ソフトウェアテスト技術

Software Testing Technology

大塚俊章, 荻野富二夫

要　約　ソフトウェアの品質が注目されつつある．品質は上流で作りこむのが品質管理の基本であるが，テストは上流工程での適切なテスト戦略とテスト設計から始まり最終工程としての検証に至る，ソフトウェア開発工程全般に渡る極めて重要な活動である．だが，テスト技術に関する誤解や認識の違いが，ソフトウェア品質の低下を引き起こしたり，場合によってはプロジェクトの成否すら左右する場合があることが，日本ユニシス品質保証部が実施している第三者レビュ等の品質保証活動を通して明らかになってきた．オフショア企業に開発委託しているプロジェクトで特にその傾向があるが，昨今のシステム構築プロジェクトが抱えている共通の問題とも思われる．本稿では，特に誤解が生じやすく，よって委託先と共通理解を確立しておくべきテストの考え方と技術について述べる．

Abstract　The quality is becoming a main issue of software developments. It is fundamental to the quality control that the quality is elaborated at the upper process. Software testing is extremely important activities throughout the entire software development process which begins with planning an appropriate test strategy and designing test scenario and cases. However, through the quality assurance activity such as third-party code review in our company, it has emerged that the misunderstanding or perception gaps of testing technology causes the lowered software quality and sometimes influences the success of a project. This tendency is especially true with the project execution outsourced to the offshore developer, however, it is the common problems which system development projects these days face. The purpose of this report is to discuss the concept and technology of software testing, subject to misunderstanding with subcontractors, on which agreements must be reached with them.

1. はじめに

ソフトウェアの品質低下が社会問題となりつつある．携帯電話やデジタル家電は言うに及ばず，自動車などに搭載される組込ソフトウェアの規模も増大の一途をたどっており，ソフトウェアの品質が企業の経営のみならず人命や財産に及ぼす影響も大きくなりつつある．その一方でソフトウェア開発の失敗によるデスマーチ[1]も話題にこと欠かない．「品質は上流で作り込む」のが品質管理の基本であり，ソフトウェア開発のライフサイクル（SDLC）の上流工程でのインスペクションやレビュなどの技法や管理方式が導入されてきたが，昨今ではテスト技術に対する反省と関心も高まりつつある．これまで大学や企業の中でテスト技術に関する体系的な教育が十分に行われて来たとは言い難く，極めて属人的な技術であった．特に現在のソフトウェア産業の管理職の世代がソフトウェア開発に携わった時代は，プログラム設計とコーディングに十分な机上検査の時間を割き，コンパイルも実行も一度で通すことが技術力の証しであり，テストは必要悪と考える風潮があったことは否定できないであろう．一方，東証の誤発注

問題に対するシステム情報学会の報告[2]では，テストの不十分さが指摘されている．またソフトウェア開発の現場では社内外・国内外問わず，テストに関する様々な誤解，認識違いが存在し，それが場合によってはプロジェクトの成否を左右する問題にまで発展するケースも見られる．ソフトウェア開発の下流工程をオフショア企業に委ねることが常態化しつつある現在，テスト工程の品質強化策は重大な意味をもつ．

日本ユニシスの品質保証部では，テスト工程のみならず，プロジェクトや成果物の品質向上のための全社的な品質活動を展開してきたが，このような環境変化に合わせ，テスト技術の更なる強化の必要性を認識し専門組織を設立した．社内のみならずオフショア企業を含めた開発委託先企業と，テストの考え方や技術についての誤解を解消し，戦略的テストプロセスやテスト技術の普及・定着を図り，テスト品質を向上することが目的である．本稿では，特に誤解が生じやすく，よって委託先と共通理解を確立しておくべきテストの考え方と技術について述べる．

2007 年 3 月，日本ユニシスは株式会社アイ・ブイ・スクエア（以下 IVS 社）と，テスト分野での技術協定を締結した．IVS 社が日本総代理店となっているインドの STAG software 社（以下 STAG 社）が保有する統合的テスト方法論「STEM™」[*1] をテストプロセスに取り入れ，開発プロジェクトへの適用を通して品質向上を図るのが狙いである．STAG 社はテスト技術やテクニックに傾倒することなく，STEM™ の中に，テストの原理や合理的なテスト戦略・方法論を確立し，それらに忠実にテスト技術者を育成しテストビジネスを展開している．その活動方針が日本ユニシスの目指す方向性と一致したことが最大の選定理由である．なお，インドでは開発発注元企業が受入テストを STAG 社のようなテスト専門会社に発注するケースが増加しており，ソフトウェアテストが既にビジネスとして成立している．

2. ソフトウェアテスト技術概観

ソフトウェアテストを支援するツールも多数登場し，テスト品質の向上やテスト工数の削減が期待できるようになってきた．しかし，開発依頼元と委託先の間での，テスト技術や概念の理解の差異が，テスト品質不良やプロジェクトスケジュール遅延などを引き起こすケースも発生している．こうした問題を解決するためには，テストの意味と原理を正しく理解した上で，正しい技法を選択する必要がある．本章では，STEM™ の考え方を引用しながら[3]，理解のレベルを合わせておくべき技術的なポイントに絞って解説する．

2. 1 テストの姿

G.J.Myers は「ソフトウェアテストの技法第 2 版」[4]で次のように述べている．

- **ソフトウェアテストは技術的な作業であるが，それは経済学的また心理学的に重要な考慮が必要である．理想的な世界では，プログラムのすべての組合せをテストしたいと考える．しかし，ほとんどの場合，これは簡単にいえば不可能である．**
- **一般的に，プログラムのすべてのエラーをみつけることは，非現実的でもあり，しばしば不可能でもある．**

テストとは，「エラーが無いことを示していく過程」ではなく，「存在するはずのエラーを発見するための活動」である．「存在するはずのエラー」の規模が，日本ユニシスが採用している障害検出率の目標値ということになる．また「テストをし尽くすこと」「すべてのエラーを

見つけること」が非現実的である以上，その事実を理解した上で，経済的・合理的なテストを行うための戦略と技術がテスト成功の鍵を握る．

2. 2　**直交原理**

STEM^{TM} では，テストを次の 3 軸（図 1）で捉える．

- レベル（Level）　　単体テスト，統合テスト，システム・受入テスト
- タイプ（Type）　　機能性テスト，効率性テスト，信頼性テスト，…
- 技法（Technique）　ブラックボックス技法，ホワイトボックス技法

この原理が意味するところは，ソフトウェアの欠陥はそのタイプに応じてそれを検出すべき最適な 3 軸（Level/Type/Technique）の組合せが決まるというものである．例えばプログラムの「メモリーリーク」というタイプの欠陥は，レベル = 単体テスト，タイプ = 機能性テスト，技法 = ホワイトボックス技法で検出するのが最も合理的であり経済的である．このタイプの欠陥が，統合テストやシステムテストあるいは稼働後に発症した場合には，再現性の乏しい追及困難な現象となることが多い．2. 6. 1 項で示すとおり，テスト戦略立案段階では，リスクの高い欠陥を特定し，直交原理によりその欠陥を摘出すべき手段（3 軸）を定める．

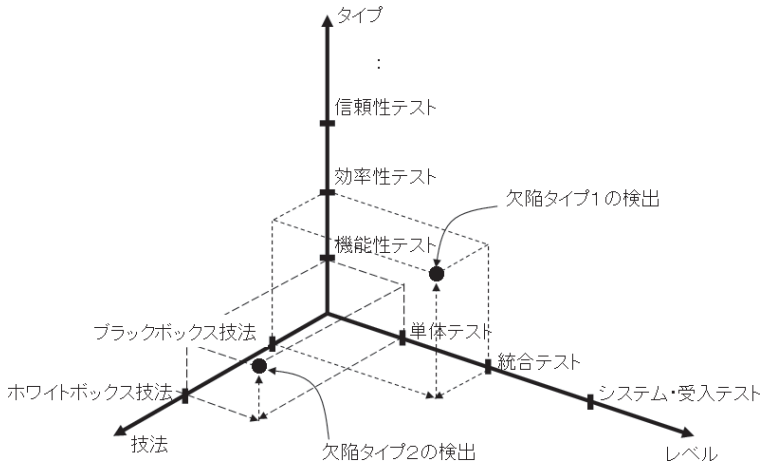

図 1　STEM^{TM}　の直交原理

2. 3　**テストレベル**

直交原理の「レベル軸」は，単体テスト（Unit Testing），統合テスト（Integration Testing），システム・受入テスト（System/Acceptance Testing）の各テスト工程を意味し，図 2 に示した日本ユニシスの標準的な開発モデルのテスト工程と対応する．

直交原理によれば，それぞれのテストレベルでどんな欠陥を摘出すべきかを考え，適切なテストのタイプと技法を選択しなければならない．

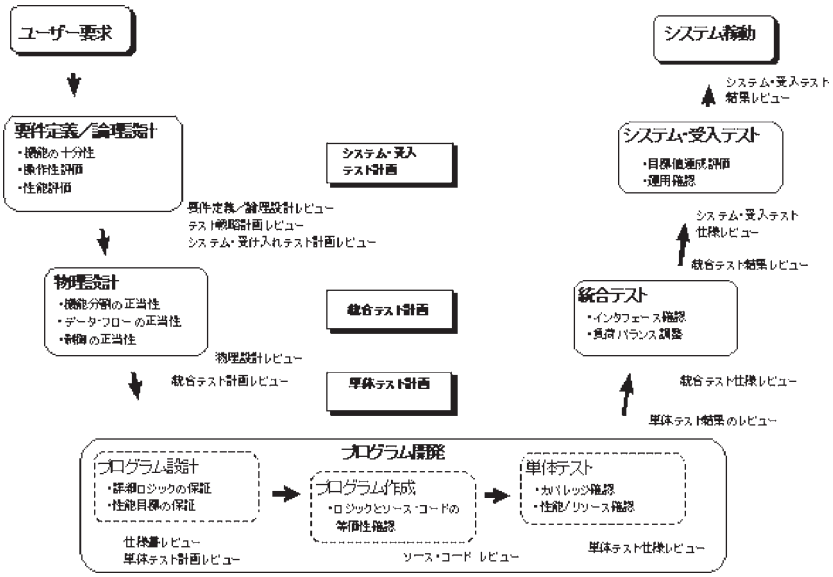

図2　日本ユニシスの標準的な開発モデル，V モデルと V & V（検証および妥当性確認）

2.3.1　単体テスト

単体とは「意味を持った動作をする最も小さなソフトウェアの単位」[1]であり，単体テストとは「ひとつのプログラムの中のサブプログラム，サブルーチン，またはプロセジュアのテスト過程をいう」[4]．オブジェクト指向などのプログラミングパラダイムの変化や，JAVA 等の開発言語の変化などにより「単体」の捉え方も変わってきているが，ソフトウェアがサブプログラム，関数などと呼ぶ最小単位のプログラムの集まりとして構成される以上，この最小単位の品質を保証することは不可欠であり，その手段が単体テスト，その対象はそれらのプログラムと考えるべきであろう．自動車を例にとると，最小単位の（それ以上分解できない）部品，例えばボルト 1 本に対する耐久試験・強度試験がそれにあたる．しかし，どうしても全てのプログラムの単体テストやコードレビュに投入する工数が捻出できないという場合には，入念にテストすべきものと，そうではないものを見極めるための合理的な基準を設定すべきであろう．例えば，次のようなプログラムに対しては工数をかけてテストを行う．

・経験の浅いプログラマが作成したプログラム
・多数のプログラムから引用される使用頻度の高いプログラム
・プログラムステップ数が，ある数を超えるプログラム
・サイクロマチック複雑度がある水準を超えるプログラム（2.4.2 項参照）

2.3.2　統合テスト

統合テストの目的は，単体テストが完了し単体としての品質が保証されたプログラムが相互に連携して正しく機能するかどうかを検証することである．その手法として，非増加テストと増加テストがある[4]．図 3 のプログラム構成例を用いて，両手法の概要を説明する．

非増加テストでは，図 4 のように，ドライバ・スタブを用いてそれぞれのプログラムを個別に単体テストした後で，全てのプログラムを一気に統合したテストを行う．この統合方式をビッグバン方式と呼んでいる．この場合，5 個のドライバと 5 個のスタブを開発する必要がある．

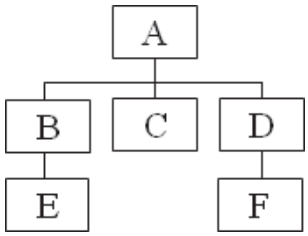

図3　プログラム構成の例

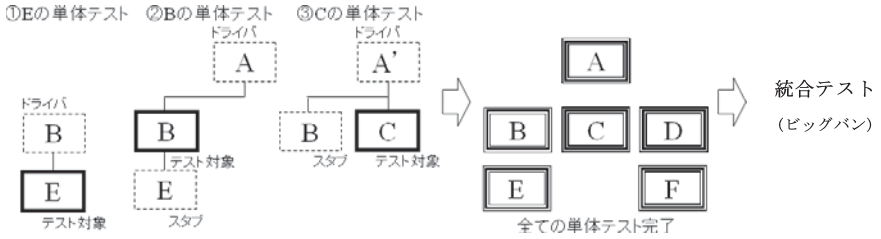

図4　非増加テストと統合テスト

増加テストでは，新たなプログラムのテストに際して，テスト済みのプログラムを利用する．図5では，①でBのドライバを使いEをテストした後，②ではドライバAとテスト済みのEを使用してBをテストする．この例は最下位のプログラムから順次テストしており，ボトムアップ方式と呼ばれる．このケースでは五つのドライバが必要となる．このほかに，トップダウンやサンドウィッチと呼ばれる統合戦略もある．尚，増加テストでは統合テストは別種のテスト形態とはみなされず，単体テストの完了⑥と同時に統合テストも完了する．

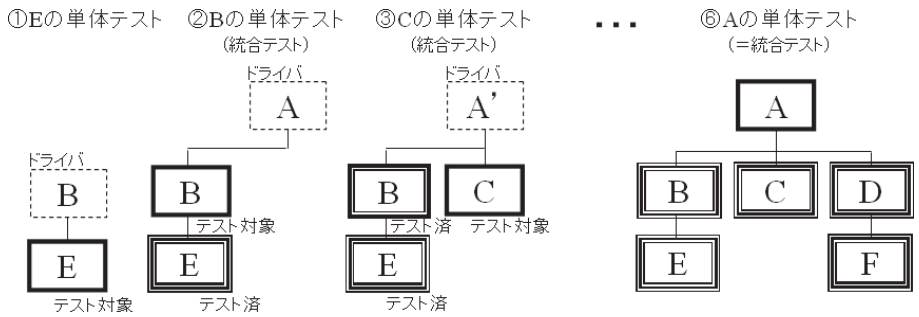

図5　増加テスト

参考文献[4]では述べられていないが，現実にはあり得るケースとして，真のビッグバンテストについても簡単に述べる（図6）．単体テストを一切行わずに統合テストを行う方式である．ドライバやスタブを作成する手間も，個々に単体テストする手間も不要であるが，この方式が最も生産性が高いと考えてよいかどうかは一概には言えない．後述するように，Aに対するブラックボックス技法に基づくテスト設計と，全てのプログラムに渡るホワイトボックス技法による網羅性の検証などを行うことで，この方式にもある程度合理的な評価を与える余地はある．

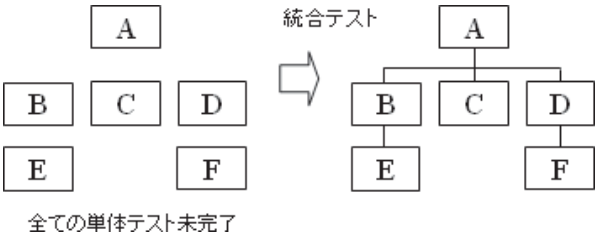

図 6　真のビッグバンテスト

統合テストには，ITa,ITb と呼ばれる一種の工程分割とも言える考え方がある．一般に，前者はサブシステム内の統合テスト，後者はサブシステム間の統合テストとみなされている．統合テストをコンポーネント間の相互作用の正しさを証明することと捉えることもある[5]．サブシステムやコンポーネントの意味の厳密な定義はさておき，オブジェクト指向を持ち出すまでもなく，システムを構成するプログラムが図 7 に示すような抽象的な概念の階層を持つことは一般に知られている．複数のコンポーネントが更に上位のコンポーネントに包含されることもある．コンポーネント内の統合テスト，コンポーネントを含む上位コンポーネントとしての統合テスト，サブシステム間の統合テストと，それぞれの場面において前述の統合の方式や順序の最適な選択が必要となる．

- Comp-a3 は真のビッグバンによる統合で十分か
- Comp-a1 は増加テストによる綿密なテストが必要か
- Comp-a1 のテストのために Comp-a2，Comp-a3 のスタブ作成が必要か
- SubSystem-a 全体の統合テストが完成するまで，SubSystem-b はスタブとしてテストするか
- SubSystem-b が重要な機能を担うものの，システム稼働直前まで実体との統合のチャンスが無いとしたら，シミュレータを開発するなどして SubSystem-a のテストの完成度を上げる必要があるか

など，統合テストの順番や方式は一様ではなく，コンポーネントやサブシステムの構造や重要度，開発スケジュールなど様々な観点で決める必要がある．これらの方針は，テスト戦略・統合テスト計画として，プロジェクト開始時点で検討しておくべき最も重要な課題のひとつである．スタブを用いた統合テストの段階では，スタブに対して正しい処理要求を発行しているかどうかの検証をしておく必要もある．委託先がこうした検証をしておらず，スタブが呼び出されていることを以ってテスト完了としていたために，最終テストの段階で多数の障害に直面したプロジェクトがある．スタブを用いたテストも，その実体を組み入れたテストでも，ブラックボックス技法で設計したテスト仕様は同じであり（2. 4 節参照），確認の方法が異なるだけである．テストの委託先との曖昧性のない事前の合意が重要である．

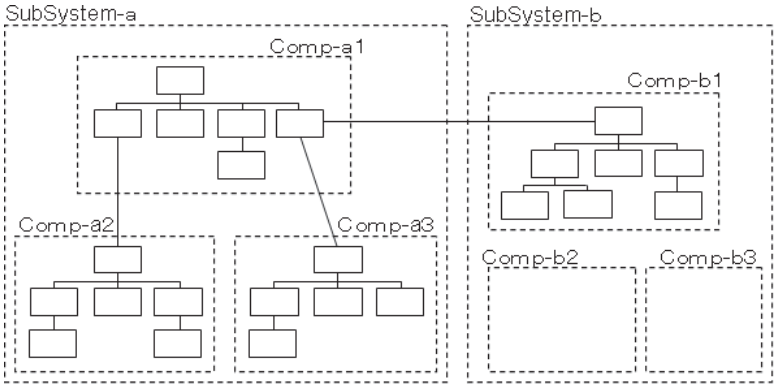

図 7　一般的なプログラム構造

2. 3. 3　システムテスト・受入テスト

直交原理の「タイプ軸」は，日本ユニシスが外部品質特性と呼ぶ以下の項目に対応する．

機能性（Functionality）	合目的性，正確性，接続性，標準適合性，セキュリティ
効率性（Efficiency）	実行効率性，資源効率性
信頼性（Reliability）	成熟性，障害許容性，回復性
使用性（Usability）	理解性，習得性，運用性
保守性（Maintainability）	解析性，変更性，安定性，試験性
移植性（Portability）	環境適応性，移植作業性，規格準拠性，置換性

これは顧客（外部）から見た品質特性であり，多くの場合，システムテスト・受入テストの目的となるが，単体テストや統合テストの目的となる場合もある．図 2 の V モデルでは，これらのテストは要求定義のプロセスと対応している．言い換えれば，顧客の要求を論理設計，物理設計，プログラム設計と翻訳しながら作り上げたシステムが，最終的に顧客の要求を本当に満たすものに仕上がったかどうかの検証を行うのが目的である．従ってテストケースは要求仕様を基に作成されることになる．

2. 4　テスト技法（ブラックボックス技法とホワイトボックス技法）

直交原理の「技法軸」は，ブラックボックス技法とホワイトボックス技法である．これらの技法の意味は，必ずしも正しく理解されていない．ここでは，正しいテストについての理解を確認する．

ブラックボックス技法はプログラムの内部構造と動作には一切関知せずに，プログラムが仕様書どおりの動きをするか否かに注目したテストである[4]．ホワイトボックス技法は，プログラムの内部構造を調べプログラムの論理に沿ったテストを行う[*2]．2 次方程式を解くプログラムを考える．プログラム A は根の公式による解法を，プログラム B は解の精度を保証するためニュートンラプソン法による収束計算法を適用しているとする．ブラックボックス技法を用いて設計したテストケースはどちらのプログラムにも適用できるが[*3]，ホワイトボックス技法を用いたテストケースは，A，B それぞれで異なるものになる．昨今のアジャイル開発におけるテストファースト手法では，テスト対象のコードを書く前にテストプログラムを作成することからブラックボックス技法によるテスト設計を行うことになる．リファクタリング（プログ

ラム仕様を維持したままコードの最適化を行うこと）時に流用できるテストケースや，プログラム改修時のリグレッション（デグレード）テストのために開発されるテストプログラムも同様である．STEM^{TM}では，テスト設計にブラックボックス技法を用い，その設計の妥当性を検証するためにホワイトボックス技法のカバレッジ分析の手法を用いる．「偶然の正確[*4]」の検出性を例にあげてホワイトボックス技法の優位性を主張する文献もある[5]が，ホワイトボックス技法の方がテストケース設計に向いているということではないので注意が必要である．これはコードレビュの領域の問題である．

2.4.1　ブラックボックス技法によるテスト設計

STEM^{TM}では，図8のとおり，テスト設計の明確なガイドラインが示されている．テスト対象のソフトウェアは，特定の入力に対して何らかのロジックが起動され所定の結果を生み出す．ここで言うソフトウェアは，単体テストでは個々のプログラムであり，統合テスト，システムテストの段階では，図7に示したコンポーネントやサブシステムのように，複数のプログラムにより構成されるより上位のサービスを指す．このロジック（プログラム仕様）からテストシナリオを導出するために使用されるのが，デシジョンテーブルや状態遷移図などの技法である．入力データに対して，同値クラス分割や境界値分析などの手法を適用してテストデータが得られ，それらの組合せとしてテストケースが導出される．簡単な具体例を3章に示す．一般に入力データは実質的に無限であることが多い．また，テストデータとテストシナリオの組合せで得られるテストケースも同様である．有限のコストと時間と労力でテストを行うために，テストすべきデータやテストケースを削減する必要がある．同値クラス分割や境界値分析，あるいは直交表などの技法はそうした目的で利用する．テスト密度のノルマを負ったテスト担当者にとってジレンマとも言えるが，これらの技法で刈り取られるべきテストケースの多くは本来は無駄なものである．テスト密度信仰の危うさはここにある．

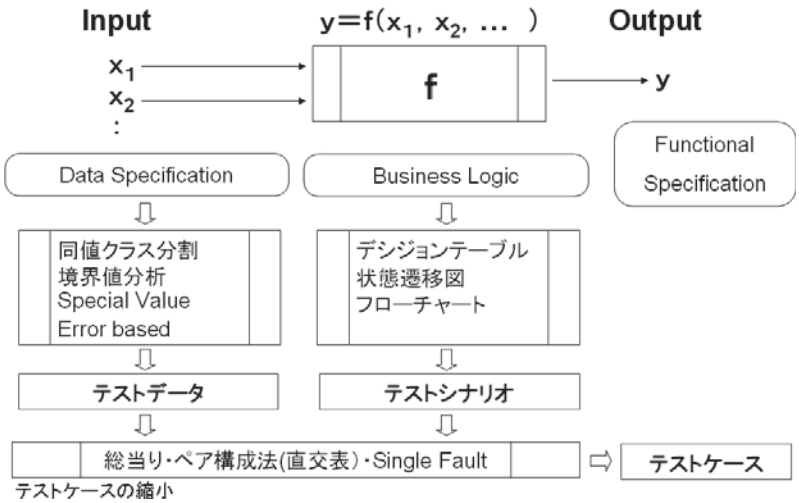

図8　ブラックボックス技法によるテスト設計（STEM^{TM}）

2.4.2　ホワイトボックス技法によるテスト設計の検証

STEM^{TM}では，ホワイトボックス技法の検証内容をデータフロー（Data flow techniques）

と制御フロー（Control flow techniques）の二つに大別している．

前者は，変数の値が参照される前に設定されているか，動的メモリ等の動的資源が正しく割当て・解放されているかなど，プログラムの制御フローの中で，データが正しく処理されているかどうかを検証する．多くのプロジェクトは，テスト活動としてではなく，コードレビュや静的検査ツールの利用（広義のテスト）で対応していると思われるが，以下に述べるカバレッジ指向テストの中で検証することも可能であろう．

後者は，作成されたコードが正しく動作することを検証する．論理網羅テストと呼ばれ[4]，以下に示す網羅基準（カバレッジ基準）により，コードが漏れなく実行され正しく動作することを保証する．最初の二つの網羅基準は，一般に C0，C1 と呼ばれている．

- 命令網羅（Statement coverage（SC））
- 判定条件網羅（Decision coverage（DC））/分岐網羅（Branch coverage（BC））
- 条件網羅（Condition coverage（CC））
- 判定条件/条件網羅（Decision/Condition coverage（DC/CC））
- 複数条件網羅（Multiple condition coverage（MCC））
- 経路網羅（Path coverage（PC））

2. 4 節で，テスト設計にブラックボックス技法を用い，その設計の妥当性を検証するためにホワイトボックス技法のカバレッジ分析の手法を用いると述べた．ブラックボックス技法で設計したテストケースの実行時にカバレッジを測定して，カバーされていないケース（そのテストケースでは実行されない分岐やパス）をテストケースとして追加するのが一般的である．C0，C1 カバレッジを測定するツールは一般に入手可能であり，カバレッジ率のほか，通過しなかったパスが通知される．通過しない原因としては次のような理由が考えられる．

1） ブラックボックス技法による設計時のミス（ソフトウェア仕様の見落としなど）

2） ソフトウェア仕様に折り込まれていない，内部ロジックに関連した条件
例えば，データ量の増加に伴いメモリ内処理からファイル IO へ切り替わるなど．

3） ソフトウェア仕様に折り込まれていない，下記違法コードのような不正処理

```
                ⋮
if (kouza_no == 511111)  nyukin_gaku *= 2;
                ⋮
```

ただしこの種の問題を検出できるのは，プログラム作成者とは別の人がテストを実施した場合に限る．

4） テストケースを発生させるのが困難なケース
実質的には発生しない（発生する可能性の極めて低い）例外ケースのエラー処理など．

上記カバレッジ基準は，下に行くほどテストケース数（テスト工数）が急激に増大する．日本ユニシスのプロジェクトの多くは，単体テストの網羅基準として C1 を採用し，品質基準を 90%程度に設定している．100%としないのは，上記 4）のケースを除外するためである．90%以下のカバレッジしか得られなかった場合にはその理由を明らかにする（報告させる）ことも必要であろう．

ホワイトボックス技法は，2. 2 節の直交原理にも示したとおり，原理的に統合テスト・システムテストでも利用できる技術である．大域変数や，プログラム間で割当・解放を行う動的資

源のデータフローが正しいかどうかを検証するためにはこの技法を利用するが，テストの難易度とコストが高いため，コスト対効果の観点で難点がある．

プログラムの制御フローに関する話題として，最後に McCabe のサイクロマチック複雑度について紹介する[7]．これは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

図 9 は，C ＝ 4 のプログラム例と有向グラフを示す．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ている．これを測定するツールも一般に提供されている．この数値とプログラムのバグ（品質）の相関関係を分析することも興味深いし，この数値が一定値を越えるプログラムは作成しないという品質基準としての利用や，コードレビュ・単体テストの念入り度合を決める基準など，活用の可能性は広い[*5]．

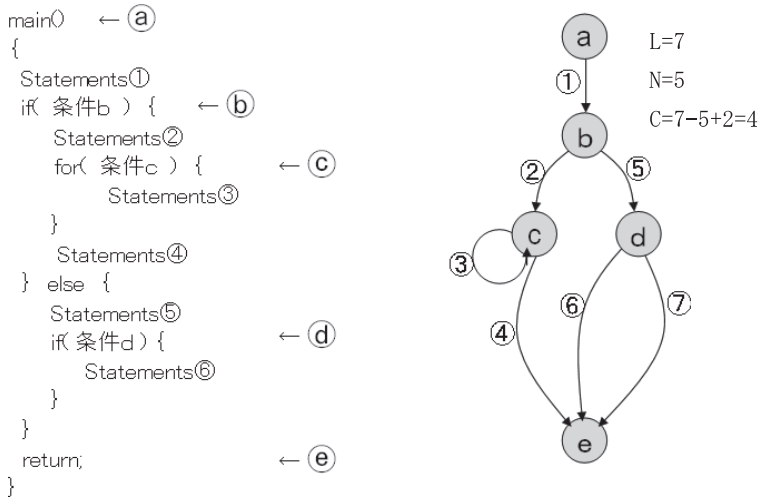

図 9　McCabe のサイクロマチック複雑度

2. 5　テストの妥当性評価

前節までに述べた技法を用いて，合理的・経済的なテストを計画する必要があるが，そのテストが妥当性を欠いていないかどうかを検証することもまた重要である．日本ユニシスの品質標準では，テストの妥当性を評価するために，テスト密度（一定のプログラムステップ数に対するテスト項目数）・障害検出率・信頼度成長曲線などの品質メトリックスが利用されている．この背景には，「数値化できないものは管理できない」という考え方があり，定性的な品質に対して定量的な尺度を導入したという点において画期的なものであった．しかし昨今，このメトリックスに対する誤った信仰が目につくようになった．テストの妥当性は，テストすべき機能がもれなくテストされているかどうか（テストケースの網羅性（カバレッジ）），致命的な影響を及ぼす恐れのある欠陥（リスク）を特定しそれを防止するためのテストが実施されているかどうかで評価すべきであるが，テスト密度は必ずしもそれを保障するものではない．図 10 では，横軸にテストすべき機能，棒グラフはそれぞれの機能に欠陥が混入した場合の影響度（リスク），曲線はテストケース数を示した．仮に①②③が全てテスト密度の目標値を同程度にク

リアしているとすれば，テスト密度ではその善し悪しを測ることはできないが，③がリスクの低い機能に貴重なテスト資源を割り当てているのに比べ，①は重要な機能ほど綿密にテストされている点で優れている．

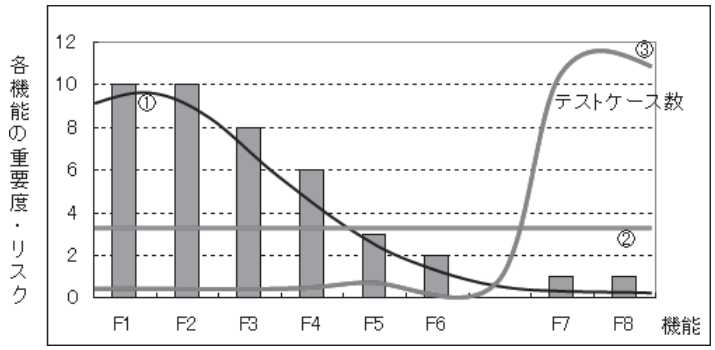

図 10　**テストケース数の妥当性**

このようなテストの妥当性評価尺度を導入するためには，システムに要求される機能（Feature: F_{ij}）を列挙し各々にリスクを割当てる必要がある．このリスクは，元になった要求（Requirement: R_i）に立ち返って，顧客と業務の視点で捉える必要がある．要求：R_i と機能：F_{ij} の対応は，一般に顧客との契約ベースラインとして管理されているはずであり，要求の変更管理（Requirement Management）の管理対象にもなっている．日本ユニシスの標準として，要求追跡検証マトリックス（RTVM: Requirement Traceability Verification Matrix）があるが[12]，こうした既存の要求管理情報と連携した管理が妥当であろう．さらに図 11 のように，各機能：F_{ij} に対して設定されたテストシナリオとテストケースを対応づけて管理することにより，図 10 で示したテストケースの妥当性評価の道が拓ける．また，テストで検出された欠陥もテストケースと結びつけて把握すれば，障害の発生状況を量として捉えるだけでなく，要求のリスクとの関連で評価できるようになる．これが日本ユニシスの品質管理が次に目指す方向性のひとつである．

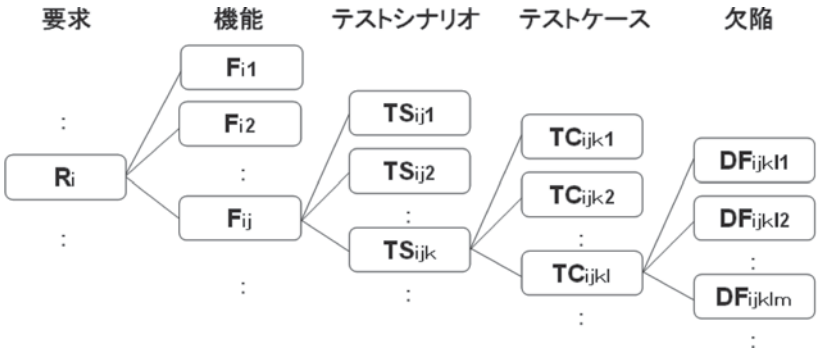

図 11　**テストシナリオ・テストケースのトレーサビリティ**

2. 6　テスト戦略とテスト計画

効果的なテストを行うためには適切な戦略と計画の立案が不可欠である．図 12 に示すとおり，テスト戦略はテスト活動全般に渡る方針や技術的アプローチを定め，テスト計画は各レベルの

テストごとに立案される．これらは本来プロジェクト計画立案の段階，即ち開発プロジェクトの初期段階で立案しなければならない．テスト計画はスケジュールのみを定めるものではない．

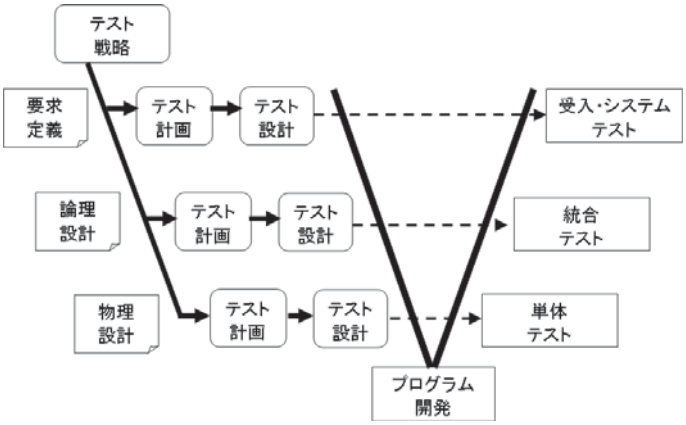

図 12　テスト戦略とテスト計画

2. 6. 1　テスト戦略

テスト戦略の中に定めておくべき最も重要な事項について簡単に紹介する．

1)　リスクの明確化

テスト戦略検討の第一歩は，どの要求，どの機能に焦点を当てるべきか，どのような潜在的な課題・欠陥に焦点を当てたテストを行うべきかを考えることである．有限の時間とコストとパワーを，最も効果の得られる点，決して欠陥を見逃してはいけない点に集中する必要がある[*6]．そのために，2. 3. 3 項に示した外部品質特性（非機能特性）や，2. 5 節に示した要求および機能に対して，想定される欠陥とそのリスクの大小を分析し割当てる．プログラム作成要員に初級者が多い場合には，「データフロー」の課題に焦点を当てたり[*7]，難易度の高い技術の開発・適用を行うような場合にはそこに焦点を当てることも必要であろう．

2)　テストのタイプ・レベル・技法の選定

2. 2 節の直交原理に従い，洗い出された欠陥をどのレベルのテストで，どの技法を用いて検出すべきかを決定する．この方針は，単体・統合・システムテストの各テスト計画に継承されることになる．1）2）を通した重要な考え方は，検出すべき欠陥を明らかにして，それらを検出すべき最適なテストのレベルと技法を計画することにある．道具に合った仕事をすることではなく，仕事の目的を決めてそれに合わせた道具を選ぶということである．

3)　テストの自動化の検討

ソフトウェアの多くは継続的な保守を必要とする．テストを自動化しておくことは，コスト削減のみならず，品質を一定に保つことに役立つ．納品すれば終了する受託開発においても，テスト自動化の投資対効果は決して小さくない．スタブを使った単体テスト，スタブを実体に置き換えた統合テスト，機能テストが終わった後の効率テスト，不具合改修後の確認（リグレッション）テストなど，保守工程だけではなく，開発工程においても同じテストを繰り返し行う場面は多い．発注元の顧客にとっても十分に価値のある資産となり得ることを考えれば，開発コストとしての損益分岐点も変わってくるはずである．STAG 社のある技術者は，50%～ 60%のテストの自動化は可能であり，意味があると語っている．GUI を記

録して再生する形態の自動化ツールも数多く市販されており利用価値も高いが，入力仕様の変更等に伴い，自動プログラムの改修が必要となる点がネックとなっている．プログラムインターフェースレベルの不変性の高い層で，記録・再生を行う仕組みを構築することも可能であるし，日本ユニシスでもこうした取り組みを行い効果を挙げているプロジェクトもある．何れにしてもテストプログラムの使い棄て，納品されたテストドライバの死蔵は，環境保護の観点でも避けるべきである．

4） 品質メトリックス

テスト戦略が正しく機能しているかどうかを判定できる尺度を定めることである．日本ユニシスでは先に述べた，テスト密度・障害検出率・信頼度成長曲線などのメトリックスを導入し，それまで定性的と思われてきた品質を数値化する努力を行い大きな効果をあげることができた．ただ，本稿でも述べたとおり反省すべき点も多い．日本ユニシスでは 2. 5 節で述べた考え方などを折り込んだ新しい品質尺度を開発し，社内標準として制定したいと考えている．

2. 6. 2 **テスト計画**

テスト戦略を継承する形で，単体テスト・統合テスト・システムテストの各レベルのテストの詳細を定める．戦略と計画に沿った，ツールの選定，適用する技法やツールに関する教育，協力会社への作業指示書（SOW）への展開，テストチームの組織化と見積りなどを行う必要があるため，テスト計画の立案はテスト設計に着手する時期では遅い．例えば，統合テストのテスト計画では，2. 3. 2 項で述べたような統合の順序や方式などを定める必要がある．これに応じてプログラムの開発の順番を最適化する必要もあり，スタブやシミュレータの開発など，開発計画をも左右する．

以下，すべてのレベルのテスト計画に共通に折り込むべき内容の概要を示す．

1） テストのスコープ

テストの対象機能，即ち，何をテストし，何をテストしないかを明記する．

2） テスト作業と見積り

テスト戦略に基づき必要な作業を洗い出し，その依存関係を分析して WBS を作成する．メンバーとのテスト戦略・計画の共有や，ツールの選定，要員教育なども重要である．

3） テスト作業スケジュール

各々のテスト作業に適切なチーム・要員を割り当て，すべての作業が所定期間内に収まるようなスケジュールを作成する．テストで検出された障害の改修期間や，改修によるデグレードの検証などの作業も折り込まなければならない．

4） テストクライテリア（Entry/Exit 条件，Abort/Suspend 条件，Release 条件など）

Entry 条件は，テストの前提となる前工程の成果物の品質が適切な水準に到達しているかどうかを判定するものであり，スモークテストと呼ばれるテストが使用されることがある．Exit 条件はテストの完了，即ち所定の品質が達成されたことを判定する条件である．発注にあたっては特にこの条件を曖昧性なく定義し，説明し，合意し，検査することが重要である．Abort/Suspend 条件は，テストを中止/中断する際の判断基準である．昨今の統合テストでは，検出された欠陥の中で，本来単体テストで検出されるべきであった欠陥の占める割合を使うことがある．

5)　テスト環境

テストに必要なハードウェア，ソフトウェア，テストツール，テストデータなど．

6)　テストチーム体制

テストの実行や，テスト設計・テスト結果のレビュなどのテスト活動の役割分担を個人名レベルで定義する．

7)　テストトレーニング

テスト計画を実行するために必要な教育とトレーニングについて明確にする．

8)　リスク・コンティンジェンシ

通常のリスク管理の一環で，テスト活動に焦点を当てたリスクを明確にする．

9)　テスト管理

障害の重要度判定，障害レポートの管理（ワークフロー），進捗管理・レビュ・検収などのプロセスを定義する．システムテスト終盤では，「対策するリスク」と「対策しないリスク」を評価する意思決定プロセスなども重要となる．

10)　テストサイクル

統合テストの統合順序・方式の決定問題と類似するが，一般に，重要な機能・非機能要件から順番に開発・テストを行い，あるいは障害を順次解決しながら，ソフトウェアを段階的に完成させていく．そのようなテストのサイクルを開発計画・開発進捗と整合した形で計画する必要がある．前のサイクルで検証された品質や効率が，後ろのサイクルで必ずしも（多くの場合）維持されないことを私たちは経験してきた．そのためにリグレッションテストや繰り返しの評価が必要となる．

3.　ブラックボックス技法によるテスト設計例

2. 4. 1 項で述べたブラックボックス技法を用いたテスト設計事例を紹介する．

3. 1　問題の定義

入力されたログイン名とパスワードが許容できるか否かを判定するプログラムを考える．

ログイン名，パスワードとも，使用できる文字は半角英数字であり，大文字と小文字は区別され，許容される文字数は以下の通りである．

ログイン名：3 文字以上 10 文字以下，　パスワード：1 文字以上 10 文字以下

入力されたログイン名とパスワードが許容できるのは，以下の条件がすべて成立する場合である．

①ログイン名，パスワードともに有効である（上記条件を満たす）

②ログイン名が存在する（登録されている）

③パスワードが合致する（登録されているものと合致する）

3. 2　ビジネスロジックの分析→テストシナリオの作成

このプログラムの使命である判定の論理を分析する必要がある．ここでは表 1 のデシジョンテーブルを使用したが，状態遷移図やフローチャートなどの技法が有効な場合もある．Y は条件の成立，N は条件の不成立，－はどちらでもよいことを意味する．各列 TS1 ～ TS5 をテストシナリオと呼ぶ．TS1 はポジティブシナリオ，他はネガティブシナリオである．障害発生

時にテスト不足の責めを受けるのは，ネガティブシナリオが不足している場合が多い．

表1　ディシジョンテーブル

条件	TS1	TS2	TS3	TS4	TS5
ログイン名は有効か	Y	N	Y	Y	Y
パスワードは有効か	Y	-	N	Y	Y
ログイン名は存在するか	Y	-	-	N	Y
パスワードが合致するか	Y	-	-	-	N
結果	許可	拒否	拒否	拒否	拒否

3. 3　入力データ仕様の分析→テストデータの選定

正しいログイン名，パスワードとして使用すべきテストデータについて検討する．ログイン名，パスワードに使用できる文字は次のとおりであり，総数は 62 文字となる．

①半角英字（小文字）26 種，②半角英字（大文字）26 種，③半角数字 10 種

ログイン名は 3 文字以上 10 文字以下であるから，正しいログイン名の可能な数は，

「$(62)^3+(62)^4+\cdots+(62)^{10}$」となる．これが（現実的には）天文学的な数字であり，全ての可能性をテストし尽せないことは明らかである．上記①②③それぞれを同値クラスと考え，同値クラス内のデータとして任意のひとつを選定すればよいとするのは合理的である[*8]．ただし同値クラスの組合せとしては表 2 のような考慮は行うべきであろう．

表2　同値クラスの組合せ

	1	2	3	4	5	6	7
①半角英字（小文字）	✓			✓	✓		✓
②半角英字（大文字）		✓		✓		✓	✓
③半角数字			✓		✓	✓	✓

次に，使用できる文字数の制約を考える．ログイン名は 3 文字以上 10 文字以下である．境界値分析の考え方に立てば，テストすべきログイン名の長さは｛3, 4, 9, 10｝の 4 種となる．プログラムの中には恐らく「3 <= 文字数」という条件判定を含むが，「<=」ではなく，誤って「<」「>=」「>」などとプログラムしてしまうことがあるので，3 だけでなく 4 もテストデータに加えておくことで，予期せぬ結果が得られたときの原因の見当ををつけやすくなる．パスワードの文字数も同様に｛1, 2, 9, 10｝を考えればよい．

最後に，不正なログイン名，パスワードについて考える．文字長の不正データとしては，ログイン名は｛2, 11｝，パスワードは｛0, 11｝の検証を行えば十分であろう．文字種の不正として，特殊文字｛@#$%^&*!｝や空白文字，全角文字など多くの可能性があり，テストデータの選定に苦慮するかもしれない．ひとつの解決策は，代表的な特殊文字をいくつか選んでテストして C1 カバレッジを測定し，未実行パスを確認することである．普通のプログラムであればそれで所定のカバレッジを満たす．もし「文字 == '@'」「文字 == '!'」などの条件分岐があ

るためにカバレッジを満たさなければ，テストケースを補充すればよい．

3.4　テストケースの作成

前述のテストシナリオとテストデータを組み合わせてテストケースを作成する（表3～表7）．テストシナリオとテストデータの総当りのケースを選定する必要はない．テストシナリオの中に，そのシナリオに対応するテストデータを漏れなく設定すればよい．期待される結果を明記しておくことは重要である．当然のことではあるが，意外な盲点になっていることが多い．

表3　TS1のテストケース

TC#	TCタイプ	ログイン名	パスワード	期待される結果
1-1	＋	abc	a	許可
1-2	＋	ABC	A	許可
1-3	＋	123	1	許可
1-4	＋	AbcD	Aa	許可
1-5	＋	abc123	abc123	許可
1-6	＋	UNISYS123	UNISYS 567	許可
1-7	＋	Unisys1234	Unisys5678	許可

表4　TS2のテストケース

TC#	TCタイプ	ログイン名	パスワード	期待される結果
2-1	－	Ab	Abc123	拒否
2-2	－	“”	Abc123	拒否
2-3	－	Unisys12345	Abc123	拒否
2-4	－	Abc!	Abc123	拒否
2-5	－	A bc	Abc123	拒否

表5　TS3のテストケース

TC#	TCタイプ	ログイン名	パスワード	期待される結果
3-1	－	Abc123	“”	拒否
3-2	－	Abc123	Unisys12345	拒否
3-3	－	Abc123	Abc!	拒否
3-4	－	Abc123	A bc	拒否

表6　TS4のテストケース

TC#	TCタイプ	ログイン名	パスワード	期待される結果
4-1	－	NotPresent	Abc123	拒否

表7　TS5 のテストケース

TC＃	TCタイプ	ログイン名	パスワード	期待される結果
5-1	―	Abc123	Notmatch	拒否

4. 新たな取り組み

日本ユニシスの品質保証部は，更なる品質強化へ向けた今後の活動の一環として，テスト技術に焦点を当てた以下のような活動を開始した．

1）　テストプロセス標準の確立

戦略的テストプロセスのガイドラインと標準を策定し，日本ユニシスのシステム構築のエンジニアリングプロセス標準（ISEP）への組み込みを行っている．テストの意味と原理を正しく理解するための情報提供と，この理解の上に立脚した合理的・経済的なテストプロセスの構築と普及活動が目的である．

2）　テストアセスメントの実施

日本ユニシスの品質保証部が実施する QAR（第三者品質保証レビュ）では，成果物や管理資料等の現物確認を行っているが，更にこの活動を拡張する形で，テスト工程の成果物の抜き取り検査を開始した．現物のプログラム品質の早期確認と問題の早期発見，開発プロジェクト及び協力企業への啓発活動と指導の強化を図るのが目的である．

3）　W モデル型開発[*9] の推進

開発の初期段階から，「開発チーム」＋「テスト/品質保証チーム」の編成を採用しているプロジェクトのソフトウェア品質は概して良好である．要求定義工程からテスト専門チームがプロジェクトに参画し，開発チームは「如何に作るべきか」を考え，テストチームは「如何にテストすべきか」を考える．これにより開発チームの設計の曖昧さや不備が明らかになることも多く，設計レビュとしての効果も大きい．このような W モデル型開発を，他の多くのプロジェクトへ展開し品質のフロントローディングを推進している．

5. お　わ　り　に

ソフトウェアテスト技術について，一般のテスト技術教育コースでも誤解を招くような指導がなされている場合がある．テストの意味や原理について正しく理解しておかないと，合理性に欠くテストを行ってしまう危険もある．例えば，「物理設計とコード作成を別の技術者が担当する分業制をとり，コード作成者がホワイトボックス技法を使用して単体テスト設計とテスト実施を行い，経路網羅（PC）相当の網羅性までテストする」という戦略は合理性を欠いており，不経済なテスト[*10]，無意味なテスト[*11] を行うことになる．テスト工程も含めた開発委託を行う場合には，テストを担当する若手・中堅の技術者だけでなく，プロジェクトマネージャや管理層も，改めてテストの意味と原理について見直し，適正なテスト戦略の立案と明確な品質基準を定め，委託先との共通理解を得ておく必要がある．

本稿で述べた活動のきっかけとなるインド STAG 社でのテスト教育受講のチャンスを頂い

た稲泉常務に感謝したい．また，IVS 社谷崎社長，STAG 社 Ashok 社長他，両社のテストエンジニアの方々には数多くのご支援とご助言を頂いた．この場を借りて感謝の意を捧げたい．

＊ 1　STEM は，STAG software 社の登録商標である．

＊ 2　箱に白いペンキを塗っても中は透けて見えないので，むしろ「グラスボックス」「クリアボックス」と呼ぶべきだという主張もある．

＊ 3　厳密に言えばそうとも言い切れない．ニュートンラプソン法を用いたプログラムでは，プログラムの仕様に，収束計算の初期値や，収束判定条件など，数値解法独特の条件が折込まれる可能性があるため，ブラックボックス技法によるテストケースが根の公式によるプログラムのそれと異なることはあり得る．

＊ 4　参考文献[4]では，「y=2x」とプログラムすべきところを誤って「$y=x^2$」とした場合，テストデータとして「2」を使用すると誤りに気がつかないこと（プログラムが正確であると判定してしまう偶然）をこう呼んでいる．別の参考文献[5]はこの問題はブラックボックステストでは検出できないのでホワイトボックステストの方が有効であるとしている．本稿ではブラックボックステストとは言わず，一貫してブラックボックス技法という用語を用いている．テスト設計に利用できるのはブラックボックス技法であり，ホワイトボックス技法はそうではない．最も難しい Path coverage をクリアしても，そのプログラムが仕様を満たすことを保証したことにはならない．目的（検出できる欠陥のタイプ）が異なる技法についての優劣の議論は，得てして本質を見失うことにつながるので注意が必要である．

＊ 5　10 以下は良好，30 を越える場合は構造の見直しが必要，50 を越えるとテスト不能，75 を越えると誰がどんな改修をしても誤りを犯すという報告がある[8]．

＊ 6　東証誤発注問題では，誤発注直後の数分間の内に，証券会社と東証の担当者それぞれが事態を認識したものの，東証の売買システムが取り消し注文を受け付けなかったと報じられている．システム側は入力ミスを疑い警告を発したが，その警告は無視された．新規上場株のみなし処理が発生し，かつ買い気配から一気に売り気配に変わる段階でのみ発生する不具合であり，障害発生の数年前の運用テスト時の改修で混入した問題とのことである．筆者の想像では，担当者は恐らくこの改修がそのような影響を及ぼすことに思いが至らずテストの必要性も感じていなかったのではないか．その状況で，テストによりこの問題を検出しなければならなかったとすれば，「取り消し注文」の機能に最大級のリスクが設定されていて，このテストが自動化（省力化）されていて，プログラム改修後の何らかのタイミングでリグレッションテストを実施することが規定されていて，その中にこのテストが組み入れられていなければならなかったと思う．これが「当然そうあるべきだった」かどうかを論じるつもりはないが，テスト戦略の第一番に，リスクを評価しておくことの重要性は十分に示唆しているのではないだろうか．

＊ 7　変数の初期化忘れや動的資源の割り当て/解放不正など「Data flow」にまつわる問題はプログラミングの初級者が犯し易い問題であり，統合テスト以降の工程で発覚すると再現性が低く上級技術者の貴重な工数を奪うたちの悪い欠陥となる．この種の欠陥の検出には，コードレビュ，静的検査ツール（広義のテスト）が利用されることが多いが，単体テスト工程でホワイトボックス技法を適用するのも有効である．動的資源の割り当て/解放のプログラムインタフェースに検証用のコードを組み込んでおけば，単体テストでの自動的な検証も可能となる．

＊ 8　文字種の判定を文字コードを数値とみなして大小関係で判定しているとしたら，境界値分析の考え方で，各文字種の境界「a」「z」「A」「Z」「0」「9」を選定すべきかもしれない．

＊ 9　開発チームとテスト/品質保証チームが連携（並行）して V モデルに沿った開発を行う形態を意味する．2 チームが V に沿って動くことによりその軌跡 V が重なって W に見えることによる．

＊10　膨大なコストをかける割には得るものが少ないテスト．同値クラス分割や境界値分析など，テストすべきデータ数を合理的に極小化する技術は，これ以上やっても時間の無駄であることを教えてくれる．

＊11　何も得られないテスト．得られるのは本来行うべきテストがなされていない，バグを抱えたプログラムだけ．（＊ 4）の例で，物理設計者が誤って物理設計書に「$y=x^2$」と書けば，プログラマはこれを信じてプログラムを作成するだろう．プログラマに，プログラム仕様に基づくブラックボックス技法を用いたテスト設計を命じるのは無理があると考え，ホワイトボックス技法を用いたテスト設計とテストの実行を命じたとする．彼は言われたとおり「$y=x^2$」が実行されることを確認し，安心してテストを終了する．この種の問題は決して笑い話ではなく，実際に我々の周りで起きている怖い現実である．

参考文献
［1］ Edward Yourdon，デスマーチ第2版ソフトウェア開発プロジェクトはなぜ混乱するのか，日経BP社，2006.5
［2］ https://issj.nuis.jp/teigen.pdf，東証における誤発注問題に関する提言，情報システム学会東証問題検討プロジェクト，2006.12
［3］ stag PCSTEplus program（テスト技術者教育テキスト），STAG社，2006
［4］ G.J.Myers他，ソフトウェアテストの技法第2版，近代科学社，2006.8
［5］ Rick D.Craig他，体系的ソフトウェアテスト入門，日経BP社，2004.10
［6］ Lee Copeland，はじめて学ぶソフトウェアのテスト技法，日経BP社，2005.11
［7］ Rex Black，ソフトウェアテスト実践ワークブック，日経BP社，2007.1
［8］ Capers Jones，ソフトウェア開発の定量化手法第2版，共立出版，1999.7
［9］ Boris Beizer，ソフトウェアテスト技法，日経BP社，1994.2
［10］ Boris Beizer，実践的プログラムテスト入門，日経BP社，1997.8
［11］ ソフトウェア・テストPRESS Vol.1-4，技術評論社，2005.6-2007.1
［12］ 三宅晴夫，要求管理のプロセス紹介，ユニシス技報第67号，Nov.2000

執筆者紹介 大 塚 俊 章（Toshiaki Otsuka）
1990年日本ユニシス(株)入社．汎用機通信制御プログラムの開発・保守，および，医療情報システムの開発・保守に携わった後，2005年より品質保証に関する作業に従事し，現在は品質保証部プロジェクト支援室に所属．

荻 野 富 二 夫（Fujio Ogino）
1979年，東京工業大学大学院修士課程修了（情報科学専攻）．同年日本ユニシス（株）入社．入社以来CAD/CAMシステムの開発に携わり，2007年4月より品質保証部プロジェクト支援室長．