，実行時点で設定するものがある．本研究では，おのおのを表 [ ] に示すよう定義する．その上で，事前入力と事前条件をまとめたものをテストパラメータ，イベントと操作をまとめたものをテストアクションと呼ぶこととする．

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 テストケースの抜け漏れが起きる課題と関連する先行研究

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アプリケーションソフトウェアの側面の総称である．これらの側面について記述した成果物は仕様とよばれる．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際にこれらの側面が記述されている仕様に着目して詳細化を行う．その際は，テスト対象の詳細化をするときの起点や中間分類が人によって異なってバラバラになってしまわな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の整理方法は，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当てはめるアプリケーションソフトウェアの側面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．混乱が起きている例を表 2.1 に示す．表 2.1 の例には以下

表 2.1: Add caption

大項目	中項目	小項目	細目	補足項目	テスト条件
印刷	設定	印刷部数	ー	ー	100 部印刷した場合
設定	プリント設定	一般	異常系	エラーメッセージ	「印刷部数が 99 部を超えました」と表示されること

のような問題がある．

1. 設定というカテゴリが大項目に出ている場合と中項目に出ている場合が混在している．

2. 階層数も一定でないため，各階層がどのような意味を持つものかがばらついている．

3. 上段は，期待結果が書かれていない．

4. 上段と下段は同じテスト条件について書かれている．

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．ISTQB では，テスト分析は「…テスト分析の期間中，何をテストするか決定するため，すなわち，テスト条件を決めるために，テストのベースとなるドキュメントを分析する [ ] 」と説明されている．ISTQB では，この説明のとおり，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．これは，Ostrand [　], Grindal [　] などのテスト開発に関する先行研究でも同様である．更に言うと，G.J.Myers や B.Beizer といった先行研究の多くは，テスト分析にてテスト条件が特定された後のテスト設計で行わ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[10], Akiyama[11], Yumoto[12] がある．それらの研究はテスト分析手法の論理に着目をしているけれども，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．Eldh は，Understanding Instruction の不足によるテストケースの品質低下について調査をしており，複数の解釈による間違いが起きることを報告している [10] ．テスト分析のルールが一貫性を持っていないことは同様の問題を引き起こす．また，ルールに一貫性がないことはテストの再利用を困難にする．

2.2.2　テストケース数が増えてしまう課題と関連する先行研究

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.[7]

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．[8] 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．

2.3 テストカテゴリベースドテスト

本研究では，テストカテゴリベースドテストという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 [12]. この分析手法を採用する理由は，3 つある．1 つは，前述するテスト分析の課題を解決するために自身で提案し，現場にて適用している手法であるためである．2 つめは，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研究結果があるためである．3 つめは，本研究で合理的にテストケースの抽出を行う手法を提案する基の考え方として，テストカテゴリベースドテストを基にしていることである．

本節では，テストカテゴリベースドテストの概要を説明する．このテスト分析手法のアプローチでは，テスト条件群を，テスト対象のサブセットとなるフィーチャセットに属するタスクとテストケースの構造をベースに階層化することで，テスト条件と言う用語の持つ曖昧さを排除している．タスクとは，前述した通り，アプリケーションソフトウェアにて何らかの入力を出力に変換する処理のことである．また，階層の要素としてテストカテゴリという，テスト対象の知識とフォールトの知識使って定義した分類を構造に追加した．テストカテゴリをテスト分析のガイドとして使い，抜け漏れや重複の少ないテスト条件の選択を可能にする．

2.3.1 テスト条件群の構造

前述した通り，テスト条件とは，機能，トランザクション，品質特性，構造的要素といったアプリケーションソフトウェアの側面の総称である．通常，これらはアプリケーションソフトウェアの仕様として記載されるものである．ブ

ラックボックステストにおけるテスト条件群の要素は，図 2.6 に示した構造で整理できる．

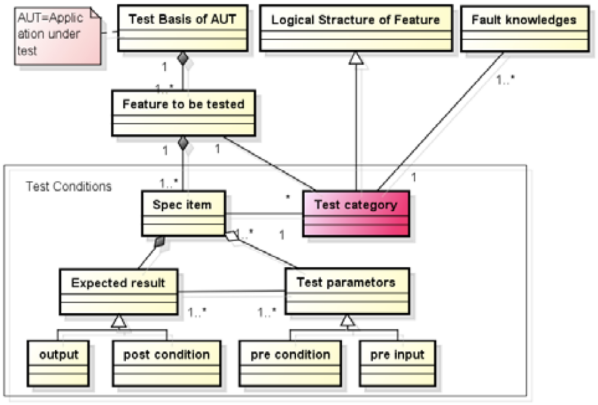


図 2.6: テスト条件の構造

テストベースは，テストケースを抽出する基になる文書のことであり，開発時に作成する要件や設計内容が書かれた文書が該当する．テストベースにはテスト対象フィーチャを実現するタスクを明確に定義する仕様項目が 1 つ以上記述されている．フィーチャは，利用者が観察可能なソフトウェアシステムのサブセットであり，利用者とテスト対象のインターフェースとなる [　]．ブラックボックステストは，外部観察によるテスト設計の方法であるため，テスト条件をフィーチャから選択することが必要になる．テスト対象から選択したフィーチャはテスト対象フィーチャと呼ばれている [　]．

仕様項目とは，テスト対象フィーチャに属するタスクの要件を綿密に定義し文書化したものである．タスクの要件とは，テスト対象フィーチャの振る舞いのひとつであり，たとえば「ボリュームは 1 から 10 の間で設定できる．1 は消音であり，10 は 100dbs になる」が該当する．この記述が仕様項目である．テスト分析では，テストすべき仕様項目を選択していく．その仕様項目の内容をテストケースの構成と同じように期待結果とテストパラメータに分類し，整理する．テストパラメータとは，テストケースの構成要素のひとつで，事前入力と事前条件を汎化したものである [9]．このような分類，整理によって，明確なルールにそったテスト分析が可能になる．

2.3.2 論理的機能構造

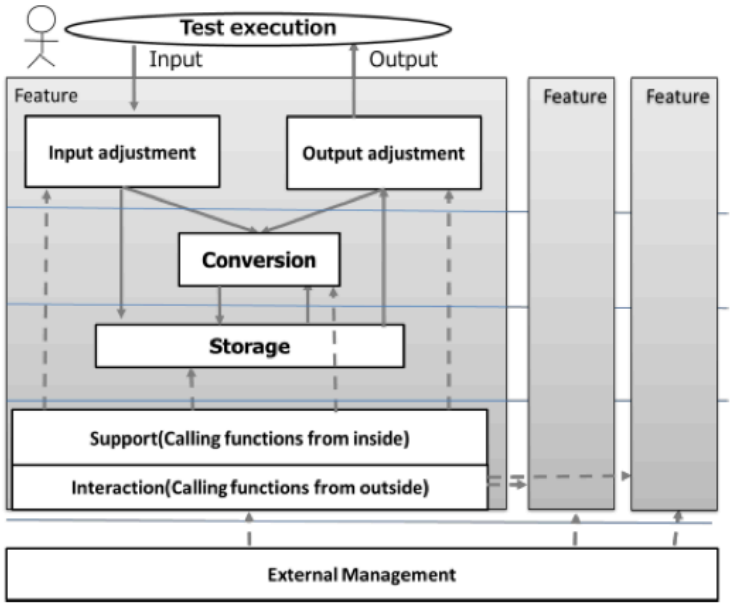


図 2.7: MECE にフィーチャからテスト条件を識別する方法

ブラックボックステストの場合，テスト対象の内部構造を完全に知ることはできなく，テスト実行は入力と出力だけが頼りになる．大村は「…人工のシステムとは，インプットを変換し付加価値を与えアウトプットする変換装置であるため，論理的には，必ず図「　」のような構造を持つ [　　]」と主張している．提案しているテスト分析手法は，同様のコンセプトを利用している．つまり，テスト対象のフィーチャ[　] は同様の論理構造を持つ人工システムだと考えている．この論理構造は，図「　」のようにフィーチャを MECE(互いに相容れなくて完全に徹底的)[13] な方法でテストをするために利用できる．図「　」に示す各箱は，テスト対象の内部構造を推定し，テストが必要なタスクを特定する有用なモデルとして利用できる．そして，タスクに関する記述がされている仕様項目がテスト条件となる．

テスト分析をしていく際に論理的機能構造を使って内部構造を推定してタスクを特定していく方法を導入すると，テストに必要なテスト条件の特定が容易になるという仮説を立てている．現状，次に示す課題はテスト条件の特定を困難にしている．

1. 明白に必要だと思われる仕様の一部分が記述されていない．

2. 機能間の組み合わせでどのように振舞うかといった仕様は，ドキュメント中の該当する単一のセクションには完全に記載し切れていない．

2.3.3 テストカテゴリ

論理的機能構造は抽象的な概念であるため，テスト分析をするそれぞれの技術者の間にて解釈の違いが生じる可能性がある．テスト条件を決定する際に，その解釈に一貫性を持たせるため、論理的機能構造の箱に対してテスト対象で使われる用語を使った名前付けをする．そのようなテスト対象に特化して付けた論理的機能構造の各箱の名前をテストカテゴリと呼ぶ．テストカテゴリはテスト条件を特定するための有用なガイドである．特定したテスト条件にはフィーチャの仕様項目，期待結果，テストパラメータが含まれている．決定したテストカテゴリに対する合意形成は，最も重要なことになる．各メンバーが各テストカテゴリの意味を明確に理解していることを確認するために，メンバー間での各テストカテゴリに分類したテストにて発見する可能性のある欠陥および故障を，例を挙げてディスカッションすることを必須にしている．

表 2.2: テストカテゴリ一覧の例

論理的構造	テストカテゴリ	意味づけ（想定する欠陥）
入力調整	画面入力	入力チェック，入力画面の制御
	ボタン操作	画面遷移のルール，処理起動
出力調整	表示	処理結果の表示，出力数の制御
	帳票出力	印刷内容，印刷フォーマット
変換	計算	料金計算
貯蔵	検索	検索条件の組み合わせ，検索結果
	登録/更新/削除	**DB** 処理
相互作用	反映	**DB** 処理結果の他機能への反映
サポート	エラー処理	エラー復旧処理

テストカテゴリに関するディスカッションの結果は表 **??**で示した表にまとめる．

それによりテスト開発プロセス活動にかかわるメンバーは認められたテストカテゴリに対して合意形成をすることができる．合意形成のねらいは次のとおりである．

1. テスト開発にかかわるテスト担当が AUT に関して同様の理解に達することができる．
2. テスト担当間のテスト条件の解釈のぶれを最小限にとどめることができる．

2.3.4 実施手順とドキュメントフォーマット

構造化したテスト条件群を順番に導くために，テスト分析の活動を図 2.8 のような作業ステップに分割し，各ステップでのインプットとアウトプットを定義する．

以降に各作業ステップについて説明をする．

Step1 テスト対象のフィーチャを選択

テストベースからテスト対象のサブセットとなるフィーチャを特定する．フライト予約をするソフトウェアで例えた場合，フライト予約（搭乗したい飛行機の条件を照合し，予約を成立させる一連の機能群）がテスト対象フィーチャとなる．

Step2 テストカテゴリの設計

テストカテゴリを設計する方法は，階層ホログラフィックモデリング法 (HHM 法) におけるサブトピックの設定方法と類似している [14]. HHM 法でいうところのメイントピックにテスト対象フィーチャを置く．メイントピックを構成するサブトピックとして論理的構造毎にテスト対象フィーチャの動作条件や振る舞いを列挙する. 列挙する際はどのようなフォールトが起きることが考えられるかを検討材料にして列挙する．選択したテスト対象フィーチャ全部に対してサブトピックを列挙した後に，サブトピック全体を眺めて，象徴する名称を付与し，それをテストカテゴリにする．テストカテゴリは，メンバ間で内容を説明し，意味づけ（フォールトの言い換え）を共有することで，解釈のぶれを防ぐ．

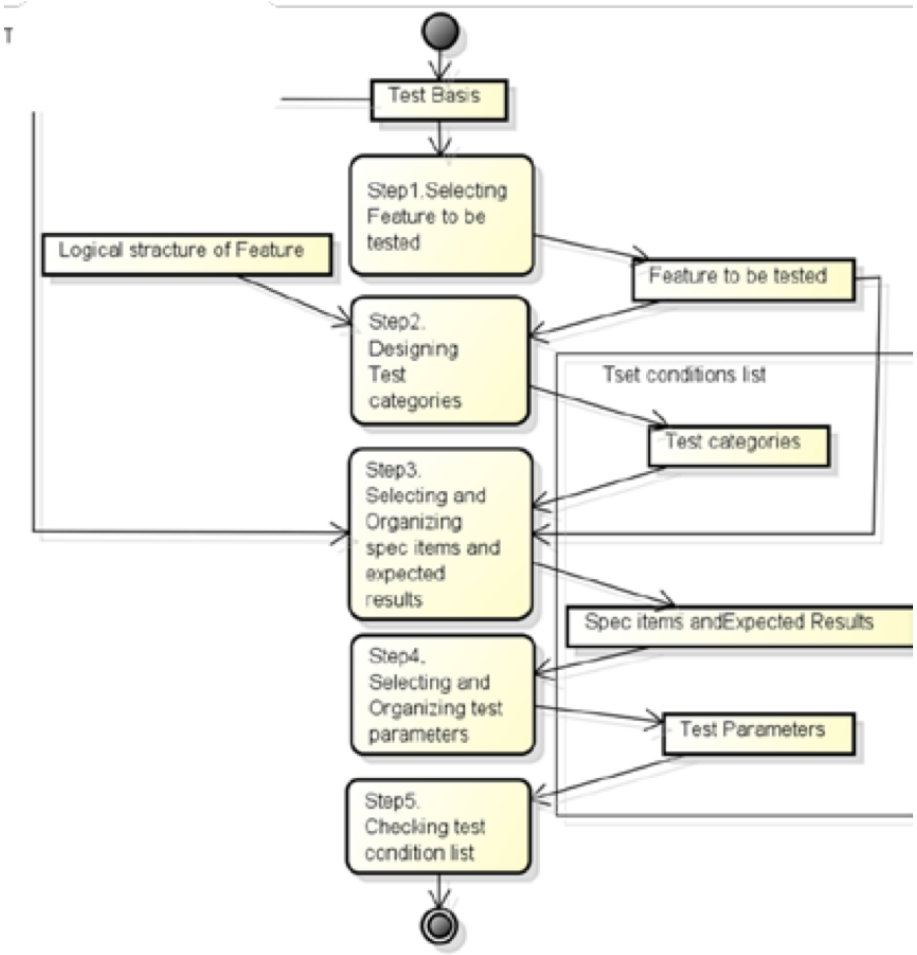


図 2.8: テスト分析の実行ステップ

Step3 テストカテゴリを使い仕様項目と期待結果を選択，整理する．

テストカテゴリでフィーチャの内部構造を推定し，実現するタスクを特定する．テストベースからそのタスクの仕様項目と期待結果の記述を抽出する．

Step4 テストパラメータを選択し，整理する．

選択した仕様項目と期待結果からテストパラメータを選択する．テストパラメータは選択した仕様項目と期待結果にヒントとなることが書かれているため，それを手がかりにテストベースを分析して選択する．

表 2.3: テスト条件一覧の例

テスト対象フィーチャ	テストカテゴリ	仕様項目	期待結果	テストパラメータ
TF-a	TC-a	SI-a	ER-a	TP1,TP2
		SI-b	ER-b-1	TP1,TP3,TP4
			ER-b-2	TP1,TP4
	TC-b	SI-c	ER-c	TP5,TP6,TP7
TF-b	TC-a	SI-d	ER-d	TP9,TP10
	TC-b	N/A	N/A	N/A

テストベースからテスト条件を特定する際は，その結果を表 3.1 に示すテスト条件リストにまとめる．

各テスト対象フィーチャには同じテストカテゴリのセットを列挙する．そして，各テストカテゴリに対応する仕様項目と期待結果を列挙する．テスト対象フィーチャによってはテストカテゴリに対応する仕様項目が無い場合もあり，その場合はテストカテゴリの中に列挙されるものが何も無いので N/A と記載する．

仕様項目によっては，条件によって期待結果が複数になることがあるため，その際は，リストには仕様項目に対して複数の期待結果を記載する．テスト条件リストの作成を通じて，各仕様項目と期待結果のセットに要求されるテストパラメータの組み合わせが特定できるようになる．テストパラメータはテスト設計プロセスの中で同値クラスと組み合わせを設計する．セクション I にて示し

表 2.4: テスト条件一覧の例　未作成

機能構造	テストカテゴリ	例	故障の例
変換	計算	料金の計算	計算ミス
入力調整	UI からの入力	画面からの入力	エラー
相互作用	反映	画面からの入力	エラー

たとおり，テストパラメータを設計するための手法や技法に焦点を当てた研究は数多くなされている．

多くのテスト開発に従事するテスト担当者が上記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，開発されたテスト条件のまとまりは，より包括的で，重複が含まれない．これは総合的に見て高いテストカバレッジを確かにし，高品質のテストを提供することにつながる．これは本手法の主たる効果となる．更に，この手順には 3 つの効果がある．

1. この手順は，本手法で提案しているテスト条件の構造をベースにしている [6]． テストベース内の要素は，仕様項目，期待結果，テストパラメータに分類できる．この手順を通して，各要素は 1 つずつ順番に特定，選択する． テスト分析にて同じロジックと手順で作成し，同じカテゴリに分類できた各メンバーの最終結果の可読性が向上する．

2. テストカテゴリに対する合意形成によって，チームメンバは仕様項目の特定と選択が容易になる．

3. 本手法は体系化され，標準化され，進めていくのか用意になるため， テスト担当者がテスト分析を繰り返すことができるようになる．

テストカテゴリ活用有無によるテスト条件導出結果の比較

テスト設計に関するワークショップを開催し，2 チームに分かれてテスト分析の作業ステップ 3「テストカテゴリを使った仕様項目と期待結果の選択」の演習を行った．ひとつのチームは，テストカテゴリを使わずに仕様項目，期待結果，テストパラメータを列挙してもらい，もうひとつのチームは，テストカテゴリ

を使って仕様項目，期待結果，テストパラメータを列挙してもらった．題材として音楽再生機器を選定した．出席者は全てこの機器のテストに関わった経験があり，製品知識はある．テスト対象フィーチャは，音楽再生機器のボリュームコントロール機能を選定した．

これまでの実験では，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を行なった．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．

一貫性のあるルールを適用することでテストケース作成に必要な仕様項目の特定の際に抜け漏れが少なくなることを確認している．しかしながら，今までの研究にて提案した手法は，テストベースの分析に論理的機能構造をガイドとして使用することを明示しているだけであり，具体的な分析手順について定義できていない．そのため，実験の際に被験者に対して，テスト分析にて仕様項目の選択を網羅的に行う具体的な方法を明確に提示できていない．

第3章　予備実験—テスト分析でのばらつき傾向とテストケース抜け漏れへの影響

3.1　検証実験の背景

2.4 節の「テストカテゴリベースドテスト」にて示した実験では，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，傾向を見るためには不十分である．

そのため，テスト分析手法を適用する前のテスト分析での結果のばらつき傾向，及びその傾向と適用後の結果との相関を調べることを目的に，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

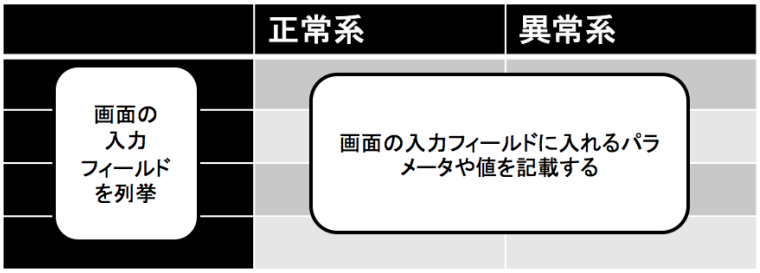
3.2.1　検証実験の手順

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，テスト開発プロセスを説明する．その後，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．

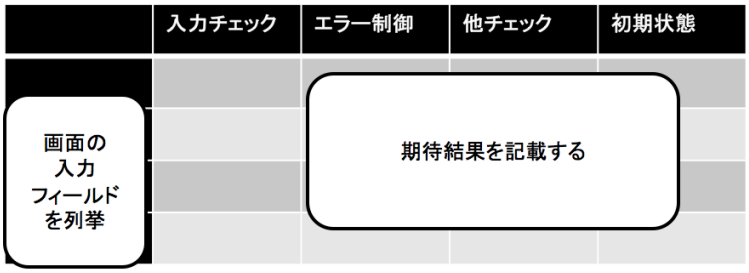
最初の分析結果のまとめが終わり，全出席者のが各グループの分析結果を理解した後，テストカテゴリを用いたテスト分析手法と実施手順を説明して手順

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.

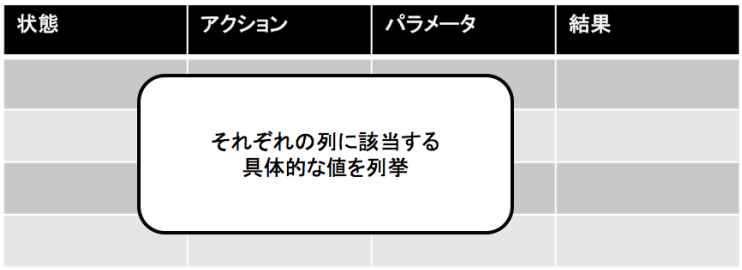
3.2.2　テスト分析手法適用前のテスト分析の結果



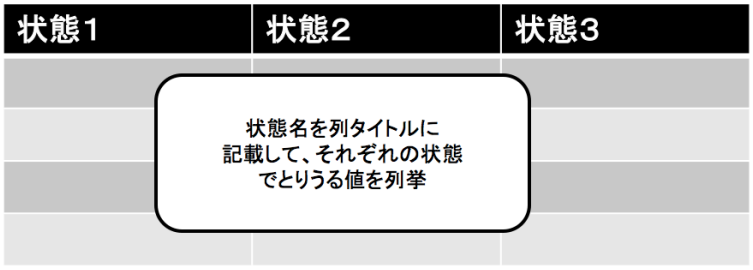
(a) テスト分析の事例：CS1



(b) テスト分析の事例：CS2



(c) テスト分析の事例：CS3



(d) テスト分析の事例：CS4

図 3.1: テスト分析の事例

テスト分析結果のばらつきは図 **??**から図 **??**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 **??**から図 **??**で示す．

1. CS1 と CS2:両方ともテスト対象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: テスト対象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．題材は，1 回目の実験では音楽再生機器を AUT にして実施した（6 チーム）．2 回目の実験では，飛行機予約システムを AUT にして実施した（2 チーム) ワークショップの時間は４時間である．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．表 3.1 は最初の演習結果のうちの 1 つの結果を比較した表である．

手法を知らないでテスト分析をした結果は Fig．1．のようにばらつくので，ワークショップの後に講師が仕様項目の数を計算できるように仕様項目の分類などをしている．

実験結果の評価のために，表 3.2 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 3.3 に示したとおりである．

テストカテゴリにそった演習による仕様項目の結果と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で比較をした．図 3.1 は最初の演習結果のう

表 3.1: Fig. 4. A comparison result table from team2(TM2) of the firstverification experiment.

Feature	**Test-Category**	**Spec item**	**result**	**Parametor**		**Feature**	**Test-Category**	**Spec item**	**expe**
	input	N/A	N/A	N/A			input	N/A	N/A
	ConvA	1	1	3			ConvA	1	
	SupportA	N/A	0	6			SupportA	N/A	
	OutputA	1	1	2			OutputA	1	
	StorageA	1	1	3			OutputB	1	
	StorageB	N/A	0	0			StorageA	1	
	Intraction	N/A	0	0				1	
		N/A	0	0			managementA	1	
		N/A	0	0				N/A	

表 3.2: 評価レベルの定義

評価レベル	比較結果
B	リストしたテスト条件数は増加していない, かつ実験の期待結果よりも少ない.
-	リストしたテスト条件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	リストしたテスト条件数は増加している, しかし実験の期待結果よりも少ない.
A+	リストしたテスト条件数は増加している, かつ実験の期待結果に達している. テスト条件の数は増加していない,.

表 3.3: 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

ちの 1 つの結果を比較した表である．手法を知らないでテスト分析をした結果は図**??**から図**??**のようにばらつくので，ワークショップの後に講師が仕様項目の数を計算できるように仕様項目の分類などをしている 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テスト条件の列挙数が増えた．

1. 最初の実験は，音楽生成機器が AUT 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．5 チームにてテストカテゴリ内に列挙した仕様項目の数が増えている．論理構造の項目ごとに集計すると，管理にて 5 チームの列挙数が増加している． 出力と貯蔵では，列挙数が増加したチーム以外は特定すべき仕様項目がすでに最初の演習で列挙できているため，効果があったと結論付けることが可能である．

2. 二回目の実験は，フライト予約システムが AUT 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仕様項目の数が増えており，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，図 3.2 のように，テスト分析手法を知らない状

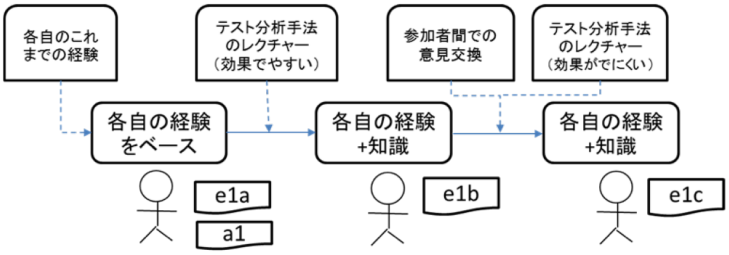


図 3.2: e1a から e1c までの演習の前提条件の変化

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．

テスト分析手法のレクチャーを 2 回に分けた理由は， 前述したように手法の実施手順がデータフローを使った手順とそれ以外の手順に分けられるため， 2 段階にわけることがワークショップの参加者にとって知識習得が容易になるであろうという判断からである．

e2 では，図 3.3 のように，e1a から e3a までで行ったレクチャーと演習を通じて得た知識とスキルが別の題材で活用できることを確認する． e1a と比較することで， スキルが向上しているかを観察する．このワークショップでは，57 名分の IT 技術者が参加した．参加者の構成は図 3.4 と図 3.5 のとおりである．年齢構成，業務領域ともに偏りがなく，産業界のサンプリングとして意味があると考える．参加者の技術経験と，テスト技法の研修受講有無（テスト技法に関する知識習得）については， 記述式の調査紙にて確認を行った．

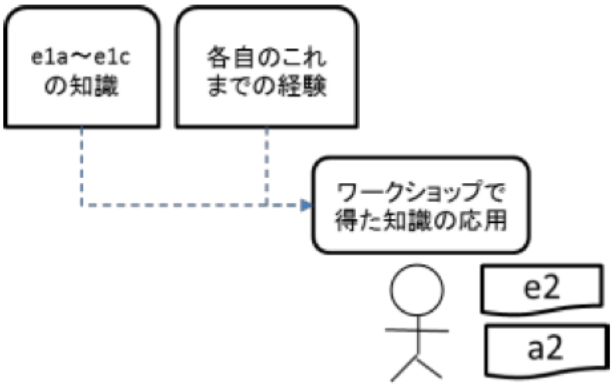


図 3.3: 図 6 図 6 e2 の演習の前提条件

e1a，e1b，e1c，e2 の演習結果において，各参加者が特定できたテスト条件数 (回答数) を図 3.6 の箱ひげ図を用いて比較する．図 3.6 の Y 軸は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．

例えば e1a では，最高点は 5 であり，中央値は 1 であった．正解数とした数は 9 なので，非常に低い値であった．レクチャー後の e1b では中央値が 3，e1c では 4，e2 では 7 と，演習が進むごとに中央値が増えているので，テスト条件数を特定するスキルが向上したと考えられる．ただし，e2 とそれ以外は演習題材が異なり，正解とした条件数が異なる (e2 は 23 で，それ以外は 9 である)．そのため，単純な数値の比較では不十分である．正解とするケース数を 100 とした箱ひげ図を図 8 に示す．

図 3.7 からは， e1a では約 10 % であったテスト条件の特定数の割合の中央値が，e2 では約 40 %まで向上したことが確認できる．ただし，図 3.6 と図 3.7 からわかるように参加者全員のテスト条件特定数が一律に上がったわけではない．効果があった部分とそうでない部分がどこであるかを調べるために，テスト条件ごとの特徴，および参加者の特徴でさらに分析をすすめた．

各参加者が特定できたテスト条件数 (回答数) を図 3.8 のように論理的機能構造で分類して比較した．図 3.8 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．

表 4.4 は， 1 から 9 までのテスト条件を対応する論理的機能構造とテストカ

表 3.4: e1a から e1c までの演習結果の変化表

	論理的機能構造	テストカテゴリ	難しさ	効果
1	入力調整	ボタン	難	中
2	出力調整	音声出力	中	高
3	変換	音量	易	低
4	貯蔵	設定保存 1	難	中
5		設定保存 2	難	高
6	サポート	状態遷移	難	中
7	相互作用	対向機反映	難	低
8		設定情報共有 1	難	中
9		設定情報共有 2	難	中

難しさ　0-10 難　11-25 中　26-易

効果　0-10 低　11-25 中　26-高

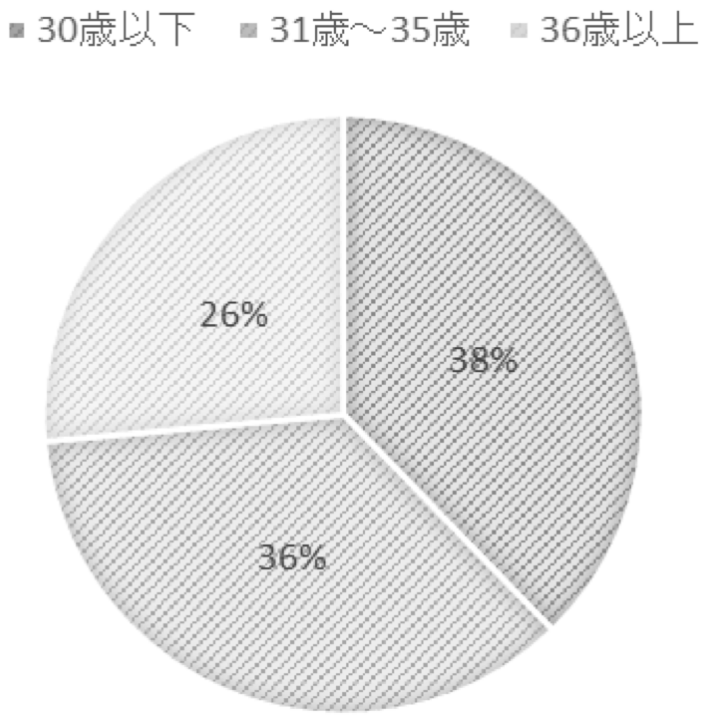


図 3.4: 参加者の年齢分布

テゴリを示し，難しさと教育効果を示した．難しさは，正解数の分布から 3 段階に分けた．教育効果は，e1a(教育前) と e1c(教育後) の差を 3 段階で分類した．

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 3.9 に示す通り、大きく 4 パターンに分類できた．

今回のワークショップでは，e1a の演習にて，解答をこれまでの経験に基づいて自由に書いてもらうようにした．この結果，テストの記載パターンが 4 つに分類できることがわかった．

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．

これらの実験結果から特定の要因は見出せなかったが，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．

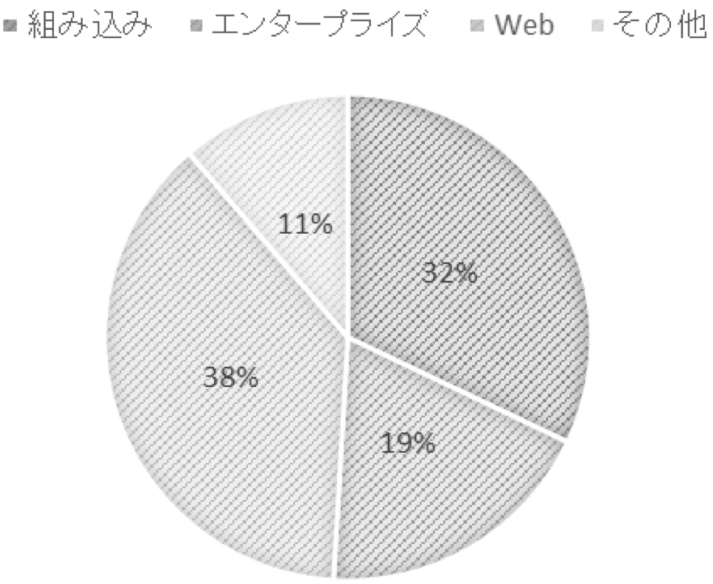


図 3.5: 参加者の業務領域分布

表 3.5 の 1 と 3 は中間成果物的であり， 記載した内容を見てそのままテストを実行するには不向きである．一方，2 と 4 はそのままテスト実行時に利用できる． 一方， 分析や設計をすると 1 と 3 が成果物になる．自由に記載してもらう際に分析結果から書くことは，普段の業務でも分析や設計をしていると想定できる． なので， 2 と 4 を直接書くのは， 普段の業務であまり分析や設計行為をしていないのではないかという仮説を持った．仮説が正しければ， 普段から業務にて分析や設計をしている参加者のほうが， 慣れているために知識の習得が早いと想定し， 今回のワークショップを通じた演習結果にてこれまでの記載方法と演習の成果に相関があるかを調査した．図 3.10 がスピアマンの順序相関分析をした結果である， e1a では， 仕様項目から記載する参加者と特定できたテスト条件数には， 0.51(0.4 以上の値は相関ありといえる) の相関が出たたがそれ以外は 0.4 以上の値は出なかった． グラフの傾向からは， e2 では分析的な記述をしていた参加者のほうが実装的な記述をした参加者より正の相関となったが，分析結果の値は 0.2 をきっているため，相関があるとは結論付けられない．

これらの検証実験にて，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

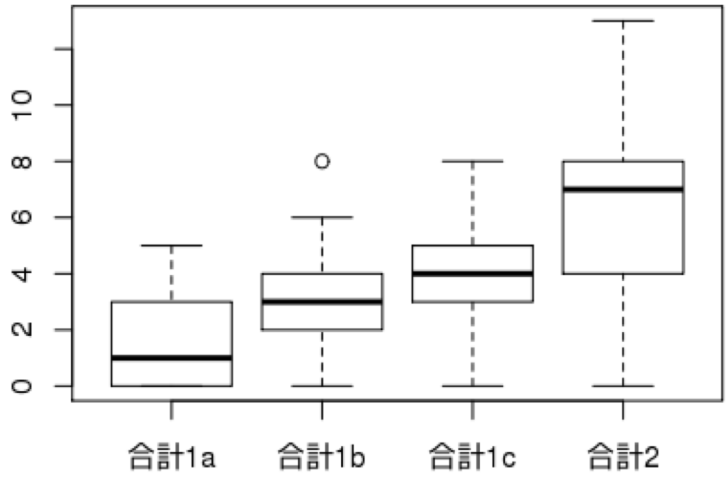


図 3.6: 参加者あたりのテスト条件特定数

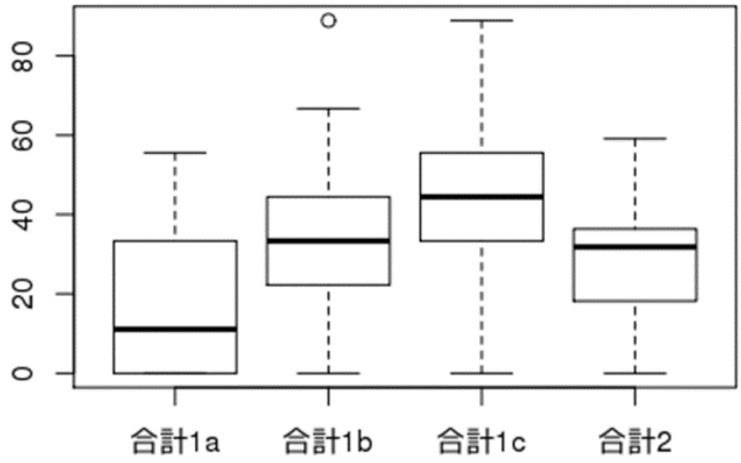


図 3.7: 参加者あたりのテスト条件特定割合

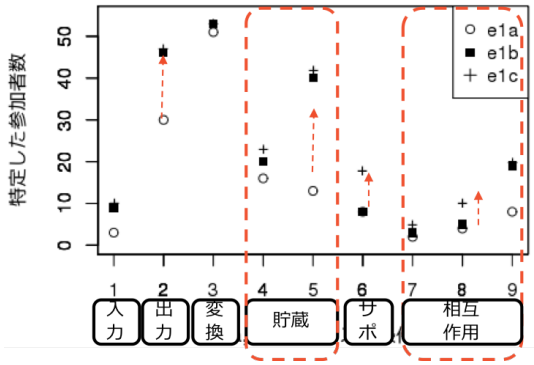


図 3.8: 参加者あたりのテスト条件特定数

表 3.5: テスト記述パターン

	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

	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 3.9: テスト分析パターン

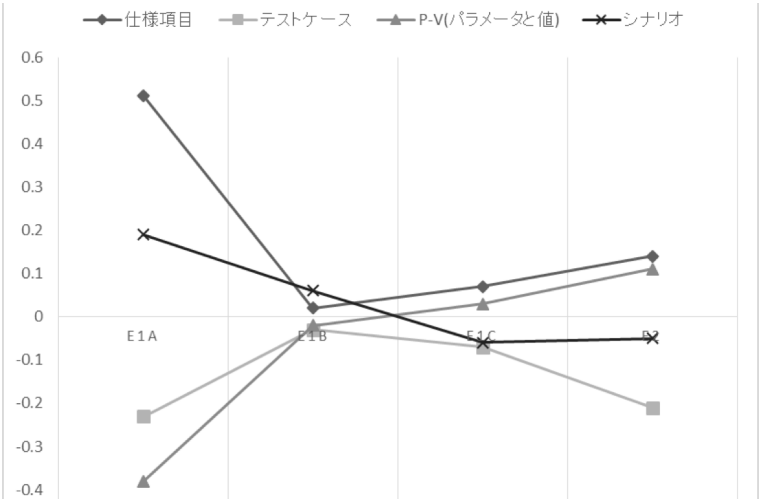


図 3.10: e1a の参加者業務分野別テスト条件特定数

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．

e1a の仕様項目を記載する参加者だけ相関が出たのは，今回の演習で特定するテスト条件が仕様項目そのものであるため， 最初の演習では仕様項目を記載した参加者と結果の相関が出たと考えられる．

3.4 終わりに

3.5 謝辞

本研究は，WACATE(http://wacate.jp/) の場で行ったワークショップの結果を基に行いました，WACATE 実行委員と WACATE2015 夏の参加者の皆様の協力で

実現することが出来ました．ここに感謝の意を表します．

第4章　I/O テストデータパターンを使ったテストケース抽出手法

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し，テストケースを抽出する方法として，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，データをテスト対象にインプットし，テスト対象のアウトプットを期待結果と実際の結果で比較する．

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，テスト対象の外部から複数の値を入力し，テスト対象がそれらの値を計算し，計算結果をテスト対象の外部にアウトプットする．

これは図 4.1 で示している「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．

また，固定比率が計算結果に適用されるとき (他の計算も適用されると言う意味で)，テスト対象は適切な比率を呼び出し，利用してから計算結果をアウトプットする．これは図～**??**で示している例「外部と内部からのデータ入力，外部へのデータ出力」となる. 注意すべきこととして I/O データパターンはテスト対象の外部からの観察が可能なものを選ぶことがあげられる．処理の起動終了に使われる内部のコマンド（シグナルやイベント）は，データパターンへ分類

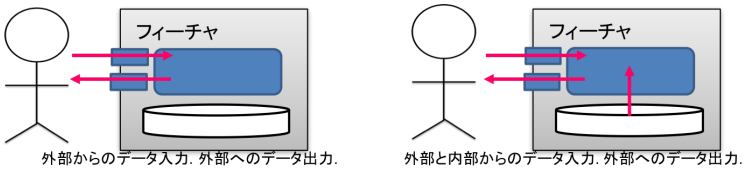


図 4.1: テスト対象へのデータ入出力の説明

をするときに考慮しない．なぜならこの手法はブラックボックステストのためのテストベースの分析手法であり，AUT 内部のシグナルやイベントのような内部コマンドは，システムテストレベルでのブラックボックステストでは，明示的に考慮しないからである．

テスト実行をするときのテスト対象へのデータのインプットの方法は，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．同じようにテスト対象からのアウトプット方法は外部への出力，内部に保持したデータの出力，外部と内部からデータの出力の 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．

	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 4.2: I/O データパターンの説明

テスト対象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 4.2 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．

Whittaker によって提案されているフォールトモデル [15] は，AUT の入力と出力のモデルに関する類似の研究である．フォールトモデルの目的はフォールトを見つけることであり，AUT の入出力モデルはテストベースから特定するテ

スト条件である，仕様項目を分類するモデルとして使われる．

テスト分析のアウトプットである仕様項目は最終的に全て実行して確認することができなければならないので，全てが 9 パターンのどれかに分類できる．

4.1.3 I/O テストデータパターンとテストカテゴリ

検証実験からの結果を I/O データパターンに分類するために，P1 から P9 の識別子を仕様項目の属性として付与した．その後，3 章の表表 3.2 で示した評価レベルのアプローチをベースに整理した．

実験結果は表 4.1 と表 4.2 に示したとおりである．前節の結果と本節の結果では行数が異なるの理由は， 仕様項目のいくつかは同じテストカテゴリに属する仕様項目でも異なった I/O データパターンを付与している場合があり，そのためテストカテゴリを論理構造の項目で集約をしていないためである．表 4.1 と表 4.2 が示すとおり， P1， P2， P4， P7 が検証実験で使った AUT に対応するデータパターンであった．

表 4.1: Fig. 8-1. The evaluation results of the I/O data pattern.

Logical Structure				Team					
				TM1	TM2	TM3	TM4	TM5	TM6
Conv	P1	ボリューム強弱		-	-	-	-	-	-
Conv	P1	対向機		B	A+	B	B	B	-
Support	P1	状態遷移		B	B	B	B	B	B
Mngt	P1	電話と音楽の音量の影響		B	A+	A+	A+	A+	A+
Storage	P2	ボリューム値保存		-	A+	-	A+	A+	-
Conv	P4	デフォルト値/設定値		B	B	B	B	-	B
Mngt	P4	リセット		B	B	B	A+	B	A+
Output	P7	ビープ音		-	-	-	-	-	A+

I/O データパターンで集約した結果を表 4.3 と表 4.4 である．各 I/O データパターンにて A and A+ の割合を確認すると，P2 のみが両方の検証実験で 100 %となったため，P2 のデータパターンに対する効果が最も高い．

表 4.2: Fig. 8-2. The evaluation results of the I/O data pattern.

Logical Structure			Team	
			TM1	TM2
Input	P1	P1	B	B
Support	P1	P1	-	A+
Support	P1	P1	B	B
Storage	P2	P2	A+	A+
Input	P4	P4	-	-
Mngt	P4	P4	B	A
Output	P4	P4	B	B
Output	P4	P4	A	A
Input	P7	P7	A	-
Conv	P7	P7	A	A
Output	P7	P7	B	B

音楽再生機器の場合，P2 に分類された仕様項目はボリューム値の保存である．この仕様項目はテストベースに記述はされているものの，ボリュームコントロールのセクションとは別のセクションに記述されていた．飛行機予約システムの場合，P2 に分類された仕様項目は注文内容のデータベースへの保存である． データベースへの保存に関して，テストベースには，詳しい記載は無かった． これらの観察結果はセクション III に前述した推測である「明白に必要だと思われる仕様の一部分が記述されていない.」と一致している．

表 4.3: The summarized results of the I/O data patterns.

I/O pattern	Evaluation level				
	A+	A	-	B	
P1	6	0	7	11	35.3%
P2	3	0	3	0	100.0%
P4	2	0	1	9	18.2%
P7	1	0	5	0	100.0%

the persentage = (A+ + A) /(A+ + A + B)

表 4.4: The summarized results of the I/O data patterns

I/O pattern	Evaluation level				
	A+	A	-	B	
P1	1	0	1	4	20.0%
P2	2	0	0	0	100.0%
P4	0	3	2	3	50.0%
P7	0	3	1	2	60.0%

the persentage = (A+ + A) /(A+ + A + B)

I/O テストデータパターンと論理的機能構造の要素を対比させて，各パターンがどの要素に該当するかを表 4.5 にまとめた.

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．

表 4.5: Add caption

		入力調整	出力調整	変換	貯蔵	サポート	相互作用
入力	外部	P1，P2，P3		P1，P2，P3			
	内部			P4，P5，P6	P4，P5，P6		
	外部と内部	P7，P8，P9		P7，P8，P9	P7，P8，P9		
出力	外部		P1，P4，P7				
	内部				P2，P5，P8		
	外部と内部		P3，P6，P9		P3，P6，P9		

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 4.3 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．そのため，表〜ref tbl:D-4-tbl1 の 3 箇所にプロットされている．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．

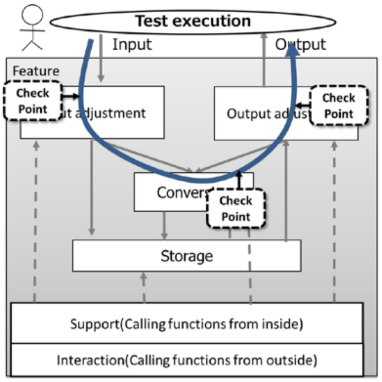


図 4.3: I/O テストデータパターンのデータの流れ

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．

また，表 4.5 では，サポートと相互作用についてはデータパターンがプロットされていない．なぜなら 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だからである．

論理的機能構造の要素である入力調整，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るのに対して，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仕様項目を特定するための分類である．

サポートと相互作用に分類される仕様項目は，呼び出した後の出力で期待結果を確認する．呼び出された機能の I/O テストデータパターンは，論理的に全パターンが発生する可能性がある．そこで，これまでの研究で行った被験者を使ってテストカテゴリを使った分析手法の習得前と習得後を比較する実験にて使用したテスト分析の講師解答例を同じフォーマットの表に当てはめて，実際の傾向を調査した．

4.1.4 これまでの実験データを使った調査

3 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した．この実験では，ヘッドセットのフィーチャであるボリュームコントロールと，フライト予約システムのフィーチャである新規フライト予約の 2 種類の異なった AUT を実験に使用した．実験で解答例として作成したテスト分析結果を I/O テストデータパターンで整理した結果が表 4.6 である．この結果からわかるとおり，実際に現れたパターンは，P1 と P2 と P4 と P7 だけであり，P1 から P9 のパターンの全てが現れなかった．また P1 でプロットされているのは Input Adjustment と Conversion のみであり，Output Adjustment に該当する仕様項目は無かった．同様に P2 は Storage のみ，P4 は Conversion と Output Adjustment のみであり，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．

表 4.6: Add caption

		入力調整	出力調整	変換	貯蔵	サポート	相互作用
入力	外部	P1(F)		P1(V)			
	内部			P4(V)			
	外部と内部	P7(F)		P7(F)	P7(F)		
出力	外部		P4(F)P7(VF)			P1(V)	P1(V)P4(VF)
	内部				P2(VF)	P2(F)	
	外部と内部						

4.1.5　サポートと相互作用に関する考察

前述したように，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．

TableII では，Support と Interaction に分類できる I/O テストデータパターンを特定できていなかったが，実際のテスト分析結果から調査した結果，TableIII で示したとおり，P1 と P2 と P4 と P7 に仕様項目を分類できた．

TableIV には，TableIII と同実験にて Support と Interaction に分類した仕様項目を列挙した．

サポートと相互作用として特定する仕様項目の傾向について以下のような考察が出来る．

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．この例では，全てテスト対象の入力に対して結果を返すだけであるため，I/O テストデータパターンは P1 としている．

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．

この例では，ボリュームコントロールの 2 つの仕様項目は，副作用を確認する際に，該当する他フィーチャにてテスト実行時に外部から入力を与えて結果を確認するため P1 にしている．フライト予約システムの場合は，新規フライト予約にて登録した新規予約が反映していることを他のフィーチャにて確認することを指しているが，テスト実行の際は該当のフィーチャに対して外部入力を与えずともすでに保存された結果の出力で確認ができるため P4 としている．

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．整理した結果を TableV に示す．

TableV には，各仕様項目を特定する際のきっかけとなる呼び出し方法を列挙した．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することで，他の AUT に適用する際に活用できると考えられるためである．呼び出しのきっかけと I/O テストデータパターンの組み合わせは TableV のように整理できた．

表 4.7: Add caption

フィーチャ	テスト条件	**I/O** テストデータパターン	呼び出し	論理的機能構造
ボリュームコントロール	再生中，通話中以外音量値の調節を無視する	P1	割り込み	サポート
	通話と再生の音量値を調節しても互いに影響を受けない	P1	リソース共有	相互作用
	リセットで音量値がデフォルト値に戻る	P4	リソース共有	Interaction
新規フライト予約	「既存注文検索」へ注文が反映すること	P4	他への反映	Interaction
	「注文件数グラフ」へ注文が反映すること	P4	他への反映	Interaction
	「注文履歴」へ注文が反映すること	P4	他への反映	Interaction
	登録時にチケット在庫なしの場合エラーになること	P1	他処理連動	Support
	注文挿入中に強制終了すると処理をロールバックすること	P2	他処理連動	Support

表 4.8: Add caption

呼び出し 論理的機能構造	割り込み	リソース共有	他への反映	他処理連動
サポート	○			○
相互作用		○	○	

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

これまでの実験では，被験者の学習過程に対する効果を検証してきた．また，実験用に用意した小さなサンプルを利用してきた．

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テスト条件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．

この実験は以下の目的で行う．

I/O テストデータパターンの効果実証

- 目的：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確認する．
- 評価方法：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　実験の題材

実験対象のテスト対象は，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．簡単なサンプルでは出現しないデータパターンもあるため，現実の複雑なアプリケーションを対象にした．全機能のうち,「アップロード（デバイス上の写真をオンラインサーバへアップロード)」,「グリッドビュー（オンライン上の写真をデバイスにてサムネイルの一覧として閲覧)」という二つのFeature をテスト対象として選択した．この二つの機能を選択した理由は，データの内部への投入を行う機能とデータの照会のみ行う機能とで，I/O テストデータパターンの出現傾向が異なること調査することが出来ると考えたためである．この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．

4.2.3　実験の実施手順

実験は以下の手順で行なった．

- 実際に作られたテストケースをテスト条件にまとめなおす．
- 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．
- I/O テストデータパターンを付与する．
- 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．

1) 実際に作られたテストケースをテスト条件にまとめなおす

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．テストケースは，入力値や事前条件を組み合わせた複数のインスタンスであるため，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仕様項目と期待結果としてまとめなおす必要がある．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．まとめ直す際には，図～ref fig：D-4-Fig8 のように，テストケースの要素を整理し，同じアク

ションを行い，同じ期待結果を確認しているテストケースはひとつの仕様項目としてまとめた．

Step summary	Step Description	Expected Result
Sort order using 2 photos	1.Prepare 2 photos which are different photographing date. 2.Upload 2 photos to an album that is prepared. 3. Select Grid view from a main menu. 4. Click sort button. 5. Change sort order to photographing date in ascending order.	Grid view is ordered in date in ascending order.
Sort order using a lot of photos which display more than 2 pages	1.Prepare 40 photos which are different photographing date. 2.Upload 2 photos to an album that is prepared. 3. Select Grid view from a main menu. 4. Click sort button. 5. Change sort order to photographing date in ascending order.	Grid view is ordered in date in ascending order.

Test Category	Spec-item	Expected Result	Test parameter
Image display	Change sort order	Grid view is ordered in decided sorting	·Ascending or descending, ·number of photos · sorting rules

図 4.4: テストケースから仕様項目をまとめる方法の説明

現場で作られたテストケースの数は，アップロードが 491 ケース，グリッドビューが 151 ケースであった．仕様項目として整理した結果，アップロードが 59 項目，グリッドビューが 22 項目の仕様項目となった．このように数量が変わる理由は，たとえば「デバイスからサーバーへ画像ファイルをアップロードして保存が出来ること」というひとつの仕様項目に対して，テストケースは，画像の種類（Jpg，Bmp など），画像のサイズ，アップロードする画像の枚数，画像情報のパターン（ファイル名，撮影日など）といったテストパラメータを組み合わせたものがテストケースとなっているためである．

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以下の 4 つを入力データ，出力データとして扱うこととした．

- 画像データ
- 画像の情報
- 設定データ
- 外部コマンド

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) で明らかにした仕様項目に対して Fig.9. のリストのように Input data ,Output data 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．図～ref fig：D-4-Fig9 は，ソート順の情報を外部から入力し，内部からの入力となる画像データと一緒になり，外部にソートした画像データを表示している例である．この場合の I/O テストデータパターンは P7 となる．

Test Category	Spec-item	Expected Result	Input data	Output data
Image display	Change sort order	Grid view is ordered in decided sorting	Outside: Sorting data Inside: Image data	Outside: Image data Inside:

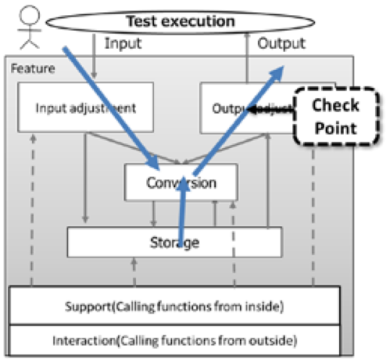


図 4.5: 仕様項目に入力データと出力データを加える方法の説明

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う．作業ステップは以下のとおりである：

- TableVI，VII のようにテストカテゴリを特定する
- テストカテゴリ毎に入力データと出力データを明らかにする
- I/O テストデータパターンごとのデータの流れをシミュレーションして仕様項目を選択する

- 現場のテストケースを分析した結果をテストカテゴリに分類し，差異を比較する．

サポートと相互作用については，テストカテゴリとして特定した Trigger で呼び出す機能から仕様項目を選択した．

4.2.4 実験結果

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にまとめた．

表 4.9: Add caption

	P1	P2	P3	P4	P5	P6	P7	P8	P9
アップロード	○	○	○	○	X	○	○	X	○
グリッドビュー	○	X	X	○	X	X	○	X	○

今回の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 両方でテストカテゴリと 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が適用できた．そして，両方のフィーチャにて，現実のテスト対象に おけるテスト設計と比較し，現場のテスト設計に仕様項目が不足していることが実証できた．分類に利用した I/O テストデータパターンは，P5 と P8 を除く全てであった．

実プロジェクトのテスト条件との比較をした結果を図 **??**に示す．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テスト条件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，Table IX. である．

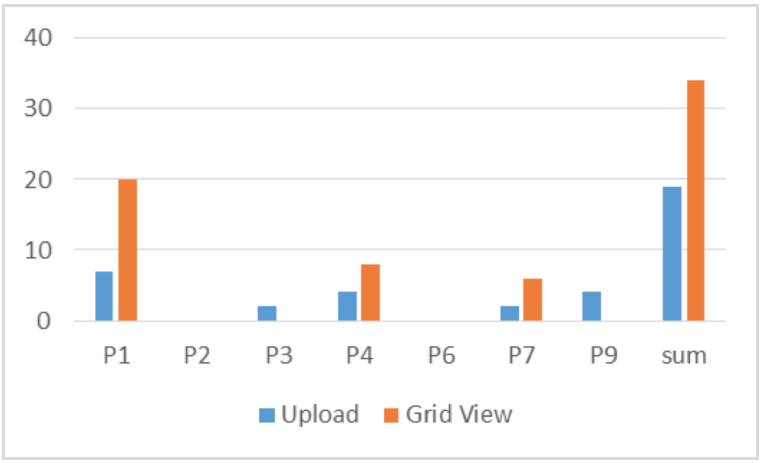


図 4.6: IO テストデータパターンごとの違い

現実のプロジェクトにて不足していたテスト条件には，I/O テストデータパターン別にみると 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行い，外部に出力する最も単純なパターンであり，I/O テストデータパターンから見ても単純なことを確認するテスト条件が漏れている．

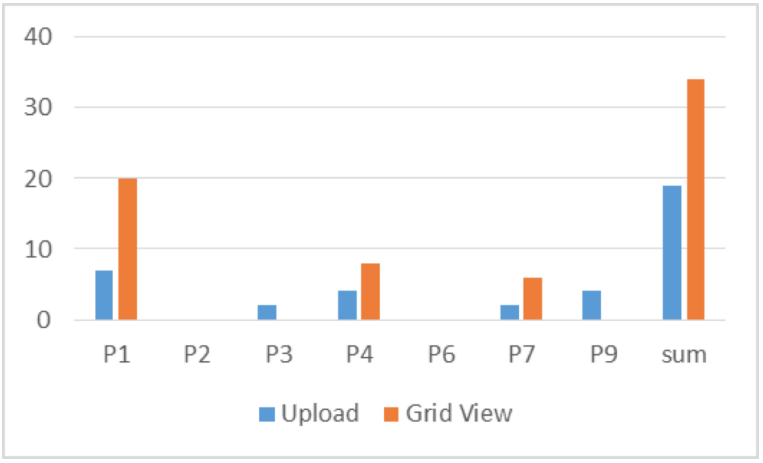


図 4.7: IO テストデータパターンごとの違い

3) 現実のプロジェクトにて不足していた仕様項目

TableX には，現実のプロジェクトにて不足していた仕様項目をいくつか抜粋して列挙した．これらの仕様項目は，全て今回のデータの I/O のシミュレーションを行い網羅的にテストベースを確認することで特定できたものである．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．

4.3 終わりに

本論文では，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．

4.4 謝辞

We would like to thank NPO ASTER to use test cases in the real project for the experiment and for allowing us to publish these results.

第5章　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 入力情報

一般的にテストケース抽出のために必要な入力情報をテストベースと呼ぶ [16]. 提案手法に必要なテストベースは DFD，ER 図，CRUD 図である．以下，DFD，ER 図，CRUD 図を簡潔に説明する．

- DFD（データフローダイアグラム）

  DFD はシステムにおけるデータの流れを表現した有向グラフであり，要求分析において用いられている．DFD はデータ指向設計の要として用いられ，オブジェクト指向設計においても抽象化する前段階として実践の場で用いられている．

  DFD は，最上位のコンテキストレベルから階層として詳細化され，各階層は 1 枚以上の DFD から成る [16]．テストベースとして用いる場合，テストの範囲は DFD で与えられるとする．DFD の階層が下がると単体テストとなり，上がると統合テストとなる．

  DFD はノードとエッジからなる．ノードは 3 種類の要素である N 個のタスク（プロセス）Ta と，M 個の保持データ（データストア）Ds と，L 個

の源泉（外部エンティティ）So から構成されている．３種類の要素を一意に特定する際は Ta_i，Ds_j，So_k と表記する．

エッジは，ノードからノードへのつながりを有向線分で表記している．エッジはデータの流れを表しており，制御の流れは表していない．エッジの特定は，起点ノードと終点ノードを用いて行う．ある特定のタスクからデータストアへの入力がある場合のエッジの特定は，Ta_i/Ds_j となり，源泉から出力してタスクで処理をする場合は，So_k/Ta_i と表す．

- ER 図

  ER 図はシステムにおけるエンティティ間の関係を示す図であり，UML のクラス図に対応している．DFD では表現できないエンティティの詳細化やエンティティ間の関係について示しており，DFD と共に用いられている．

  ここでは，DFD のデータストア Ds_j が持つエンティティと，CRUD 図の対応から，後述する拡張 CRUD 図を作成するために用いる．よって，テストベースとしては，システムすべての ER 図を必要とするものではない．

- CRUD 図

  CRUD 図とは，タスク Ta_i からデータストア Ds_j への C : 生成，U : 更新，R : 参照，Ds : 削除の操作を表した図である [17]．CRUD 図から，DFD と ER 図では表現されていないタスクのエンティティへの操作を知ることができる．

  本論文では，タスクがデータストアに対して行う操作を特定するために CRUD 図を用いる．タスク Ta_i のデータストア Ds_j に対する操作が U : 更新であればタスクによる操作はエッジを介した操作として Ta_i/Ds_jU と表記する．ただし，タスクが操作するデータストアが 1 つだけの場合は，Ta_iU といった省略した表記を使う．

5.2.2 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．

- ルール 1 : 変更タスクの特定

表 5.1: 拡張 CRUD 図

タスク	データストア			源泉		
	Ds_1	...	Ds_j	So_1	...	So_k
Ta_1						
...						
Ta_i						

対象とする DFD 内の変更タスクのうち，データストア Ds へ出力エッジを持つタスクを選択し，順序組合せの変更タスク群 $P\{Ta\}$ とする．変更タスク群からの出力するデータストア群を $P\{Ds\}$ とする．

- ルール 2：波及タスクの特定

  ルール 1 で求めた $P\{Ds\}$ からの入力エッジを持つタスクを波及タスク群 $S\{Ta\}$ として特定する．

- ルール 3：順序組合せテストケースの抽出

  拡張 CRUD 図を基に変更タスク群 $P\{Ta\}$ とそのデータストア群 $P\{Ds\}$ を介する波及タスク群 $S\{Ta\}$ を組合せ，順序組合せのテストケースとする．

以降からは，順序組合せを抽出してテストケースとするまでの実施手順を詳細に説明する．

5.2.3 ルール 1：先行タスクの特定

ルール 1 を用いて先行タスクとそのデータストアを特定し，拡張 CRUD 図の先行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．表 5.1 に拡張 CRUD 図の表記を示す．拡張 CRUD 図のデータストアに対する情報は C，U，R，D のいづれか，または組合せか空白である．源泉に対する情報は In か Out，または組合せか空白である．空白は関係が無いことを示す．

表 5.2: 中間の拡張 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		

1. 源泉からの入力エッジを持つ変更タスクの特定

   テストケースは，外部からのテスト対象への入力から，外部への出力結果を確認するものであるため，テスト入力とテスト結果のペアで構成されている．そこで，テスト対象範囲の外からの入力，即ち So_k からの入力エッジを持つ Ta_i を見つける必要がある．この特性を持ったタスクのうち，さらに変更のあるタスク群を変更タスクの集合となる $P\{Ta\}$ 候補とする．変更が特定の状態でのみ起こり得る場合は，タスクの後に変更が起きる状態を［St_l］と記載する．

2. データストアへの出力エッジを持つタスク特定

   Ta_i から Ds_j への出力エッジは，C か U か D の操作を行うことを意味する．CRUD 図から該当する出力エッジを持つ Ta_i を選択する．$P\{Ts\}$ 候補の中から，該当する Ta_i を選び，変更タスク群 $P\{Ta\}$ を確定する．

3. 中間の拡張 CRUD 図作成

   拡張 CRUD 図には，変更タスク群 $P\{Ta\}$ に該当する So_k から Ta_i への入力 (In)，もしくは Ta_i から So_k への出力 (Out) の情報を付加する．特定した Ta_i に対して，入力となる So_k に In を記入し，Ds_j については CRUD 図を参照して C か U か D かその組合せかを記入する．中間の拡張 CRUD 図として例示した表 5.2 では，3 つの源泉 $\{So_1, So_2, So_3\}$ と 3 つのデータストア $\{Ds_1, Ds_2, Ds_3\}$ があり，2 つのタスク $\{Ta_1[St_1], Ta_3[St_1]\}$ が変更タスクである．この段階で作成する拡張 CRUD 図は，作業途中のものである．

5.2.4 ルール 2：後続タスクの特定

ルール 2 を用いて後続タスク群を特定し，拡張 CRUD 図へ後続タスク部分を追加し図を完成させる．

1. データストアを介した後続タスク特定先に作成した中間の拡張図から先行タスクの操作が 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候補となる．波及タスクとして選択するタスクは表 5.3 に示す表の○印の組合せに該当するタスクである．波及タスクは，C : 生成，U : 更新，D : 削除を選択する．”- ”をつけた組合せは，データストアを介した影響が生じないため，組合せテストの対象としない．”× ”をつけた組合せは仕様上有り得ない組合せであり，ありえないことの確認は，順序組合せを網羅しなくともよいため，組合せテストの対象としない．

   着目したデータストアに対してエッジを持つタスクが後続タスクのうち，源泉に出力エッジを持つタスクを後続タスクとして特定する．特定した後続タスクを拡張 CRUD 図に追記し完成させる．

   表 5.3: タスク間のデータ共有の組合せパターン

		$P\{Ta\}$			
		C	R	U	D
$S\{Ta\}$	C	×	×	×	○
	R	○	-	○	-
	U	○	-	○	×
	D	○	-	○	×

2. 拡張 CRUD 図の完成波及タスク候補のうち，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．波及タスクの特性を DFD より読み取り，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 5.4 に示す．この例では，データストア Ds_1 から源泉 So_2 への流れをタスク $Ta_2[St_1]$ が行い，データストア Ds_2 から源泉 So_3 への流れをタスク $Ta_5[St_1]$ が行っていることを示している．

表 5.4: 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

5.2.5 ルール 3：順序組合せテストケースの抽出

1. 変更タスクと波及タスクの組合せを抽出

   拡張 CRUD 図から変更タスクを選ぶ．先に作成した拡張 CRUD 図の例（表 5.4 を参照）であれば，$Ta_1[St_1], Ta_3[St_1]$ である．次に変更タスクが操作しているデータストアと，それを操作している波及タスクを対応付ける．例では，$Ta_1[St_1] \underset{Ds_1}{\longrightarrow} Ta_2[St_1]$ と $Ta_3[St_1] \underset{Ds_2}{\longrightarrow} Ta_5[St_1]$ である．

2. データストアに対する操作の組合せ

   操作の組合せとは変更タスクと波及タスクの操作の組合せである．表 5.4 の例であれば，変更タスクのデータストアに対する操作である Ta_1 は，Ds_1 に対して C と U の操作を行っている．波及タスク Ta_2 の操作は R である．組合せは $C \rightarrow R$ と $U \rightarrow R$ となる．変更タスクと波及タスク間に介在するデータストアが 1 つであれば $\underset{Ds_1}{\longrightarrow}$ を省略して $\rightarrow$ で表してもよい．また変更の発生条件となる状態が 1 つであれは，$[St_1]$ を省略してもよい．表 5.4 の例における全組合せは，$Ta_1C \rightarrow Ta_2R$，$Ta_1U \rightarrow Ta_2R$，$Ta_3C \rightarrow Ta_2R$，$Ta_3C \rightarrow Ta_2U$ の 4 個である．

3. テストケース表の完成

   変更タスクと波及タスクの操作の組合せをテストケースとしてまとめる．表 5.5 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．ここで用いたテストケースとは，ISTQB の定義による論理的テストケースに相当する [9].

表 5.5: 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	$Ta_3C \rightarrow Ta_2U$

具体的な値や期待結果，該当の処理までの状態を遷移させていく手順まで定義した記述を具体的テストケースと呼ぶが，本論文では扱わない．

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．

5.3.2 題材の概要

フライト予約システムの概要を以下に示す．

題材となるフライト予約システムの仕様は，本研究の一環として評価実験の際に題材として使っているものである [18]．テスト対象の分析と，テストケース設計に関する用語は，国際標準である ISO/IEC/IEEE29119 の定義に従い，テスト対象の論理的なサブセットを機能セット（Feature set）と呼ぶ [19]．本論文では，表 5.6 の新規フライト予約を，変更が入った機能セットとする．

新規フライト予約からテストケースを抽出するための前提として用意した仕様は，新規フライト予約に関連する DFD と ER 図（図 5.2），CRUD 図（表 5.7）とする．DFD に含まれるタスク数 N は 6，データストア数 M は 2，源泉数 L は 4 である．

＜フライト予約システム概要＞
・旅行代理店用に開発したフライト予約サーバにインターネット経由でアクセスできる専用のクライアントアプリケーション．
・旅行代理店の窓口での利用を想定しており，ユーザ認証されたユーザのみ利用可能である．
・旅行代理店の窓口数（クライアント数）は 50 としており，同時に予約処理を行うことができる．
・旅行代理店にて取り扱う全ての航空会社の飛行機の予約が可能である．
・本システムは，フライト予約サーバを仲介して複数の航空会社のシステムと同期をする．
・チケット情報や残チケット数は同期することで最新に更新される．
・フライトの新規予約、予約内容の更新，削除が可能である．更新と削除は新規予約したユーザのみ可能である．
・以下はシステム範囲外
–チケット代金の決済（別システムと連携して行うため）．
–マスタ情報設定（他システムとの共用マスタ設定アプリケーションがあるため）．

図 5.1: フライト予約システムの概要

5.3.3 ルール 1：変更タスクの特定

テストベースである DFD に含まれるタスク数 N は 6 であるが，変更が入った新規フライト予約の変更タスクは，表 5.7 の CRUD 図を確認するとフライト検索 Ta_1 とフライト予約 Ta_2 であることがわかる．図 5.2 から，Ta_1 と Ta_2 の外部入力を確認する．Ta_1 は，CustomerSo_1 から ETD と Destination を外部入力し，Ta_2 は，CustomerSo_1 から FlightNo，ClASs，Order number，PAX を外部入力している．

続いて，Ta_1 と Ta_2 の内部出力を確認する．Ta_1 は Flight infoDs_1 に対して検索条件を与えているのみで内部入力はしていないため，変更タスク群 $P\{Ta\}$ からは除外する．Ta_2 が Flight infoDs_1 で U，Booking infoDs_2 で C を行っていることが表 5.7 から読み取れる．これらから，拡張 CRUD 図（表 5.8）を作る．表 5.8 から，ルール 1 に適合する Ta_2/Ds_1U，Ta_2/Ds_2C を特定できる．

5.3.4 ルール 2：波及タスクの特定

ルール 2 にて波及タスク群 $S\{Ta\}$ を抽出するために，タスクの外部出力を図 5.2 の DFD から調べる．$P\{Ds\}$ に含まれる Ds_1 と Ds_2 とエッジを持ち，かつ So へ出力するタスク群が $S\{Ta\}$ 候補である．図 5.2 では，全てのタスクが Ds_1 および Ds_2 からのエッジを持つ．しかし，So への出力に着目すると，Ta_4

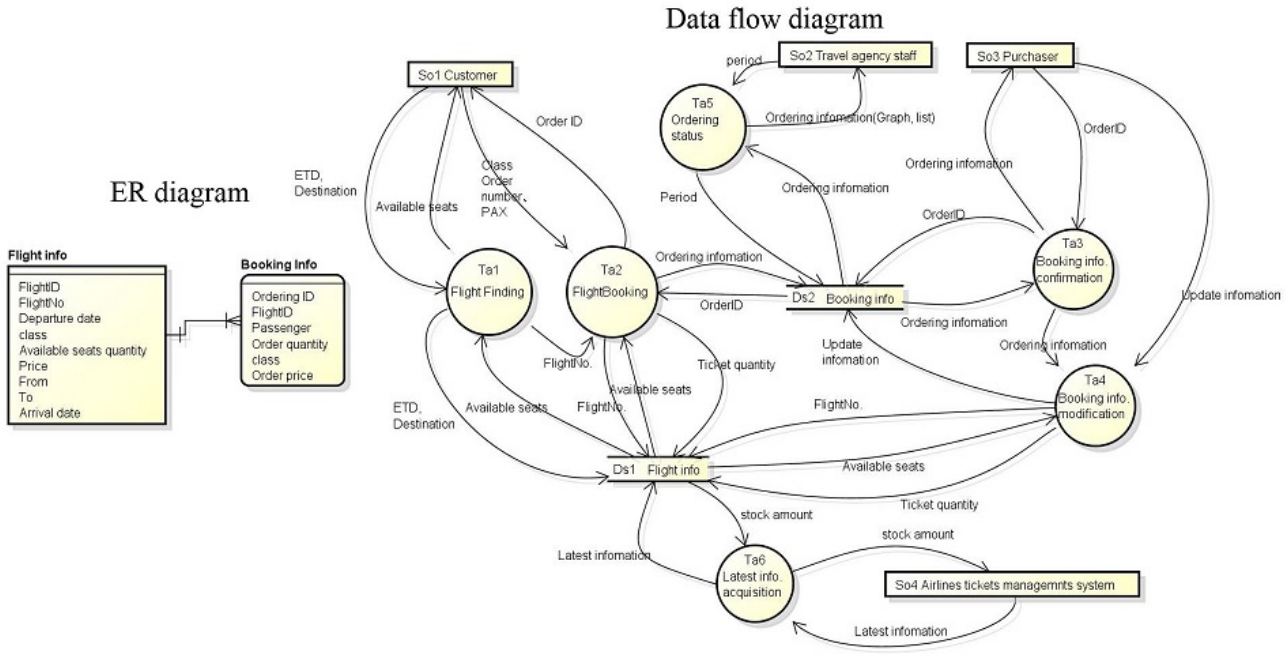


図 5.2: 新規フライト予約のデータ設計（一部分）

は該当するエッジがないため，$S\{Ta\}$ 候補には入らない．

$S\{Ta\}$ 候補のうち、表 5.3 の○がつく組合せに相当する Ta_i が，ルール 2 で特定したタスクとなる．本章の例の場合，$P\{Ta\}$ での操作は，C と U であるため，$S\{Ta\}$ 候補の中で C の操作をする Ta_i 以外は全てルール 2 で特定したタスクとなる．

これらに該当する Ta_i と Ds への CRUD 操作，そして So への Out を追記し，表 5.9 を完成させる．

5.3.5 ルール 3：手順 順序組合せテストケースの抽出

表 5.9 の拡張 CRUD 図から変更タスクと波及タスクの組合せを抽出する．抽出した変更タスクと波及タスクの組合せに対して，データストアに対する操作を明記したものは以下のとおりとなる．

表 5.6: フライト予約システムの機能セット一覧

テストアイテム	機能セット
フライト予約システム	メニュー
	ログイン
	新規フライト予約
	予約変更 & キャンセル
	予約一覧
	予約グラフ
	同期処理

- $Ta_2/Ds_1U \xrightarrow[Ds_1]{} Ta_1R$
- $Ta_2/Ds_1U \xrightarrow[Ds_1]{} Ta_2/Ds_1U$
- $Ta_2/Ds_1U \xrightarrow[Ds_1]{} Ta_6U$
- $Ta_2/Ds_2C \xrightarrow[Ds_2]{} Ta_3R$
- $Ta_2/Ds_2C \xrightarrow[Ds_2]{} Ta_5R$

これらの変更タスクと波及タスクの操作の順序組合せがテストケースとなる．抽出した順序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論理的テストケースとしてまとめる．表 5.10 に論理的テストケースとしてまとめた結果を示す.

5.3.6 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 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

表 5.7: フライト予約システムの CRUD 図

機能セット	タスク		エンティティ	
			Ds_1 Flight info.	Ds_2 Booking info.
新規フライト予約	Ta_1	フライト検索	R	
	Ta_2	フライト登録	RU	C
予約変更	Ta_3	予約情報確認		R
キャンセル	Ta_4	予約情報修正	RU	UD
予約リスト予約グラフ	Ta_5	注文状況		R
同期処理	Ta_6	最新情報取得	CU	

表 5.8: フライト予約システムの中間拡張 CRUD 図

タスク	データストア		源泉			
	Ds_1	Ds_2	So_1	So_2	So_3	So_4
Ta_1						
Ta_2	U	C	In			

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である図 5.3 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．図 5.3 は，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．それは，画面遷移のイベントでのガード条件に記載したデータが DFD のエッジに記載したデータ，ER 図のエンティティの属性と一致していることから確認できる．S1 網羅基準を適用した結果として，表 5.11 に 28 の状態遷移パスを示した。この表の提案手法（Proposal method）の列には，提案手法で抽出した順序組合せに該当する順序組合せを示した。

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，表 5.10 テストケース No.1，2，

表 5.9: フライト予約システムの拡張 CRUD 図

タスク	データストア		源泉			
	Ds_1	Ds_2	So_1	So_2	So_3	So_4
Ta_1	R		Out			
Ta_2	RU	C	$InOut$			
Ta_3		R			Out	
Ta_5		R		Out		
Ta_6	CU					Out

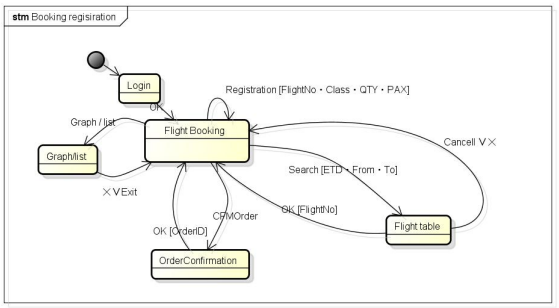


図 5.3: フライト予約システムの画面遷移図（一部分）

4，5 の 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，No.3 の $Ta_2/Ds_1U \underset{Ds_1}{\longrightarrow} Ta_6U$ である．このテストケースは，必要なテストケースと考えられる．適用評価にて利用したテストベースは，変更が入った機能セットに焦点を絞ったものである．この例では，新規フライト予約が該当する。そのため，図 5 では，新規フライト予約に隣接する画面遷移が，該当するテストベースとなっている．表 5.10 のテストケース No3 における波及タスクである Ta_6U は，表 5.7 から同期処理のタスクであることがわかるが，新規フライト予

表 5.10: 順序組合せテストによる論理的テストケース

新規フライト予約

No	論理的テストケース	順序組合せ
1	フライト予約後の空き情報問合せによる同一フライトの参照	$Ta_2/Ds_1U \xrightarrow[Ds_1]{} Ta_1R$
2	フライト予約後の再度同一フライトの予約	$Ta_2/Ds_1U \xrightarrow[Ds_1]{} Ta_2/Ds_1U$
3	フライト予約後の同期処理によって最新のチケット残数の計算	$Ta_2/Ds_1U \xrightarrow[Ds_1]{} Ta_6U$
4	既存注文開く画面での予約したフライトの参照	$Ta_2/Ds_2C \xrightarrow[Ds_2]{} Ta_3R$
5	注文件数グラフ・注文履歴の一覧への新規予約フライト予約の反映	$Ta_2/Ds_2C \xrightarrow[Ds_2]{} Ta_5R$

約とは別の機能セットに含まれるタスクであり，フライト予約画面と隣接する画面遷移図には現れない．そのため，S1 網羅基準では抽出することができない．このテストケースを抽出するためには機能セットのサイズを大きくする必要があり，その機能セットで状態遷移テストを適用するとテストケースの数はさらに爆発する．

5.4 変更波及解析への応用

5.5 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．

———

表 5.11: S1 paths in screen transition of flight booking registration

No	State	Event	State	Event	State	Proposal Method
1	Flight Booking	Registration [FlightNo・Class・QTY・PAX]	Flight Booking	Registration [FlightNo・Class・QTY・PAX]	Flight Booking	$Ta2/Ds1U \rightarrow Ta2/Ds1U$
2	Flight Booking	Search [ETD・From・To]	Flight table	Cansell ∨ ×	Flight Booking	
3	Flight Booking	Search [ETD・From・To]	Flight table	OK [FlightNo]	Flight Booking	
4	Flight Booking	CFMOrder	OrderConfirmation	OK [OrderID]	Flight Booking	
5	Flight Booking	Graph	Graph/list	× ∨ Exit	Flight Booking	
6	Flight Booking	Registration [FlightNo・Class・QTY・PAX]	Flight Booking	Search [ETD・From・To]	Flight table	$Ta2/Ds1U \rightarrow Ta1R$
7	Flight Booking	Registration [FlightNo・Class・QTY・PAX]	Flight Booking	CFMOrder	OrderConfirmation	$Ta2C \rightarrow Ta3R$
8	Flight Booking	Registration [FlightNo・Class・QTY・PAX]	Flight Booking	Graph	Graph/list	$Ta2C \rightarrow Ta5R$
9	Flight table	Cansell ∨ ×	Flight Booking	Registration [FlightNo・Class・QTY・PAX]	Flight Booking	
10	Flight table	OK [FlightNo]	Flight Booking	Registration [FlightNo・Class・QTY・PAX]	Flight Booking	
11	Flight table	Cansell ∨ ×	Flight Booking	Search [ETD・From・To]	Flight table	
12	Flight table	OK [FlightNo]	Flight Booking	Search [ETD・From・To]	Flight table	
13	Flight table	Cansell ∨ ×	Flight Booking	CFMOrder	OrderConfirmation	
14	Flight table	OK [FlightNo]	Flight Booking	CFMOrder	OrderConfirmation	
15	Flight table	Cansell ∨ ×	Flight Booking	Graph	Graph/list	
16	Flight table	OK [FlightNo]	Flight Booking	Graph	Graph/list	
17	OrderConfirmation	OK [OrderID]	Flight Booking	Registration [FlightNo・Class・QTY・PAX]	Flight Booking	
18	OrderConfirmation	OK [OrderID]	Flight Booking	Search [ETD・From・To]	Flight table	
19	OrderConfirmation	OK [OrderID]	Flight Booking	CFMOrder	OrderConfirmation	
20	OrderConfirmation	OK [OrderID]	Flight Booking	Graph	Graph/list	
21	Graph/list	× ∨ Exit	Flight Booking	Registration [FlightNo・Class・QTY・PAX]	Flight Booking	
22	Graph/list	× ∨ Exit	Flight Booking	Search [ETD・From・To]	Flight table	
23	Graph/list	× ∨ Exit	Flight Booking	CFMOrder	OrderConfirmation	
24	Graph/list	× ∨ Exit	Flight Booking	Graph	Graph/list	
25	Login	OK	Flight Booking	Registration [FlightNo・Class・QTY・PAX]	Flight Booking	
26	Login	OK	Flight Booking	Search [ETD・From・To]	Flight table	
27	Login	OK	Flight Booking	CFMOrder	OrderConfirmation	
28	Login	OK	Flight Booking	Graph	Graph/list	

第6章　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テスト条件の一貫性と特定する数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，テスト対象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テスト条件と実プロジェクトで作られるテストケースと比較して，不足しているテスト条件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．

第7章　転記前の部分

7.1　はじめに

ソフトウェアの一部に変更を加えた場合，その変更の波及を探る変更波及解析（Change Impact Analysis）は，実務上の大きな課題である．産業界において変更にかかる活動は，新規開発よりも大きな割合を占めている．ゼロから新規にソフトウェアを開発するケースは稀であり，何らかの流用を基に変更を加える開発が主流となっている．開発方法においてもアジャイルが主流となり，変更の積み重ねによって開発が行われている．

しかし，変更波及を合理的に制御する技術は，ソフトウェア工学とって未完成の分野である [20]．変更の背景は，時代と共に課題を難しくしている．ソフトウェアの多様化と複雑化，再利用範囲の増大などから変更波及の範囲が拡大し，かつ安易な変更による弊害など課題が山積している．これらの課題に対してソフトウェア工学は十分な解を提供できていない状況にある [21].

本論文では，状態遷移を持つソフトウェアにおいて，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ついて考える．課題の一つは，網羅基準である. データフローテストの全使用法（AU 法）[8] を基にして, 変更波及のテスト網羅基準を波及全使用法（Impact Data All Used : IDAU）として提案する．もう一つの課題は，IDAU 法を満たす具体的な変更波及のテストケースの抽出方法である. 変更波及のテストの設計手法として, 順序組合せテストを提案する．最後に，ここで提案する IDAU 法のコストの評価、すなわちテストケースの数を従来技法である状態遷移テストの S1 網羅基準と比較をして考察を行い, 提案する方法が合理的であることを示す.

7.2 変更波及とその解析

本節では，提案する網羅基準やテスト設計手法の前提となる，システムの構成と変更について定義し，変更波及テストの課題について詳細を示す．

7.2.1 変更と変更波及

AS に対して，何らかの変更を加える場合について考える．変更には，なんらかの意図があり，AS が持つ機能の変更であったり，不具合に対する変更や，性能や保守性の改善のためのリファクタリングであったりする．本論文では，変更の意図については取り扱わず，AS の構成要素（タスク，状態，保持データ）に対する具体的な変更について考える．ただし，テスト実行するためには，タスクを動かすことが必要となる．そのため，以降の議論はタスクに焦点を絞る．

ひとつの変更 Q について考える．変更 Q は，タスク群 Ta のあるタスク Ta_i に対して行われたとする．変更 Q は，コードの削除や追加を含み，その結果 Ta_i の版 R が $R+1$ に変更される．この変更の結果を Ta_i^R から Ta_i^{R+1} とする．

変更 Q の波及には，３つのケースが考えられる．

1. 変更波及が無い場合.（リファクタリングに相当）
2. 変更波及が他のタスクへ波及しない場合.
3. 変更波及が他のタスクへ波及する場合.

3. の変更波及は，タスク間の参照が図 7.1 に示すように状態と保持データに限られるならば，該当する状態や保持データの参照を介した範囲が限られると考えられる．本論文では，この考え方から波及を受けるタスクを特定し，その合理的なテスト設計について論じる．

変更波及，あるいはその解析（Change Impact Analysis）に関する研究は古くから行われている．プロダクトラインや UML 図面群を基に依存関係生成モデルを用いて波及解析を行う研究がある [22, 23, **?**]．タスク内のデータフローを基に変更波及を詳細に解析した研究がある [24]．データベースなど保有データを基に変更波及解析を行う研究も行われている [25, **?**]．一方，状態と状態遷移はマルコフ過程として実装されているので，過去の状態が未来の状態に影響し

ない．状態遷移に関する波及解析の研究は見当たらないのはそのためだと推測する．

7.2.2 状態と変更波及のテスト

変更波及は，保持データを介して波及タスクへ伝達する．状態や状態遷移自身は変更波及に関与しないが，変更波及のテストにおいて与えるデータの順序において状態を考慮する必要が生じる．

保持データの構成について考える．その要素を $Ds = \{Ds_1, Ds_2, \cdots, Ds_j, \cdots, Ds_d\}$ とし，変更波及を受ける保持データの要素を Ds_j とする．保持データに対する操作は，データのライフサイクルである「生成:C」「参照:R」「更新:U」「削除:D」を記した CRUD 図で定義する．Ds_j を介して変更波及が生ずるのは，変更タスク Ta_i において「生成:C」あるいは「更新:U」が行われ，他のタスクで「参照:R」が行われた場合に波及タスクとなる．

保持データのライフサイクル上の「生成:C」「参照:R」「更新:U」「削除:D」などの操作は，無条件に行われるのではなく，操作するタスクの制御フローに沿って行われる．制御フローは，２階層として捉えることができる．上位の制御フローはタスクの実行順序により決定される大きな制御の流れに相当する．個々のタスク内の制御フローが下位にあたる．個々のデータ参照実行文はその制御フロー上の条件文で実行が決定される．条件にはタスクへの入力，保持データ，状態が含まれる．

変更波及を確認するには，変更が波及した保持データを参照するデータフローに沿ってテストを設計することになる．このデータフローを決定するのは，関係

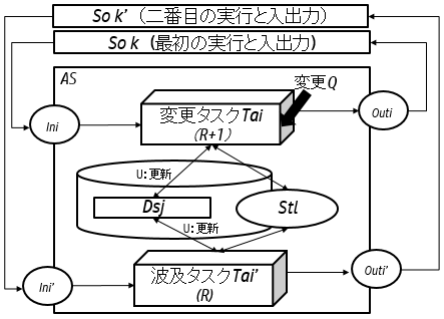


図 7.1: 変更タスクと変更波及

するタスクの実行順序とタスク内の制御フローであり，その制御フローを決定する際に状態が影響する．状態による制御が想定通りに行われない欠陥は，タスクの実行順序により決定される大きな制御の流れの判断のための状態の確認を，タスク実行のあるタイミングでのみ行っていることが原因となる．よって，実際のテスト実行においては状態を考慮する必要が生ずる．

7.2.3 変更波及テストの網羅基準

テストにおける網羅基準については，その強度を含め制御フローとデータフローの観点から研究が行われ体系が作られている [8]. 最も弱い網羅基準は制御フローのみに着目した実行文網羅，次が分岐網羅であり，最も強い網羅基準はデータフローを含めた全パス網羅（All Paths）である．全パステストは，すべての分岐の積であり現実的には実現不可能のため，全使用法（All Uses）が推奨されている [8].

変更波及をテストする場合，波及に関与するデータに着目し，そのデータフローテストを行う．一般的な全使用法は「データを定義したすべての場所から始まり，データを使用するすべての場所に至るまでのパスセグメントを最低限１つを含むテストケース」と定義されている [8]．この定義を変更タスクと波及タスクとの関係に置き換え「変更タスクにおいてデータの生成および更新があるデータを使用するすべてのタスク（波及タスク）を２つのタスクを実行するまでに経由するルートにかかわらず最低限１つ含むテストケース」とし，波及全使用法とする．

関連図書

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

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

[3] C Ebert and T Capers Jones. Embedded software: Facts, figures, and future. *IEEE Computer*, Vol. 42.4, pp. 42–52, 2009.

[4] 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.

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

[6] 大和田尚孝. システム統合の「正攻法」. 日経 BP 社, 2009.

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

[8] 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)).

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

[10] 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.

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

[12] 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.

[13] M Rasiel Ethan. The mckinsey way, 1999.

[14] Stan Kaplan Haimes, Yacov Y. and James H. Lambert. Risk filtering, ranking, and management framework using hierarchical holographic modeling. *Risk Analysis*, No. 22.2, pp. 383–397, 2002.

[15] James A Whittaker. *How to break software*. Addison-Wesley, 2003.

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

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

[18] 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.

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

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

[21] 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.

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

[23] 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.

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

[25] 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.