基幹システムにおけるテスト十分性の確保

Securing Exhaustive Testing in Mission-Critical System

飯 田　志 津 夫

要　約　基幹系システムの開発規模や複雑性は増大を続けているが，企業活動に欠くべからざる存在として，また，社会のインフラストラクチャとして，高い信頼性が求められている．信頼性を確保するには，非機能要件も含めて十分なテストを実施する必要があるが，テスト活動だけを対象としても品質の向上には限度がある．先ず，テストの有効性を確保できる環境を，開発活動全体を通して確立する必要がある．次に，複数の技法を組み合わせてテスト設計を実施しテストの網羅性を高めるが，この結果テスト項目数が増えコストや納期に悪影響を与えてはならない．このため，効率化の観点から，テストの網羅性を確保しつつ合理的にテスト項目数を削減する技法の有効性を示す．

Abstract　Although the development scale and complexity of the mission-critical system are continuing to increase, the high reliability is required as the indispensable presence for corporate activities, and social infrastructures. In order to secure the reliability, it is necessary to carry out the exhaustive testing also including non-functional requirements, but there are limits to improvement in quality only for testing activities. First, it is necessary to establish the environment where the validity of testing can be secured throughout the whole development activities. Next, although the test design is carried out combining two or more techniques and the testing coverage is improved, as a result, the number of testing items increase, and it must not have a bad influence on neither cost nor time for delivery. For this reason, the validity of the technique which reduces the number of testing items rationally is shown from a viewpoint of increase in efficiency, securing the testing coverage.

1. は　じ　め　に

プロダクト品質保証では，テストの十分性が特に重視される．テスト工程で除去できなかった欠陥は，稼働後に障害となって顕在化するが，1 件の障害を解決するまでに要するコストは，欠陥を作り込んだ工程に近いほど少ないといわれている．稼働後に障害が発生すると，その対応コストと利害関係者に与える影響の双方が非常に大きなものとなる．

日本情報システム・ユーザー協会（JUAS）調査[1]では，ユーザによる総合テストで観測された不具合件数の 13%程度に相当する不具合件数が稼働後に発生していると報告されている．また，情報処理推進機構 ソフトウェア・エンジニアリング・センターの「ソフトウェア開発データ白書 2007」[2]によれば，システム稼働後 6 ヶ月間のソースコード 1,000 行当りの不具合発生数（不具合密度）の平均が，0.121 件となっている．

本稿では，基幹系システムを対象とするが，その特性は，開発規模が大きく，高効率と高信頼性が求められ，利用者が多数となるということに要約される．このような特性を持つシステムで，網羅性の高いテストを設計するための技法と，コストと開発期間の制約の中で効率的にテストを実施するために用いるべき技術を明確にする．更に，テストの有効性を向上させるた

めに，開発工程全体を通して実施すべき活動内容についても考察する．

2. テストの現状

情報システムが社会のあらゆる局面に組み込まれネットワーク化されており，高い信頼性が求められることから，テストでは，単にバグを検出するだけに止まらず，検証すべき範囲が拡大し難易度が高まっている．一方，提供する機能の高度化と拡大から，開発するシステムの規模が増大しているが，日本ユニシスの 2005 ～ 2007 年度の開発実績調査でも，図 1 に示す通り開発規模の増大に比例してテスト工数の占める割合が増加している．

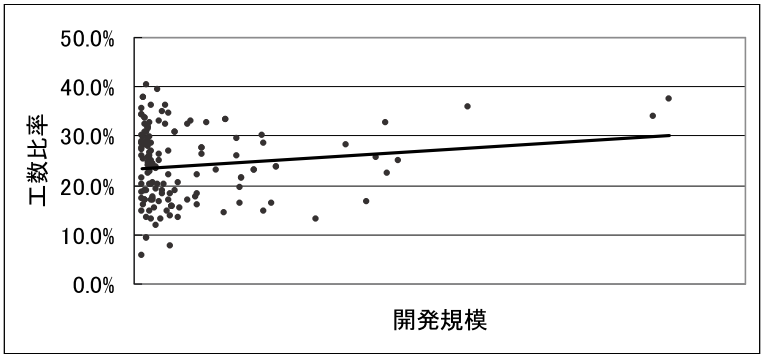

図 1　開発規模とテスト工数比率の関係

ところが，ビジネス面での情報システムの重要度が増すほど，より短期間で効率的な開発が要請されており，十分なテスト工数と期間を確保できていない状況が見られる．テストの網羅性を向上させつつテスト作業を効率化するため，テスト項目の単なる絞り込みに終始するのではなく，開発工程全体を俯瞰して，テスト工程のプロセス品質と生産性向上に向けた見直しが求められている．

2.1　開発規模の増加がテストに与える影響

開発規模の増加がテスト作業を難しくし，テスト工数の占める割合を高くする理由として以下の事項が考えられる．

1）　データ項目数が増え，各データ項目が取りうる値の組み合わせ数が増加する．
2）　提供する機能数の増加に伴い，複数の機能間を連携する処理パターン数が増加する．
3）　基幹システムの場合，発生が稀な異例ケースにも全てシステム対応が求められるため，処理ロジックの複雑性が増す．

2.2　非機能要件の重要性増加

機能要件の充足はもとより，非機能要件の利用者の観点での検証が重視されるようになってきた．特に，基幹系システムでは，以下の点から非機能要件の重要性が増す．

1）　多様なバックグラウンドを持つ利用者が使用することから，簡潔で分かり易いユーザインタフェースが求められる．
2）　一般的に基幹系ではデータの発生件数が多く，業務上の締日や月末，期末などのピーク

日やピーク時間帯が存在するので，十分な効率性を確保する必要がある．

3)　システムの重要性が増し，高信頼性が求められるため，システム基盤構築の難易度が高まる．

非機能要件のテストが本格的に実施されるのは，最終工程であるシステムテストとなることが多い．ところが，この段階で非機能要件の未充足が判明しても，アーキテクチャレベルに問題があることが多く，設計工程まで巻き戻す大きな手戻りが発生する．また，非機能要件の検証には，特別なスキルやテスト環境が必要な場合が多く，十分な検証が実施されないまま稼働を開始してから処理能力の不足が明確となり，急遽ハードウェア能力の増強で対応する場合もある．

2.3　ウォーターフォール型開発に起因する課題

機能要件に関しても，上記の非機能要件と同様に，システムテスト工程にならないと顧客要求の充足が最終的に確認できないという課題が惹起される．ウォーターフォール型開発プロセスを採用している場合は，要件定義と設計工程に対応するテスト工程の関係を表すＶモデルで示される通り，顧客と開発に合意した要件は，最後の工程であるシステムテストにならないと確認できないためである．

開発の早い時点で確認するため，プロトタイプやスパイラル開発などの手法が提唱されているが，ビジネスの成否を左右する基幹系システムでは，稼働時期の厳守が必須であり，開発に多数の要員が参画することから，大規模開発では，その問題点を認識しつつも，ウォーターフォール型開発プロセスを採用せざるを得ないのが実状である．

2.4　稼働後に障害が顕在化する理由

テストの十分性を確認するために，通常は以下の方策が用いられているが，これらは代用特性である開発プロセスの品質を評価しているもので，真の品質であるプロダクト品質を評価していない．

1)　ホワイトボックステストでカバレッジを計測し，未テスト箇所を無くする．

2)　バグの検出目標値を設定し，目標の達成状況を評価する．

3)　バグの検出状況をグラフ化して，潜在バグ数を予測し信頼性の向上度を確認する．

2.4.1　ホワイトボックステストでのカバレッジの限界

ホワイトボックステストは，入力とした設計書へのプログラム作成者の理解力と意図をテストしており，設計書通りに作成されていることは検証していない．プログラム作成者が過った判断をしていたり，機能の実装が欠落していても判明しない．また，カバレッジでは，条件網羅（C2 メジャー）を実施するとテスト項目が膨大な件数となるため，実用上は命令網羅（C1 メジャー）で止めているのが現状である．

2.4.2　バグ検出目標の設定

予めバグ検出目標件数を設定し，その達成状況で品質を判断するもので，日本電気の品質会計[*1]が有名である．一般的には，開発開始時に過去の経験からバグ件数を予測し，開発中に予測値を見直すが，予測の精度が問題となる．組織として開発プロセスが確立し，十分な過去の

実績値（パフォーマンスベースライン）を保持している場合は信頼性の高い予測が期待できるが，プロセスが安定していない場合は予測値のバラツキが大きくなり，実力と乖離した目標値を設定する可能性がある．なお，CMMI[*2] [3]では，前者のような実力をそなえた組織を「レベル 4：定量的に管理されたレベル」として，高成熟度組織と位置付けており，そのハードルは高い．一方，後者のような未成熟な組織の場合は，品質のバラツキが大きいので，現物を精査しないと実態が把握できない．

2. 4. 3　グラフによる信頼度向上の可視化

バグの発生件数の累積を時系列でグラフ化すると，テスト活動の進捗につれて単位時間当たりのバグの発生件数が次第に減少し，信頼性が向上している様子が観察できる．これを，「信頼度成長モデル」や「傾向曲線モデル」と呼び，テスト終了時のバグ数を予測[4]し，曲線の傾きがサチュレートしてきたことで収束を判断する．

このようなモデルは変曲点を持っており，モデルが適切に機能するためには，変曲点後のデータが必要である．Stephen H. Kan[5]によれば，妥当な曲線の適合と予測を提供するためには，システムテスト全体の約 60%弱のデータが必要であるとされており，工程の前半では判断に利用できない．また，テストの実施順序に偏りがあると，対象箇所の難易度や開発者の力量のバラツキによってグラフの形状が歪むことがあり，判断を誤らせる可能性がある．

3.　テストの有効性向上に向けて

必要にして十分なテストを実施する目的は，高品質の成果物を開発することである．これをテスト工程だけで実現しようとすると，不良品の顧客への流出を防止するため検査を強化することになるが，検査に投入できるコストと期間には限度がある．検証活動としてのテストが有効に機能するためには，開発工程全体で品質を作り込む必要がある．正しい要件定義，正しい設計，正しい実装，正しい検証が担保されることをテストの観点から概観する．

3. 1　開発工程毎の精度向上

開発の各工程で実施する活動の精度向上策を，以下の観点から考察する．

1）　要件に起因する問題への対応

要件には，発注者から明示された要件だけでなく，業界や発注者の企業内で常識となっており改めて明示されない暗黙の要件が存在する．更に，実際に操作しないと，実感として把握することが困難であるという情報システムの特性から，「JIS　X　0129-1：ソフトウェアの品質[6]」に記載されている通り，発注者も気付いていない要件が存在し，更に明示されたことで要件が変化するため，設計の開始前に品質要求を完全に定義できないとされている．

このため，先ず「共通フレーム 2007[7]」では，開発側だけに要件定義作業を押し付けるのではなく，上流工程にかかわるステークホルダの役割を明確にしており，UML[*3] などのモデリング技法を用いて図示することで，発注者の理解力の向上をはかっている．

2）　デザインレビューの質的向上

デザインレビューは，上流の工程で品質を確保するために，専門家やステークホルダなどの異なる背景を持つ複数の要員によるさまざまな視点でのレビューである．これによって，アーキテクチャなどのシステムの根幹にかかわる部分に問題がないことを確認する．このレ

ビューを効果あるものとするためには，十分な技術力や見識を備えたレビュアを確保することが必須となる．

3)　設計仕様の厳格性追求

設計工程では，前工程の成果物を入力として，詳細化と変換を行ない新たな設計書を作成する．この過程で誤りが混入したり，曖昧な記述や仕様の欠落などが発生することがある．これを回避する技法として，開発の最初から正しいものを作成していくという「クリーンルーム手法[8]」がある．クリーンルーム手法では，テストを検証の手段には用いず，テストはあくまでも完成したソフトウェアの品質を評価するための工程と位置付けられている．また，仕様を厳密に記述し曖昧さを排除する「形式手法[9]」も提唱されている．

しかしながら，理論として完全であっても，高スキル保有者を対象とした方法論を実際の開発現場に適用しようとすると，多くの要員が関与する大規模システム開発では要員スキルや開発期間など，解決しなければならない問題が多く採用が難しい．

4)　静的テスト技術

テストには，実機上での動作を確認する動的テストと，レビューやツールを使用して確認する静的テストがあり，目的に応じて効果の高い方法を使用する．非機能要件のうち保守性と移植性に関しては，設計文書とソースコードを目視とツールで確認する必要がある．保守性や移植性に問題があっても直ぐには顕在化しないが，後刻の保守コスト増加要因となる．

このため，設計の最初で十分に検討し，達成に向けて守るべきルールを基準書や規約で定め，その設計方式が開発の各工程で遵守されていることを厳格に検証する必要がある．動的テストでは，プログラム構造（解析，変更，試験が容易な構造）や使用している API と関数を具体的に確認できないためである．保守性を向上させるためには，ソースコードだけでなく保守用ドキュメント開発規約の遵守度も検証する必要がある．

3.2　テスト活動を支援する構築技術

テスト活動を支援し，更にテストの位置付けを見直す構築技術に関して考察する．

1)　テストを軸とした開発

開発の中でテストを中心に据える，アジャイル開発におけるテストファーストや，テスト駆動開発[10]が提唱されている．開発するシステムの特性とチームを構成する要員のスキル次第で，非常に有効な開発技法であるが，各開発者の担当する役割の範囲が広く高スキルが求められるため，プログラム作成工程で多数の要員を動員する必要のあるウォーターフォール型プロセスでは適用が難しい．

2)　モジュールの独立性

プログラムモジュールに分割する際は，各モジュールの独立性が高まるように配慮することで，他モジュールとの相互作用を減少させることができる．これによって，相互作用に関連するテスト項目を削減できるだけでなく，保守性の高い構造が実現できることになる．独立性を表す尺度には，以下の二つの基準[11]が用いられる．

ⅰ）凝集度

ⅱ）結合度

実現する機能がモジュール内に絞り込まれている程度を示す「凝集度」は，情報隠蔽（カプセル化）から導出される概念である．これが弱いと，修正による副作用（デグレード）が

発生しやすくなる．モジュール間の関連性を示す「結合度」は，モジュール間のインタフェースや参照関係が複雑なほど強くなり，テストすべき組み合わせ条件の選定が難しくなる．このような観点からモジュールの独立性を確認することによって，品質に対する脆弱性を早期に把握することが可能である．

3） 部品化と再利用

古くて新しい話題に「部品化と再利用」がある．オブジェクト指向が期待されているが，情報隠蔽の観点からは非常に有用である．一方，継承に関しては，自然法則が支配する世界では，その有効性が喧伝されているが，ルールを恣意的に決められるビジネスの世界では，自社内で開発方式と基盤を統一しないと難しい．部品の粒度にもよるが，ERP[*4] は業務のベストプラクティスを詰め込んだ巨大な部品ともいえる．また，SOA[*5] では，サービスの集まりとしてシステムを構築するが，コンポーネント間の通信方法の特性から，即時処理の要求に応えることは難しい．

更に，長期間にわたって複数の開発で必要となる要件を予測し，共通利用するアーキテクチャや基盤を予め準備しておくことで，ソフトウェアの製品系列を展開できる「ソフトウェアプロダクトライン[12]」がある．大量生産と同じような低コストで，個々の顧客毎の要望に応える商品を提供するマスカスタマイゼーションを実現しようという動きである．

4） 頑健性の向上

Glenford J. Myers がテストに関する古典的名著[13]で述べている通り，全ての条件をテストすることは不可能である．このため，バグが発生しても停止せず，その拡大を防止する頑健性の高いシステム，つまり，品質副特性の「障害許容性」を高める必要がある．これを実現する手法に「アサーション」がある．あるモジュールの呼び出しを行っているとき，モジュールが正しく動作している際に満たされるべき条件として，以下の3条件を規定しておき，これが満たされない限り何らかの誤りが発生したとみなして処理を中止することで，致命的な誤りの発生を回避する手法である．

ⅰ）事前条件：モジュール呼び出し時に引数が満たすべき条件

ⅱ）事後条件：モジュールでの処理終了時に引数が満たすべき条件

ⅲ）不変条件：モジュールによる処理により変化しないものを規定する条件

3.3 個人レベルの技術力向上

組織レベルでの継続的な改善モデルである CMM/CMMI[*2] を示した Watts S. Humphrey は，同じフレームワークで個人レベルでの改善モデル[14]である PSP[*6] および，チームレベルでの改善モデル[15]の TSP[*6] を提唱している．大規模開発では組織レベルでの対応が必須であるが，最終的には個人の力量に依存するウエイトが高い．このため，開発活動を個人レベルで測定し，そのフィードバックを通して学習し改善をはかっていくことが重要である．

4. テストの網羅性向上のアプローチ

テストの網羅性を向上させるためには，種々の仕組みを動員してテスト設計を実施し，その結果をクロスチェックして漏れを無くする方法と，過去の経験やベストプラクティスの蓄積などの先人の知恵と成果を活用し，テスト設計の精度を高める方法が考えられる．

4.1　テストの観点からの設計内容検証

機能設計と同期（並行）してテスト設計を実施する「W モデル」が提唱されている．工程の流れが，前述の V モデルに対して，W 字に見えることからこう呼ばれる．この場合，テスト項目が設計できるか，結果を検証できるかといったテストの観点から要件定義と設計の内容を検証できるので，設計の不具合や曖昧な箇所を当該工程内で検出できる．

「日経コンピュータ[16]」誌によると，東京証券取引所が 2009 年後半に稼働を目指す次世代システム「arrowhead」では，要件定義の段階から W モデルを採用し，検収テストで使用する 5,000 パターンを超えるチェックリストを要件定義の段階で既に作成しており，この段階で要件の矛盾や漏れを検出できるため，検収テストでの手戻りが減少するというメリットが紹介されている．

4.2　顧客視点での設計

顧客と開発者の認識に齟齬が生じていたり，顧客の意図を誤って理解してしまうと正しい設計書が作成されず，その設計書に基づいて実施されるテストも意味が無いことになる．これを防止するためには，顧客視点で設計を進め文書化することが重要である．このため，日本ユニシスを始めとする IT ベンダ 9 社では，顧客がシステムと直接インタフェースする部分を設計する外部設計に着目して，開発現場の設計事例から以下の観点で抽出した「コツ」を集約・整理した「発注者ビューガイドライン[17]」を公表している．

1）　誤った理解を防止したり，見つけ出すためのポイント
2）　誤った理解に導いたり，検出を難しくするポイント

4.3　非機能要件の充足

非機能要件の抽出と具体化および検証には，特別なスキルと事前の周到な準備が必要な場合が多い．従来から重視されてきた性能や障害回復，運用面だけでなく，セキュリティや環境面への配慮（グリーン IT）のウエイトも高まっている．特に，大部分の情報システムがネットワークで接続されている状況下では，セキュリティ充実の要請が強い．セキュリティ面の機能要件は満たしているが，脆弱性が存在している場合，悪意を持った攻撃にさらされると情報漏洩や権限昇格による制御権の奪取などが発生することがあるが，これは機能の充足確認テストでは検証できない．

また，連続稼働が要求される環境下では，占有したメモリ領域の解放処理が適切でないために使用可能なメモリ容量が減少していくメモリリークも大きな問題となり，コーディング標準の遵守とツールによる検査が必要となる．

日本ユニシスでは，開発したシステムが稼働を開始できる品質に到達していることを確認するために，満たすべき要件を列記した「カットオーバークライテリア」を作成し，リリース判定会議を実施している．このカットオーバークライテリアは，業態や業務種別に共通する項目と，業態独自の項目から構成されており，先ず要件定義段階で顧客との検討に使用し，暗黙の要件の顕在化や要件の抽出漏れの防止に活用している．ちなみに，非機能要件のテスト項目が充実していると，機能要件のテスト項目も十分である場合が多いので，テスト品質の指標としても活用している．

なお，非機能要求について，顧客と開発ベンダの両者で共通認識を持てるようにする方法を共同検討する業界団体の「システム基盤の発注者要求を見える化する非機能要求グレード検討

会[18]」では，276 項目の「システム基盤の非機能要求に関する項目一覧」を公開している．

4. 4　トレーサビリティの確保

V モデルは，開発工程の進展に伴い利用者から次第に遠く離れていく様子を表しており，利用者が成果物の内容を判断することが次第に難しくなってくる．このため，利用者が当初に提示した要求が，各工程で漏れなく対応され欠落が発生しておらず，余計なものも追加されていないことを確認するため，当初要求とそれを確認するテスト項目までの間のトレーサビリティを確保することが重要となる．CMMI では，成熟度レベル 2 の「要件管理」で，要件の両方向のトレーサビリティを維持することを求めている．

4. 5　モデルベースのテスト

何らかの形でテスト対象を記述したものを，「テスト設計モデル」と呼び，制御フローや状態遷移図，CRUD 図のような図や表であらわされるものだけでなく，機能一覧表も該当する．これらのモデルを用いてテスト項目を設計する技術の総称であり，何らかの条件や機能を全て網羅するテストを設計するときは，必ず用いる技術である．網羅性の観点からテスト設計を支援するため，開発するシステムの特性に応じた最適なモデルを選定することがポイントである．設計段階で作成されたモデルをテスト設計で使用するため，テストの観点からもモデルが検証されると共に，テスト設計に必要なモデルの開発を設計側に要請できるため，前述の W モデルに基づく開発に最適の技法である．更に，定式化されたモデルであるため，第三者もテスト設計の内容が理解しやすいという利点がある．

5.　テストの効率性向上アプローチ

テストの効率性を向上させるためには，ツールの使用による自動化や再利用の推進，テスト順序の工夫などの生産性向上と，テスト終了の判断基準を明確化した上でのテスト項目数の合理的な削減の二通りの策が考えられる．テスト終了の判断基準には，網羅性やバグ検出件数，残存するリスクなどが考えられるが，ここでは，テスト項目数の合理的な削減策について検討する．

5. 1　リスクベースアプローチ

リスクベーステストでは，最初にリスク分析を実施し，以下の観点からリスクを評価して，高リスクのものからテストを実施することで，欠陥の多い部分や重要な部分に優先して対応するすることができる．

1）　欠陥を作り込む可能性
2）　欠陥によってもたらされる重大性

但し，リスクベーステストではバグの見逃しを防止できないので，テスト対象部分に対するテストの網羅性の確保が必須である．リスクベーステストの特徴は，設定されたコスト内で最大限の成果を得ることにあるので，リスクの分析と評価の手法が重要なポイントを握ることになる．以下で，リスクの分析と評価の手法を検討する．

5. 1. 1　IT セキュリティマネジメントでのリスク分析手法の流用

IT セキュリティマネジメントのガイドラインである「ISO/IEC TR 13335（Guidelines for

the Management for IT Security)」では，リスク分析手法としてベースラインアプローチ，詳細リスク分析，非形式的アプローチ，組み合わせアプローチが紹介されている．この中でベースラインアプローチは，リストアップされた一覧表を利用する方法であり，比較的導入が容易である．項目が包括的にリストされていると，経験の少ない要員でも対応可能であるが，プロジェクトの特性に合わせてテーラリングが必要で，項目毎に採否の判断が求められる．一覧表の限界が，テストの限界とならないように常に見直す必要がある．

5. 1. 2　FTA の適用

社会のインフラストラクチャを構成する重要システムは，高い信頼性が求められ，何らかの不具合が発生しても完全なサービス停止に陥らない頑健性が求められている．システム障害を予防し回避する観点から，信頼性工学のアプローチを取り入れる必要性があるが，ソフトウェアはハードウェアと異なり，同じ条件では必ず同じ振る舞いをするので，冗長化が対策とはならない．

「FTA（fault tree analysis）[19]」は，望ましくない故障（不安全事象）を原因側へ末広がりに樹木状に分解し，論理的，定量的に多重故障の評価を行なうものである．これは原因追求のプロセスと同じであるが，FTA の狙いは，このことを予め設計の段階で系統的に実施し，必要な改善をほどこすことである．人的要素（誤操作，見落とし，操作忘れなど）や環境要因などまでも考慮し，2 個以上の複合した事象の組み合わせを扱うことが特徴である．FTA では，故障原因のうちで，システムに致命的な影響を与える尺度を重要度と呼んでいる．この重要度には，構造重要度と確率重要度の 2 種類があり，リスクの高低の判断に適用可能である．具体的に考慮する要因として，任務達成を妨げる要因（システムダウン），安全に対する脅威となるもの，収益に重大な影響を与える要因（重要サービスの停止）などが挙げられる．

図 2 に銀行勘定系システムの普通預金がサービス停止となる FTA のマクロなレベルでの例を示すが，社会のインフラストラクチャとして，自動機の入出金および為替取引と連動した振替入出金のサービス停止の与える影響が非常に大きい．

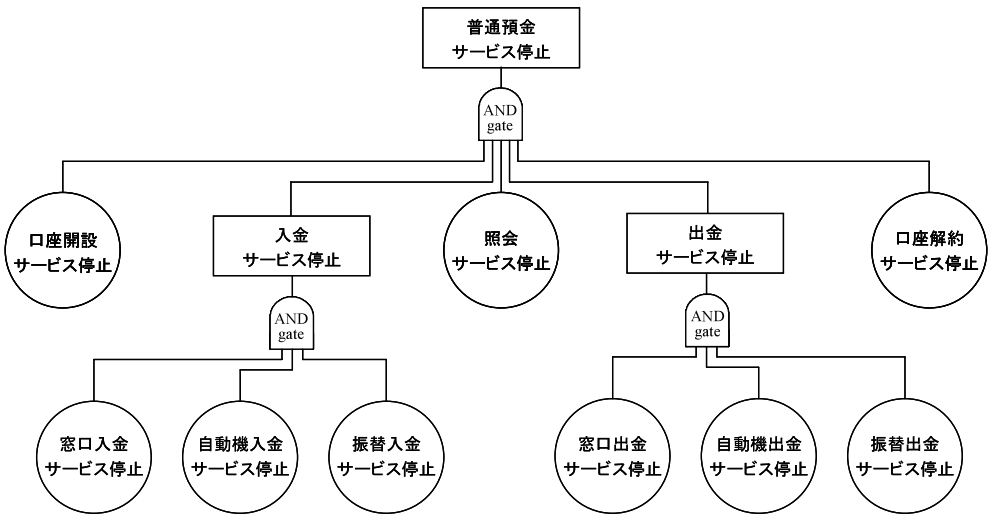

図 2　銀行勘定系システムで普通預金のサービス停止となる FTA の例

5. 2　実験計画法の適用

システムの開発規模が大きくなると，条件の組み合わせや業務シナリオのパターンが増加する．このため，開発規模で正規化したテスト項目数は，単体テストでは殆んど増加は見られないが，結合テストとシステムテストでは増加する傾向が見られる．日本ユニシスの 2005 年度の開発実績によれば，1 件のテスト項目の設計から完了までに要する工数は，単体テストを 1 としたとき，結合テストで約 3.0，システムテストでは約 5.4 となっており，結合テスト以降のテスト項目数の合理的な削減が，テスト工程全体の効率化に寄与する．実験計画法[20]では，実験条件を直交表に割り付けることで実験回数を削減するが，この直交表をテスト項目設計に適用して，組み合わせテスト項目数を削減する事例が報告されている[21]．

5. 2. 1　直交表の適用

一般に複数のパラメータ（因子と呼ぶ）を組み合わせる実験では，少なくともパラメータが取り得る値（水準と呼ぶ）の数の積の回数だけ実験数が必要になり，因子数が多くなると実験回数は膨大な数になってしまう．直交表は，任意の 2 因子間の全水準の組み合わせが同数回ずつ現れるという特徴を持っている．このため，因子間に交互作用がない，つまり直交している場合は，直交表を用いることによって，多くの因子に関する実験を比較的少ない回数で行うことができる．例えば，各因子が 2 水準（True/False）の場合，3 因子の組み合わせは 8 となるが，直交表上では 4 となる．同様に，7 因子の場合は，128 が 8 に，15 因子の場合は，32,768 が 16 となり，組み合わせる因子数が多いほど削減効果が高くなる．

この直交表を組み合わせテストに応用すると，互いに独立な複数の因子が特定の値で組み合わされた時に，バグが発生しないかを確認することができる．表 1 に 2 水準の 7 因子の組み合わせとして，普通預金の入金処理テスト項目を直交表 L8 に割り当てた例を示す．2 因子間の全ての組み合わせが網羅されている．表 1 では，主要な条件の組み合わせを示しているが，実務では，各普通預金元帳オブジェクトの状態（残高と付利単位の関係，未決済自動振替データ有無，事故届有無，便宜扱有無，本日自動機出金額累計と限度額の関係など）や時間軸（利息元加日や金利変更日前後など），環境（営業店端末や自動機の種類）などにより振る舞いが変わり組み合わせる因子数が増加するので，削減効果が一層期待できる．

表 1　普通預金入金処理テストの直交表 L8 への割り当て例

因子 / テスト	通帳有無	未記帳有無	他店券有無	未資金残高	未決済自振	勘定締上	起算日取引
テスト項目1	無通帳	未記帳無	現金入金	無	無	締上前	無
テスト項目2	無通帳	未記帳無	現金入金	有	有	締上後	有
テスト項目3	無通帳	未記帳有	他店券入金	無	無	締上後	有
テスト項目4	無通帳	未記帳有	他店券入金	有	有	締上前	無
テスト項目5	有通帳	未記帳無	他店券入金	無	有	締上前	有
テスト項目6	有通帳	未記帳無	他店券入金	有	無	締上後	無
テスト項目7	有通帳	未記帳有	現金入金	無	有	締上後	無
テスト項目8	有通帳	未記帳有	現金入金	有	無	締上前	有

5.2.2　基幹システムでの考察

D. Richard Kuhn[22]は，1因子単独および2個の因子間の組み合わせで発生するバグが，全体の7～9割を占めると報告している．日本ユニシスの基幹システムで稼働後に発生したバグ（障害）を分析したところ，図3に示す通り1因子に起因する障害が38%～65%，これに2因子の組み合わせで発生した障害を加えると，全体の79%～100%を占めており，高い効果が見込めることが判明した．

システム	組合せ因子数				開発規模（相対）	障害検出率（相対）
	1	2	3	4以上		
A	38%	79%	96%	100%	1	1
B	43%	100%	－	－	0.8	0.4
C	52%	100%	－	－	0.6	1.5
D	64%	97%	100%	－	0.8	1.5
E	65%	100%	－	－	0.5	1.6

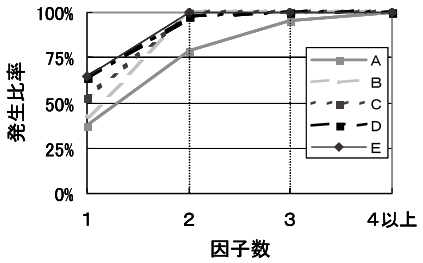

図3　基幹システムで稼働後に発生した障害の因子数別分析

また，図3の表からは，1因子での障害の占める比率と，開発規模で正規化した障害検出率に逆の相関性が見られるため，開発規模が大きく障害検出率の高い「システム－D」に関して，稼働後8ヶ月間に検出された障害の月別の因子数別比率を図4に示す．障害発生件数は，稼働月をピークに毎月大幅に減少し品質副特性の「成熟性」が向上してくるにつれて，1因子に起因する障害の占める比率が低下する傾向が見られる．稼働後の時間の経過につれて，複数の因子が組み合わさったいわゆる難しい障害が検出されてくるという実務経験とも合致する．この状態を一般的に「枯れてきた」と称するが，潜在する障害の多寡の判断に1因子に起因する障害の占める比率を利用できる可能性がある．この技術を用いると，少ないテスト項目数でプロダクト品質の良否を比較的早く把握できるので，開発側のテスト効率化に資するだけでなく，品質保証側のテストの効率化や探針[*8]にも有効である．

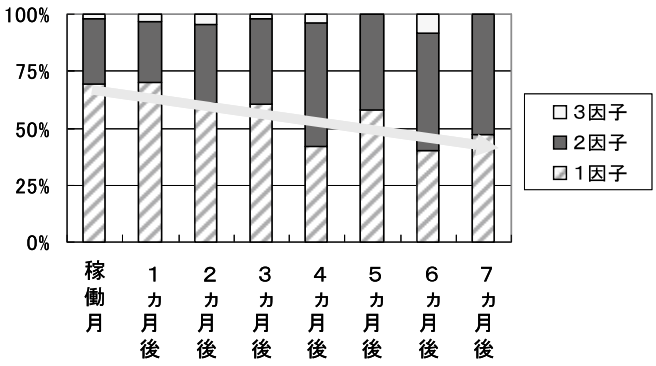

図4 「システムーD」の稼働後8ヶ月間の障害の月別の因子数別比率

直交表では，通常はエラーとなる値（無効同値クラス）は因子が取り得る値としては採用しない．採用するとエラーとなり，出力結果が得られないためである．従って，事前に単体テストで，エラーデータが適切に処理されていることを検証しておく必要がある．更に，因子が取

り得る水準数を減らすと組み合わせ数を減少できるため，ブラックボックステスト技法の同値分割と境界値分析を使用し，更に業務要件からの代表値の選定などを通して絞り込む必要がある．この技法は，組み合わせ数が多くなるほど効果を発揮するが，適用方法の理解が前提となるため，最初は対象を限定して経験を深めるのが適切である．特に，発生頻度が実務では非常に少ない組み合わせ条件に適用すると効果が高い．

6. お わ り に

全ての条件をテストし尽くすことは現実的には困難であるが，テストの網羅性の向上を通してプロダクト品質を高めていく努力を怠ってはならない．これは，テスト単体で解決できることではなく，有効性が示されている各種の技法を組み合わせて，開発工程の全体にわたって適用していく必要があり，ソフトウェアエンジニアリングへの深い理解が求められる．

一方，テストの網羅性の向上が，単なるテスト項目数の増加となり，テストに要するコストと期間の増大をきたしては片手落ちである．テストの網羅性を確保しつつ，合理的にテスト項目数を削減する効率化の観点が必要である．製造業を始め多くの分野で評価の高い実験計画法は，更なる高信頼性を求められつつ，開発規模の拡大と複雑性の増加が止まらない情報システムのテストにおいても強力な技術である．IT 業界で馴染みの少ない概念や用語などが用いられているが，積極的に新技術に取り組み実績を積み上げていかないと，いつまでも従来技術での限界を超えられない．

技術とは，目的に合致し種々の条件を満たす合理的な方法を考え出すことである．科学的な厳密さを追求する必要はなく，実際的で使いやすい方法が望まれる．テストの世界でも既に多くの技術が存在しており，各技術の長所を理解して，対象システムの特質に合わせて複数の技術を組み合わせて実施すべきである．

＊1　品質会計は，作り込まれたバグを過去の経験から予測し，それを負債とみなし，検証活動で摘出していくことによって負債を返済していくという考え方で，障害予測は，開発開始時の予測に加えて，開発途中の予測見直しを重視している．
＊2　CMM/CMMI は，CMU/SEI の登録サービスマークである．
＊3　UML（Unified Modeling Language）は，モデリング用の意味付けされた統一表記法である．
＊4　ERP（Enterprise Resource Planning）は，日本では統合業務パッケージと呼ばれており，受注・販売管理，在庫管理，生産管理，会計といった企業の基幹業務をサポートする情報システムパッケージである．
＊5　SOA（Service Oriented Architecture）は，大規模なシステムを，外部から標準化された手順によって呼び出すことができるサービスの集まりとして構築する設計手法である．
＊6　PSP（Personal Software Process）は，CMU/SEI の登録サービスマークであり，個人としてソフトの品質と工程の改善をはかる自己改善プロセスである．
＊7　TSP（Team Software Process）は，CMU/SEI のサービスマークであり，チームとしてのプロセスに従ったソフトウェア開発をガイドする．
＊8　探針は，日立製作所で実施されている信頼性予測技法である．開発部署でのテスト中にサンプリングでソフトウェアを検査し，早い時点で品質を推定して品質向上に結び付ける．

参考文献　［1］「ユーザー企業 ソフトウェアメトリクス調査報告書 2008 年版」，日本情報システム・ユーザー協会，2008 年 6 月，pp.144
［2］ 情報処理推進機構 ソフトウェア・エンジニアリング・センター，「ソフトウェア開発データ白書 2008」，日経 BP 出版センター，2008 年 8 月，pp.219
［3］ “CMMI for Development, Version 1.2”, Carnegie Mellon University Software Engineering Institute, August 2006

［4］ 山田　茂,「ソフトウェア信頼性モデル」, 日科技連出版社, 1994年11月, pp.125-157
［5］ Stephen H. Kan, “Metrics and Models in Software Quality Engineering (2nd Edition)”, Addison-Wesley Professional, 2003（古山　恒夫, 富野　壽 監訳,「ソフトウェア品質工学の尺度とモデル」, 構造計画研究所, 2004年11月, pp.206）
［6］ 板倉　省吾,「JIS X 0129-1（ISO/IEC 9126-1）ソフトウェア製品の品質―第1部：品質モデル」, 日本規格協会, 2003年2月, pp.5
［7］ 情報処理推進機構 ソフトウェア・エンジニアリング・センター,「共通フレーム2007」, オーム社, 2007年10月, pp.101
［8］ 二木　厚吉 監修, 大槻　繁, 金藤　栄考, 佐藤　武久 著,「ソフトウェアクリーンルーム手法」, 日科技連出版社, 1997年12月, pp.28-65
［9］ 佐原　伸,「形式手法の技術講座」, ソフト・リサーチ・センター, 2008年4月, pp.145-154
［10］ Kent Beck, “Test-Driven Development; By Example, 1st Edition”, Addison Wesley Professional, 2003（長瀬　嘉秀 監訳,「テスト駆動入門」, 2003年9月, pp.189-201）
［11］ Roger S. Pressman, “Software Engineering / sixth edition”, The McGraw-Hill Companies, Inc., 2005（西　康晴, 榊原　彰, 内藤　裕史 監訳,「実践ソフトウェアエンジニアリング」, 日科技連出版社, 2005年2月, pp.165）
［12］ Paul Clements, Linda Northrop, “Software Product Lines : Practice and Patterns”, Addison-Wesley, Professional, August 2003（前田　卓雄 訳,「ソフトウェアプロダクトライン」, 日刊工業新聞社, 2003年9月）
［13］ Glenford J. Myers,「The Art of Software Testing」, John Wiley & Sons, Inc., 1979（長尾　真 監訳, 松尾　正信 訳,「ソフトウェア・テストの技法」, 1980年3月, pp.9-11）
［14］ Watts S. Humphrey, “A Discipline for Software Engineering”, Addison-Wesley Longman, Inc., 1995（松本　正雄 監訳,「パーソナルソフトウェアプロセス技法」, 共立出版, 1999年5月）
［15］ Watts S. Humphrey, “Introduction to the Team Software Process”, Pearson Education, Inc., 2000（岡　真由美 訳, NTTソフトウェア 監修,「チームソフトウェア開発ガイド」, コンピュータ・エージ社, 2002年10月）
［16］「品質向上は発注力強化から 東証, 世界に挑む300億円プロジェクト」, 日経コンピュータ, 日経BP社, 2008年11月1日号, pp.102-109
［17］ 情報処理推進機構 ソフトウェア・エンジニアリング・センター,「発注者ビューガイドライン」, 2008年7月, http://sec.ipa.go.jp/reports/20080710.html
［18］ システム基盤の発注者要求を見える化する非機能要求グレード検討会,「システム基盤の非機能要求に関する項目一覧」, 2008年9月, http://www.nttdata.co.jp/nfr-grade/
［19］ 塩見　弘, 島岡　淳, 石山　敬幸,「FMEA, FTAの活用」, 日科技連出版社, 1983年9月, pp.69-99
［20］ 田口　玄一 編,「実験計画法　改訂版」, 日本規格協会, 1987年2月
［21］ 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年7月
［22］ D. Richard Kuhn, Dolores R. Wallace, Albert M. Gallo Jr., “Software Fault Interactions and Implications for Software Testing”, IEEE Transactions on Software Engineering, Vol.30, No.6, June 2004

執筆者紹介　飯 田　志 津 夫（Shizuo Iida）
日本ユニシス(株)入社以降. 金融機関勘定系システムの開発に20年間従事. 金融マーケティング部署と共通利用技術部署を経験後, 品質管理部署にてISO9001認証取得, CMM/CMMI導入・展開を担当し, 現在, 品質保証部に所属.