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

２０１８年　３月

湯本　剛

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

システム情報工学研究科

筑波大学

２０１８年　３月

概要

昨今のソフトウェア開発は，効率的に短納期で開発を行うことが求められている．新規にソフトウェアを開発するのではなく，既存のソフトウェアの改良や連携機能の追加で対応する保守開発が増えている．既存のソフトウェアに対する追加や改良が進むにつれ，ソフトウェア開発の複雑性と規模は急激に増加している．ソフトウェアテストを十分に行うために求められるテストケースの数は，これらに起因するソフトウェアの複雑性と規模の増大に伴い，増加の一途をたどっている．これらの状況から，昨今のソフトウェア開発では，多くのテストケースを多くの人員が短期間に作成して実行していかなければならない．これが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人員が，適切に対象を理解することで，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．

まず，テスト対象の仕様書を分析する際に，テスト実行時のデータの入出力のパターンで分析を進めることで網羅的にテストすべきことを特定する方法として，I/O テストデータパターンを提案する．このパターンは単一の処理をテスト実行する時に取りうるデータフローのパターンの全体像を定義したものである．現実に使われたモバイルアプリケーション開発プロジェクトのテストケースを入手し，そのテストケースと I/O テストデータパターンを使って特定したテスト条件の内容を比較し，このパターンを使うことで，テスト対象を理解する際に網羅的な確認が可能になることを検証する．

次に，統合テストにおける，複数の処理の実行順序を組み合わせた確認を行う際に，同一の保持データの操作を行う処理の組み合わせを抽出する順序組み合わせテストを提案する．予備実験にて利用したエンタープライズ開発の仕様書と設計書を入力情報として，この手法の適用評価を行う．また，既存の順序組み合わせを抽出する手法である状態遷移テストのの 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．

目 次

図 目 次

表 目 次

第1章　緒論

ソフトウェア開発工程の中で，品質を確保する主要な活動として，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となるソフトウェアの品質を可視化することができる．品質を可視化するためには，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．ソフトウェアテストを十分に行うために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大に伴い，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 倍に増大している [2]．組み込みソフトウェア開発のソフトウェア規模の増大は，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]．

ソフトウェアの規模の増大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]．昨今のソフトウェアの開発は，新規開発が減少傾向にあり，システム統合，派生開発 [5][6]，保守開発，コンポーネントベース開発 [7] といったすでに利用されているものに対して追加，改良をする開発が多くを占めてきている．日本における新規開発の比率がエンタープライズシステム開発で 51 パーセント [4]，組み込みソフトウェア開発で 5 パーセント [8] だという調査結果からも新規開発の減少傾向が読みとれる．ソフトウェア保守開発においては，開発対象のソフトウェアを分析し，理解するための工数が工数全体の 40 パーセントから 60 パーセント必要となる．しかし，現実的には，十分にその工数を確保できないことが課題だといわれている [9]．

また，この種類の開発は，テスト工数の比率が大きくなることが多い．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 人がテスト工程に投入されたという事例もある [10]．これらの調査結果から，開発全体に占めるソフトウェアテストの割合が多いほど開発コストに与える影響も大きくなるといえる．そのため，ソフトウェアテストを効率的に行うことが開発コストを左右すると考えられる．効率のよいテストとは，テスト対象の品質を可視化するために使うリソースが少ないことである．

ソフトウェアテスト工程全体の中の活動のうち，テスト実行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活動となる．特に，開発の要件が期待通りに実現しているかを確認するシステムテストのレベルにてソフトウェアをテストする局面では，開発工程で作られるソフトウェアだけでなく，既存のソフトウェアや実行環境など，実運用で必要となるものがすべて合流する．ここでのテスト実行は，クリティカルパス上にある活動となる．テスト実行がクリティカルパス上に滞在する期間をどの程度短くできるかが，開発コストだけでなく開発期間にも大きく影響を及ぼす．

テスト実行がクリティカルパス上に滞在する期間を短くする方法は，対象範囲を絞る [11]，欠陥の修正を効率化する [12] などがあり，多くの研究がある [13][14]．その 1 つとして，テスト実行の開始よりも早い段階でテストケースを抽出し，テストすべき内容の全体を示すことがあげられる．これにより，効率の良いテスト実行を計画できるためである．テストケースを開発する活動が遅延して，テスト実行の活動を逼迫しないようにするためには，複数の人員を投入し，計画した期間内でテストケースを作る必要がある．テストケースを開発する工数は，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 [15]．この調査結果は，テストケースの開発には多くの人員が必要となることを示している．昨今のソフトウェアの規模と複雑性の増大から，必要となるテストケース数はとても多くなり，数万から数十万となることも珍しくない．前述した大規模なシステム統合プロジェクトの事例では，統合テストとシステムテストのテストケース合計が 1,030,000 ケースであったと報告されている．

多数の人員が大量のテストケースを抽出する活動に必要とされているにもかかわらず，テストケースを開発するための明確に定義されたルールがないことが多い．そのために，投入された人員は個々の考え方に基づいてテストケースを開発することになる．この方法は，ソフトウェア保守開発の課題と同様に，それぞれのテスト対象を分析し，理解した内容に一貫性がなく，対象範囲の整理が不十分なまま大量のテストケースが作られていくことを意味している．ソフトウェア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．しかし，ソフトウェア規模の増大，また短納期のプレッシャーを受けながら，上記したように大量の人員で大量のテストケースを効率的に作らなければならないことは，以下の問題を引き起こす．

1. テストケースが重複する．同じテストを複数人が実行することになるため作業効率が低下する．
2. テストケースが欠落する．テスト実行に入ってからテストケース追加が必要になり作業効率が低下する．

これらの問題は，コスト増や納期遅延の原因となるだけでなく，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くなる問題となる [16] [17] [18].

また，テスト対象の規模が大きくなるほど，複数の機能を組み合わせたテストケースを実行する必要が出てくる．機能の組み合わせを単純なルールで網羅するようにテストケースを抽出すると，テストケース数が乗算で増えてしまうため，テストケースを増やさない工夫が必要となる．そのためには，対象を理解し，目的に沿ったテストだけを行うことが重要である [19]．投入された多くの人員がテストケースを増やさないように目的に沿ったテストケースだけを作るためには，目的に沿った網羅基準と適切な抽出方法が必要である．

本研究では，テストケースを開発する活動に携わる人員が，適切に対象を理解することで，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．テストケースを開発する対象として，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する．テストレベルの中でもシステムテストは，開発の規模に伴い規模が大きくなり，それに伴いテストケースの開発にも多くの人員が投入されることになるためである．そして，システムテストのレベルでのテスト対象への入出力データの順序情報を分析し，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，手法の適用評価を行う．

本論文は 6 章で構成される．2 章では，システムテストのレベルでの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，本研究のベースとなるテスト分析手法であるテストカテゴリベースドテストの概念，作業ステップ，そして適用時の効果を調査した実験結果を述べる．

3 章では，前章で述べた課題を更に分析するためにおこなった実験の結果を述べる．実験は 3 回行なった．3 回の実験から，テストケースの抽出結果には，ばらつきがあること，また，手法を取り入れることによって漏れていたテスト

ケースが抽出できるようになることを確認する．また，実験結果から読み取れる傾向を考察する．

4 章では，システムテストレベルにて，テストを実行する際の入出力データに着目する．テスト実行はデータの入出力を行うことであり，この全体像を I/O テストデータパターンとして定義し，このパターンを網羅することでテスト対象の分析を網羅的に行うことができる．この I/O テストデータパターンを適用した分析手法を提案する．I/O テストデータパターンの適用評価では，現実のプロジェクトで実際に使用されたテストケースと，提案する手法で作ったテストケースを比較し，現実のプロジェクトにてどのようなテストケースが不足するのかを考察する．

5 章では，入出力の実施順序から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として，テスト実行時のデータフローに着目する．複数の機能の統合を確認するためには，複数回のテストデータの入出力が必要となる，統合は多くの機能を組み合わせていくため，2 回以上のテストデータの入出力を組み合わせる必要も出てくる．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，単純に 2 回のテストデータ入出力の組み合わせ，それ以上の回数の入出力の組み合わせを洗い出してテスト実行順序を網羅しようとすると，実行順序の組み合わせ数を乗算で求めるようになるため，テストケース数が爆発する．そこで，機能間の統合を確認するためのテストケースを抽出するために，データフローに関する設計文書をベースに順序組み合わせを抽出する方法である順序組み合わせテストとその網羅基準である波及全使用法（Impact Data All Used : IDAU）を提案する．この適用評価では，順序組み合わせテストを使って実在の仕様書，設計書からテストケースが抽出できることの実証行う．また，順序組み合わせのテストケースを抽出する既存の手法である状態遷移テストを比較し，テストケース抽出内容を考察する．

最後の 6 章では，これらの研究をまとめて，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

本研究は，アプリケーションソフトウェアを開発する際に行うソフトウェアテストの中で，システムテストレベルでのブラックボックステストを研究対象にする．ブラックボックステストのテストケースを開発するプロセスの中では，テスト分析を対象にする．本章では，これら研究対象の範囲を明確にするために定義を説明し，そこで起きている課題を述べる．また，この研究のベースになるテスト分析手法である，テストカテゴリベースドテストについて説明する．

2.1 テストケースの開発方法とテストレベル

2.1.1 アプリケーションソフトウェアの構成

本研究では，図 2.1 に示す状態 St と保持データ Ds を持つアプリケーションソフトウェア AS に対するソフトウェアテストを研究対象にする．アプリケーションソフトウェア AS 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 In を出力 Out に変換する処理により実現されていると考えられる．この処理を本研究ではタスク Ta と呼ぶ [20]．タスク Ta 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 Ta のサイズは，テストレベルによって決まる．ユニットテストのレベルであれば関数となり，システムテストのレベルであれば，システムを利用するユーザが操作する機能となる．ソフトウェアの構成要素であるタスク Ta の出力 Out 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．たとえば，Web アプリケーションにて予約を行うタスク Ta について考えると，予約が可能か否かを示す状態 St と，予約オブジェクトの予約状況を示す保持データ Ds によって予約の成否が決まる．

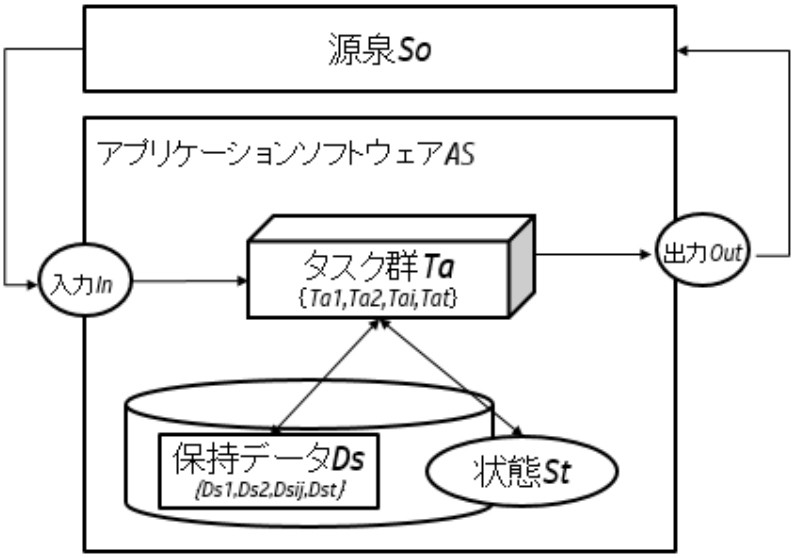


図 2.1: アプリケーションソフトウェアの構成

アプリケーションソフトウェア 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，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 ホワイトボックステストとブラックボックステスト

テストケースの種類は，ソフトウェアの物理的な構造をベースにテストケースを抽出するホワイトボックステストと，ソフトウェアの仕様をベースにテストケースを抽出するブラックボックステストに大別できる [21] .

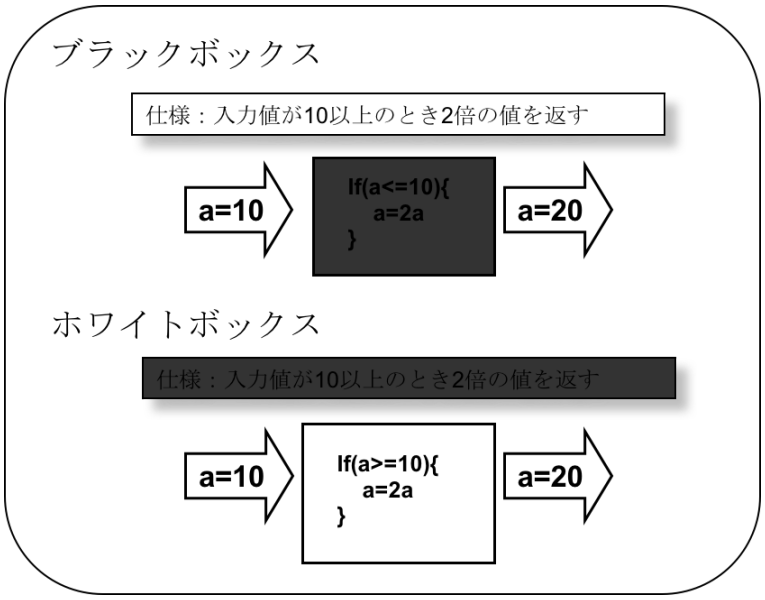


図 2.2: ホワイトボックステストとブラックボックステストの違い

ホワイトボックステストとブラックボックステストの違いを図 2.2 に示す．両者の違いはテストの網羅基準とテストケースの抽出方法の違いである．ホワイトボックステストは，テストケース抽出のベースがテスト対象となる AS の内部要素である Ta の構造になる．ユニットテストのレベルで例えると，Ta の内部構造となるプログラムのソースコードの行を網羅，分岐を網羅するように In を与えて Out を確認するといったように，網羅すべきアイテムを明確に選択してテストケースを開発する．網羅基準はテスト設計技法として提唱されている [22] [23] [24] [25] [26].

一方，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，Ta に対する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では，仕様に対する網羅基準が数多く提唱されている [27] [28] [29] [30] [31] [32].

しかし，ブラックボックステストは，テストケース抽出の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．これに起因する課題については，2．2 節にて述べる．

本研究では，テストケースの設計方法の種類は，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

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 [33]．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 [34]．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 [35].

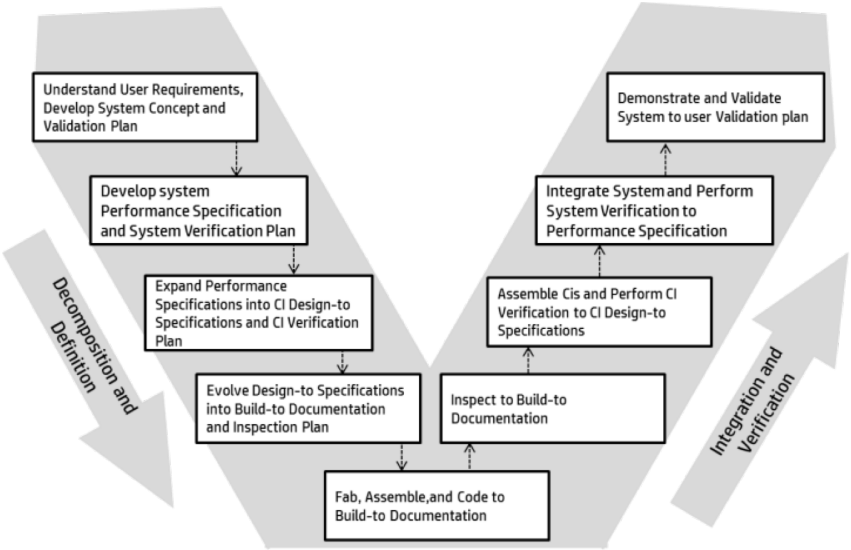


図 2.3: V モデル

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 [36]．本研

究は，複数のテストレベルの中で，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，つまりシステムテストのレベルで実行するブラックボックステストに焦点を当てている．システムテストのレベルは，開発した単体のソフトウェアがすべて統合されるため，規模と複雑性の増大による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

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．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

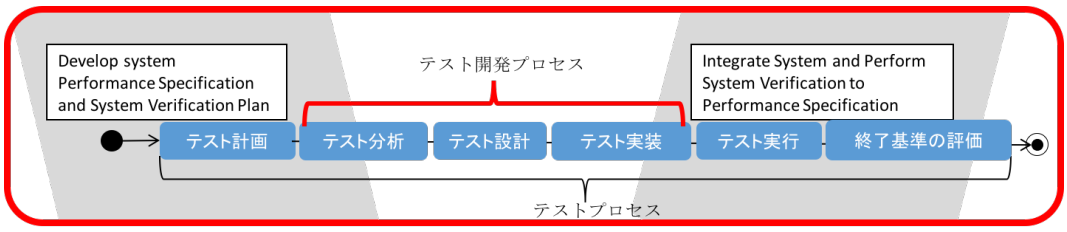


図 2.4: テスト開発プロセス

設計，テスト実装の 3 つのテストケースを抽出するための活動はテスト開発プロセスと呼ばれている [37]．

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，テストすべきアプリケーションソフトウェア AS 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ケース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 AS の仕様である [38]．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．各テストレベルにてテストケースを開発するベースとなる仕様をテストベースと呼ぶ [39]．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 での成果物に記述された仕様がテストベースとなる．

テスト分析では，テストベースに記述された仕様から，テストすべき AS の動作条件や振る舞いを特定する．仕様には，テストでの期待結果も記載されているので，一緒に特定する必要がある．更に，動作条件や振る舞いを実現するための事前条件や事前入力は，期待結果と照らしあわせて適切なものを仕様から取捨選択する．このようなテスト分析を行なった際のアウトプットは，テスト条件と呼ばれている [37]．つまり，テスト条件とは，図 2.5 のように仕様項目と該当する事前条件と事前入力のことを指している．

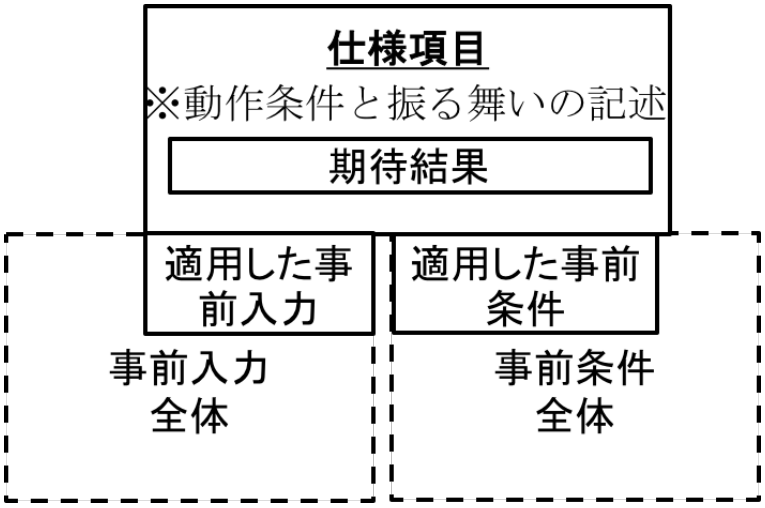


図 2.5: テスト条件の構成要素

これら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．

IEEE610 では，テストケースを，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出力を確認する，と定義している [40]．すなわち，機能テストとは，ブラックボックステストと同義である．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．おのおのは，表 2.1 に示すよう分類できる．

その上で，本研究では，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 [41]．テスト条件を網羅するテストケースを開発する際，テストアクションは Ta から Out を

表 2.1: テストの構成要素の再分類

	事前に設定 (パラメータ)	実行時点で設定 (アクション)
テスト入力	事前入力	イベント
実行条件	事前状態	操作

導く直接的な要因である．そのため，テストケースを実行する際の Out を導くテストアクションは 1 つに特定できる．しかし，テストパラメータは，1 つのテストアクションに対して多くのバリエーションを取り得る．バリエーションは，事前入力となる In や Ds だけでなく事前状態となる St も含まれる．

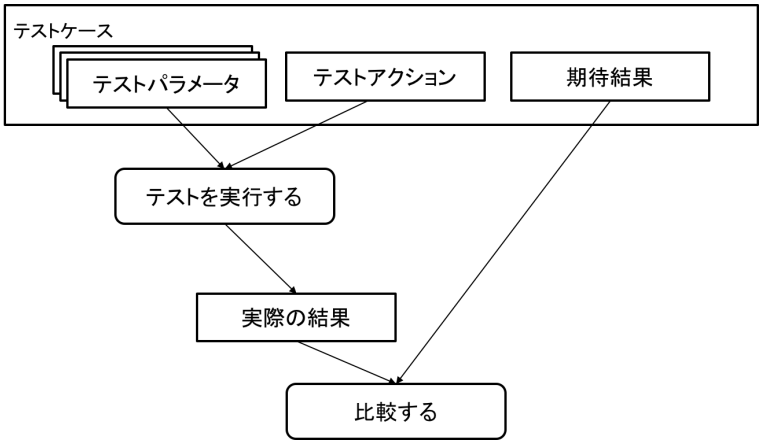


図 2.6: テストケースを実行するプロセス

図 2.4 のテスト実行においてテストケースを実行するプロセスを図示すると，図 2.6 のように表すことができる．テスト入力，実行条件となるテストパラメータ，テストアクションを入力としてテストを実行し，出力した実際の結果が期待結果と一致するかを比較する．期待と一致すればそのテストケースは成功であり，期待と一致しなければ，故障（Failure）として報告し，欠陥（Fault）が特定されれば，それを修正することになる．

2.2 テスト対象詳細化の問題

2.2.1 分類に対する一貫性の欠如

テスト分析の活動のアウトプット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 [37]．これらの側面について記述した成果物は，仕様である．ブラックボックステストのテスト分析では，テスト対象をテストケースが作れるサイズまで詳細化していく際に，これらの側面が記述してある仕様をベースにする．詳細化する際は，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違うものにならないよう分類の関係を定義して整理していく必要がある．しかし，実務において，テスト分析におけるテスト条件群の整理方法は，経験則や個人の考え方に基づいている．

実務の世界では，一般的に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あてはめるアプリケーションソフトウェアの側面の分類方法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，同じ項目が複数の階層に現れてしまったり，同じ意味の項目が別の名称で選択されるといった混乱を引き起こす．混乱が起きている例を表 2.2 に示す．

表 2.2: 一般的なテスト分析の詳細化の例

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	–	–	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

表 2.2 の例には，以下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，詳細化で現れる項目の内容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可能性が高くなると考えられる．このような課題については，Eldh が，指示内容理解（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 [42].

ISTQB では，テスト分析の活動を「…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」と説明している．しかし，この説明は，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だけであり，テストベースを分析していくための詳細化の方法を具体的に定義していない．テスト分析手法に関する研究にて，詳細化するモデルがいくつか提案されている [43] [44] [45] [46] [47]．しかし，複数の人数でテストケースを作る際に起きる課題については言及していない．テストケースを開発する人員の成熟度の向上は，重要な課題 [48][49][50] であり，手法が有用であるための要因となる．

テストケースの開発に関する先行研究は，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．[51] [52] [53] [54] テストパラメータを仕様書から自動摘出する研究 [55] [56] や，摘出したパラメータを使って仕様書とソースコードの比較をする研究 [57] [58] [59] [60] などが進んでいる．それらの研究では，テスト条件を特定するまでの詳細化はすでに行われた前提となっている．

2.3 機能間の統合に対するテストケース抽出の問題

2.3.1 既存の網羅基準によるテストケース数の増大

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 [61]. 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，複数機能を統合したもののテストでは，状態遷移に伴う時系列の組み合わ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．テストケース数の爆発への対処としては，回帰テストにおけるテストケースの優先順位づけに関する研究がある [62] [63]. しかし，これらは，何かしらの基準に対する網羅性を示すものではない．必要なテストケースの抽出方法とその網羅性に関する手法は，多くは機能や制御構造を基にした方法である．そのため，機能間の統合と状態遷移に伴う時系列の組み合わせには対応していない．

状態遷移間の組み合わせに対するテストケースを開発するための手法は，数多く提案されている [64] [65] [66] [67]．状態遷移の組み合わせの網羅基準としては，N スイッチカバレージがある．N スイッチカバレージでは，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したがって組み合わせテストケースを抽出する [68]．N=0 では遷移パスの組み合わ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，膨大なテスト工数が必要となる．

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．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 [69]．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，状態遷移の組み合わせに対して直

交表を応用し 2 因子間の組み合わせを中心に，一部 3 因子の組み合わせも抽出する研究がある [70][71]．この方法は，デシジョンテーブルを用いて機械的に組み合わせを抽出でき，2 因子間の組み合わせ即ち 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．

現実的な方法としては，設計で用いられる UML のシーケンス図を基にテストケースを抽出する方法が知られている [72]．この方法によるテストケース数はシーケンス図で定義されたシーケンスで決まる．シーケンス図が状態遷移の S1 網羅基準を満たすか否かは，シーケンス図が表すテスト対象のサブセットの範囲による．多くの場合，設計者が意図したシーケンスは，起こり得る状態遷移の組み合わせの一部しか表してないため，漏れが生じる課題がある．

2.4　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 [73] というテスト分析手法を利用した予備実験を行い，テスト分析の課題の調査，およびテスト分析の知識を与えることによるテストケースを網羅的に抽出できるスキル向上傾向の調査を行なった．この結果は 3 章に記載をする．この分析手法を使って調査を行う理由は，以下のとおりである．

1. 前述したテスト分析の問題のうち，分類に対する一貫性の欠如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．
2. 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．
3. 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利用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，テスト対象のサブセットに属するタスクの仕様項目を特定していく方法を提示する．また，テストケースの構造をベースにテスト条件を分解することで，テスト条件という用語の持つ曖昧さを排除する．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故障の知識を使って定義した分類を構造に追加している．

2.4.1　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をテストケースの構成要素で整理すると，図 2.7 に示した構造で整理できる．

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

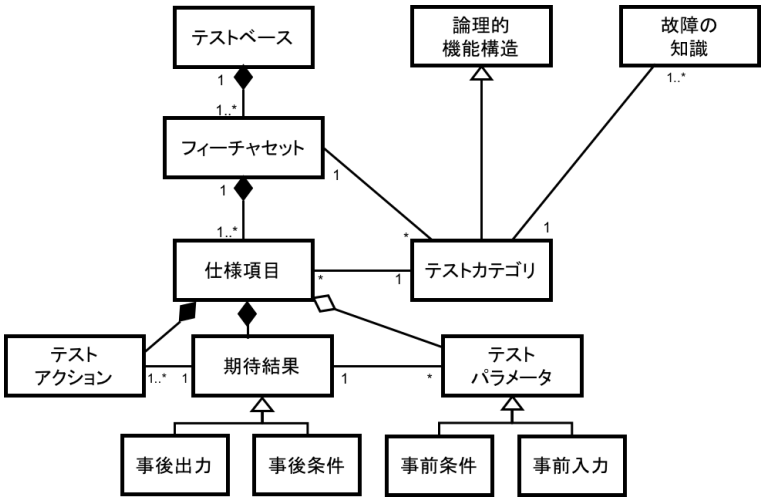


図 2.7: テストケースの構成要素で整理したテスト条件の構造

いる．フィーチャとは，利用者が観察可能なソフトウェアシステムの論理的なサブセットであり，利用者とテスト対象のインターフェースとなる [74]．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．テスト対象となるフィーチャは ISO/IEC/IEEE29119 の定義に従い，フィーチャセットと呼ぶ [75].

仕様項目とは，フィーチャセット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，フィーチャセットの振る舞いの 1 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特定していく．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 1 つで，事前入力と事前条件を汎化したものである．期待結果は，出力と事後条件を汎化したものである．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．

2.4.2 論理的機能構造

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 2.8 のような構造を持つ [76]」と主張している．

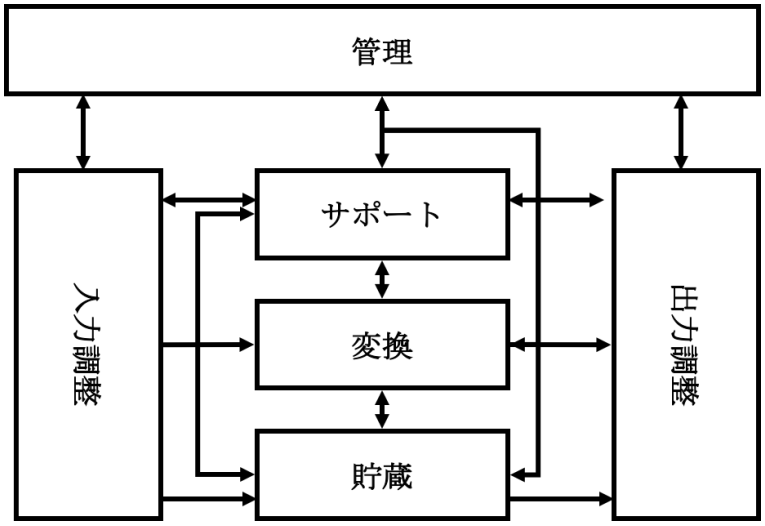


図 2.8: 人工システムの論理構造

入力調整 入力される資源は，直接変換機能に送り込まれることなく，変換しやすいように入力機能によって整えられたのちに送り込まれる．

出力調整 不十分であったり不必要なものを含む変換されて出てくる資源をシステムから出力する前に，有用資源に調整したり不必要なものを始末する．

変換 システムの中核となる基本機能．システムが目的を達成するのに直接関わり，システムを特徴付ける．

貯蔵 入力資源を変換機能に安定的に供給したり，出力資源を外部の要求に合わせて送り出すために，さらには他の基本機能がスムーズに働くためにシステム内に資源を蓄える．

サポート 変換をはじめとする他の機能が円滑に働くために，それぞれの機能を裏から支える．

管理 変換装置全体が様々な制約の中で秩序関係を維持しながら目的を達成できるようにする．

テストカテゴリベースドテストは，同様のコンセプトを利用している．つまり，テスト対象となるフィーチャセットは同様の論理構造を持つ人工システムだと捉える．フィーチャを MECE(互いに相容れなくて完全に徹底的)[77] な方法でテストをするために，この論理構造を利用する．テストカテゴリベースドテストでは，人工システムが持つ論理構造を基にして，図 2.9 のように定義し，論理的機能構造と命名する．

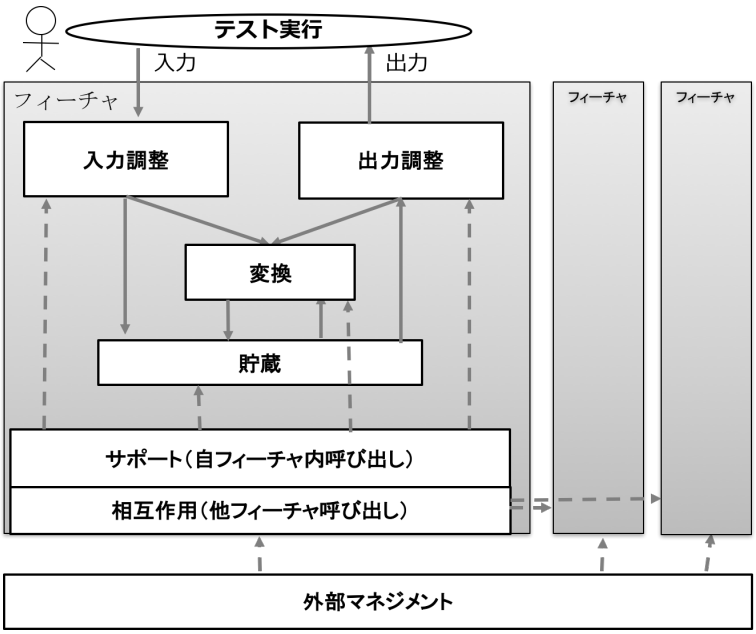


図 2.9: 論理的機能構造

図 2.9 に示す各要素は，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．論理的機能構造では，前述した人工システムの論理構造でサポートと呼んでいる要素をサポートと相互作用の二つに分けている．サポートは，割り込みや同時処理のロック機構など，変換や貯蔵などフィーチャの内部に属するタスクが正しく処理を行うためのサポートタスクのテストのための分類である．通常，テスト対象のフィーチャに対する 1 回のテストアクションで確認できる．相互作用は，変換や貯蔵などの内部に属するタスクでの処理の別タスクへの副作用の確認であり，通常，2 回以上のテストアクションを必要とするものである．管理は，外部マネジメントと呼び，フィーチャセットの外側で，システム全体の秩序を維持するタスクを分類する．そのため，ブラックボックステストでは使わない．

飛行機のフライトを予約するシステムのシステムテストにて,「新規フライト予約」をフィーチャセットとした時のテスト条件を特定する例で考えてみる．新規フライト予約では，まず予約したいフライトが何であるかをシステムに入力するが，過去の日付など購入できない日付や，運行していない行き先など，不正な入力情報をあらかじめチェックするタスクがあり，テストが必要となる．これは入力調整へ分類する．入力情報から，予約可能なフライトの購入金額を計算するタスクのテストは，変換に分類する．そして，予約が成立するとその情報はシステムに登録される．登録を行うタスクのテストは貯蔵に分類する．このシステムが状態を持っていて，状態によって予約ができないといった制御があれば，そのタスクに対するテストが必要になる．これはサポートに分類する．また，予約が成立したことによる副作用を別のタスクに対するテストアクションで確認するテストは，相互作用に分類する．外部へ結果を伝える際にフォーマットの変換を行うといったタスクがある．このタスクのテストは出力調整へ分類する．このようなテストを実行するためには，タスクに関する振る舞いや条件が記述がされている仕様項目を特定する必要がある．この特定した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，テストベース中の該当する単一の節以外に記載される．

2.4.3 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人員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，論理的機能構造の要素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論理的機能構造の各要素を具体的に表現したものをテストカテゴリと呼ぶ．テストカテゴリの命名には，テスト対象の知識が必要である．そして，テストカテゴ

リは，テストケースの抽出に携わる各人員がテストカテゴリの意味を同じように理解することが必要であり，そのための合意形成を行う．テスト対象を表す命名で合意したテストカテゴリは，テスト条件を特定するための有用なガイドとなる．

表 2.3: テストカテゴリ一覧の例

論理的機能構造	テストカテゴリ	意味づけ（想定する故障）
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

テストカテゴリは，表 2.3 にて示したテストカテゴリ一覧にまとめる．そして各テストカテゴリに分類したテストにて検出したいと考えている故障を列挙する．これら故障に対して，テストケースを開発する人員の間で例を挙げてディスカッションし，その結果を表 2.3 で示したテストカテゴリ一覧の故障の欄に反映する．ディスカッションにより，テスト開発プロセス活動にかかわる人員は，テストカテゴリの意味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テストカテゴリに対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．

2.4.4　実施手順

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 2.10 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．

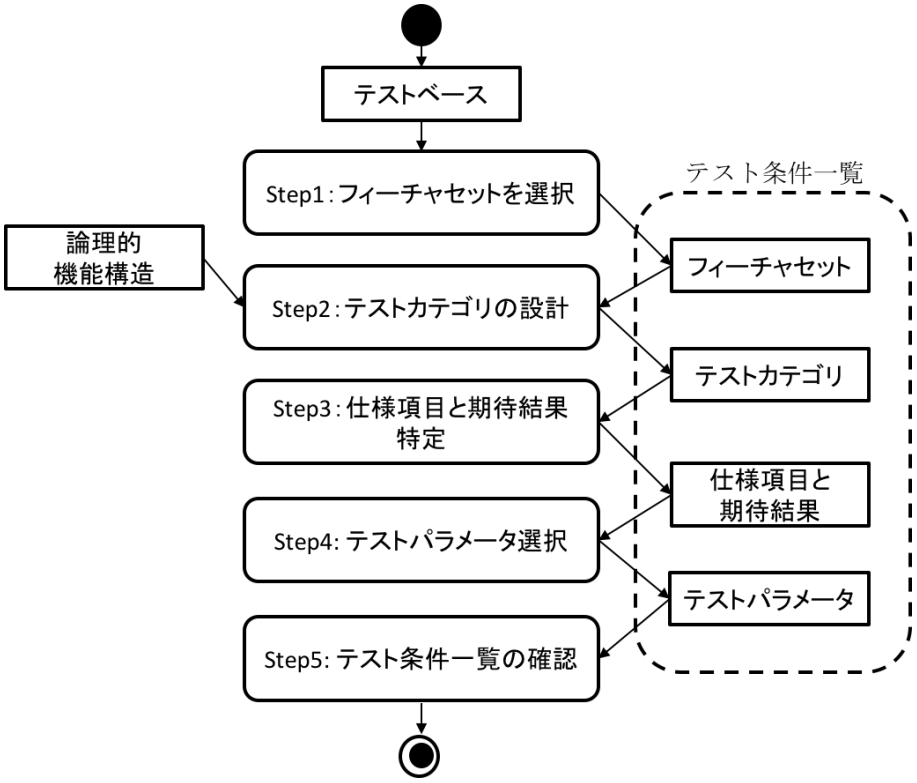


図 2.10: テスト分析の実行ステップ

以降に各作業ステップについて説明をする．

Step1 フィーチャセットを選択

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セットを特定する．フライト予約をするアプリケーションソフトウェアで例えた場合，新規フライト予約（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）がフィーチャセットとなる．

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 [78]．HHM 法でいうところのメイントピックにフィーチャセットを置く．メイントピックを構成するサブトピックとして論理的機能構造毎にフィーチャセットの動作条件や振る舞いを列挙する. 列挙する際は，どのような故障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．選択したフィーチャセット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（どのような故障が起きるか）を共有することで，解釈のぶれを防ぐ.

Step3 テストカテゴリを使い仕様項目と期待結果を特定，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．

特定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特定した仕様項目と期待結果にそってテストベースを分析して選択する．

表 2.4: テスト条件一覧の例

フィーチャセット	テストカテゴリ	仕様項目	期待結果	テストパラメータ
TFa	TCa	$SIa1$	$ERa1$	$TP1$,$TP2$
		$SIa2$	$ERa2-1$	$TP1$,$TP3$,$TP4$
			$ERa2-2$	$TP1$,$TP4$
	TCb	$SIb1$	$ERb1$	$TP5$,$TP6$,$TP7$
	TCc	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 2.4 に示すテスト条件一覧にまとめる．

テスト条件一覧の最初の列には，フィーチャセット TF を列挙する．表 2.4 の例では，TF の 1 つである TFa を列挙している．隣には，テストカテゴリのセット

$TC = \{TCa, TCb, TCc, \cdots\}$ を列挙する．そして，TC に対応する仕様項目 SI と期待結果 ER 列挙する．表 2.4 の例だと，TCa に対応して $SIa1$,$SIa2$ を列挙している．仕様項目によっては期待結果が複数になることがある．そのため，$SIa1$ に対しては $ERa1-1$ のみを列挙しているが，$SIa2$ には，$ERa2-1$,$ERa2-2$ を列挙している．TFa 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る．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ため，N/A と記載する．

テストパラメータには，各仕様項目と期待結果のセットから見て適切なテストパラメータの組み合わせを選択する．$SIa1$，$ERa1-1$ に対しては $TP1$,$TP2$ が選ばれた組み合わせとなる．テストパラメータは，テスト分析の後の活動となるテスト設計にて同値分割を行い適切な同値クラスにする．複数の同値分割の組み合わせは，ディシジョンテーブル技法やペアワイズテスト技法といったテスト設計技法使って適切な組み合わせを設計する．

テストケースの開発に従事する多くの人員が上記のステップに従うと，同じルールにしたがって分析を行うことができる．結果として，特定したテスト条件の一覧は，包括的で，重複が含まれない．このようなテスト条件一覧を作成することは，高いテストカバレッジを確かにし，高品質な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，この手順にしたがってテスト分析を行うことには，以下の 3 つの効果がある．

1. この手順は，図 2.7 で示した「テストケースの構成要素で整理したテスト条件の構造」をベースにしている．この手順を通して，テストベースを仕様項目，期待結果，テストパラメータをそれぞれ順番に特定，選択していく．テストケースの開発に携わる人員は，テスト分析の活動を通じて同じ順番でテスト条件を特定し同じカテゴリに分類できるため，全員の成果物をまとめて確認する際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，テストケースの開発に携わる人員は仕様項目の特定と選択を同じ認識を持って行うことができる．人員間での情報共有も容易になる．

3. 本手法は体系化，標準化して進めていくことが容易になるため，組織に関わるテストケースの開発に携わる人員がテスト分析を繰り返すことができるようになる．

2.4.5　テスト条件特定結果の比較

テストケースの開発に関するワークショップを開催し，同一組織内にてソフトウェアテストに従事している人員を 2 グループに分けて，テストカテゴリベースドテストの説明で記したテスト分析の作業「Step3：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．

1 つのグループは，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，もう 1 つのグループは，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た．題材として音楽再生機器を選定した．出席者はすべて類似の機器のシステムテストに関わった経験があり，製品知識はある．フィーチャセットは，音楽再生機器のボリュームコントロールである．

2 つの演習結果と演習の模範解答を比較し，解答例と同じだけの仕様項目を特定できれば網羅的に分析ができているとした．そして，テストカテゴリベースドテストを適用した場合とそうでない場合の違いを分析した．グループの回答を比較データに使った理由は，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．

表 2.5: 仕様項目の選択割合の比較

	入力調整	変換		サポート		出力調整		貯蔵		相互作用	合計	比率
解答例	0	2	1	0	2	1	0	1	0	2	9	100%
テストカテゴリ未適用	0	2	0	0	0	1	0	1	0	1	5	56%
テストカテゴリ適用	0	2	0	0	2	1	0	1	0	1	7	78%

演習結果にて，仕様項目の選択数を比較すると表 2.5 のようになった．1 番上の行は解答例であり，講師が予め準備したものである．2 番目と 3 番目の行はワークショップの中で各グループが演習中に作成したものである．比率は，解答例を母数にした場合の合計数の割合である．テストカテゴリを利用したグループは講師と同じテストカテゴリを利用し，テストカテゴリを利用していないグループは，演習後にこちらでテストカテゴリにマッピングして比較可能にした．両方のグループとも解答例と同じ数の仕様項目の特定はできなかった．グルー

プ間の比較をした場合，テストカテゴリありのグループのほうが仕様項目の特定数が2つ多く，より高い結果となった．

また，各仕様項目として列挙した期待結果とテストパラメータ数の集計は表2.6のような結果となった．

表 2.6: 期待結果とテストパラメータ数の選択結果

テストカテゴリ	仕様項目		期待結果数		パラメータ数	
	適用なし	適用あり	適用なし	適用あり	適用なし	適用あり
入力 A			N/A	N/A	N/A	N/A
変換 A	○	○	1	1	5(3)	N/A
	○	○	N/A	1	1	N/A
変換 B			N/A	N/A	N/A	N/A
サポート A			N/A	N/A	N/A	N/A
サポート B		○	N/A	1	N/A	2
		○	N/A	1	N/A	2
出力 A	○	○	1	1	N/A	2
出力 B			N/A	N/A	N/A	N/A
貯蔵 A	○	○	1	1	3(1)	2
貯蔵 B			N/A	N/A	N/A	N/A
相互作用 A	○	○	N/A	1	1	1
			N/A	N/A	N/A	N/A
合計	5	7	3	7	10(4)	9(0)

テストカテゴリを利用しなかったグループは，期待結果が明記されていなく，テストパラメータをあらわすテスト条件のみ記載している仕様項目がテストカテゴリを利用したグループと比較して4つ多かった．列挙されていたパラメータ数は，テストカテゴリを利用しないグループのほうが11多く，4倍以上であった．また，テストパラメータの内容を被験者に確認して，同じ仕様項目のパラメータになるものが別の仕様項目として記述されていたものを講師が集計時に分類し直した．パラメータ数の欄に（）にて記した．テストカテゴリを利用し

ないグループは，パラメータ数の欄に（）で記した数が 4 つあった．これらはテスト設計時に重複したテストケースを設計してしまう可能性がある．

2.5 まとめ

本章では，研究の対象となるアプリケーションソフトウェア，テストケースの種類，テストレベル，テストプロセスを明記し，ソフトウェアテストの中で，システムテストレベルでのブラックボックステストを研究の対象にすること，ブラックボックステストのテストケースを開発する活動の中では，テスト分析を対象にすることを述べた．そして，そこで起きている問題として，テスト対象を詳細化するときに分類に対する一貫性が欠如していることを述べた．また，機能間の統合に対する問題として，既存の網羅基準を適用するとテストケース数が膨大になることを述べた．

これらの問題に対して，テストカテゴリベースドテストというテスト分析手法をベースに研究をすすめるため，前提知識としてこの手法の概要と，既出の実験結果を説明した．

第3章　テスト分析結果のばらつき傾向とその影響

本章では，前章で述べた課題を更に分析するためにおこなった予備実験の結果を述べる．実験では，被験者に対して，テストベースを与えてテスト分析を実施してもらい，その後，テスト分析手法であるテストカテゴリベースドテストの知識を与えた上で再度テスト分析をしてもらう．テストカテゴリベースドテストの知識を与える前と後の結果から，ルールに関する知識を与えていない状態でのテスト分析結果にはばらつきがあること，また，2 章で説明をしたテストカテゴリベースドテストによるルールを与えることによる変化を調査する．

3.1 予備実験の目的

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，手法を適用したグループが，適用していないグループよりも抜け漏れが少ない結果となった．しかし，実験データは 1 組のみであり，傾向を結論づけるには不十分である．テストカテゴリベースドテストを適用する前のテスト分析での結果のばらつきの傾向，及びばらつきの傾向と分析手法を適用後の結果との相関をより多くのデータで調べることを目的にした予備実験を行った．実験はワークショップを通じてグループ単位で 2 回，個人単位で 1 回行なった．

グループ単位 1　製造メーカーの同一製品開発チームのメンバーを 4 〜5 人にグルーピングして実施（6 サンプル収集）

グループ単位 2　オープンな研修にて，別々の企業の参加者を 4 〜5 人にグルーピングして実施（2 サンプル収集）

個人単位 1　テスト技術者コミュニティ主催のワークショップを通じて実施（57 サンプルを収集）

3.2 グループ単位の実験

3.2.1 実験の概要

グループ単位の実験は2回行った [79]．実験は，両方とも4時間のワークショップを通じて実施した．ワークショップの進め方を，図 3.1 に示す．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，参加者に各自の考えに基づき，2.4.5 節の実験と同様に「テスト分析の作業 Step3：テストカテゴリを使った仕様項目と期待結果の選択」を実施してもらう．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解答としてまとめる．

最初の分析結果のまとめが終わり，全出席者が各グループの分析結果を理解した後，テストカテゴリベースドテストと実施手順を説明して，手法の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解答をまとめる．

解答例と同じ数の仕様項目を特定できれば，網羅的にテスト分析ができているとする．そのうえで，同一グループに対するテストカテゴリベースドテストの知識を与える前と，知識を与えた後のグループ解答を実験のデータとして利用した．

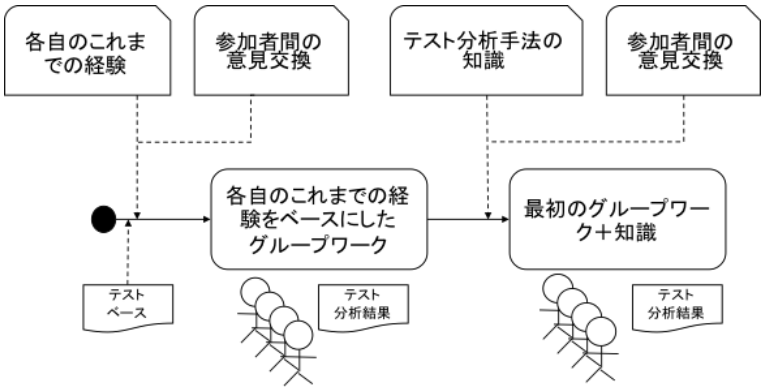


図 3.1: 演習の前提条件の変化

3.2.2　実験の題材

実験の題材は 2 種類用意した．[グループ単位 1] では，組込みソフトウェア開発の演習題材として音楽再生機器を使った．フィーチャセットはボリュームコントロールである．2.4.5 節で行なった実験とは別の製品の仕様書を題材にしている．[グループ単位 2] では，エンタープライズシステム開発の演習題材として，フライト予約 Web システムを利用した．フィーチャセットは新規フライト予約である．テストベースとなる仕様書として，両方とも 12 枚のパワーポイントのスライドを用意した．講師側が用意した解答例である，テスト分析で特定すべきテスト条件の一覧を表 3.1 と表 3.2 に示す．

3.2.3　テストカテゴリベースドテスト適用前のテスト分析の結果

テストカテゴリベースドテストの知識を与えていないため，ルールがない状態で行われたテスト分析結果のばらつきについて，実験による典型的な 4 パターンのテスト分析結果を図 3.2a から図 3.2d 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．

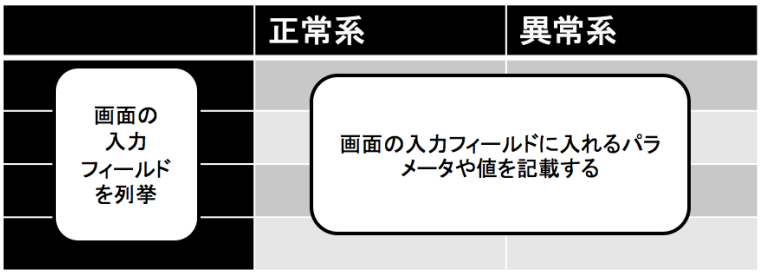
この結果は複数の個人のテスト分析が，それぞれ複数の結果に到達することを示している．複数の結果とは，テスト分析を通して特定したテスト条件にばらつきがあることを意味している．テスト分析が多くの人たちで行われるときには，テスト条件の重複，もしくは完全に抜け落ちるといった可能性が考えられる．

表 3.1: 音楽再生機器の講師解答例

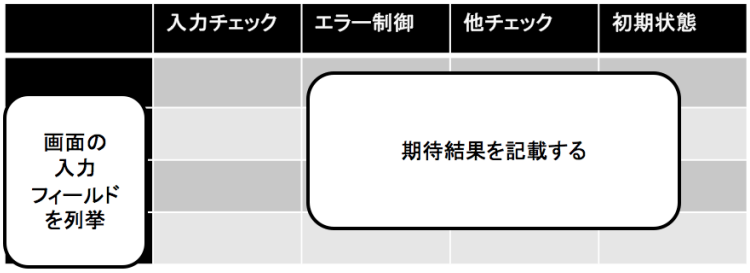
論理的機能構造	テストカテゴリ	仕様項目	期待結果／確認内容
入力調整	対向機入力	–	–
	ボタン	・キューイング	・早押ししたとき無視すること
出力調整	音声出力	・ボリューム上限下限を知らせるビープ音	・ビープ音が押下に対して 1 回なること
	LED 点灯	–	–
変換	音量	・音量調節	・ボタン押下でボリュームが上下すること
	対向機操作	–	–
貯蔵	設定保存	・デフォルト値 ・音量調節の保存	・ボリュームを変更させてリセットするとデフォルト値に戻ることで確認 ・PowerON 後前回 Off した際の結果と同じ音量ができることを確認する
サポート	ヘッドセット状態遷移	・状態により音量調節アクションを受け付ける／受け付けない	・再生中，通話中以外音量調節ボタンを無視する
相互作用	対向機への反映	・対向機のボリューム値への影響	・ヘッドセットのボリュームを変更したあと，対向機のボリュームが変更していないことで確認
	設定情報の共有	・複数の設定情報の設定影響 ・他の処理で音量設定の削除	・通話と再生で交互に音量を調節した際に互いに影響を受けないこと ・初期化，リセットで初期値に戻ることを確認

表 3.2: フライト予約システムの講師解答例

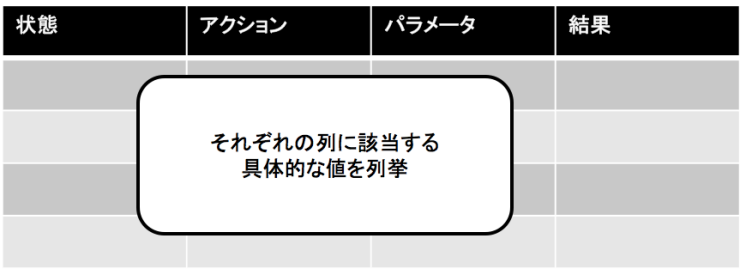
論理的機能構造	テストカテゴリ	仕様項目	期待結果／確認内容
入力調整	画面入力	・画面表示内容 ・年，月，日の入力範囲チェック ・入力桁数チェック ・型チェック ・うるう年判定，末日判定 ・出発地と到着地の組み合わせ	・画面仕様書に沿ったレイアウトであること ・入力フィールド／ボタンの制御が仕様どおりであること ・Min 日以降 Max 日が入力できること ・フライト予約可能日以前の入力はクリアされること ・桁数/文字数チェックをすること ・型（文字，数値）のチェックをすること ・うるう年，末日判定のチェックをすること ・同じ出発地と到着地を選択できないこと
	ボタン操作	・画面遷移	・画面仕様書に沿った画面の遷移ができること
出力調整	結果表示	・プログレスバー/登録完了メッセージ ・注文不成立メッセージ ・金額計算表示 ・注文番号 ・フライト情報	・登録中と登録後表示が正しいこと ・メッセージが出ること ・フィールドに収まること ・同上 ・同上
	帳票出力	–	–
変換	計算	・合計額の計算 ・登録時の採番 ・チケット在庫の計算	・クラス単価×人数となること ・直前の No ＋ 1 となること ・在庫数から購入数をマイナスし 0 以下にできない
貯蔵	検索	・フライト検索結果	・検索条件 (and) で該当するフライトが検索されること
	登録，更新，削除	・フライトの登録	・フライトが登録が出来ること
サポート	エラー処理	・登録時にチケット在庫なし ・注文挿入中の強制終了	・エラーになること ・ロールバックすること
相互作用	反映	・「既存注文開く」への反映 ・「注文件数グラフ」への反映 ・「注文履歴」への反映	・検索条件 (and) で検索されること ・注文件数がプラスされること ・すべての状況が書き込まれること



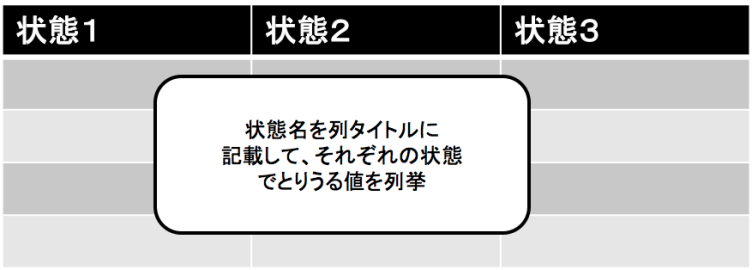
(a) テスト分析の事例：CS1



(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 3.2: ルールがない状態でのテスト分析の事例

3.2.4 実験結果の評価

1 回目，2 回目の実験では，グループでの解答を実験結果として利用した．1 回目の実験は，組み込み開発を行なっている組織内での 6 グループの実験結果を収集し，2 回目の実験結果ではオープンセミナーの参加者をグルーピングした 2 つの実験結果を収集したので，合計で 8 つの実験結果を収集した．そして，テストカテゴリベースドテストの知識をを与える前の演習結果と，知識を与えた後の演習結果とを比較した．テストカテゴリベースドテストの知識を与える前にテスト分析をした結果は，図 3.2 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をしている．

表 3.3: 1 回目の実験のグループ 2（TM2）での比較結果

テストカテゴリ	期待結果数		パラメータ数	
	適用なし	適用あり	適用なし	適用あり
入力	0	0	0	0
変換	1	1	3	3
出力	1	1	2	2
貯蔵 A	0	1	0	1
貯蔵 B	0	1	0	3
サポート	0	0	6	0
相互作用	0	1	0	2
	0	0	0	0
	0	0	0	0
合計	2	5	11	11

表 3.3 は最初の演習結果のうちの 1 つグループの結果を比較した表である．知識を与える前と後では，特定した期待結果の数は 3 つ増えている．テストパラメータを特定した数は，合計としては変わらないが，サポートにかかるテストパラメータに関してだけ，知識を与える前よりも減っている．このテスト条件は，状態により処理を受け付ける場合と受け付けなくなる場合があることを確認するテスト条件である．知識を与える前は，状態を 6 つ列挙していたが，期

待結果の記載がなかった．知識を与えた後は，列挙していた状態に関する記述がなくなっていた．グループの人員に確認したところ，期待結果と紐つけようとしたが，期待結果が不明であるために記載するのをあきらめたことが理由であった．

表 3.4: 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.

8 つの実験結果を比較するために，表 3.4 に示した評価レベルを設定した．評価レベルは 4 段階であり，特定したテスト条件が増加し，期待結果に達している A^+ が最も高い．最も低いレベルは B で，特定したテスト条件が増加せず，かつ期待結果よりも少ない場合である．

予備実験にて収集した 8 つのグループの評価結果を表 3.5 に示す．TM1 から TM6 までが最初の実験の結果である．*AS* は音楽再生機器であり，フィーチャセットはボリュームコントロールである．この実験結果では，演習の解答例と同じだけのテスト条件を特定できたグループは 0 であった．ただし，6 グループ中 5 グループにおいて，列挙したテスト条件の数が増えている．論理的機能構造の要素ごとの比較では，相互作用にて 5 グループの列挙数が増加している．出力調整と貯蔵では，列挙数が増加したグループ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，効果があったかどうかはこの結果からは判断できない．ただし，サポートは全グループにてテスト条件の増加がないため，効果は出ていないと考えられる．

表 3.5: グループごとの実験の評価結果

論理的機能構造	グループ							
	TM1	TM2	TM3	TM4	TM5	TM6	TM7	TM8
変換	B	A	B	B	B	B	A	A
入力	B	B	B	B	B	B	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

TM7 と TM8 は 2 回目の実験である．AS はフライト予約システムである．新規フライト予約がフィーチャセットである．この実験でも知識を与えた後の演習結果にて解答例と同じ数のテスト条件を特定したグループはいなかった．ただし，2 グループすべてにおい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いる．論理的機能構造の要素ごとの比較では，変換と出力調整と貯蔵にて 2 グループともにテスト条件の列挙数が増えた．それ以外の要素では 1 グループのみが列挙数が増えた結果となっている．

全体的に定量的な向上が見られたが，特定のグループの著しい成長，もしくは論理的機能構造の要素毎に特徴的な傾向は見出せなかった．サンプルとなるデータ数が 8 つと少ないため，3 回目の予備実験では，実験データ取得の方法を変更した．

3.3 個人単位の実験

3.3.1 実験の概要

3 回目の実験では，グループ単位ではなく，各参加者の実施結果を収集した [80]．ワークショップを通して，図 3.3 のように，知識を与えていない状態での

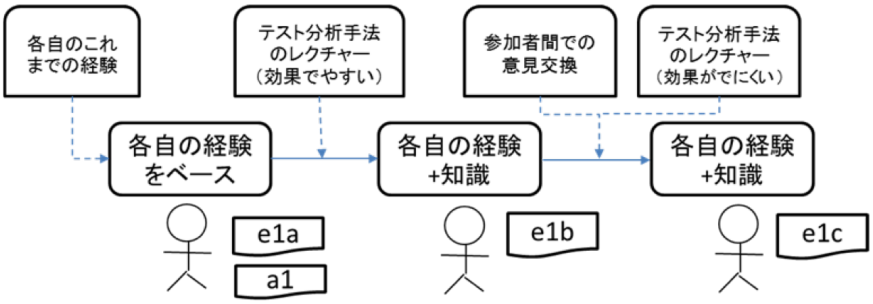


図 3.3: e1a から e1c までの演習の前提条件の変化

演習実施（e1a），一部分だけ知識を与えた状態で演習実施（e1b），すべての知識を与えた状態で演習実施を（e1c）行い，各参加者の演習結果を実験データとして収集した. e1a のタイミングで記述式のアンケート (a1) をとっている．

テストカテゴリベースドテストのレクチャーを 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．

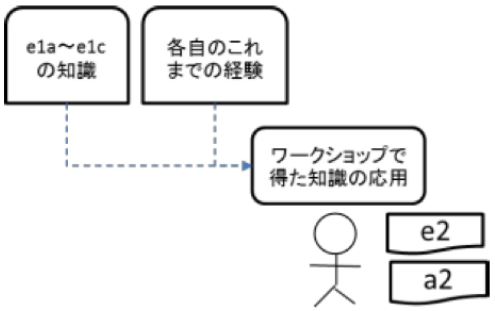


図 3.4: e2 の演習の前提条件の変化

e2 では，図 3.4 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．1c の後には再度アンケート (a2) をとっている．

このワークショップでは，57 名分の IT 技術者のサンプルを収集した．参加者の年齢，業務領域，経験年数とテスト技法の知識は図 3.5，図 3.6，図 3.7 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．

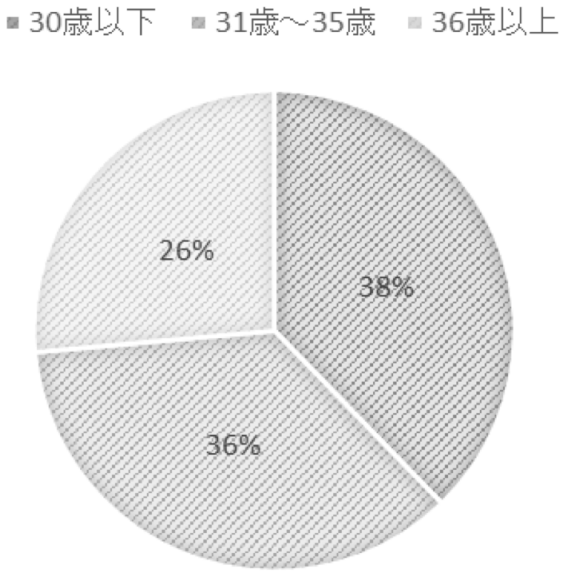


図 3.5: 参加者の年齢分布

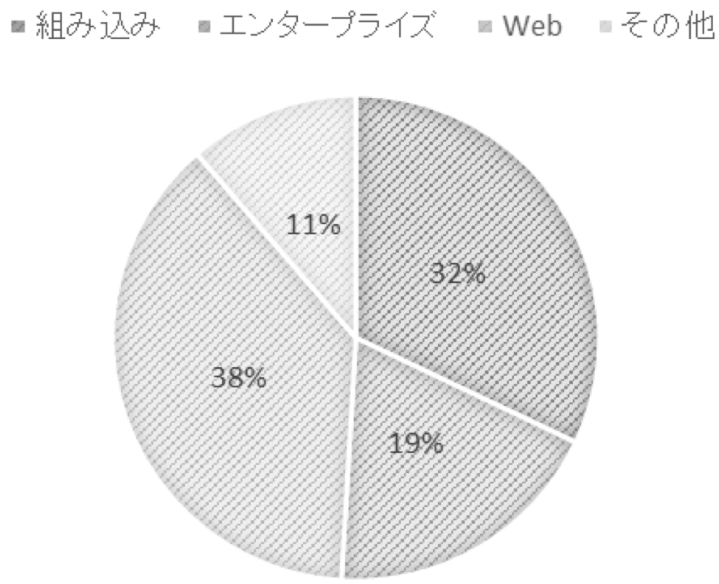


図 3.6: 参加者の業務領域分布

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，a1 にて記述式のアンケートにて確認を行った．テスト技法の研修受講経験のある技術者が 38 人と半分以上をしめているのが，いままでの実験の被験者とは異なる．このワークショップ参加者の特徴として，今までに何らかの研修を受けた経験が一般的な技術者と比較して多いことがあげられる．この理由は，このワークショップは実費を参加者自身が支払い，土日を費やして実施するものであるためで，参加者のキャリア意識が高いことにある．

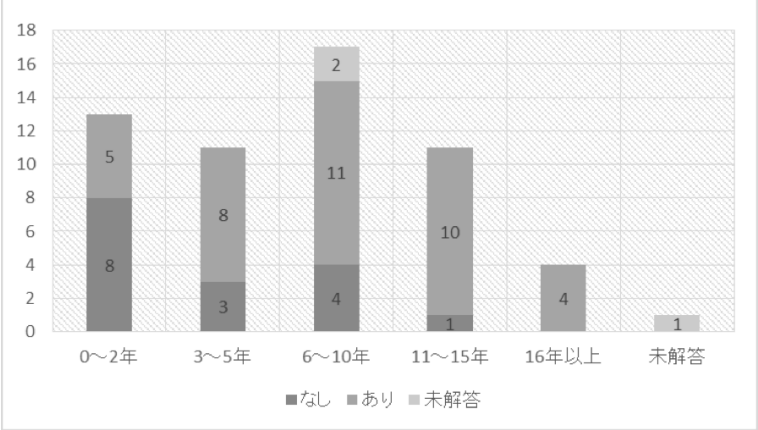


図 3.7: 実務経験とテスト技法研修受講実績

3.3.2 実験の題材

演習題材は，組み込みソフトウェア開発とエンタープライズシステム開発のシステムレベルの仕様を用意した．組み込みソフトウェア開発の演習題材は音楽再生用機器のボリュームコントロールをフィーチャセットとしたものである．3.2 節の [グループ単位 1] の演習で利用した題材と同じものである．エンタープライズシステム開発の演習題材は，フライト予約システムの新規フライト予約をフィーチャセットとして演習問題を用意した．3.2 節の [グループ単位 2] の演習で利用した題材と同じものである．講師側が用意した解答例である，テスト分析で特定すべきテスト条件の一覧は，3.2 節同様に表 3.1 と表 3.2 となる．

3.3.3 実験結果の評価

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 (解答数) を図 3.8 の箱ひげ図を用いて比較する．図 3.8 の Y 軸は解答したテスト条件数を示し，X 軸は，各演習における解答数の分布を箱ひげ図で示している．

e1a(合計 1a) 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(合計 1b) では中央値が 3，e1c(合計 1c) では 4，e2(合計 2) では 7 と，演習が進むごとに中央値が増えているので，テスト条件数を特定するスキルが向上したと考えられる．ただ

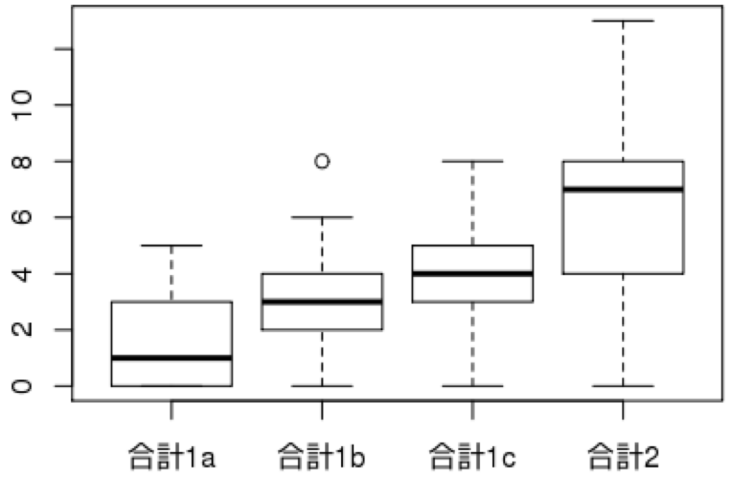


図 3.8: 参加者あたりのテスト条件特定数

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 (e1 は 9 で，e2 は 24 である)．演習で費やす時間が同じため，時間単位での特定できる能力という意味では向上しているものの，単純な数値の比較だけでは不十分である．正解とするケース数を割合で示した箱ひげ図を図 3.9 に示す．

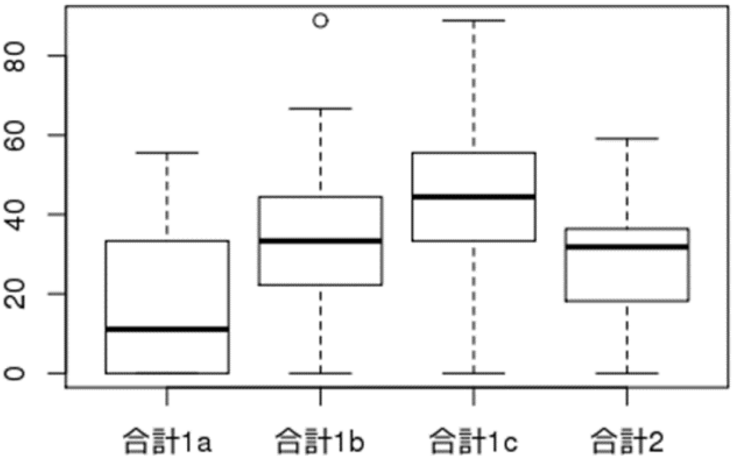


図 3.9: 参加者あたりのテスト条件特定割合

図 3.9 からは， e1a(合計 1a) では約 10 パーセントであったテスト条件の特定数の割合の中央値が，e2(合計 2) では約 40 パーセントまで向上したことが確認

できる．ただし，図 3.8 と図 3.9 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．

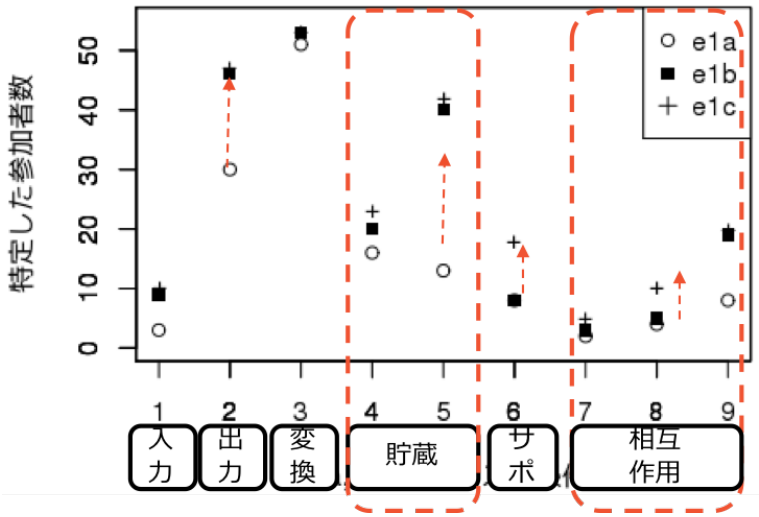


図 3.10: e1a から e1c までの演習解答の分布

各参加者が特定できたテスト条件数 (解答数) を図 3.10 のように論理的機能構造で分類して比較した．図 3.10 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．テストカテゴリベースドテスト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 3.6 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テゴリを示し，難しさと知識を与えることによる教育効果を示した．難しさは，e1a の正解数から 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．表 3.6 から，テスト条件 2 の音声出力と，テスト条件 5 の設定保存は，演習が進むにつれテスト条件が特定できた参加者が増加したため，特に効果が高かったといえる．これは，表 3.5 で示したグループ単位での実験における TM1 から TM6 と同様の結果である．2 章にて，仕様書には明確に記述がないものは，テストカテゴリのようなガイドを使うことで特定が容易になるのと仮説をたてて実験を行ったが，今回の実験でもその仮説を実証する結果となった．

表 3.6: e1a から e1c までの演習結果の変化

	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		設定共有 1	難	中
9		設定共有 2	難	中

難しさ　0-10 難　11-25 中　26-易

効果　0-10 低　11-25 中　26-高

続いて，ワークショップ参加者の業務経験年数ごとに，テスト条件を特定できた数にどのような傾向があるかを調査した結果を図 3.11 の箱ひげ図にて示す．ワークショップ参加者の実業務の経験年数は a1 でのアンケート結果を利用した．図 3.11 のように，e1a では，業務経験 1〜2 年の参加者以外は，すべての経験年数で中央値がほぼ同じになった．テスト分析に関して，入社 3 年でほぼ同様の能力になり，その後は業務経験を重ねてもテストケースを作る能力はあまり変化していない．入社 1〜2 年では，テスト設計技術に関するスキルの伝達がなされていない，もしくは行われていても効果が出ていないことが推定できる．

そして，レクチャーと演習を通じて得たスキルを確認するため，e2a の結果を比較できるようにした箱ひげ図が，図 3.12 である．テスト条件の特定数を中央値で比較すると図 3.11 では 1〜3 だったものが図 3.12 では 6〜7 になっており，全体的に向上している．しかし，図 3.9 で考察したように正解数が違うことを考慮すべきである．この比較で触れておくべき特徴は，業務経験が 0-2 年度の参加者が伸びたたことである．この比較から，演習の前と後では，業務経験とテスト条件を特定できる能力に差がほとんどなくなっていることが確認できた．

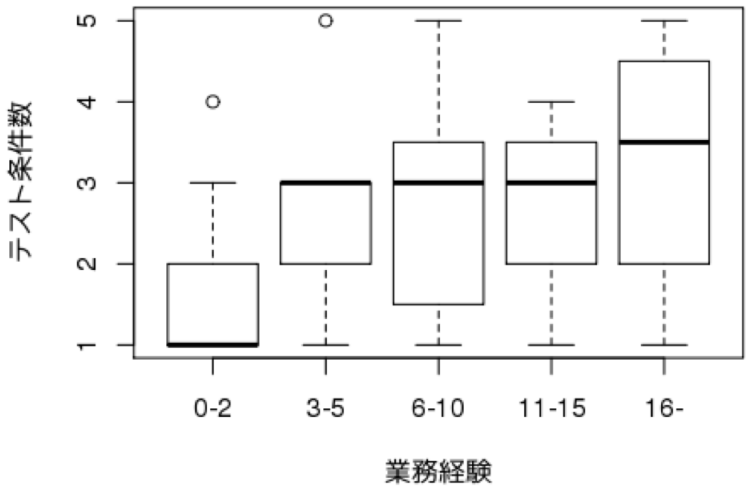


図 3.11: e1a の参加者/業務経験別テスト条件特定数

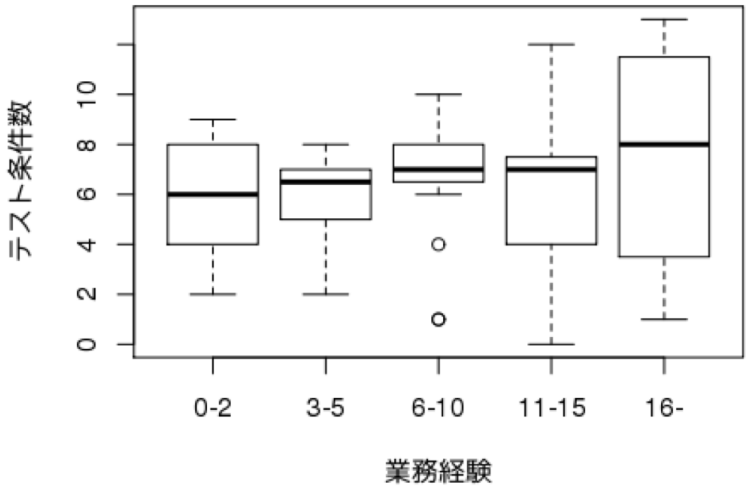


図 3.12: e2 の参加者/業務経験別テスト条件特定数

3.3.4 テストカテゴリベースドテスト適用前のテスト分析方法の分類

今回のワークショップでは，e1a の演習にて，テスト分析結果をこれまでの業務経験に基づいて自由に書いてもらうようにした．テストカテゴリベースドテスト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 3.7 に示す通り 4 つに分類できた．

表 3.7: テスト分析結果記述パターン

	パターン	記載内容	分類
1	仕様項目	「〇〇な場合に××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

表 3.7 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が，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．ここから 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たてた．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，分析，設計の活動に慣れているために知識の習得が早いと仮定し，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 3.13 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 を割り込んでいるため，相関があるとは結論付けられない．

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

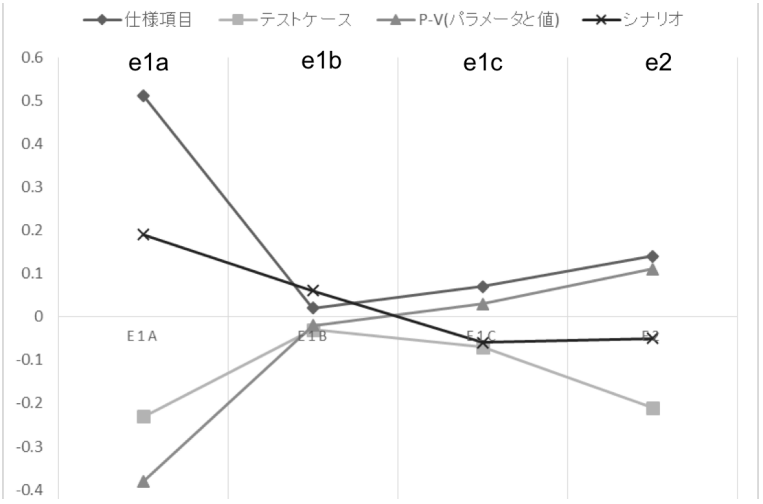


図 3.13: e1a の参加者業務分野別テスト条件特定数

した参加者と結果の相関が出たと考えられる.

3.4 まとめ

本章では，3 回の予備実験を通して，テストカテゴリベースドテストの知識を与える前のテスト分析での結果のばらつきの傾向，及びばらつきの傾向と分析手法を適用後の結果との相関を調査した．実験はワークショップを通じてグループ単位で 2 回，個人単位で 1 回行なった．テスト対象を詳細化するときに分類に対する一貫性が欠如していることによるテスト分析結果のばらつきを確認することができた．また，分類ルールの手法としてテストカテゴリベースドテストの知識を与えることで，テスト条件を特定できる数が増えることが確認できた．仮説として立てた「仕様書には明確に記述がないものは，テストカテゴリのようなガイドを使うことで特定が容易になる」ことが実証できる傾向になった．ただし，テストカテゴリベースドテストの知識を与えても期待した数のテスト条件を特定できるわけではないことが確認できた．また，業務経歴 3 年未満の技術者には有効であったが，3 年以上の技術者にはあまり効果が出ないことも確認した．

第4章　I/O テストデータパターンによるテストケース抽出手法

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）に着目する．テストベースを分析する際に，テスト実行時の I/O の要素で分析を進めることで，網羅的にテスト条件を特定する方法として I/O テストデータパターンを提案する．3 章の題材を使って提案手法の適用評価を行う．また，現実に使われたモバイルアプリケーション開発プロジェクトのテストケースを入手し，そのテストケースと I/O テストデータパターンを使って特定したテスト条件の内容を比較し，手法を適用することで特定できるテスト条件にどのようなものがあるかを確認する．

4.1 I/O テストデータパターンの概要

4.1.1 テストカテゴリベースドテストの課題

3 章で示した予備実験を通して，テストカテゴリベースドテストの適用に一定の効果は確認できたものの，テストカテゴリベースドテストの知識を与えることで期待した数のテスト条件を特定できるようになる状況を観察することはできなかった．また，業務経歴 3 年未満の技術者には有効であったが，業務経歴 3 年以上の技術者には効果を観察できなかった．テストカテゴリベースドテストは，テストベースの分析に論理的機能構造を基にしたテストカテゴリ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ことが理由だと考えられる．

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，テスト対象となる AS のタスク Ta に対して，源泉 So からデータを入力し，Ta から So へ出力されたデータと期待結果とを比較する．

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，AS の外部となる源泉 So_i から計算の対象となる複数の値 In_i を入力し，タスク Ta_i がそれらの値を計算し，計算結果 Out_i をテスト対象の外部である So_i へ出力する．

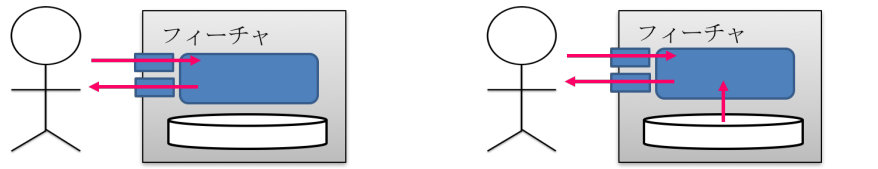


図 4.1: テスト対象へのデータ入出力の説明

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，So_j から入力する数値 In_j に対して，保持データ Ds_j である固定比率を使って計算を行う場合，Ta_j は Ds_j を呼び出し，計算に

その値を利用してから計算結果 Out_j を So_j へ出力する．これは図 4.1 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．

テスト実行をするときの Ta へのデータ In のパターンは，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データ Out のパターンは外部への出力，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 パターンに分類できる．これら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．

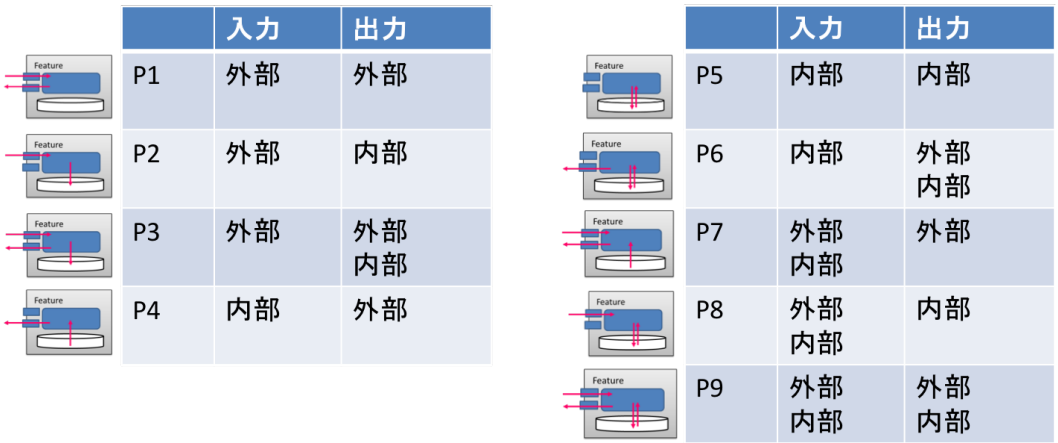


図 4.2: I/O テストデータパターン

これを I/O テストデータパターンと呼ぶ [81]．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 [82] は，AS の Ta に対する In と Out 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，I/O テストデータパターンで提示しているデータの入出力モデルはテストベースからテスト条件の中の仕様項目を特定するためのモデルとして使われる．

注意すべきこととして I/O テスト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．Ta 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，I/O テストデータパターンとしては考慮しない．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，AS 内部のシグナルやイベントのような外部観察ができないコ

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，明示的に考慮できないからである．

テスト分析で特定した仕様項目は，テスト実行をした際の入出力にて確認することができなければならないので，すべてが 9 パターンのどれかに分類できると考えられる．

4.1.3 テストカテゴリベースドテストとの関係

P1 から P9 の I/O テストデータパターンは，単一の Ta に対するデータ入出力の全体像となる．テスト実行の際，So から入力を行い，So へ出力する間に，データは，テスト対象となる Ta によって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．I/O テストデータパターンの P1 から P9 のそれぞれが論理的機能構造のどの要素に該当するかを表 4.1 にまとめた．

表 4.1: I/O テストデータパターンと論理的機能構造

データ I/O		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を行う Ta 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，P1 は，表 4.1 の以下の 3 箇所にプロットされている．

- 列：入力調整　行：入力／外部
- 列：出力調整　行：出力／外部
- 列：変換　行：入力／外部

このデータフローで通過する 3 箇所が，P1 の場合の期待結果を確認する候補となる．これらを本研究ではチェックポイントと呼ぶ．チェックポイントのうち，期待結果を意図的に確認する必要があると判断したものがテスト分析で特定すべき仕様項目，つまりテスト条件となる．

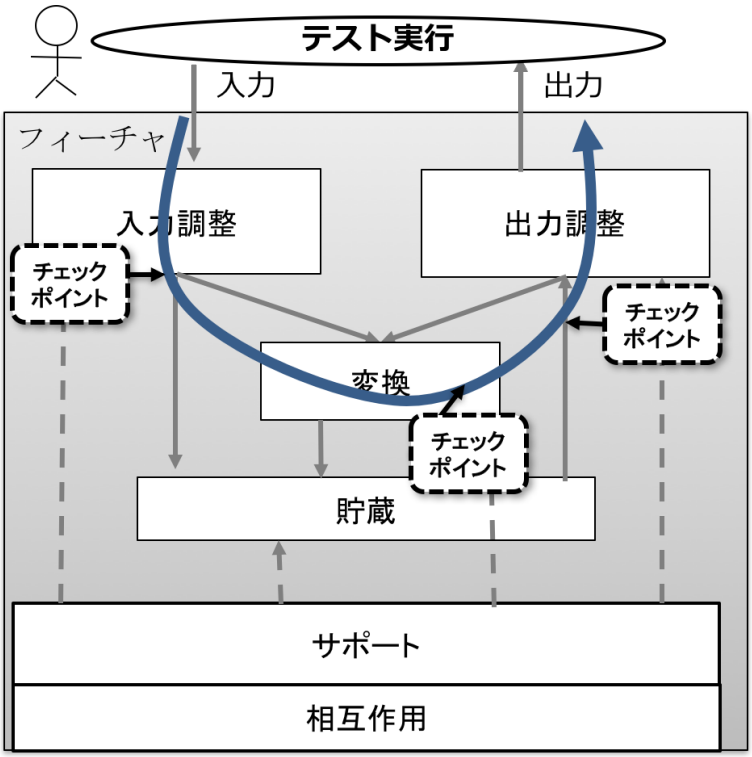


図 4.3: I/O テストデータパターンのデータの流れ

データフローの中で通過する 3 箇所のチェックポイントのうち，実際には，すべてが期待結果を確認する仕様項目になるとは限らない．たとえば，テストすべき仕様項目を特定する際に，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はテストすべきであるが，出力調整が適切にされることは，すでに自明であるためにテスト

ケースとする必要がないということが考えられるためである．少なくとも 1 つ以上のチェックポイントを特定すればよい．

また，表 4.1 では，サポートと相互作用については I/O テストデータパターンがプロットされていない．P1 から P9 の I/O テストデータパターンは，単一の Ta に対するデータ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 Ta に対するデータ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 Ta に対するデータ入出力だけではなく，関係する他の Ta 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，単一の Ta に対するデータの入出力の後，何かしらの方法で呼び出した他の Ta の出力で期待結果を確認する. 呼び出されたタスクの 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．

4.2 I/O テストデータパターンの適用評価

本節では，3 章で行なったテストカテゴリベースドテストの知識を与える前と後を比較する予備実験の結果を使って I/O テストデータパターンの適用評価を行う．最初に，演習にて利用した題材の講師解答例を I/O テストデータパターンに分類するために，解答例のテスト条件に対して，P1 から P9 の識別子を属性として付与する．そして，3.2 節のグループ単位での予備実験結果を 3 章の表 3.4 で示した評価レベルで整理する．その後，表 4.1 と同じフォーマットの表に出現結果を当てはめて，これらの題材にてどの I/O テストデータパターンが現れるのかを確認する．

4.2.1 I/O テストデータパターンの出現傾向の調査

グループ単位の予備実験では，音楽再生機器のフィーチャセット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セットである新規フライト予約の 2 種類の異なった題材を使用した．この予備実験では，被験者のグループがテストカテゴリベースドテストの知識を得ることでテスト条件をどのように多く特定できるようになるかを確認している．この実験結果を I/O テストデータパターンを使って表 3.4 の評価レベルにて評価した結果は，表 4.2 と表 4.3 に示したとおりである．予備実験と I/O テストデータパターンの評価結果では行数が異なる理由は，同じテストカテゴリに属する仕様項目だとしても I/O テストデータパターンが異なるものは別の行にしているためである．たとえば，表 4.2 では，相互作用が 2 行あるが，この中に含まれている仕様項目の I/O テストデータパターンは P4 と P1 となることが該当する．この実験結果からは，表 4.2 と表 4.3 が示すとおり， P1， P2， P4， P7 がフィーチャセットに対応する I/O テストデータパターンとなる．

I/O テストデータパターンで評価レベルの数を集約した結果が表 4.4 と表 4.5 である．各 I/O テストデータパターンにて A と A^+ の割合を確認すると，P2 のみが音楽再生機器，フライト予約システムの両方の検証実験で 100 パーセントとなっているため，P2 のデータパターンに対する効果が最も高いと言える．

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る．飛行機予約シ

表 4.2: 音楽再生機器の演習結果と I/O テストデータパターン

論理的機能構造		グループ					
		TM1	TM2	TM3	TM4	TM5	TM6
入力	P4	B	B	B	B	B	B
出力	P7	-	-	-	-	-	A^+
変換	P1	B	A^+	B	B	B	-
貯蔵	P2	-	A^+	-	A^+	A^+	-
貯蔵	P4	B	B	B	B	-	B
サポート	P1	B	B	B	B	B	B
相互作用	P1	B	A^+	A^+	A^+	A^+	A^+
相互作用	P4	B	B	B	A^+	B	A^+

表 4.3: フライト予約システムの演習結果と I/O テストデータパターン

論理的機能構造		グループ	
		TM1	TM2
入力	P1	B	B
入力	P7	A	-
出力	P4	B	B
出力	P4	A	A
出力	P7	B	B
変換	P7	A	A
貯蔵	P2	A^+	A^+
貯蔵	P7	A	A
サポート	P1	-	A^+
サポート	P1	B	B
相互作用	P4	B	A

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，詳しい記載は無い．これらの観察結果は 2 章にて述べたテスト条件の特定を困難にしている課題の一つである「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.」と一致する．

表 4.4: I/O テストデータパターンごとの集計結果（音楽再生機器）

音楽再生機器

I/O パターン	評価レベル				割合
	A^+	A	-	B	
P1	6	0	1	11	35.3%
P2	3	0	3	0	100.0%
P4	2	0	1	15	11.8%
P7	1	0	5	0	100.0%

割合 = $(A^+ + A) / (A^+ + A + B)$

表 4.5: I/O テストデータパターンごとの集計結果 (フライト予約システム)

フライト予約

I/O パターン	評価レベル				割合
	A^+	A	-	B	
P1	1	0	1	4	20.0%
P2	2	0	0	0	100.0%
P4	0	3	0	3	50.0%
P7	0	5	1	2	71.4%

割合 = $(A^+ + A) / (A^+ + A + B)$

実験にて利用した 2 つの演習題材に対するテスト分析の講師解答例を I/O テストデータパターンに分類した結果が表 4.6 である．この実験の題材にて使われた I/O テストデータパターンは，P1 と P2 と P4 と P7 の 4 つであったため，表 4.6 にはそれらのみを記載している．また P1 としてプロットされているのは入力調

整と変換のみであり，出力調整に該当するテスト条件は無かった．同様に P2 としてプロットしているのは貯蔵のみ，P4 としてプロットしているのは変換と出力調整のみであり，各 I/O テストデータパターンでは，テスト実行のデータ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: I/O テストデータパターンの出現傾向の調査結果

データ I/O		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(VF)		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(VF)	P1(V), P4(VF)
	内部				P2(VF)	P2(F)	
	外部と内部						

4.2.2 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の中で，サポートと相互作用は，単一の Ta に対するデータの入出力だけではなく，関係する他の Ta の呼び出しに着目している．テスト条件として特定するのは，最初のテストアクションで操作されるタスク Ta_k の入出力のデータフローのチェックポイントからではなく，他のタスク Ta_l の入出力のデータフローのチェックポイントである．

表 4.1 では，サポートと相互作用に分類できる 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，表 4.6 で示したとおり，P1 と P2 と P4 にテスト条件を分類した．

表 4.7: サポートと相互作用に分類されたテスト条件表

フィー チャセット	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	相互作用
	対向機のボリュームに影響しないこと	P1	他への反映	相互作用
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	相互作用
	「注文件数グラフ」へ注文が反映すること	P4	他への反映	相互作用
	「注文履歴」へ注文が反映すること	P4	他への反映	相互作用
	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	サポート
	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	サポート

表 4.7 は，表 4.6 にてサポートと相互作用に分類したテスト条件の一覧である．

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．サポートに分類されるテスト条件として特定するのは，フィーチャセットでのテスト実行時のアクションによって操作されるタスク Ta_k から内部的に連動して呼び出される別のタスク Ta_l のチェックポイントをテスト条件とすべき仕様項目として特定する場合である．表 4.7 では，テスト対象の Ta_k へのデータ入力に対して Ta_l の結果を出力するものは，I/O テストデータパターンを P1 としている．また，テスト対象の Ta_k へのデータ入力に対して Ta_l が連動して Ds へ書き込みをしているものは P2 としている．

一方，相互作用は，テスト実行時に，テスト対象の Ta_k へのテストアクションによるデータ入力による副作用を確認する．副作用が起きる例としては，入力データを Ds へ登録した際の状態の変化などが挙げられる．この確認をするためには，他フィーチャセットの Ta_l に対するテストアクションを行い，Ta_l に対するデータフローのチェックポイントをテスト条件とすべき仕様項目として特定する場合である．

音楽再生機器のボリュームコントロールにて特定した仕様項目は，Ta_k へのテストアクションを行なった後に，その副作用を確認するために該当する他フィーチャセットの Ta_l へ外部 So から入力を与えて，そのデータフローの中のチェックポイントを確認するため P1 にしている．フライト予約システムの場合は，Ta_k となる新規フライト予約にて登録した新規予約が Ta_l の結果に影響を与えることを確認するため，他のフィーチャセット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する外部からの入力を与えなくても Ds にて保持している結果を出力すること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トリガーと相互作用に該当する仕様項目の特定に使うトリガーを整理して，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．整理した結果を表 4.8 に示す．

表 4.8 では，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一覧にした．本研究では，各仕様項目を特定する際の単一の Ta に対するデータ入出力の後に他の Ta が動作するきっかけのことをトリガーと呼ぶ．サポートに該当する仕様項目の特定に使う呼び出しのトリガーとしては，割り込みと他処理連動を挙げた．割り込みは，Ta_k の実行中に他の優先度の高い Ta_l が強制的に Ta_k とは違う結果を返すことをさす．他処理連動は，Ta_k の結果から Ta_l が連動して実行され流ことを指す．例外処理を必要とする場合が該当する．相互

表 4.8: サポートと相互作用の仕様項目の呼び出し方法

トリガー 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		○	○	

作用は，リソース共有と他処理への反映を挙げた．リソース共有は，Ta_k と Ta_l が並列に動作している際，データや設定などの共有による副作用があることを指す．他処理への反映は，Ta_k と Ta_l を順番に実行した際に，Ta_l が Ta_k に影響を受けることを指す．これらはサポートや相互作用に該当する仕様項目の特定に使う呼び出しのトリガーを整理することで，テスト条件の特定を行う際に汎用的なパターンとして活用できると考えられる．

4.3 I/O テストデータパターンの効果検証

4.3.1 実験の目的

3 章の実験では，被験者の学習過程に対する効果を検証してきたため，実験用に用意した小さなサンプルを題材として利用した．

この実験では，今回提案している 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より網羅的にテスト条件を特定できるかを検証する [83]．そのために現実にとある開発プロジェクトで利用したテストケースを入手し，そこで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．

- 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．

4.3.2 実験の題材

実験対象のテスト対象では，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 I/O テストデータパターンもあったため，現実の複雑なアプリケーションを対象にすることでより I/O テストデータパターンの適用範囲が広がると仮定したためである．オンラインのモバイル写真共有アプリケーションが持つ全フィーチャセットのうち,「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 2 つをフィーチャセットとして選択した．この 2 つのフィーチャセットを選択した理由は，1 つがデータの内部への投入を行うフィーチャセットであり，もう 1 つがデータの照会のみ行うフィーチャセットであるため，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仮定したためである．このモバイル写真共有アプリケーションの開発のシステムテストレベル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した．

4.3.3 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．
- フィーチャセット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．

以降，各手順で行う内容を具体的に述べていく．

Step1 テストケースのテスト条件への変換

今回の実験の比較対象となるシステムテストレベルにて実際に使われたテストケースには，テストケースの網羅対象となるテスト条件の一覧は成果物として作られていない．このテストケースは，入力値や事前条件を組み合わせた複数のインスタンスとなっていた．今回の実験のためには，このテストケースをテスト分析の成果物であるテスト条件一覧となるように，仕様項目と期待結果をまとめなおす必要がある．まとめ直す際には，図 4.4 のように，テストケースの要素を整理し，同じテストアクションを行い，同じ期待結果を確認しているテストケースを 1 つの仕様項目として集約していった．

実プロジェクト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．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 1 つの仕様項目に対して，現場で作られたテストケースは，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複数のテストケースが作られている．このようなテストケースを整理し，仕様項目として集約していった結果，テスト条件となる仕様項目の数は，アップロードが 59 項目，グリッドビューが 22 項目となった．

手順概要	手順詳細	期待結果
2枚の写真の並び順の確認	1.撮影日が異なる写真を2枚準備する. 2.準備した2枚の写真をアルバムにアップロードする. 3. メインメニューからグリッドビューを選択する. 4. ソートボタンをクリックする. 5. 並び順が撮影日の昇順となる.	グリッドビューは決められた並び順になる.
2ページ以上にまたがって表示されるだけの写真を使った並び順の確認	1.撮影日が異なる写真を40枚準備する. 2.準備した40枚の写真をアルバムにアップロードする. 3. メインメニューからグリッドビューを選択する. 4. ソートボタンをクリックする. 5. 並び順が撮影日の昇順となる.	グリッドビューは決められた並び順になる.

テストカテゴリ	仕様項目	期待結果	テストパラメータ
画面表示	並び順変更	グリッドビューは決められた並び順になる	・昇順. 降順, ・写真の枚数 ・ページ数 ・並び順のルール

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

Step2 入力データ，出力データの特定

フィーチャセット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た結果，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

Step3 I/O テストデータパターン付与

Step2 で特定した入力データと出力データは，Step1 で明らかにした仕様項目に対して図 4.4 のように入力データと出力データという列に追記していった．テスト実行時の追記したデータフロー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 4.5 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．

テストカテゴリ	仕様項目	期待結果	入力データ	出力データ
画面表示	並び順変更	グリッドビューは決められた並び順になる	外部: 並び順情報 内部: イメージ	外部: イメージ 内部:

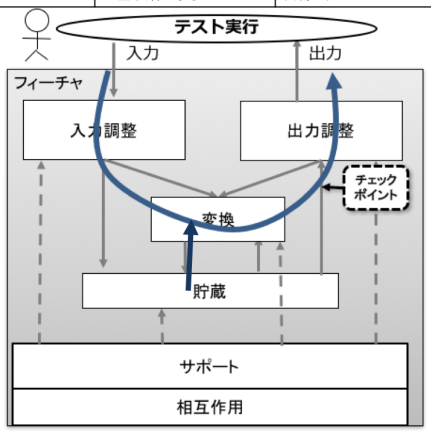


図 4.5: 仕様項目に入力データと出力データを加える方法の説明

4.3.4 I/O テストデータパターンを使ったテストベース分析

I/O テストデータパターンと既存のテスト分析手法であるテストカテゴリベースドテスト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：

- 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フローをシミュレーションしてチェックポイント候補から仕様項目を選択する．
- 実プロジェクトのテストケースの分析結果をテストカテゴリに分類し，差異を比較する．

特定したテストカテゴリは，表 4.9 のようになった．サポートと相互作用については，テストカテゴリがトリガーとなる．トリガーから呼び出す Ta へのデータフローをシミュレーションして，仕様項目を特定した．

表 4.9: テストカテゴリ

入力調整	出力調整	変換	貯蔵	サポート	相互作用
画面上操作	画面表示 メッセージ 初期表示	計算	設定保存 画像保存	割り込み 中断 データ同時変更	リソース共有 反映 並列処理

4.3.5 I/O テストデータパターンの効果検証の結果

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で利用した I/O テストデータパターンは表 4.10 のようになった．実験に使ったフィーチャセット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，両方のフィーチャセットにて，現実の開発で行われたシステムテストレベルのテストケースと 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，現実の開発で使われたテストケースに，テストすべき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 を除くすべてであった．

表 4.10: I/O テストデータパターンの出現傾向

	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	○

実プロジェクトのテスト条件との比較をした結果を図 4.6 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．

4.3.6 I/O テストデータパターン毎の出現傾向の評価

実プロジェクトで作られたテストケースと I/O テストデータパターンを使ってテストベースの分析して特定した仕様項目の特定数を P1 から P9 の分類で出現割合を比較した結果が，表 4.11 である．

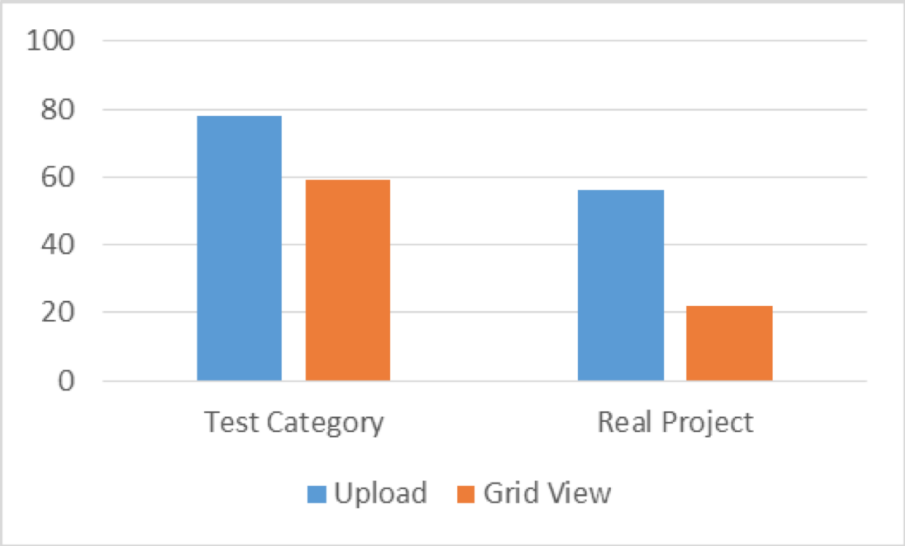


図 4.6: 実プロジェクトで特定したテスト条件の比較

表 4.11: I/O テストデータパターン毎の特定数比較

アップロード

パターン	P1	P2	P3	P4	P6	P7	P9	合計
TC と I/O	24	6	12	14	3	7	12	78
現実の PJ	17	6	10	10	3	5	8	59
	71%	100%	83%	71%	100%	71%	67%	76%

グリッドビュー

パターン	P1	–	–	P4	–	P7	P9	合計
TC と I/O	25	–	–	13	–	14	4	56
現実の PJ	5	–	–	5	–	8	4	22
	20%			38%		57%	100%	39%

上段の TC と IO は，テストカテゴリと I/O テストデータパターンを使ってテスト条件を特定した数であり，下段の現実の PJ は，現実のプロジェクトでのテスト条件数である．欄外に記載した割合は，上段の数を母数にした場合の下段の現実のプロジェクトのテスト条件数の割合である．両方の結果を合計したグラフが図 4.7 である．図 4.7 からは，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．

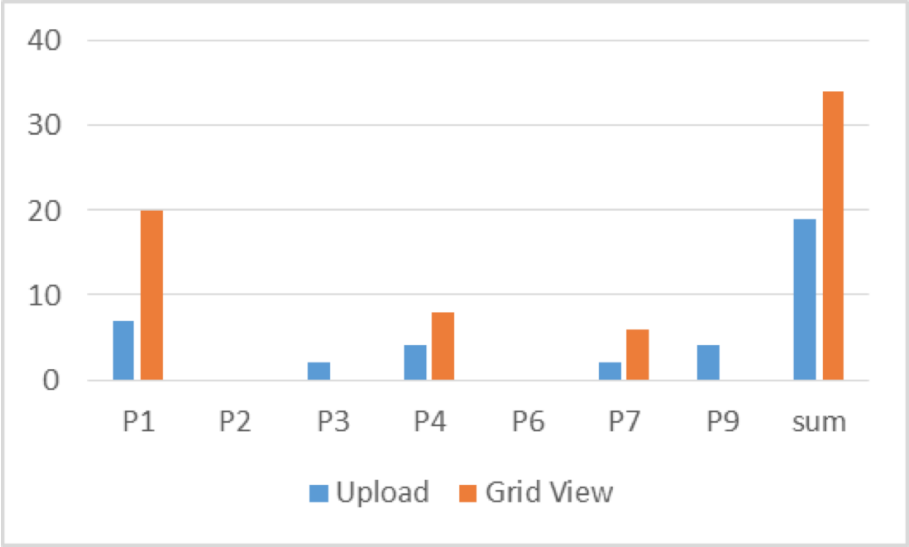


図 4.7: I/O テストデータパターンごとの違い

4.3.7　I/O テストデータパターンにて特定したテスト条件

表 4.12 から表 4.15 に，I/O テストデータパターンにて新たに特定したテスト条件となる仕様項目をすべて列挙した．これらのテスト条件は，すべて今回のデータの 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．I/O テストデータパターンにて新たに特定したテスト条件には，論理的機能構造の要素別に見ると入力調整, 出力調整に分類できるテスト条件，たとえばメッセージが現れることや入力制御といった単純な仕様項目でも漏れていることが確認できる．

一般的に，テスト条件となる仕様項目の一覧を作成せずに具体的なインスタンスとなるテストケースを列挙していく場合，テストパラメータ組み合わせの数量の多さに合わせてテストケースの数量が膨大になるため，網羅すべきテスト条件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．

表 4.12: I/O テストデータパターンにて特定したテスト条件 (1/4)

フィーチャセット	テストカテゴリ	仕様項目	期待結果	I/O	入出力データ
アップロード	データ同時変更	アップロード中のアップロード済み画像のアルバム移動	ロックがかかりできないこと	P9	外入 : 画像　内入 : 移動元画像外出 : 件数　内出 : 画像
アップロード	データ同時変更	アップロード中のアップロード済み画像の情報変更	ロックがかかりできないこと	P9	外入 : 情報　内入 : 画像情報外出 : 変情　内出 : 画像情報
アップロード	データ同時変更	アップロード中のアップロード先アルバム名変更	ロックがかかりできないこと	P9	外入 : 画像，アルバム名　内入 : 元アルバム名外出 : 件数　内出 : 画像
アップロード	データ同時変更	アップロード中のアップロード前画像のアルバム移動	ロックがかかりできないこと	P9	外入 : 画像　内入 : 移動元画像外出 : 件数　内出 : 画像
アップロード	メッセージ	「画像を選択」時の選択枚数表示	「○枚の画像を選択中」というメッセージが出るか	P1	外入 : 画像選択　内入 : – 外出 : メッセージ　内出 : –
アップロード	メッセージ	重複画像だけでアップロードした際の画像のアップロード	「アップロードできない」というメッセージがでるか？	P1	外入 : 画像選択　内入 : – 外出 : メッセージ　内出 : –
アップロード	メッセージ	「画像を選択」画面初期表示	「画像を選択」と表記が出ること	P1	外入 : 文字　内入 : – 外出 : 文字　内出 : –
アップロード	メッセージ	「画像を選択」時の選択枚数表示にて，選択済みの画像を押下	「○枚の画像を選択」というメッセージの枚数が減ること	P1	外入 : 画像選択　内入 : – 外出 : メッセージ　内出 : –
アップロード	メッセージ	「画像を選択」時の選択枚数表示にて，選択済みの画像を押下	選択数が 0 になった場合,「画像を選択」と表記が出ること	P1	外入 : 画像選択　内入 : – 外出 : メッセージ　内出 : –
アップロード	画像表示	「アルバムに追加」でのアルバム一覧	アルバム一覧の表示順が正しいこと	P4	外入 : –　内入 : 画像サムネイル，アルバム名外出 : 画像サムネイル，アルバム名　内出 : –
アップロード	画像表示	「画像を選択」時の画像表示順	撮影日昇順で表示されること	p4	外入 : –　内入 : 画像外出 : 画像　内出 : –
アップロード	初期表示	「アップロード」画面の操作	新規アルバム YYYY/MM/DD というアルバム名が表示されること	P4	外入 : –　内入 : 初期設定外出 : アルバム名　内出 : –
アップロード	初期表示	「アップロード」画面の操作	「すでにアップロード済みの画像を保存しない」が ON になっていること	P4	外入 : –　内入 : 初期設定外出 : 設定　内出 : –
アップロード	画面上操作	「アップロード」画面の操作	「すでにアップロード済みの画像を保存しない」が ON → OFF → ON に変更できること	P1	外入 : 設定入力　内入 : – 外出 : 設定　内出 : –

表 4.13: I/O テストデータパターンにて特定したテスト条件 (2/4)

フィーチャセット	テストカテゴリ	仕様項目	期待結果	I/O	入出力データ
アップロード	画面上操作	「アルバムに追加」に表示されるアルバム名	「全画像」を選択できること	P7	外入 : 操作入力　内入 : 登録済画像外出 : 登録済画像　内出 : –
アップロード	画面上操作	「画像を選択」での画像選択	画像にタップするとチェックがつくこと	P7	外入 : 選択　内入 : 画像外出 : チェックアイコン　内出 : –
アップロード	中断	ネットワーク切断から再開したときのアップロード	一定以上の時間が過ぎるとアップロードが再開しない（要確認）	P1	外入 : 画像　内入 : – 外出 : メッセージ　内出 : –
アップロード	画像保存	アルバムの枚数が 200 枚以上になる枚数で画像を選択してアップロードのエラーが出た後に別のアルバムを選択してアップロード	アップロードできた画像が NIS にて表示できること	P3	外入 : 画像　内入 : – 外出 : 枚数　内出 : 画像
アップロード	画像保存	アップロードした画像の画質	アップロードした画像が崩れていないこと	P3	外入 : 画像　内入 : – 外出 : 枚数　内出 : 画像
グリッドビュー	メッセージ	選択画面初期表示	「画像を選択」と表記が出ること	P1	入力 : 文字出力 : 文字
グリッドビュー	メッセージ	選択解除ボタン押下	「画像を選択」と表記が出ること	P1	入力 : 選択　文字出力 : 文字
グリッドビュー	メッセージ	選択済みの画像を押下	「〇枚の画像を選択」というメッセージの枚数が減ること	P1	入力 : 選択出力 : 文字
グリッドビュー	メッセージ	選択済みの画像を押下	選択数が 0 になった場合,「画像を選択」と表記が出ること	P1	入力 : 選択出力 : 文字
グリッドビュー	メッセージ	選択ボタン押下	「アルバムを選択中」というメッセージが出ること	P1	入力 : 選択出力 : 文字
グリッドビュー	メッセージ	未選択の画像を押下	「〇枚の画像を選択」というメッセージが出ること	P1	入力 : 選択出力 : 文字
グリッドビュー	画像表示	1 画面あたりの表示ファイル数	縦 : 4 × 10. 横 : 3 × 7 になること	P4	入力 : 既存設定出力 : 文字画像

表 4.14: I/O テストデータパターンにて特定したテスト条件 (3/4)

フィーチャセット	テストカテゴリ	仕様項目	期待結果	I/O	入出力データ
グリッドビュー	画像表示	スクロール時の枚数表示	複数ページのスクロールでファイルの日付と，現在の枚数順/全体の枚数が画面中央に表示されること	p7	入力 : スクロール出力 : 文字 画像
グリッドビュー	画像表示	画像ファイルの並び順	左から右方向に埋まっていくこと。	P4	入力 : 既存設定出力 : 文字 画像
グリッドビュー	画像表示	画像ファイルの並び順	2 行目になるとまた左から埋まること	P4	入力 : 既存設定出力 : 文字 画像
グリッドビュー	画像表示	更新ボタン押下	別デバイスで新しい画像の追加，名称の変更などをしていた場合，変更した画像に更新されること	P4	入力 : 既存設定出力 : 画像
グリッドビュー	共有	他ビューの並び順影響有無	他のアルバムやビューで設定した並び順の影響を受けないこと	p4	入力 : 既存設定出力 : 画像
グリッドビュー	設定保存	画面の並び順設定用画面	前回選択した並び順にチェックがついていること	p7	入力 : 並び順出力 : 並び順
グリッドビュー	設定保存	画面の並び順設定用画面	デフォルトでチェックされていること	p7	入力 : 並び順出力 : 並び順
グリッドビュー	画面上操作	OK ボタン押下	全画像，カテゴリから移動したグリッドビューの場合のみ OK ボタンが現れ，共有設定画面に遷移する	P1	入力 : 既存設定出力 : 画像
グリッドビュー	画面上操作	アルバム追加ボタン押下	アルバムから移動したグリッドビューの場合，アルバム追加画面に遷移する	P1	入力 : 既存設定出力 : 文字 画像
グリッドビュー	画面上操作	ダウンロードボタン押下	アルバムから移動したグリッドビューの場合，DL サイズ選択画面に遷移する	P1	入力 : 既存設定出力 : 文字
グリッドビュー	画面上操作	削除ボタン押下	アルバムから移動したグリッドビューの場合，削除画面に遷移	P1	入力 : 既存設定出力 : 文字
グリッドビュー	画面上操作	選択画面初期表示	全画像から遷移したグリッドビューの場合，選択，解除，削除，アルバム追加，DL ボタン出ない．OK ボタン出る．	P1	入力 : 既存設定出力 : 文字 画像
グリッドビュー	画面上操作	選択画面初期表示	カテゴリから遷移したグリッドビューの場合，全画像からの遷移と同じであり，かつカテゴリを選択する選択用ビューがグレーアウトする	P1	入力 : 既存設定出力 : 文字 画像

表 4.15: I/O テストデータパターンにて特定したテスト条件 (4/4)

フィーチャセット	テストカテゴリ	仕様項目	期待結果	I/O	入出力データ
グリッドビュー	画面上操作	選択画面初期表示	端末の写真のグリッドビューの場合，最初から選択画面となり，チェックをすることでOKボタンが有効になる	P1	入力：既存設定出力：文字画像
グリッドビュー	画面上操作	選択解除ボタン押下	画面上の全ての画像のチェックが外れること	p1	入力：選択出力：アイコン
グリッドビュー	画面上操作	選択済みの画像を押下	画像のチェックが外れること	p1	入力：選択出力：アイコン
グリッドビュー	画面上操作	全て選択ボタン押下	画面上の全ての画像にチェックがつくこと	p1	入力：選択出力：アイコン
グリッドビュー	画面上操作	未選択の画像を押下	画像のチェックがつくこと	p1	入力：選択出力：アイコン
グリッドビュー	画面上操作	未選択の画像を押下	全てチェックの場合，押下した画像以外のチェックが全て外れること	p1	入力：選択出力：アイコン
グリッドビュー	画面上操作	フロービューボタン押下	フロービューに遷移すること	P7	入力：既存設定出力：文字画像
グリッドビュー	画面上操作	マップビュー押下	マップビューに遷移すること	P7	入力：既存設定出力：文字画像
グリッドビュー	画面上操作	画像選択ボタン押下	ピクチャービューに遷移すること	P7	入力：既存設定出力：文字画像
グリッドビュー	画面上操作	画面の並び順設定用画面	ファイル名，アップロード名，撮影日，ファイルサイズ，ファイル形式，お気に入り，マイルールの順で一覧表示されていること	P1	入力：既存設定出力：文字
グリッドビュー	画面上操作	並び順の変更	一覧項目の横をタップするとチェックがつくこと	p1	入力：設定出力：設定
グリッドビュー	中断	ネットワーク切断時のグリッドビュー初期表示	キャッシュがある場合，キャッシュされた画像が一覧表示されること	P4	入力：既存設定出力：文字画像
グリッドビュー	中断	ネットワーク切断時のグリッドビュー初期表示	キャッシュの限界を超えた場合，その画像が一覧表示されないこと	P4	入力：既存設定出力：文字画像
グリッドビュー	中断	ネットワーク切断時の更新ボタン押下	キャッシュがある場合，更新ボタンを押すと別デバイスの変更が反映されず画像が表示される	P4	入力：既存設定出力：文字画像

4.4 まとめ

本章では，テスト実行時のデータ入出力の要素で分類し網羅的に分析する I/O テストデータパターンを提案した．そして，入手した現実の開発プロジェクトのテストケースを使い，I/O テストデータパターンで特定したテスト条件との比較を試みた．提案した I/O テストデータパターン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．

また，実プロジェクトで作られたテストケースからまとめ直したテスト条件の一覧を表 4.16 から表 4.21 に列挙した．これらのテスト条件は I/O テストデータパターンを使ったシミュレーションでもすべて特定可能であった．

表 4.16: 実プロジェクトのテストで使われたテスト条件 (1/6)

フィーチャセット	テストカテゴリ	仕様項目	期待結果	I/O	入出力データ
アップロード	データ同時変更	アップロード先のアルバムに複数のデバイスからアップロードして結果的に合計が 200 枚以上になる枚数で画像を選択してアップロード	（要確認）	P3	外入：画像　内入：— 外出：件数　内出：画像
アップロード	データ同時変更	アップロード中のアップロード済み画像が入るアルバム削除	未整理の画像としてアップロードされる?（要確認）	P9	外入：画像　内入：元アルバム名外出：件数　内出：画像
アップロード	データ同時変更	アップロード中のアップロード済み画像の削除	（要確認）	P9	外入：画像　内入：削除元画像外出：件数　内出：画像
アップロード	データ同時変更	アップロード中のアップロード前画像の削除	（要確認）	P9	外入：画像　内入：削除元画像外出：件数　内出：画像
アップロード	データ同時変更	アップロード中のアップロード前画像の情報変更	（要確認）	P9	外入：情報　内入：画像情報外出：変情　内出：画像情報
アップロード	メッセージ	100 枚以上の画像を選択してアップロード	「規定以上の枚数はアップロードできない」というメッセージが出て処理が終了する	P3	外入:画像選択，画像　内入:–外出：件数　内出：画像
アップロード	メッセージ	アップロード先のアルバムの枚数が 200 枚以上になる枚数で画像を選択してアップロード	「規定以上の枚数はアップロードできない」というメッセージ出てが出て処理が終了する	P1	外入：画像選択　内入：– 外出：メッセージ　内出：–
アップロード	メッセージ	マイフォト全体での枚数の MAX を超えたときの画像のアップロード	画像が上限に達したことを知らせるメッセージが出て処理が終了する	P1	外入：画像選択　内入：– 外出：メッセージ　内出：–
アップロード	メッセージ	アルバム数が MAX（3500）を超えたときの新規アルバム追加のアップロード	アルバムが上限に達したことを知らせるメッセージが出て処理が終了する	P1	外入：画像選択　内入：– 外出：メッセージ　内出：–
アップロード	メッセージ	アップロード終了時のＤＬ／ＵＬ一覧へのメッセージ	ＤＬ／ＵＬ一覧で終了したことがわかること	P1	外入：画像　内入：– 外出：メッセージ　内出：–
アップロード	画像表示	アップロードした画像の表示確認	画像にアップロードアイコンが付与されて表示されている	p4	外入：–　内入：画像外出：画像　内出：–
アップロード	画像表示	「アルバムに追加」でのアルバム一覧	アルバムが無い場合はアルバムが出てこないこと	P4	外入：–　内入：画像サムネイル，アルバム名外出：画像サムネイル，アルバム名　内出：–
アップロード	画像表示	「画像を選択」で表示される画像種類	動画（アップロードできないコンテンツ）が選択画面に出てこないこと	P4	外入：–　内入：画像外出：画像　内出：–
アップロード	画像表示	「画像を選択」時の画像表示	複数ページにまたがる場合はスクロールできること	p7	外入：フリック　内入：画像外出：画像　内出：–

表 4.17: 実プロジェクトのテストで使われたテスト条件 (2/6)

フィーチャセット	テストカテゴリ	仕様項目	期待結果	I/O	入出力データ
アップロード	画像表示	アップロード状況確認画面	アップロードできた画像が先に表示されこれからアップロードする画像が色が黒っぽく表示されること	P9	外入 : 画像　内入 : 登録済画像外出:画像登録結果　内出: 画像
アップロード	画像表示	アップロード状況確認画面	ドラッグ&ドロップでアップロード済とアップロード前の画像の表示を入れ替えることができないこと	P7	外入 : 操作入力　内入 : 画像　外出 : 表示位置　内出 : –
アップロード	画像表示	アップロード状況確認画面	複数ページにまたがる場合はスクロールできること	p7	外入 : フリック　内入 : 画像　外出 : 画像　内出 : –
アップロード	画像保存	画像のアップロード実施	アップロードできた画像が NIS にて表示できること	P2	外入 : 画像　内入 : – 外出 : –　内出 : 画像
アップロード	画像保存	画像のアップロード実施	ＪＰＧとして保存されること	P2	外入 : 画像　内入 : – 外出 : –　内出 : 画像
アップロード	画像保存	重複画像のアップロード無効	重複画像がアップロードされていないこと	P2	外入 : 画像（なし）　内入 : – 外出 : –　内出 : 画像（なし）
アップロード	画像保存	アップロード画像の名称が MAX を超える	要確認 ???	P2	外入 : 画像（なし）　内入 : – 外出 : –　内出 : 画像（なし）
アップロード	割り込み	アップロード中に電話などの割り込み入る	処理は継続する（要確認）	P3	割り込みで処理をバックグランドにするタスクだが，I/O としてはその間 P3 が続いている外入 : 画像　内入 : – 外出 : 枚数　内出 : 画像
アップロード	計算	アップロード中のＤＬ／ＵＬ一覧	ＤＬ／ＵＬ一覧で何枚までアップロードしているかがわかること	P3	外入 : 画像　内入 : – 外出 : 枚数　内出 : 画像
アップロード	画面上操作	「アルバム名編集」でのアルバム名が空白	アルバム名の保存ボタンが押下ができないこと	P1	外入 : 設定入力　内入 : – 外出 : 設定　内出 : –
アップロード	画面上操作	「アルバム名編集」でのアルバム名が空白	40 文字を超えて入力できないこと	P1	外入 : 設定入力　内入 : – 外出 : 設定　内出 : –
アップロード	画面上操作	「アルバムに追加」でのアルバム名変更	新規アルバム名変更でアルバム名が変更できること	P1	外入 : アルバム名入力　内入 : – 外出 : アルバム名　内出 : –
アップロード	画面上操作	「アルバムに追加」に表示されるアルバム名	既存のアルバムを選択できること	P1	外入 : –　内入 : アルバム名　外出 : アルバム名　内出 : –
アップロード	画面上操作	「画像を選択」での画像選択	ドラッグによる表示順移動ができないこと	P7	外入 : 操作入力　内入 : 画像　外出 : 表示位置　内出 : –
アップロード	画面上操作	「画像を選択」でのアップロードボタン押下	ＤＬ／ＵＬアイコンが表示されもとの画面にもどること	P3	外入 : 選択，画像　内入 : – 外出 : 枚数　内出 : 画像

表 4.18: 実プロジェクトのテストで使われたテスト条件 (3/6)

フィーチャセット	テストカテゴリ	仕様項目	期待結果	I/O	入出力データ
アップロード	画面上操作	「画像を選択」でのアップロードボタン押下	画像が選択されていないときにはアップロード画面へ遷移しないこと	P1	外入：選択　内入：– 外出：メッセージ　内出：–
アップロード	画面上操作	ＤＬ／ＵＬ一覧のＸボタン押下	キャンセルダイアログに遷移すること	P1	外入：選択　内入：— 外出：メッセージ　内出：—
アップロード	画面上操作	ＤＬ／ＵＬ一覧の件数	ＤＬ／ＵＬ一覧が 20 件を超えると古いものから削除されること	P1	外入：選択　内入：– 外出：メッセージ　内出：–
アップロード	画面上操作	ＤＬ／ＵＬ中に新規にアップロードを行う	Waiting 状態になり，前の処理が終了するとアップロードを開始すること	P3	外入：選択　画像　内入：– 外出：メッセージ　内出：画像
アップロード	画面上操作	ＤＬ／ＵＬアイコンからの遷移	ＤＬ／ＵＬ一覧へ遷移すること	P3	外入：選択　画像　内入：– 外出：登録済み件数　内出：画像
アップロード	画面上操作	ＤＬ／ＵＬ一覧で UL 中を選択	状況確認画面へ遷移する	P9	外入：選択　画像　内入：登録済画像情報外出：登録済み件数　内出：画像
アップロード	画面上操作	ＤＬ／ＵＬ一覧でのクローズボタン押下	ＤＬ／ＵＬアイコンに戻る。遷移先画面は遷移元画面と同じであること。	P3	外入：選択　画像　内入：– 外出：アイコン　内出：画像
アップロード	画面上操作	キャンセル実行	ＤＬ／ＵＬ一覧にてキャンセルボタンが出てこなくなること。	P1	外入：選択　内入：– 外出：表示　内出：–
アップロード	中断	アップロードのネットワーク切断	アップロードが止まる（要確認）	P1	外入：止めるための情報　画像　内入：– 外出：メッセージ　内出：–
アップロード	中断	ネットワーク切断から再開したときのアップロード	中断していたアップロードが再開すること。	P3	外入：画像　内入：– 外出：件数　内出：画像
アップロード	中断	アップロード中にアプリからログアウトした際のアップロード	アップロードを継続するか？（要確認）	P2	外入：画像　内入：– 外出：　内出：画像
アップロード	中断	キャンセル時のアップロード画像の扱い	キャンセルするまでの画像がアップロードできて，その後の画像はアップロードしていないこと	P1	外入：選択　内入：– 外出：表示　内出：–
アップロード	中断	アップロード中にアプリからログアウトした際のＤＬ／ＵＬ一覧	ＤＬ／ＵＬ一覧が全てクリアされる	P1	外入：選択　内入：– 外出：表示　内出：–
アップロード	中断	アップロード中にアプリを終了した際のＤＬ／ＵＬ一覧	ＤＬ／ＵＬ一覧が全てクリアされる	P1	外入：選択　内入：– 外出：表示　内出：–
アップロード	中断	アップロード中にアプリを終了した際のアップロード	中断する（要確認）	P1	外入：選択　内入：– 外出：表示　内出：–
アップロード	中断	アップロード中にデバイスの電源を切った際のＤＬ／ＵＬ一覧	ＤＬ／ＵＬ一覧が全てクリアされる	P1	外入：選択　内入：– 外出：表示　内出：–

表 4.19: 実プロジェクトのテストで使われたテスト条件 (4/6)

フィーチャセット	テストカテゴリ	仕様項目	期待結果	I/O	入出力データ
アップロード	反映	アップロードした画像の詳細情報	位置情報など全部の情報が登録されていること	P4	外入:– 内入:画像外出:画像 内出:–
アップロード	反映	アップロードした画像の場所	指定したアルバムの中にだけ表示されていること	P4	外入:– 内入:画像外出:画像 内出:–
アップロード	反映	アップロードした画像の表示	各種ビューでバリエーションとして確認	P4	外入:– 内入:画像外出:画像 内出:–
アップロード	反映	アップロード中画像のダウンロード	ダウンロードができること	P4	外入:– 内入:画像外出:画像，枚数 内出:–
アップロード	反映	アップロードした際に作成したアルバム確認	マイフォトにてアルバムが出来ていること	P4	外入:– 内入:アルバム外出:アルバム，枚数 内出:–
アップロード	並列処理	アップロード中に共有フォルダを作成する	影響を与えないでアップロードが続く	P3	割り込みで処理をバックグランドにするタスクだが，I/Oとしてはその間 P3 が続いている外入:画像 内入:– 外出:枚数 内出:画像
アップロード	並列処理	アップロード中に設定画面へ遷移し操作する	設定をしている間もアップロードが続くこと	P3	割り込みで処理をバックグランドにするタスクだが，I/Oとしてはその間 P3 が続いている外入:画像 内入:– 外出:枚数 内出:画像
アップロード	並列処理	アップロード中に他のビューで閲覧する	各種ビューでバリエーションとして確認	P7	割り込みで処理をバックグランドにするタスクだが，I/Oとしてはその間 P3 が続いている外入:選択 内入:画像外出:画像 内出:–
アップロード	並列処理	アップロード中に他のビューで閲覧する	閲覧中もアップロードが続く	P3	割り込みで処理をバックグランドにするタスクだが，I/Oとしてはその間 P3 が続いている外入:画像 内入:– 外出:枚数 内出:画像
アップロード	データ同時変更	アップロード中のアップロード済み画像が入るアルバム削除	アルバムを削除した後に同じアルバム名を作って再度アップロードを繰り返した際にアップロードができること	P9	外入:画像 内入:削除元画像外出:件数 内出:画像
アップロード	画像保存	アルバムの枚数が 200 枚以上になる枚数で画像を選択してアップロードのエラーが出た後に別のアルバムを選択してアップロード	アップロードした画像の詳細情報の登録ができていること	P2	入力:画像情報出力:画像情報
アップロード	画像保存	一度アップロードした画像を削除してまた同じ画像をアップロードする	アップロードできた画像が NIS にて表示できること	P3	外入:画像 内入:– 外出:枚数 内出:画像

表 4.20: 実プロジェクトのテストで使われたテスト条件 (5/6)

フィーチャセット	テストカテゴリ	仕様項目	期待結果	I/O	入出力データ
アップロード	画面上操作	ＤＬ／ＵＬ中にダウンロードが失敗やキャンセルなど処理が中断した後に新規にアップロードを行う	直前のパターンが失敗する，キャンセルする，失敗が続く，エラーがでるといった状況を複数組み合わせてアップロード対象がアップロードできることを確認する	P3	外入：画像　内入：– 外出：枚数　内出：画像
アップロード	反映	アップロード中画像がエラーになる場合のダウンロード	エラーなったときのその直前の成功した画像，アップロードが失敗する画像のダウンロードができること	P4	外入：–　内入：画像外出：画像，枚数　内出：–
グリッドビュー	メッセージ	規定枚数以上の画像を押下	「一度に選択できる画像は 100 枚までです」というメッセージが出る	P1	入力：選択出力：文字
グリッドビュー	画像表示	デフォルトの画像ファイル表示順	デフォルト設定にあわせて表示されること	P4	入力：既存設定出力：文字 画像
グリッドビュー	画像表示	デフォルトの画像レイアウト	縦横が基の画像のとおりに表示されること	P4	入力：既存設定出力：文字 画像
グリッドビュー	画像表示	画像枚数がゼロ	空白で表示されること	P1	入力：既存設定出力：文字
グリッドビュー	画像表示	並び順設定後の画像ファイル表示順	ファイル名で表示されること（昇順，降順）	p7	入力：並び順出力：文字　画像
グリッドビュー	画像表示	並び順設定後の画像ファイル表示順	アップロード名の昇順で表示されること	p7	入力：並び順出力：文字　画像
グリッドビュー	画像表示	並び順設定後の画像ファイル表示順	撮影日の昇順で表示されること	p7	入力：並び順出力：文字　画像
グリッドビュー	画像表示	並び順設定後の画像ファイル表示順	ファイルサイズの昇順で表示されること	p7	入力：並び順出力：文字　画像
グリッドビュー	画像表示	並び順設定後の画像ファイル表示順	ファイル形式の昇順で表示されること	p7	入力：並び順出力：文字　画像
グリッドビュー	画像表示	並び順設定後の画像ファイル表示順	お気に入りの昇順で表示されること	p7	入力：並び順出力：文字　画像
グリッドビュー	画像表示	並び順設定後の画像ファイル表示順	マイルールで設定した順番で表示されること	p7	入力：並び順出力：文字　画像
グリッドビュー	画像表示	戻るボタン押下	マイフォト画面に遷移すること	P4	入力：既存設定出力：画像
グリッドビュー	設定保存	ドラッグドロップによるマイルール作成	ドラッグドロップの表示位置変更結果をマイルールとして保存すること	p9	入力：画像，並び順出力：画像，マイルール
グリッドビュー	画面上操作	選択画面初期表示	全て選択ボタン，選択解除ボタンの位置が上もしくは下のどちらかに現れること	P1	入力：既存設定出力：文字
グリッドビュー	画面上操作	「並び順」ボタン押下	画像の並び順画面に遷移すること	p1	入力：既存設定出力：並び順

表 4.21: 実プロジェクトのテストで使われたテスト条件 (6/6)

フィーチャセット	テストカテゴリ	仕様項目	期待結果	I/O	入出力データ
グリッドビュー	画面上操作	スクロール	上下にスクロールできること	p1	入力：選択出力：表示
グリッドビュー	画面上操作	ドラッグドロップによる表示位置変更	ドラッグドロップした位置に移動できること	p9	入力 : 画像，並び順出力 : 画像，マイルール
グリッドビュー	画面上操作	画像押下	ピクチャービューに遷移すること	P7	入力 : 既存設定出力 : 文字画像
グリッドビュー	画面上操作	並び順の変更	変更したい並び順を選択するとグリッドビュー画面に遷移すること	p9	入力 : 画像，並び順出力 : 画像，並び順
グリッドビュー	画面上操作	並び順の変更	変更したソート順で表示されること	p9	入力 : 画像，並び順出力 : 画像，並び順
グリッドビュー	中断	ネットワーク切断時のグリッドビュー初期表示	キャッシュがない場合，グリッドビューを表示すると画像が表示されないこと	P4	入力:既存設定出力:画像（なし）
グリッドビュー	中断	ネットワーク切断時の更新ボタン押下	キャッシュがない場合，更新ボタンを押すと「インターネット接続がオフラインのようです」のメッセージがでる	P4	入力:既存設定出力:画像（なし）

第5章　データ共有タスク間の順序組み合わせテストケース抽出手法

本章では，統合テストにおける複数のタスクの組み合わせに着目する，1 つのタスクによる別のタスクへの副作用を決定するのは，関係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．実践の場においては，このような確認には状態遷移テスト手法を使うことが主流である．しかし，その網羅基準である S1 網羅基準を達成することに膨大なテスト工数を要する課題があることから，その対処として経験的なノウハウを基に重要なテストケースを抽出する方法が用いられている [84] [85].

その 1 つの方法として，状態遷移の S1 網羅基準のうち，重要な順序組み合わせを見つけ出す実践的なナレッジとして，状態はタスク内で保持するデータとして実装されることに着目したテストケースの抽出方法がある．この方法では，保持するデータに対する操作順序が影響することから，データに対するタスクの操作順序パターンを使ってタスクの順序組み合わせをテストケースとして抽出する [86]．順序組み合わせのテストは，ソフトウェアに変更を加えた場合の変更波及のテストケースとして有効になる．

この方法で抽出したテストケースの網羅基準は，データフローテストを基にする [87] [88] [89]．データフローテストの網羅基準である全使用法（AU 法）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 [90].

そして，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.

5.1 データを共有する複数タスク間のテストの概要

5.1.1 I/O テストデータパターンの課題

前章では，I/O テストデータパターンで単一の Ta に対する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．論理的機能構造でのサポートと相互作用に分類されるテスト条件に関しては，トリガーを基にテスト条件を特定する方法を提案したが，具体的な抽出手順については言及できていなかった．

そのため，本章では，テスト条件を特定する際のトリガーの一つである，他処理への反映に着目する．これは，単一の Ta に対するデータの入出力による副作用を他のタスクで確認するテストである．他処理への反映をテスト条件に加える目的は，AS の一部に変更を加えた場合のその変更に対する波及の確認が必要となるためである．

現実的に，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となっている．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 [91]．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 [92]．

これらの課題に対応するため，状態遷移を持つ AS 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ケースの抽出手法を提案する．

5.1.2 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．

1 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 5.1 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．

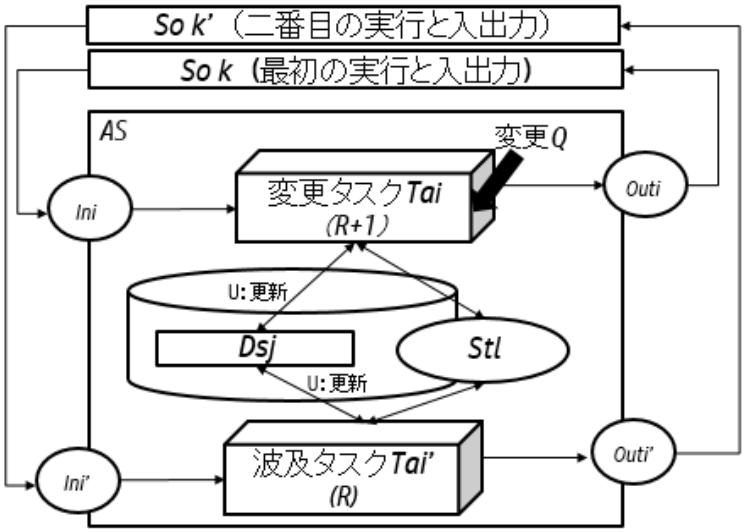


図 5.1: 変更タスクと変更波及

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 図面群をベース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 [93][94][95]．Ta のデータフローを

ベースに変更波及を詳細に解析した研究がある [96]．データベースなど保有データをベースに変更波及解析を行う研究も行われている [97][98]．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が見当たらないのはそのためだと推測する．

5.1.3 順序組み合わせテスト

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，上位の制御フロー上での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ケースを抽出することになる．このテストケースの抽出手法を順序組み合わせテストと呼ぶ．

順序組み合わせテストでのデータフローを決定するのは，関係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．

5.1.4 波及全使用法

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 [99] [100]. 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．

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 [101]．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 (Impact Data All Used:IDAU) とする．

5.2 順序組み合わせによるテストケース抽出法

本節では，順序組み合わせテストについて，入力情報となるテストベースとテストケース抽出時のルールと手順を述べる．

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 [102]．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 [102]．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 [103]．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 順序組み合わせテストの実施手順

提案する手法は，2 タスク間の順序組み合わせを対象とする．2 タスク間の順序組み合わ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

  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，順序組み合わせの変更タスク群 $P\{Ta\}$ 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 : 順序組み合わ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 $S\{Ta\}$ を組み合わせ，順序組み合わせのテストケースとする．

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						

以降からは，順序組み合わ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：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．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，または組み合わせか空白である．源泉に対する情報は In か Out，または組み合わせか空白である．空白は関係が無いことを示す．

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

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		

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 かその組み合わせかを記入する．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：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．

1. データストアを介した波及タスク特定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 $P\{Ds\}$ からの入力エッジを持つタスクが波及タスク $S\{Ta\}$ の候補となる．波及タスク $S\{Ta\}$ として選択するタスクは表 5.3 に示す表の○印の組み合わせに該当するタスクである．波及タスクは，C : 生成の場合は変更タスクが D の場合のみ選択可能になる．R : 参照，U : 更新，D : 削除を選択する場合は，変更タスクが C か U の場合である．”-”をつ

表 5.3: タスク間のデータ共有の組み合わせパターン

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

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

けた組み合わせは，データストアを介した影響が生じないため，組み合わせテストの対象としない．”×”をつけた組み合わせは仕様上ありえない組み合わせであり，ありえないことの確認は，順序組み合わせを網羅しなくともよいため，組み合わせテストの対象としない．

2. 拡張 CRUD 図の完成波及タスク候補のうち，源泉への出力エッジを持つタスクを波及タスク $S\{Ta\}$ として特定する．波及タスク $S\{Ta\}$ の特性を DFD より読み取り，特定する．特定した波及タスク $S\{Ta\}$ 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 5.5: 順序組み合わ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	順序組み合わせ
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	$Ta_3C \rightarrow Ta_2U$

5.2.5 ルール 3：順序組み合わせテストケースの抽出

1. 変更タスクと波及タスクの組み合わせを抽出

   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み合わせ

   操作の組み合わせとは変更タスクと波及タスクの操作の組み合わせである．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．組み合わせは $C \rightarrow R$ 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 の例における全組み合わせは，$Ta_1C \rightarrow Ta_2R$，$Ta_1U \rightarrow Ta_2R$，$Ta_3C \rightarrow Ta_2R$，$Ta_3C \rightarrow Ta_2U$ 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み合わ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，当該組み合わ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，順序組み合わ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 [37].

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，順序組み合わ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．

＜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すべての航空会社の飛行機予約が可能である．
・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 5.2: フライト予約システムの概要

題材となるフライト予約システムの仕様は，3 章での実験にて演習題材として使っているものである．本論文では，表 5.6 の新規フライト予約を，変更が入ったフィーチャセットとする．

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 5.3），CRUD 図（表 5.7）

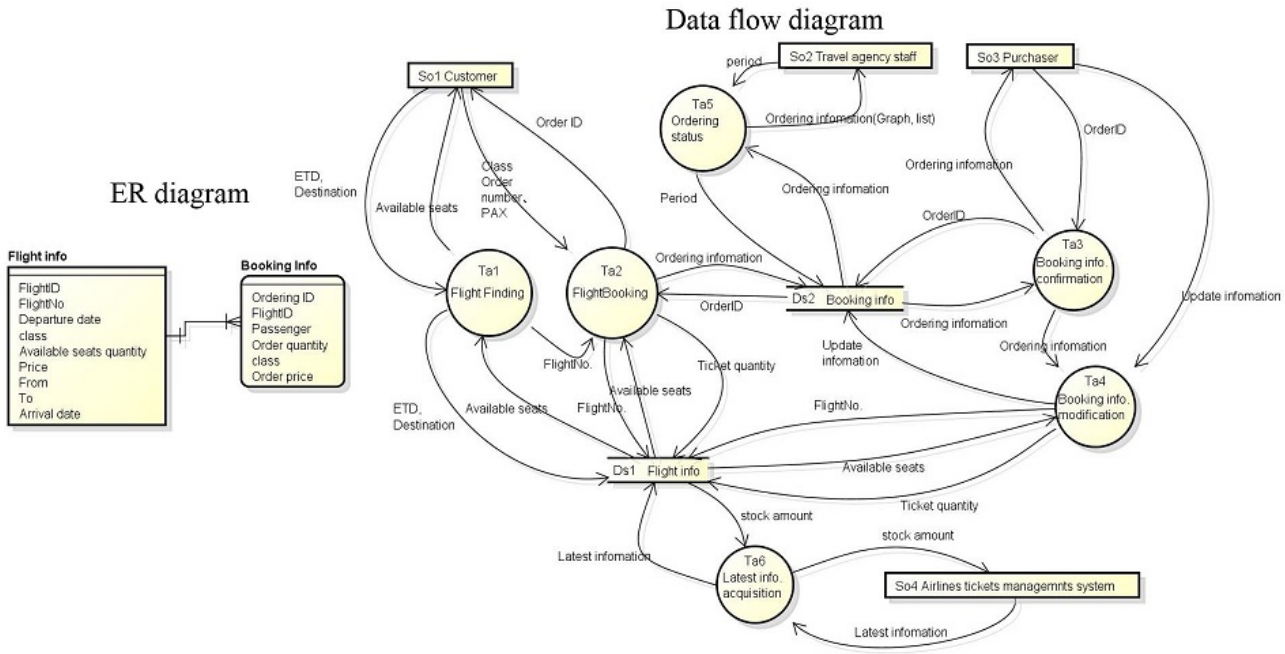


図 5.3: 新規フライト予約のデータ設計（一部分）

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．

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 5.3 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

表 5.6: フライト予約システムのフィーチャセット一覧

テストアイテム	フィーチャセット
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

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 5.3 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 5.3 では，すべてのタスクが 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 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○がつく組み合わせに相当する 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 以外はすべてルール 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 順序組み合わ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み合わせを抽出する．抽出した変更タスクと波及タスクの組み合わせに対して，データストアに対す

表 5.7: フライト予約システムの CRUD 図

フィーチャセット	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			

る操作を明記したものは以下のとおりとなる.

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

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

- $Ta_2/Ds_2C \underset{Ds_2}{\longrightarrow} Ta_5R$

これらの変更タスクと波及タスクの操作の順序組み合わせがテストケースとなる．抽出した順序組み合わ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 順序組み合わ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み合わ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である図 5.4 を使って, 順序組み合わせが確認できる網羅ベース準である S1 網羅基準を適用する．図 5.4 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。この表の提案手法の列には，順序組み合わせテストで抽出した表 5.10 のテストケース No 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

表 5.10: 順序組み合わ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	順序組み合わせ
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	$Ta_2/Ds_2C \xrightarrow[Ds_2]{} Ta_5R$

ち，対応する提案手法で抽出した順序組み合わせは，表 5.10 のテストケース No.1，2，4，5 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．

順序組み合わ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．たとえば，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 $Ta_2/Ds_1U \xrightarrow[Ds_1]{} Ta_6U$ 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，変更が入ったフィーチャ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，新規フライト予約とは別のフィーチャ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フィーチャセットのサ

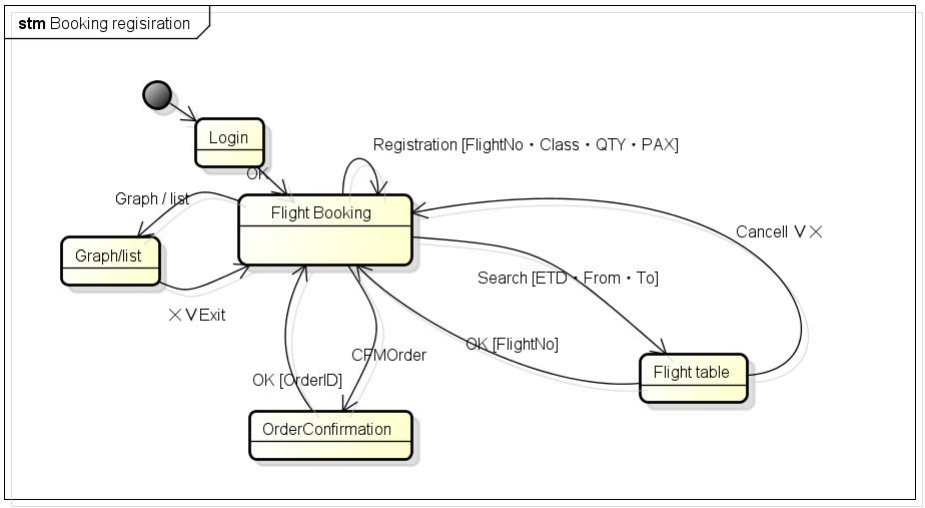


図 5.4: フライト予約システムの画面遷移図（新規フライト予約）

イズを大きくする必要があり，そのフィーチャ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 まとめ

本章では，状態遷移を持つアプリケーション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，その網羅基準としてIDAU法を提案し，変更タスクと波及タスクの順序組み合わせのテストケースを抽出する手法として順序組み合わせテストを提案した．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み合わせを抽出できることを説明した．

提案した手法で順序組み合わ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，適用評価を行った．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．

表 5.11: フライト予約登録の画面遷移図の S1 パス一覧

No	状態	イベント	状態	イベント	状態	提案手法
1	Flight Booking	Registration [FlightNo・Class・QTY・PAX]	Flight Booking	Registration [FlightNo・Class・QTY・PAX]	Flight Booking	2
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	Order Confirmation	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	1
7	Flight Booking	Registration [FlightNo・Class・QTY・PAX]	Flight Booking	CFMOrder	Order Confirmation	4
8	Flight Booking	Registration [FlightNo・Class・QTY・PAX]	Flight Booking	Graph	Graph/list	5
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	Order Confirmation	
14	Flight table	OK [FlightNo]	Flight Booking	CFMOrder	Order Confirmation	
15	Flight table	Cansell ∨ ×	Flight Booking	Graph	Graph/list	
16	Flight table	OK [FlightNo]	Flight Booking	Graph	Graph/list	
17	Order Confirmation	OK [OrderID]	Flight Booking	Registration [FlightNo・Class・QTY・PAX]	Flight Booking	
18	Order Confirmation	OK [OrderID]	Flight Booking	Search [ETD･From･To]	Flight table	
19	Order Confirmation	OK [OrderID]	Flight Booking	CFMOrder	Order Confirmation	
20	Order Confirmation	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	Order Confirmation	
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	Order Confirmation	
28	Login	OK	Flight Booking	Graph	Graph/list	

第6章　結論

本研究では，ソフトウェア開発において，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として，I/O テストデータパターンと順序組合せテストを提案し，その適用評価を行った．

まず，2 章では，本研究の対象範囲を明確にするために，対象となるアプリケーションソフトウェア，テストケース設計の種類，テストレベル，テストプロセスを明記した．そして，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を研究の対象にすること，ブラックボックステストのテストケースを開発する活動の中では，テスト分析を対象にすることを述べた，研究対象の領域で起きている問題として，テスト対象を詳細化する際の分類に対する一貫性が欠如していること，また，機能間の統合に対する問題として，既存の網羅基準を適用するとテストケース数が膨大になることを述べた，これらの問題に対して，テストカテゴリベースドテストというテスト分析手法を基に研究をすすめるため，この手法の概要と，既出の実験結果を説明した．

3 章では，テスト分析における詳細化に対する一貫性がの知識を与える前の一貫性のなテスト分析結果，及びばらつきの傾向を適用後の結果との相関をより多くのデータで調べることを目的にした予備実験を行った．実験はワークショップを通じてグループ単位で 2 回，個人単位で 1 回行なった．3 回の予備実験を通して，テスト対象を詳細化するときに分類に対する一貫性が欠如していることによるテスト分析結果のばらつきを確認することができた．また，分類ルールの手法としてテストカテゴリベースドテストの知識を与えることで，テスト条件を特定できる数が増えることが確認できた．仮説として立てた「仕様書には明確に記述がないものは，テストカテゴリのようなガイドを使うことで特定が容易になる」ことが実証できる傾向になった．ただし，テストカテゴリベース

ドテストの知識を与えても期待した数のテスト条件を特定できるわけではないことが判明した．また，業務経歴 3 年未満の技術者には有効であったが，3 年以上の技術者にはあまり効果が出ないことも判明した．

4 章では，テストカテゴリベースドテストの課題を解決するために，テスト実行時のデータ入出力の要素で分類し網羅的に分析する I/O テストデータパターンを提案した．I/O テストデータパターンの適用評価のために，3 章のグループ単位の予備実験の結果を使い，テストカテゴリベースドテストにて分類したテスト条件が I/O テストデータパターンでどのように分類されるかを確認した．単一のデータ入出力である，入力調整，出力調整，貯蔵，変換に分類されるテスト条件は，I/O テストデータパターンで特定できることが確認できた．一方，サポートと相互作用に分類されるテスト条件は，I/O テストデータパターンにて分類は可能であるが，単一のデータ入出力で動作するタスクではなく，その後に動作するタスクの出力をテストするためのテスト条件を特定する必要があることを確認できた．サポートと相互作用で確認するタスクの動作するきっかけをトリガーと呼び，トリガーをテストカテゴリとしてテスト条件を特定するようにした．最後に，入手した現実の開発プロジェクトのテストケースを使い，I/O テストデータパターンで特定したテスト条件との比較を試みた．提案した I/O テストデータパターン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，複数回行われるデータの入出力の実行順序に着目した手法である順序組合せテストを提案した．このテスト手法は，テストカテゴリベースドテストにおける相互作用に分類されるトリガーの他処理への反映に対するテストケースを抽出する手法である．このテストケースは変更の波及を確認することが目的のテストケースである．また，変更の波及を確認するための順序組合せテストの網羅基準を定義し，波及全使用法（Impact Data All Used：IDAU）とした．順序組合せテストは，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い，適用可能であることを確認した．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.

謝辞

博士課程の在学中, 公私にわたって大変お世話になった筑波大学大学院システム情報工学研究科の津田和彦教授に深く感謝いたします. そして，研究の進め方から論文の作成に渡って，貴重なご助言をいただいたデバッグ工学研究所の松尾谷徹氏に深く感謝いたします. また, 本論文の執筆に当たり, 津田研究室の方々には, 日頃より研究の進め方についての貴重な示唆やご意見を頂戴いたしました. 深く感謝いたします. 論文審査を快くお引き受けいただき, 的確なアドバイスを頂戴した筑波大学大学院システム情報工学研究科の吉田健一教授，倉橋節也教授，片岸一起准教授，徳島大学大学院の泓田正雄教授に深く感謝いたします. 発表会の場などで, 様々なアドバイスやコメントを下さった筑波大学大学院システム情報工学研究科の教員の方々に, 深く感謝いたします.

なお, 本研究の実施に当たっては，NPO 法人 ASTER を通してお借りした現実のプロジェクトのテストケースを利用しました．また，コンサルティングの実務を通して開催したワークショップ，WACATE2015 夏 (http://wacate.jp/) の場で行ったワークショップの演習結果を研究に利用いたしました．WACATE では，2 日間に渡り 57 名の参加者に協力いただき実現することが出来ました．ここに感謝の意を表します.

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] 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] 清水吉男. 要求を仕様化する技術・表現する技術: 入門 + 実践：仕様が書けていますか?. 技術評論社, 2005.

[6] 清水吉男. 派生開発における母体バグに由来するバグとその対応. *http://jasst. jp/archives/jasst09e/pdf/A2. pdf*, 2009.

[7] Jerry Gao, H-SJ Tsao, and Ye Wu. *Testing and quality assurance for component-based software*. Artech House, 2003.

[8] 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）. 組込みソフトウェア開発データ白書 2015. 独立行政法人情報処理推進機構, 2015.

[9] Alain Abran, James W Moore, Pierre Bourque, Robert Dupuis, and Leonard L Tripp. *Guide to the software engineering body of knowledge: 2004 version SWEBOK*. IEEE Computer Society, 2004.

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

[11] Gregg Rothermel, Mary Jean Harrold, Jeffery Von Ronne, and Christie Hong. Empirical studies of test-suite reduction. *Software Testing, Verification and Reliability*, Vol. 12, No. 4, pp. 219–249, 2002.

[12] Tohru Matsuodani and Kazuhiko Tsuda. Evaluation of debug-testing efficiency by duplication of the detected fault and delay time of repair. *Information Sciences*, Vol. 166, No. 1, pp. 83–103, 2004.

[13] Armin Beer and Rudolf Ramler. The role of experience in software testing practice. In *Software Engineering and Advanced Applications, 2008. SEAA'08. 34th Euromicro Conference*, pp. 258–265. IEEE, 2008.

[14] Gerald M Weinberg. *Perfect software: And other Illusions about testing*. Dorset House Publishing Co., Inc., 2008.

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

[16] Antonia Bertolino. Software testing research: Achievements, challenges, dreams. In *2007 Future of Software Engineering*, pp. 85–103. IEEE Computer Society, 2007.

[17] Mika V Mäntylä and Juha Itkonen. More testers–the effect of crowd size and time restriction in software testing. *Information and Software Technology*, Vol. 55, No. 6, pp. 986–1003, 2013.

[18] Mika V Mäntylä, Kai Petersen, Timo OA Lehtinen, and Casper Lassenius. Time pressure: a controlled experiment of test case development and requirements review. In *Proceedings of the 36th International Conference on Software Engineering*, pp. 83–94. ACM, 2014.

[19] Cem Kaner, James Bach, and Bret Pettichord. *Lessons learned in software testing*. John Wiley & Sons, 2008.

[20] Tsuyoshi Yumoto, Tohru Matsuodani, and Kazuhiko Tsuda. Analysing test basis and deriving test cases based on data design documents. In *Software Testing, Verification and Validation Workshops (ICSTW), 2017 IEEE International Conference on*, pp. 281–288. IEEE, 2017.

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

[22] 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)).

[23] 高橋寿一. 知識ゼロから学ぶソフトウェアテスト. 翔泳社, 2005.

[24] William E Lewis. *Software testing and continuous quality improvement*. CRC press, 2016.

[25] Paul Ammann and Jeff Offutt. *Introduction to software testing*. Cambridge University Press, 2016.

[26] Lee Copeland. *A practitioner's guide to software test design*. Artech House, 2004.

[27] Paul C Jorgensen. *Software testing: a craftsman's approach*. CRC press, 2016.

[28] Robert V Binder. *Testing object-oriented systems: models, patterns, and tools*. Addison-Wesley Professional, 2000.

[29] Cem Kaner and Jack Falk. *Testing computer software*. Wiley, 1999.

[30] Rex Black. *Pragmatic software testing: Becoming an effective and efficient test professional*. John Wiley & Sons, 2007.

[31] T. J. Ostrand and M. J. Balcer. The category-partition method for specifying and generating fuctional tests. *Commun. ACM*, Vol. 31, No. 6, pp. 676–686, 1988.

[32] Mats Grindal and Jeff Offutt. Input parameter modeling for combination strategies. In *Proceedings of the 25th Conference on IASTED International Multi-Conference: Software Engineering*, SE'07, pp. 255–260, Anaheim, CA, USA, 2007. ACTA Press.

[33] Michal Young. *Software testing and analysis: process, principles, and techniques*. John Wiley & Sons, 2008.

[34] Kevin Forsberg and Harold Mooz. The relationship of system engineering to the project cycle (pdf). In *Chattanooga, Tennessee: Proceedings of the National Council for Systems Engineering (NCOSE) Conference*, pp. 57–65, 1995.

[35] Roger S Pressman. *Software engineering: a practitioner's approach*. Palgrave Macmillan, 2005.

[36] 高橋寿一, 湯本剛. ソフトウェアテスト手法: 現場の仕事がバリバリ進む. 技術評論社, 2006.

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

[38] Phil Stocks and David Carrington. A framework for specification-based testing. *IEEE Transactions on software Engineering*, Vol. 22, No. 11, pp. 777–793, 1996.

[39] Rick D Craig and P Jaskiel Stefan. Systematic software testing, artech house. *Inc., Norwood, MA*, 2002.

[40] IEEE. Std 610.12-1990 (r2002, ieee standard glossary of software engineering terminology. Technical report, IEEE, 1990.

[41] 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.

[42] Sigrid Eldh, Hans Hansson, and Sasikumar Punnekkat. Analysis of mistakes as a method to improve test case design. In *Software Testing, Verification and Validation (ICST), 2011 IEEE Fourth International Conference on*, pp. 70–79. IEEE, 2011.

[43] 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.

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

[45] 大林英晶, 森崎修司, 渥美紀寿, 山本修一郎ほか. 逸脱分析を用いた要求仕様書からのテスト項目抽出手法. 情報処理学会論文誌, Vol. 57, No. 4, pp. 1262–1273, 2016.

[46] Noriyuki Mizuno, Makoto Nakakuki, and Yoshinori Seino. Test conglomeration-proposal for test design notation like class diagram. In *Software Testing, Verification and Validation Workshops (ICSTW), 2017 IEEE International Conference on*, pp. 309–312. IEEE, 2017.

[47] Lionel Briand and Yvan Labiche. A uml-based approach to system testing. *Software and Systems Modeling*, Vol. 1, No. 1, pp. 10–42, 2002.

[48] Victor Basili and Sebastian Elbaum. Empirically driven se research: State of the art and required maturity. In *Proceedings of the 28th International Conference on Software Engineering*, ICSE '06, pp. 32–32, New York, NY, USA, 2006. ACM.

[49] Juha Itkonen, Mika V Mantyla, and Casper Lassenius. How do testers do it? an exploratory study on manual testing practices. In *Proceedings of the 2009 3rd International Symposium on Empirical Software Engineering and Measurement*, pp. 494–497. IEEE Computer Society, 2009.

[50] John Rooksby, Mark Rouncefield, and Ian Sommerville. Testing in the wild: The social and organisational dimensions of real world practice. *Computer Supported Cooperative Work (CSCW)*, Vol. 18, No. 5-6, p. 559, 2009.

[51] Paul Ammann and Jeff Offutt. Using formal methods to derive test frames in category-partition testing. In *Computer Assurance, 1994. COMPASS'94 Safety, Reliability, Fault Tolerance, Concurrency and Real Time, Security. Proceedings of the Ninth Annual Conference on*, pp. 69–79. IEEE, 1994.

[52] Matthias Grochtmann and Klaus Grimm. Classification trees for partition testing. *Software Testing, Verification and Reliability*, Vol. 3, No. 2, pp. 63–82, 1993.

[53] Richard A DeMillo, Richard J Lipton, and Frederick G Sayward. Hints on test data selection: Help for the practicing programmer. *Computer*, Vol. 11, No. 4, pp. 34–41, 1978.

[54] Eckard Lehmann and Joachim Wegener. Test case design by means of the cte xl. In *Proceedings of the 8th European International Conference on Software Testing, Analysis & Review (EuroSTAR 2000), Kopenhagen, Denmark*, 2000.

[55] Satoshi Masuda, Futoshi Iwama, Nobuhiro Hosokawa, Tohru Matsuodani, and Kazuhiko Tsuda. Semantic analysis technique of logics retrieval for software testing from specification documents. In *Software Testing, Verification and Validation Workshops (ICSTW), 2015 IEEE Eighth International Conference on*, pp. 1–6. IEEE, 2015.

[56] Satoshi Masuda, Tohru Matsuodani, and Kazuhiko Tsuda. Detecting logical inconsistencies by clustering technique in natural language requirements. *IEICE TRANSACTIONS on Information and Systems*, Vol. 99, No. 9, pp. 2210–2218, 2016.

[57] Keiji Uetsuki, Tohru Matsuodani, and Kazuhiko Tsuda. An efficient software testing method by decision table verification. *International Journal of Computer Applications in Technology*, Vol. 46, No. 1, pp. 54–64, 2013.

[58] Keiji Uetsuki and Mitsuru Yamamoto. Improvement of description for reusable test type by using test frame. In *Software Testing, Verification and Validation Workshops (ICSTW), 2017 IEEE International Conference on*, pp. 289–293. IEEE, 2017.

[59] Keiji Uetsuki, Tohru Matsuodani, and Kazuhiko Tsuda. Software logical structure verification method by modeling implemented specification. *Knowledge-Based and Intelligent Information and Engineering Systems*, pp. 336–345, 2011.

[60] Keiji Uetsuki, Tohru Matsuodani, Masakazu Takahashi, and Kazuhiko Tsuda. Decision table expansion method for software testing. In *KES*, pp. 885–892, 2012.

[61] Jaffar-ur Rehman, Fakhra Jabeen, Antonia Bertolino, Andrea Polini, et al. Testing software components for integration: a survey of issues and techniques. *Software Testing, Verification and Reliability*, Vol. 17, No. 2, pp. 95–133, 2007.

[62] Gregg Rothermel, Roland H. Untch, Chengyun Chu, and Mary Jean Harrold. Prioritizing test cases for regression testing. *IEEE Transactions on software engineering*, Vol. 27, No. 10, pp. 929–948, 2001.

[63] Sebastian Elbaum, Alexey G Malishevsky, and Gregg Rothermel. *Prioritizing test cases for regression testing*, Vol. 25. ACM, 2000.

[64] James A Whittaker and Michael G Thomason. A markov chain model for statistical software testing. *IEEE Transactions on Software engineering*, Vol. 20, No. 10, pp. 812–824, 1994.

[65] David Lee and Mihalis Yannakakis. Principles and methods of testing finite state machines-a survey. *Proceedings of the IEEE*, Vol. 84, No. 8, pp. 1090–1123, 1996.

[66] Susumu Fujiwara, G v Bochmann, Ferhat Khendek, Mokhtar Amalou, and Abderrazak Ghedamsi. Test selection based on finite state models. *IEEE Transactions on software engineering*, Vol. 17, No. 6, pp. 591–603, 1991.

[67] Anneliese A Andrews, Jeff Offutt, and Roger T Alexander. Testing web applications by modeling with fsms. *Software and Systems Modeling*, Vol. 4, No. 3, pp. 326–345, 2005.

[68] Tsun S. Chow. Testing software design modeled by finite-state machines. *IEEE transactions on software engineering*, No. 3, pp. 178–187, 1978.

[69] Tomohiko Takagi, Naoya Oyaizu, and Zengo Furukawa. Concurrent n-switch coverage criterion for generating test cases from place/transition nets. In *Computer and Information Science (ICIS), 2010 IEEE/ACIS 9th International Conference on*, pp. 782–787. IEEE, 2010.

[70] 吉澤正孝, 秋山浩一, 仙石太郎. ソフトウェアテスト HAYST 法入門. 日科技連出版社, 2007.

[71] 秋山浩一, 高木智彦, 古川善吾. 直交表を用いたソフトウェアテストにおける効果的な因子選択・割り付け手法. 品質, Vol. 42, No. 4, pp. 567–576, 2012.

[72] Jean Hartmann, Claudio Imoberdorf, and Michael Meisinger. Uml-based integration testing. In *ACM SIGSOFT Software Engineering Notes*, Vol. 25, pp. 60–70. ACM, 2000.

[73] 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.

[74] Kyo C Kang, Sholom G Cohen, James A Hess, William E Novak, and A Spencer Peterson. Feature-oriented domain analysis (foda) feasibility study. Technical report, Carnegie-Mellon Univ Pittsburgh Pa Software Engineering Inst, 1990.

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

[76] 大村平. 一般システムの現象学. 技報堂出版, 2005.

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

[78] 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.

[79] 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.

[80] Tsuyoshi Yumoto, Tohru Matsuodani, and Kazuhiko Tsuda. A study on the effectiveness of test-categories based test analysis. In *Software Testing, Verification and Validation Workshops (ICSTW), 2016 IEEE Ninth International Conference on*, pp. 7–13. IEEE, 2016.

[81] 湯本剛, 松尾谷徹, 津田和彦ほか. データ i/o パターンに着目したテスト分析手法の提案. 第 77 回全国大会講演論文集, Vol. 2015, No. 1, pp. 199–200, 2015.

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

[83] Tsuyoshi Yumoto, Tohru Matsuodani, and Kazuhiko Tsuda. A study on an approach for analysing test basis using i/o test data patterns. In *Software Testing, Verification and Validation Workshops (ICSTW), 2015 IEEE Eighth International Conference on*, pp. 1–8. IEEE, 2015.

[84] Sheikh Umar Farooq and SMK Quadri. Empirical evaluation of software testing techniques-need, issues, and mitigation. *Software engineering: an international Journal*, Vol. 3, No. 3, pp. 41–51, 2013.

[85] Sebastian Elbaum, Alexey G Malishevsky, and Gregg Rothermel. Test case prioritization: A family of empirical studies. *IEEE transactions on software engineering*, Vol. 28, No. 2, pp. 159–182, 2002.

[86] 湯本剛, 植月啓次, 松尾谷徹, 津田和彦. 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. 電気学会論文誌 C (電子・情報・システム部門誌), Vol. 137, No. 7, pp. 987–994, 2017.

[87] Monica Hutchins, Herb Foster, Tarak Goradia, and Thomas Ostrand. Experiments on the effectiveness of dataflow-and control-flow-based test adequacy criteria. In *Software Engineering, 1994. Proceedings. ICSE-16., 16th International Conference on*, pp. 191–200. IEEE, 1994.

[88] Sandra Rapps and Elaine J Weyuker. Data flow analysis techniques for test data selection. In *Proceedings of the 6th international conference on Software engineering*, pp. 272–278. IEEE Computer Society Press, 1982.

[89] Sandra Rapps and Elaine J. Weyuker. Selecting software test data using data flow information. *IEEE transactions on software engineering*, No. 4, pp. 367–375, 1985.

[90] 湯本剛, 松尾谷徹, 津田和彦. 変更における状態を含むテスト網羅尺度とテストケース抽出法の提案. ソフトウェアシンポジウム 2017, Vol. 2017, No. 1, 2017.

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

[92] 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.

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

[94] 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.

[95] 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.

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

[97] 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.

[98] 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.

[99] Lloyd D Fosdick and Leon J Osterweil. Data flow analysis in software reliability. *ACM Computing Surveys (CSUR)*, Vol. 8, No. 3, pp. 305–330, 1976.

[100] Hasan Ural. Test sequence selection based on static data flow analysis. *Computer communications*, Vol. 10, No. 5, pp. 234–242, 1987.

[101] Boris Beizer. *Black-box testing: techniques for functional testing of software and systems*. John Wiley & Sons, Inc., 1995.

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

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

関連業績リスト

参考論文

[1] Tsuyoshi Yumoto, Tohru Matsuodani, and Kazuhiko Tsuda. ”A study on an approach for analysing test basis using I/O test data patterns.” 2015 IEEE Eighth International Conference on Software Testing, Verification and Validation Workshops (ICSTW), 2015, pp1-8.

[2] Tsuyoshi Yumoto, Tohru Matsuodani and Kazuhiko Tsuda. ”A Study on the effectiveness of Test-Categories based test analysis.” 2016 IEEE Ninth International Conference on Software Testing, Verification and Validation Workshops (ICSTW), 2016, pp7-13.

[3] 湯本剛, 植月啓次, 松尾谷徹, 津田和彦,「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」, 電気学会論文誌 C (電子・情報・システム部門誌), 137(7), 2017, pp.987-994.

その他の論文

・査読のない発表論文

[1] 湯本剛, 松尾谷徹, 津田和彦,「データ I/O パターンに着目したテスト分析手法の提案.」, 第 77 回全国大会講演論文集 2015.1 , 2015 , pp199-200.