第5分科会

# バグの流出防止を考える －どんなテストをすればバグを見つけられたのか？－

# Meditating on the prevention of the outflow of a bug —By what kind of test could we find the bug?—


| | | |
|---|---|---|
| 主査 | 奥村 有紀子 | （有限会社デバッグ工学研究所） |
| 副主査 | 秋山 浩一 | （富士ゼロックス株式会社） |
| 副主査 | 堀田 文明 | （有限会社デバッグ工学研究所） |
| 研究員 | 清水 剛史 | （株式会社ユニケソフトウェアリサーチ） |
| | 堀川 彬夫 | （富士フイルム株式会社） |
| | 鈴木 義昭 | （伊藤忠テクノソリューションズ株式会社） |
| | 山本 愛美 | （キヤノンソフトウェア株式会社） |
| | 光田 貴志 | （オムロン株式会社） |



**概要**

昨今のソフトウェアの開発において，納期やコストに対する市場要求が高まっている．一方で，信頼性や安全性の確保が不可欠となっている．しかし，ソフトウェアにバグが混入され，市場へバグを流出してしまうケースが後を絶たない．過去の失敗を教訓とし，バグの再発防止を図ることが重要である．

本論文では，バグ流出に焦点を当て，流出原因分析と再発防止に有効なテスト技法選定を実施するためのテスト観点－テスト技法対応表を作成した．この表では，多種多様であるテスト技法をバグ原因別に整理し，テスト技法の解説を適用ケースとともに参照できる形式となっている．バグ発見時にテスト観点－テスト技法対応表を利用し，適切なテスト技法を取り入れることで，テストプロセスを強化できる．

**Agenda**

In today's software development, the demand for shorter delivery times and costs are increasing. On the other hand, it is essential to ensure reliability and safety. However, bugs cannot remove thoroughly from the software in the development process, system troubles has occurred repeatedly. We have to learn from the past failure and plan to prevent the recurrence of a bug.

In this paper, we have analyzed the bugs which outflow to the customer, and made the bug-leakage-preventing matrix for selecting testing technique that is effective to find out outflow bug.

In this matrix we arranged various testing techniques with explanation of usage and the example of failure, according to the cause of a bug. You can strengthen testing process by using this tree on cause analysis of the bug-outflow and selection of the suitable testing technique.


## 1. 背景

昨今のソフトウェアの開発において，納期やコストに対する市場要求が高まっているものの，ソフトウェアにバグが混入され，市場に出てから不具合が発見されるケースが後を絶たない．過去の失敗を教訓とし，市場バグの再発防止を図ることが重要である[1].

一方で，本論文の検討メンバが抱える課題[表 1]を洗い出すと，各々が異なる課題を持っているも

のの，ディスカッションを進めていく中で，市場で顕在化されるバグの発生を防ぐこと，すなわち，「バグ流出の防止策の検討」が共通の課題であることが判明した．

**表 1：検討メンバの課題**

| 検討メンバ | 課題 |
|---|---|
| A | バグ分析とプロセス改善 |
| B | 数値的なバグの傾向分析 |
| B | 組織的な改善活動 |
| C | バグと技法の関係の整理 |
| D | 出荷後のバグ減少施策 |
| D | バグの再発防止と新規混入の防止 |
| E | 状態遷移に関するバグの防止 |

本論文では，流出原因の分析を実施し，再発防止のために何をすべきかを検討する．バグ流出に焦点を当てて，どのようなテストをすれば市場出荷前にバグを摘出できたのかを考察し，再発防止に有効な手段を検討する．

## 2. 研究課題

### 2.1. バグ流出防止を考える

#### 2.1.1. 問題解決のアプローチ

バグを防ぐためには，流出しないように対策する，あるいは，混入しないように対策する，と2つのアプローチが考えられる．前者は，テスト工程で実施する対策であり，後者は，仕様・設計・実装工程で実施する対策である．本論文ではテスト工程での対策「流出を防止する対策」を検討した．

また，バグ流出の再発を防止するためには，バグを発見する毎にチェックリストを作成する等の対策が考えられる．しかしながら，この対策は，特定の製品・仕様に対する対策であることから，バグの流出防止に対して個別の対策となってしまう傾向があり，対策を横展開することが難しい．本論文では，広く横展開可能な対策を講じることに主眼を置き，なぜテスト工程でバグを抽出できなかったのか根本的な原因を分析することとした．

#### 2.1.2. 流出原因の分析

本論文の検討メンバで流出原因についてディスカッションし，マインドマップ[図1]を利用して分析，議論を整理した．議論は，

1．バグが顕在化した条件・環境は何か，どんなテストを実施すれば出荷前に摘出できたのか，（テストケース）
2．そのテストをするために必要なテスト観点は何であったか（テスト観点），といった手順で真因を深掘りしていく形で進めた．
3．さらにその観点を利用して，そのバグが発見できるテストケースを設計するテスト技法を見つけることにした．

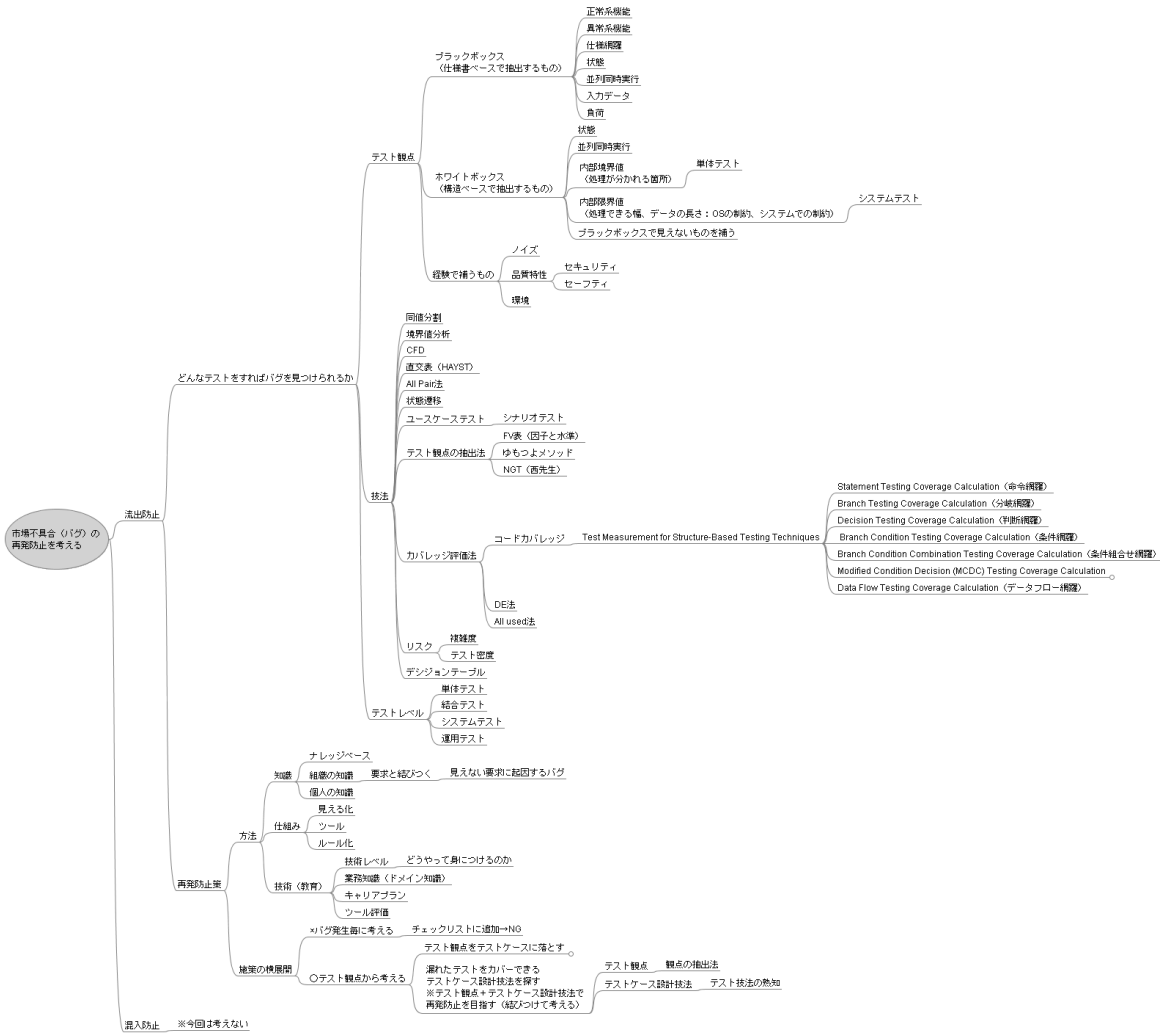


**図1：バグ流出原因防止マインドマップ**

ここで，テスト観点の定義について，その観点を使用するテストレベルごとに整理した[表 2]. 観点は仕様ベースを基本にし，入力や環境条件など例えば同値分割対象とすべき事項の種類，又はその組み合わせ方である．さらに構造ベースは仕様ベースの補完の位置付とし，データの定義参照関係などを観点とした．なお，非機能テストに関係する観点は除外している．

**表 2：仕様ベースと構造ベースで考えたテスト観点表**

| | 仕様ベースのテスト観点 | 構造ベースのテスト観点 |
|---|---|---|
| 単体テストレベル | 入力，操作，読込，表示，印刷，書込み，演算，変換，判断，状態等の処理条件 | ステートメントの流れ, 分岐, 繰り返し，獲得・返却，オープン・クローズ, |
| | 入力，操作，その他の処理条件等の組合せ | データ・変数の定義・参照，(演算，判定)，更新/再定義，消去関係性 |
| 結合テストレベル | 処理のつながりにおける単体レベルの各処理間の整合性 | 処理のつながりにおける単体レベルの各処理間の整合性 |
| | | 関数・サブルーチンへの依頼・回答 |
| システムテストレベル | 処理の組合せ，処理の順序，データの組合せ | 並行処理における処理間の同期，排他 |

### 2.1.3. バグとテスト技法の関係

テストケース設計にどのテスト技法を利用するかについては，その技法がカバーしている網羅の視点，又は狙いとしているバグ種別に着目することにした．

例えば，入力と環境の組合せについては，仕様ベースのテスト観点であるが，組合せ範囲の広がりに応じて分類することとした．単体テストレベルであれば同値分割と組合せ，結合テストレベルであればCFD，デシジョンテーブル，組み合わせる必然性が低ければAll Pairといった具合である．

テストケースを設計のためにテスト技法を導入する際には3つの条件が必要となる．

１．バグ流出がテストの問題かどうかを特定できること．

２．テストの組み立て方・目的・構造（単体・結合・システムテスト）に問題がなかったのかを確認し，テストが大きな観点で漏れているのか，細かい箇所で漏れているのかを特定できること．

３．技術的な背景としてテスト技法を知っていること．

以上の条件が揃っており，体系的な方法でテストケースを設計しているのかを検証することが出来なければ，テスト工程でのバグ流出防止の対策としてテスト技法の導入に関する分析は進めることはできない．

分析に当たっては，テスト技法に関する知識が前提となるため，テスト技法の特徴と使い方をまとめた[付録２参照]．次に，各テストレベルで摘出されるバグの例とそれを摘出しうるテスト技法の使い方の例を検討した[表 3]．

**表 3：テスト観点－テスト技法対応表**

| | 仕様ベーステスト漏れによるバグのタイプ例 | 構造ベーステスト漏れによるバグのタイプ例 | テスト技法を使用したテスト方法 |
|---|---|---|---|
| 単体テスト | 処理条件と結果の関係が仕様通りでない，<br>0 での除算，限界値越え処理でのメモリリークなど | 変数誤使用，誤消去，領域解放漏れなど | 仕様ベーステストとして同値分割，境界値分析を実施，処理・環境条件の組合せを行いテストケースを設計する．<br>仕様ベーステスト実施時に網羅性を測定し，漏れたパスのテストケースを追加する． |
| 結合テスト | 処理間/ブロック間の制御やデータ（インターフェース）不整合 | 処理，ブロック間の領域獲得・解放不整合 | 仕様と構造の両方の情報から処理（又はブロック）の流れを分析して，処理（又はブロック）間を繋ぐテストケースを設計する．仕様ベースでは見えない流は構造ベースから見つけ出す．<br>CFD，デシジョンテーブル，データの定義，参照，更新等のパス分析． |
| システムテスト | 処理順序の変化に対する機能漏れ．関係のなさそうなパラメータ，操作等の組合せに起因するバグ | TASK の処理遅延．データの二重更新，又は更新無効．デッドロック | リスク視点でテスト方式を選ぶ．All Pair，直交表，シナリオテスト，ツールを利用したタイミングテスト． |

多種多様であるバグ原因を流出防止可能なテスト技法別に整理し，テスト技法の解説と合わせて参照することで，バグ流出防止のツールとして利用できる．表3を元にして，適切なテスト技法を取り入れることにより，特定のバグだけでなく，類似するバグの流出を防止することができる等，テストプロセスの強化が可能となる．次章では，テスト技法の概要，及び，検出可能なバグ例を説明する．

## 3. テスト技法の整理

テストレベルごとのバグ例とそれを防止するためのテスト技法使用法について整理した概要を説明する（詳細な説明については付録2を参照）.

### 3.1. 単体テスト

#### 3.1.1. 仕様ベース

**【同値分割法】[1]**

テスト実施において闇雲に入力値を選択した場合，多数のテストケースを相当な時間を費やして実施したとしても，特定の同値クラスに関するテストが抜けてしまう可能性がある．同値分割法の考え方に従い，同値分割された全ての同値クラスに対してテストケースが設けられていることを確認することで，テストケースの仕様に対する網羅度を高め，効率的なテストを設計することができる.

また，同値クラスは，正常系の同値クラスである有効同値クラスと，異常系の同値クラスである無効同値クラスに分けて考えられる．仕様漏れになりやすい無効同値クラスのテストを設計することで“仕様漏れによる不具合”を検出することができる.

**【境界値分析法】[1][2]**

同値分割後，テストケースに採用する代表値を選ぶにあたり，単に同値クラスの中央値等を採用するよりも，境界値を採用することで，バグが混入しやすいプログラム中の条件判定処理が正しく動作しているかを確認することができる.

**表 4：単体テスト・仕様ベースで摘出可能なバグ**

| テストレベル | 技法 | 具体的な例（現象） |
|---|---|---|
| 単体テスト | 同値分割 | 異常終了，<br>機能実装漏れ，例外処理実装漏れ |
| | 境界値分析 | メモリリーク<br>数の境界誤り（例：100 未満を 100 以下と誤実装）<br>文字列の境界誤り（例：許容文字コードの範囲ミス） |

#### 3.1.2. 構造ベース

**【カバレッジ評価法】**

テスト対象の内部構造に着目し，内部構造の種類（制御フローやデータフロー等）の網羅率を評価する手法. ソフトウェアの論理構造に依存する障害（異常終了, エンドレスループ）を検出できる[3]. カバレッジの定義により，①コードカバレッジ，②MC/DC法（Modified Condition / Decision Coverage），③All Uses法の各評価法が定義されている.

### 3.2. 結合テスト

#### 3.2.1. 仕様ベース

**【CFD法：Cause Flow Diagram】**

図 2の場合，左の原因１から右に向かって処理が流れる．CFD法を用いて，内部構造に基づいた流れ図を集合で表すことで，仕様書では見えづらいことの多い補集合の部分に関してもテストの視点が届き，バグの流出防止につながる．CFDは仕様ベースの技法ではあるが，構造の情報も利用する[2].

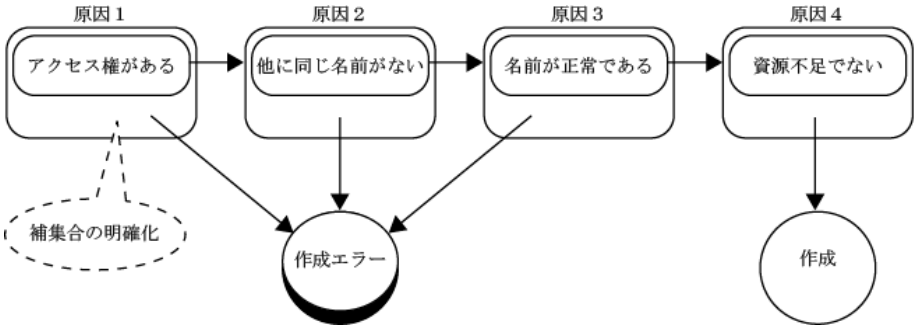


**図 2：CFD 法の流れ図**

**【状態遷移】**

状態遷移テストは，仕様の不備や誤りを検出することにも役立つ．テストすべき状態，イベント，遷移を表現することで，対象機能の全体像が見えてくる．ただし，状態やイベントが増えると複雑になり，テスト項目が増えすぎてしまう．その際には，2項間網羅や直交表を用いて項目を削減するなどが有効である[4][5].

**表 5：結合テスト・仕様ベースで摘出可能なバグ**

| テストレベル | 技法 | 具体的な例（現象） |
|---|---|---|
| 結合テスト | CFD | 処理の流れの誤り，異常系の処理 |
| | 状態遷移 | 状態の考慮漏れ，状態が異なる場合の処理の変化，<br>変数と機器状態の不一致 |

### 3.2.2. 構造ベース

3.1.2と同じ

## 3.3. システムテスト

### 3.3.1. 仕様ベース

**【HAYST法：Highly Accelerated and Yield Software Testing】**

組合せテスト技法の一つ．ソフトウェアのバグの原因は，一つの因子によるもの，および二つの因子の組合せによるものが多いという調査事例があるが，HAYST法を利用することで予期せぬ因子の組合せで起きるバグを確実に検出できる[4].

**【All Pair法】**

組合せテスト技法の一つ．All Pair法もHAYST法同様，任意の2因子間の全水準の組合せが100%となるテストケースを作成する．HAYST法との違いは，HAYST法では2因子間の水準組み合わせが「同数回」出現するのに対して，All Pair法では「同数回」という条件を外すことでよりテストケース数を削減できる点にある[2].

**【ユースケース・シナリオテスト】**

複数機能を連携する業務やシステム運用の評価，ユーザビリティ評価，業務手順の評価に活用することで，ひとつの機能としては仕様通りの結果が得られていたとしても，機能を組み合わせる，異常・例外を組み合わせるなどで処理の漏れなどを確認することができる.

**表 6：システムテスト・仕様ベースで摘出可能なバグ**

| テストレベル | 技法 | 具体的な例（現象） |
|---|---|---|
| システムテスト | All Pair<br>HAYST 法 | 機能間の干渉（変数を誤って共有），<br>プログラムのコピー&ペーストミスによる不要な条件分岐の混入 |
| | シナリオテスト | エラー回復，<br>画面切り替え，外部からの割り込み，<br>キーボードのキーの割当が標準でない，<br>不適当な情報の出力，<br>表示レイアウト，入力スタイルの不統一，<br>パフォーマンス，<br>エラー検出，<br>データ破壊，<br>不正使用 |

## 4. 研究成果

図 3 は前年度も本分科会に参加したメンバが，テスト技法を導入してテストケースを設計したシステムのバグ件数推移である．リリース毎に多くのバグを流出させていたシステムにテスト技法を取り入れてテストケースの見直しを行った結果，前年度と比べてバグ流出件数は半数近くになっている．

しかしながら，依然として 10 件前後のバグが流出している．これはバグに対して個別の対策だけを講じ，類似したバグを見逃してしまっていることに起因していると思われる．

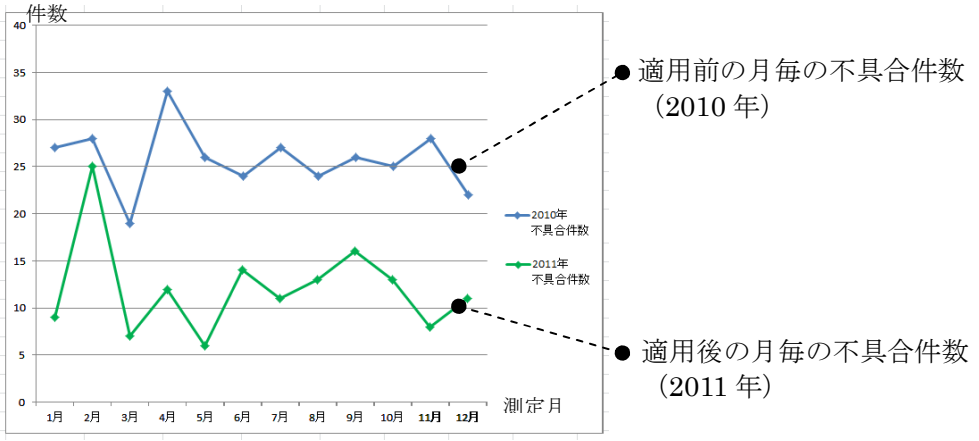


**図 3：テスト技法 適用前後の比較**

図3の結果からさらにバグの流出を防ぐための対策として，テスト技法の適用能力を高める手順を作成した．表 2，表 3を参照しながら図 4の手順でバグを分析・再発防止を検討した結果，表 7で例示する分析ができるようになった．

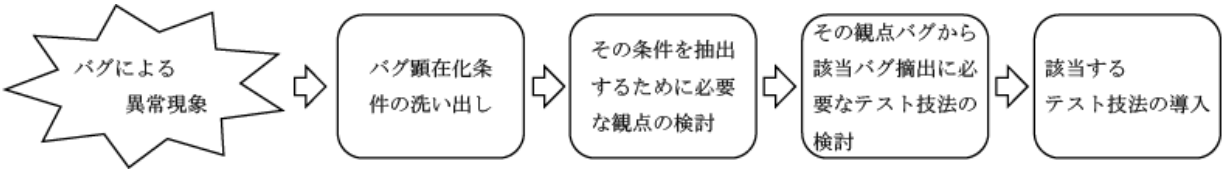


**図 4：バグ分析からテスト技法選択までの手順**

**表 7：バグの分析例**

| バグ例 | 発生契機 | テスト技法の適用 |
|---|---|---|
| 0で除算したために異常発生 | 入力値に0を指定された時に発生 | 入力値の同値分割で異常値，例外値を同値類として追加 |
| | 分母の変数が演算によってつくられる場合 | 入力値の組合せや,その引用から0が導き出される場合は入力値の完全同値分割と組合せからテストケース設計. |
| | | プログラムの実装で追加された場合は変数の定義参照関係を追跡して，該当パスをテストする．データフローテスチング |

これは一例ではあるが，テスト観点－テスト技法対応表の活用方法として以下が期待できる．

- ◆ バグ分析：テスト観点－テスト技法対応表を用いてバグをテスト観点ごとに整理することで，同様のバグだけでなく，類似バグの流出を防ぐことができる．
- ◆ テストケースの充実：開発工程で発見されるバグをテスト観点－テスト技法対応表の事象に適用することで，流出防止に効果的な技法を見出すことができる．
- ◆ テスト技法の啓蒙・教育資料：テスト観点－テスト技法対応表はテスト観点とテスト技法の視える化であり，テスト技法の初心者にもわかりやすい．付録2のテスト技法の解説などと合わせて利用することでテスト技法の啓蒙・教育資料として活用ができる．

## 5. 最後に

### 5.1. 目標の達成度合い

本研究を通じて，バグ流出防止のために，どのようなテスト技法・テスト観点を用いればよいかを検討し，適切なテスト技法の選定をするための足がかりが作成できた．

様々なテスト技法の存在を知り，理解することができたと共に，それらのテスト技法を用いてどのようなバグが検出できるかの整理ができた．また，自社での取り組みも展開し議論を交わすことで，よいアドバイスが得られ，視野が広がった．

### 5.2. 反省点

今回は，実務案件でのバグ事例を用いた検証が十分にできず，有効性の評価までは至らなかった．テスト技法に対する理解やバグ事例の効果的な利用方法などに関しても，検討の余地がある．思考ベースの解決案を実務でこなせるように技術レベルを引き上げることも必要と考える．

今後は，社内で適用し，効果を検証することが課題となる．適用事例を増やし，テスト観点－テスト技法対応表の拡大・見直しによる質向上をはかり，より役立つように充実させていきたい．

## 参考文献


[1] 太田忠雄ほか:“高信頼化ソフトウェアのための開発手法ガイドブック”, IPA（2010-9）

[2] 秋山浩一ほか:“ソフトウェアテスト技法ドリル”，日科技連出版社（2010-10）

[3] SQuBOK策定部会:“ソフトウェア品質知識体系ガイド”，オーム社（2007-11）

[4] 大西建児ほか: “JSTQB認定テスト技術者 Foundation Level試験”，翔泳社（2007-2）

[5] Rick Craigほか:“体系的ソフトウェアテスト入門”，日経BP出版センター（2004-10）

# 付録

## 付録１：観点の出し方（方法論の紹介）

テスト観点の出し方として表 8 のような手法がある．本節では，表 8 の 3 手法について説明する．

**表 8：観点の出し方**

| 観点の出し方 | 概要 | メリット |
|---|---|---|
| FV 表<br>（Function Verification Table） | 機能をあげ，その機能の検証内容と使用するテスト技法を記載する | 機能に対応した観点を表現しやすい |
| NGT<br>（Notation for Generic Testing） | ツリー状にテスト分析モデルを記載する | 観点の階層関係とその関連を検討しやすい |
| ゆもつよメソッド | 機能とテストカテゴリ（観点）から表を作成する | テストカテゴリに社内のノウハウを活用しやすい |

### 1.1. FV 表：Function Verification Table

富士ゼロックス社の秋山浩一氏の提案するテスト技法・HAYST 法でのテスト分析手法．目的機能でテストの十分性を確保する．ある機能の本来の意味を把握（目的機能）し，それが実現できたのかを確認（検証内容）し，確認方法（テスト技法）を記述することを基本フォーマットとする[図 5].

| No | 目的機能(F) | 検証内容(V) | テスト技法(T) |
|---|---|---|---|
| | | | |

**図 5：FV 表基本フォーマット**

（目的機能（F））

・機能仕様書から「機能」を取り出す（C&P 不可）．その機能が持っている目的について考える．

・USDM での要求の「理由」に当たる内容を記述する．

※USDM：Universal Specification Describing Manner,（株）システムクリエイツ清水吉男氏が提案する要求仕様の表現方法.

（検証内容（V））

・どのようにテストすればユーザ目的に合致した機能が作りこまれたかを確認できるかを記述する.

・(6W2H)で分析して考える．WHAT（機能仕様）,When（いつ使われるか）,Where（どこで使われるか）,Who（誰が使うのか）,Whom（誰のために使うのか）,How（どのように使うのか）,How much（どのくらいの価格なのか）

・マインドマップなどを使うと考え易い

（テスト技法（T））

・目的機能をテストするために使用するテスト技法を記述する

（例）

・組合せ　　：HAYST 法

・論理関係が複雑　：原因結果グラフ・CFD

・状態遷移　　：2-Switch テスト法

### 1.2. NGT：Notation for Generic Testing

電気通信大学・西氏が提案するテストのモデリングのための記法※1 であり，ツリー構造で表現される[図 6]．テストの概要設計を検討するために，テスト分析モデルとテスト設計モデルをツリー状に記述する．対象とするシステムでのテストで何をどのくらい網羅するのか決める．一方，テストの詳細設計は個々のテスト技法で記述する．そのため，どのように網羅するのかといった，個々のテスト技法で扱うものは NGT に記述しない.

※1 [参考] 西康晴, “テスト設計におけるモデリングのための記法の提案”, JaSST'06Tokyo (2006-1)

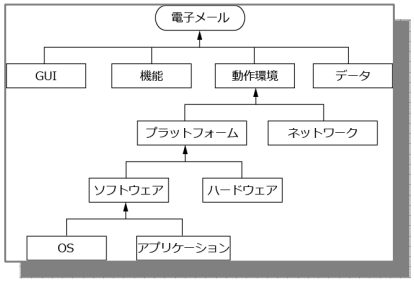


**図 6：NGT の記述例**

テスト観点は階層的に具体化していく．各レイヤーの観点は「フォーカス・クラス」と呼ばれ，詳細化されたフォーカス・クラスはテスト設計で同値クラスとなる．階層の上位のフォーカス・クラスは「ビュー」と呼ばれ，ビューの観点を「フォーカス・クラス」で表現できるように分解していく（図 7）．最終的に，各枝が MECE となるように階層的に具体化していく．

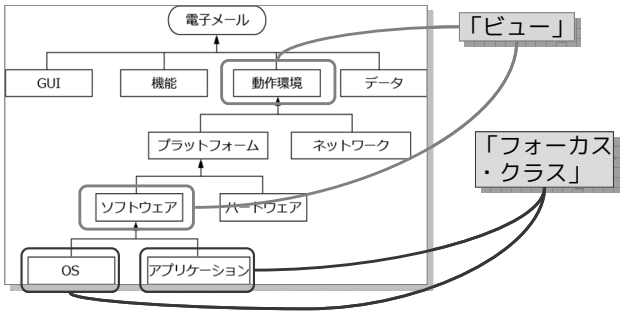


**図 7：NGT での観点の詳細化**

### 1.3. ゆもつよメソッド

HP 社の湯本剛氏が提案するテスト設計する手法であり，機能とテストカテゴリからテスト設計を考える．

1.テストカテゴリ（観点）を決める

テストタイプ毎にテスト条件をテストすべき対象やテストの方法で分類する[表 9].

**表 9：テストカテゴリの分類例**

| テストタイプ | カテゴリ | 概要 |
|---|---|---|
| 機能 | 入力表示 | 画面表示内容の確認。カーソルやフォーカスの移動等 |
| | ボタン | ボタンやリンクが押された場合の振る舞い |
| 条件やデータの組合せ | DB検索 | レコードのCRUD処理 |
| | 処理条件 | 処理が呼び出される条件等 |
| シナリオ | 業務フロー | ユーザの業務シナリオに基づいた操作 |
| イレギュラー条件 | エラー処理 | エラー発生時のメッセージ、復旧 |

2.テストマトリクスを作成する

横軸にテストカテゴリ，縦軸にテスト対象となる機能を並べた表を作成する[図 8]．テスト対象になっている機能とカテゴリでどのカテゴリでテストするかを決める．


**図 8：テストマトリクス例**

## 付録2：テスト技法概要

### 【同値分割法】[1]

仕様ベーステストの基礎であり，機能とその動作条件が仕様に記載された通りの結果を出力するかを確認するテストで使用する．本技法は，ソフトウェアの仕様から判断し同一の処理がされて同様の結果をもたらすことを期待できる入力セットや出力を想定し，テストケースを設計する．主に処理や出力結果に着目して入力を選択する．

一般的に，入力可能な値を全てテストするとテストケース数が多くなりすぎるため，同じような特性を持った部分集合(同値クラス)に分割し，同値クラスの代表値をテストすることによりテストケース数を削減できる．

同値クラスは，正常系の同値クラスである有効同値クラスと，異常系の同値クラスである無効同値クラスに分かれる．無効同値クラスのテストも忘れずに実施することが肝要である．

### 【境界値分析法】[1] [2]

同値分割法と同様に本技法も仕様ベーステストの基礎であり，同値分割法と併せて利用する．同値クラスの境界付近で誤りが生じやすいという経験則を元に，入力同値クラスと出力同値クラスの端(境界値)や，その上下の隣接値に着目したテストを実施することで効果的に欠陥を検出する．

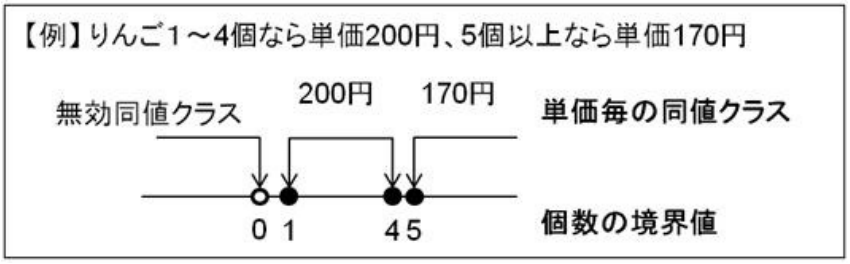


**図 9：同値分割・境界値分析**

境界値分析法には，Beizer法（ISTQBで採用），Jorgensen法（英国標準BS 7925-2で採用）などがある．図 9ではBeizer法で境界値を示している．

境界値分析法で検出できるバグの例については[表 10]の通り．

**表 10：テストケースに採用する値と検出できるバグ**

| テストケースに採用するりんご個数 | 単価200円と170円の両方の同値クラスの確認 | 単価が切り替わる個数境界値の判定処理の確認 | |
|---|---|---|---|
| | | 判定処理の個数境界値設定ミスの検出<br>例：「5<=個数なら170円」を「6<=個数なら170円」と誤って実装 | 判定処理の等号不等号記載ミスの検出<br>例：「5<=個数なら170円」を「5==個数なら170円」と誤って実装 |
| 3，7 | ○ | × | × |
| 0，1，4，5<br>(Beizer法) | ○ | ○ | × |
| -1，0，1，2，3，4，5，6<br>(Jorgensen法) | ○ | ○ | ○ |

**【CFD法】[2]**

原因結果グラフと同様に，複雑な論理関係から重要なテスト条件を漏らさない技法.

「原因の集合」と「原因どうしのつながり」に着目し，流れ線でつなぐことによって仕様を図式化する．そのため，本技法を適用する際には，処理の流れを理解しておく必要がある.

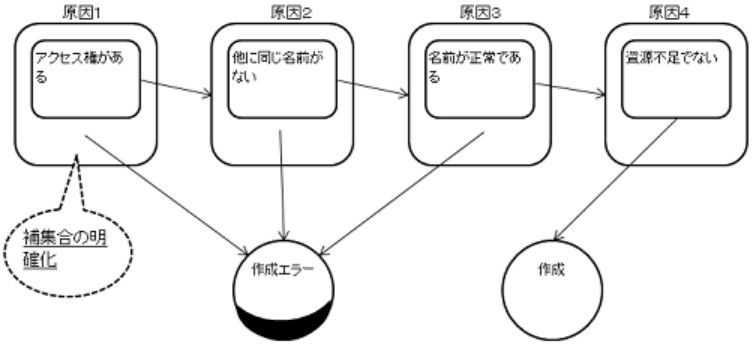


**図 10：CFD 法の流れ図**

図 10の場合，左の原因１から右に向かって処理が流れる．CFD法を用いて，内部構造に基づいた流れ図を集合で表すことで，仕様書では見えづらいことの多い補集合の部分に関してもテストの視点が届き，バグの流出防止につながる．CFDは仕様ベースの技法ではあるが，構造の情報も利用する.

**【状態遷移】[5]**

状態遷移図とは，「複数の状態をどのように遷移するのかを表現した図」のことである．状態遷移表とは，「状態とイベントをマトリクスで表現した表」のことである．マトリクスで表現することで，状態とイベントの組合せを漏れなく検討でき，無効な組合せを明確にすることができる[図 11].

状態遷移は，「状態」と，別の状態への変化を引き起こす「遷移」をもっているテスト対象に対して有効であり，すべてのテストレベルにおいて適用が可能である.

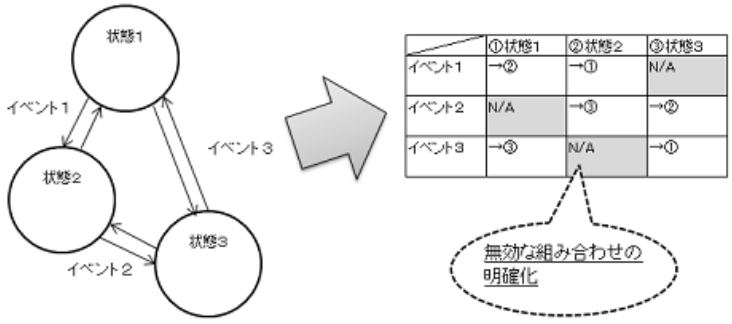


**図 11：状態遷移図，状態遷移表**

状態遷移テストは，仕様の不備や誤りを検出することにも役立つ．テストすべき状態，イベント，遷移を表現することで，対象機能の全体像が見えてくる．ただし，状態やイベントが増えると複雑になり，テスト項目が増えすぎてしまう．その際には，2項間網羅や直交表を用いて項目を削減するなどが有効である.

**【HAYST法】※2**

組み合わせテストの技法の一つ．組み合わせテスト設計において，全ての組み合わせを網羅するのではなく，実験計画法で用いられる直交表の性質「任意の二つの因子間で全ての水準の組み合わせが同一回数存在する」を利用して，二つの因子間の組み合わせを網羅したテストケースを設計する技法である．（[3]より引用）

HAYST法とは，“Highly Accelerated and Yield Software Testing”の略で，ソフトウェア組み合わせテストの技法の1 つである．大規模ソフトウェア（入力として同時に与える可能性のある機能数が300 程度の組み合わせテスト）へ適用できることと，禁則回避処理を持つ点に特徴がある．（[1]より引用）

※2 [参考] 秋山浩一ほか:“ソフトウェアテストHAYST法入門”，日科技連出版社（2007-7）

**【All Pair法】[2]**

組合せテスト技法の一つ．HAYST法同様，欠陥の多くは，2つまでのパラメータ（因子）の値（水準）の組合せで発生するという経験則をもとに，任意の2因子間の全水準の組合せが100%となるテストケースを作成する． HAYST法で用いる直交表と異なる点は，直交表が2因子間の水準組み合わせが「同数回」出現するのに対して，All Pair法では「同数回」という条件を外すことでよりテストケース数を削減できる点にある．

入力条件が多数存在する場合，同値分割，境界値分析を行ったとしても，全ての組合せパターンを網羅しようするとテストケース数が爆発的に増加してしまう．このようなケースにAll Pair法を用いることで，テストケース数を抑制しながら，効率的にバグを摘出することができる．

仕様上，パラメータ値の組合せが動作結果に影響を与えないと思われる場合であっても，予期せぬ組合せでバグが発生することがある． 例えば，帳票印刷の場合，フォントサイズ選択と，ヘッダ/フッタの有無は仕様上直接関係しないものとする．しかし，フォントサイズ＝11pt，ヘッダ/フッタ=あり，という組合せにした場合のみメモリ破損による異常が発生するとすれば，この組合せパターンのテストを予め実施し，バグ流出防止できることが望ましい[図 12].

一見関連が無いと思われるパラメータの組合せで本当に問題が発生しないかどうかを，効率的に検証する方法として有効である．

| パラメータ（因子） | 値（水準） | テストケース | | | | | |
|---|---|---|---|---|---|---|---|
| | | 1 | 2 | 3 | 4 | 5 | 6 |
| フォントサイズ | 9pt | ○ | ○ | | | | |
| | 10pt | | | | ○ | ○ | |
| | 11pt | | | ○ | | | ○ |
| ヘッダ/フッタ | あり | | ○ | | ○ | | ○ |
| | なし | ○ | | ○ | | ○ | |
| 言語 | 日本語 | | ○ | ○ | | ○ | |
| | 英語 | ○ | | | ○ | | ○ |

全組合せ網羅の場合、
3×2×2＝12ケース
となるが、
All Pair法の場合、
左記のように6ケースとなる。
(2因子間全水準組合せ網羅)

**図 12：帳票印刷での All Pair 法テストケース例**

**【カバレッジ評価法】**

テスト対象の内部構造に着目し，内部構造の種類（制御フローやデータフロー等）の網羅率を評価する手法であり，ソフトウェアの論理構造に依存する障害を検出できる（[3]より引用）．カバレッジの定義により，①コードカバレッジ，②MC/DC法（Modified Condition / Decision Coverage），③All Uses法の各評価法が知られている．

**①コードカバレッジ**

プログラムが意図したパスとは異なるパスを実行することによって発生する障害を検出できる．

プログラムの制御構造をフローグラフに表現し，グラフを網羅するようにテスト設計する技法．網羅の基準（カバレッジ）には，命令を網羅する命令網羅基準（ノード網羅基準）や，分岐文全体の真偽を網羅する分岐網羅基準（リンク網羅基準），フローグラフ上の全てのパスを組み合わせで網羅する全パス網羅基準等がある．

**②MC/DC法※3**

MC/DCは，Modified Condition/Decision Coverageの略で，ソフトウェアのテスト網羅度の一つである．条件の数をn個とすると，テストケースの数が2のn乗個にならない，現実的なテスト網羅度である．MC/DCは，条件・分岐の確認に加え，ソースコードを記述する際にしばしば間違いが起きるA or BとA and Bの記述ミスが検出できる．

与えられたテストが，MC/DC100%である，の定義は，以下を満たすことである．

① プログラムの全入口／出口を少なくとも一回はテストすること．
② プログラムの判定に含まれる全条件は可能な値を少なくも一回はテストすること．
③ プログラムの全判定は可能な値を少なくも一回はテストすること．
④ プログラムの判定の全条件は判定の出力に独立に影響することを示すこと．

※3 [引用] 松本充広，“MC/DCによる現実的な網羅のススメ”，キャッツ組込みソフトウェア研究所（2009-7）

**③All Uses法（全使用法）**

データフローテストの一つ．すべての参照－定義関係をテストする全DUパス法があるが，テストケース数が多くなり過ぎる．そこで，テストケース数を少なくするため，All Uses法が考え出された．

すべてのパスを一旦抽出し（そのままテストすると全ｄｕパス法），それから全ての変数を一度にテストできるパスを見つけ出し，整理してテストケース数を少なくしている．

**【ユースケース・シナリオテスト】**

システムテストにおけるひとつの手法として，ユーザの要求を網羅的にテストするものである．具体的な業務とシステムのかかわりを，利用者側からのシナリオとシステム側からのシナリオの両面から展開できるブラックボックステストの設計技法である．[図 13][※4]

※4 [参考] 2010 SQiP研究会 第5分科会 堀田さん配布資料.

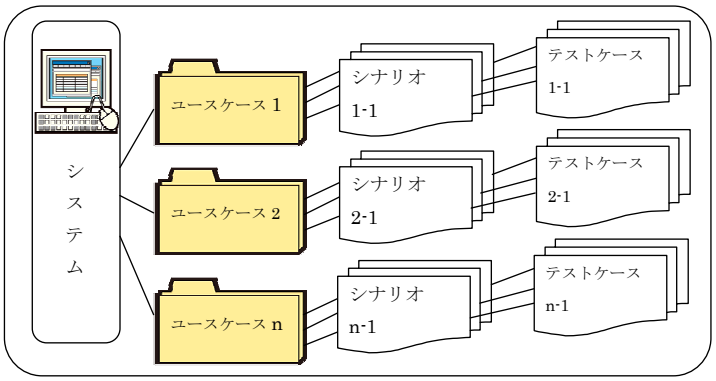


**図 13：ユースケースとシナリオの関係**

**【サイクロマチック複雑度】※5**

プログラムの複雑さを，プログラムの分岐に着目して数量化したものであり，プログラムの入口・出口・分岐をノード（N），ノード間をリンク（L）とする有向グラフとして捉え，以下の式で算出される尺度である．

サイクロマチック複雑度（C）＝ L － N ＋ 2

サイクロマチック複雑度はプログラムの入り口から出口に至る基本パスの数を意味し，サイクロマチック複雑度とプログラムのバグ・品質の相関関係を分析したい場合に利用する．例えば，この数値が一定値を越えるプログラムは作成してはならないという品質基準としての利用や，コードレビュー・単体テストの念入り度合（テスト密度等）を決める基準などに利用する．

※5 [引用] 大塚俊章ほか:“ソフトウェアテスト技術”, UNISYS TECHNOLOGY REVIEW, 第93号, p.79 (2007-8)

**【テスト密度】※6**

開発規模単位あたりのテスト数．テスト密度はテスト項目の絶対数を，規模で割って算出する．

テスト密度＝テスト項目数 / 開発規模（ステップ数等）

規模が大きくなればテストすべき項目も多くなるはずという前提に基づき，テストの十分性を推し量る指標の１つである．

テスト密度を把握することにより，バグ摘出に必要なテスト項目の絶対量を推し量ることができる．ただし，項目数は十分であっても，同じような項目ばかり数だけ増やしても意味が無いため，テスト密度だけではテストの網羅性を保証することはできない．テスト密度が十分だとしても，必要な機能を網羅したテスト項目が揃っていると判断するためには，テスト項目の内容を見てチェックする必要がある．

※6 [引用] 粟野憲一ほか: “パブリックコメント版 続 定量的品質予測のススメ 定量的品質管理実践ガイド”, IPA (2010-4)