1 ソフトウェアテスト総論

西　康晴[*1]・古川善吾[*2]

[*1] 電気通信大学 電気通信学部 システム工学科
[*2] 香川大学 工学部 信頼性情報システム工学科

はじめに

我々の周りは，ソフトウェアで占められている．毎日の生活には携帯電話や自動車が欠かせない．金融系や流通系などあらゆる企業は，情報システムを駆使してビジネスを高度化しグローバル化している．もはや我々の生活は，ソフトウェアなしでは成り立たないほど大きく依存してしまっている．

しかし残念ながら，ソフトウェアの品質事故の報道は絶えることがない．携帯電話，ディジタルテレビ，銀行，航空管制，自動車，鉄道，ダムなど，多くのバグによって我々の生活は脅かされている．近い将来に人間がロボットと共生するようになると，ソフトウェアの品質事故によって人命が奪われるという事態も発生しかねない．

こうしたソフトウェアの品質事故を防ぐ技術体系が，ソフトウェア工学である．なかでも，ソフトウェアのバグを水際で見つけ出す重要な技術分野が，ソフトウェアテストである．本稿では，テストの位置づけの変遷について整理し，テストの基本概念と主な技術を概観する．またテストの今後の進化について述べる．

ソフトウェアテストの位置づけ

To err is human ―過つは人の常，とイギリスの詩人である Alexander Pope は詠っている．ソフトウェアが人間の頭脳によって生み出される以上，人間の過ちによってバグは作り込まれ続ける．したがってバグを見つけ出すテストの重要性は，増えこそすれ減ることはない．人間が知恵を絞って開発してもなお残ってしまうバグを見つけるための，高度でクリエイティブな技術分野が本特集のテーマとなっているソフトウェアテストである．

1960 年代後半，テストはプログラマによるトライアル＆エラーの一部であり，単なる動作確認作業という位置づけに過ぎなかった．そのためテストケースが熟考されて設計されることはなく，技術分野としても認められていなかった．70 年代後半になって，テストはプログラムの正しさを実証する手段とみなされるようになったが，いずれにせよテストケースが十分検討されるには至らなかった．またテストマネジメントの面では，単なる作業進捗の管理という位置づけにすぎなかった．

しかし 1979 年に，Myers がテスト設計の面でパラダイムシフトを起こした．「テストとは，エラーを見つけようとしながらプログラムを実行する過程である」と述べ[1]，バグを見つけるためにきちんとしたテスト設計が必要であるという位置づけにテストを進化させたのである．Myers によって，テストは重要な技術分野として認識されるようになったと言ってよい．

一方テストマネジメントの面では，Beizer が 1983 年にパラダイムシフトを起こした．テストは出荷後の品質に関するリスクを下げることが目的だと述べ[2]，開発ライフサイクル全体における品質のマネジメントという位置づけにテストを進化させたのである．

ただし，その進化はまだ途上にすぎない．開発ライフサイクル全体における品質マネジメントの技術には，テストのほかにレビューやモデル検査，メトリクス，非機能特性分析，プロジェクトマネジメント，プロセス改善などもある．しかし現状では，V&V（Verification and Validation）としてテストとレビューのみを連携させ，上流からバグを予防し品質を作り込む研究が進んでいる．今後はさらに，テストを始めさまざまな技術が開発ライフサイクル全体の品質マネジメントのために包括的に連携するように進化すべきであろう．

ソフトウェアテストの基本概念

［テスト設計と戦略］

テストでバグを見つけるためには，入力値や動作環境，タイミングなどさまざまな条件でテスト対象のソフトウェアを動作させる必要がある．それぞれの条件の組合せをテストケースと呼ぶ．また，テストケースを体系的に挙げることをテスト設計と呼ぶ．質の高いテスト設計をすると，少ないテストケース数で多くの危険なバグを早く検出することができる．逆に，行き当たりばったりでテストケースを想起するアドホックテストでは，多くのバグを見逃してしまう．

テスト対象のある側面に着目してテスト設計をする技術をテスト技術と呼ぶ．たとえば制御パステスト法は，ソフトウェアの制御構造に着目したテスト技術である．

テストの研究は，テスト技術の開発と改善を中心に発展してきた．

とはいえ実際にテストで多くのバグを見つけられるような質の高いテスト設計をしようとすると，単一のテスト技術を用いるだけでは不十分である．テスト設計の意図や方向性，全体的なバランスといったテスト戦略を検討した上で，適切なテスト技術を駆使する必要がある．テストケースとテスト設計は，ソースコードとソフトウェア設計のような関係にあり，テスト戦略はソフトウェアアーキテクチャに対応する．

テスト戦略を決める際には，どのような観点でテスト設計をすればよいかを考えることが重要となる．テスト観点には，たとえばOstrandが挙げたユーザ，仕様，設計・実装，バグの4つのテストの類別や，Myersが挙げたボリューム，構成，回復，操作性などの14のシステムテストカテゴリ[1)]，ISO/IEC 9126の品質特性などがある．

しかしソフトウェアアーキテクチャのように，テスト戦略をモデリングしテストアーキテクチャとして示すような取り組みはあまり行われていない．そのため，大規模なテスト設計やパターン化，再利用に関する取り組みや研究は進んでいない．

［網羅とピンポイント］

テスト設計の基本は，網羅である．仮に，ソフトウェアを完全に網羅的にテストできるならば，すべてのバグを検出することができ，出荷後にバグが残っているかもしれないという品質リスクはゼロになる．したがって，網羅的にテスト設計を行い，カバレッジ（網羅性）を測定することが基本となる．網羅型テストの代表例には，制御パステストが挙げられる．

しかしMyersは，ごく簡単なソフトウェアでもテストケース数が爆発する例を示し，現実的な工数で完全に網羅的なテストはできないと述べた[1)]．すべての機能，すべての取り得る入力値，すべての内部状態，すべての実行タイミング，すべての動作環境など，あらゆる条件を組み合わせるとテストケースは天文学的な数となってしまう．

このように，実際には現実的な工数で完全に網羅的なテストができないため，間引きによってテストケースを削減することが必要となる．テスト設計では，間引きによってテスト工数を現実的な規模に圧縮しつつ，削減したテストケースで見つかるはずのバグによる品質リスクを許容範囲に抑えるというトレードオフが重要になる．

そのためテスト設計では，同値クラス分析などのグルーピングを行う．たとえば入力範囲が1から255の場合，入力範囲すべてに同じ処理が行われるという仮定が置ければ，どの値でも同じようにバグが発生するはずである．したがって任意の値で1度テストすればよく，網羅的にテストする必要はない．すなわちグルーピングを行うと，品質リスクを増加させずにテストケースを間引きできる．

一方，テスト設計のもう1つの基本は，ピンポイントである．仮に，バグを見つけられるテストケースだけをピンポイントに設計し実施できるならば，網羅的にテストする必要はない．たとえば入力値を判定する条件文にバグがあり，1以上の入力範囲のはずが0以上になっていたとする．この場合，ピンポイントに0だけをテストすればよく，網羅的にテストする必要はない．すなわちピンポイントにテストを行うと，品質リスクを増加させずにテストケースを間引きできる．境界値付近はバグが多いため，境界値テストはピンポイント型テストの代表例となっている．

また，もしバグが発生しても，影響が小さければ品質リスクは小さいとする考え方もある．すなわち，ユーザにとって重要性が高いデータや頻繁に用いる機能だけをピンポイントにテストできるならば，網羅的にテストする必要性は低いと考えられる．代表例として，統計モデルを用いてユーザの操作分布を推定する運用プロファイルテストが挙げられる．

このようにテスト設計では，グルーピングなどを行いながら網羅型のテスト技術を適用しつつ，ピンポイント型のテスト技術を上手に組み合わせ，テストの工数と品質リスクとのバランスを適切にとることが重要となる．

［テスト設計によるバグの予防］

テスト設計をきちんと行うには，例外事項や異常系，さまざまな動作環境の変化，予期せぬユーザの使い方など，バグを発生させ得る条件をすべて列挙することが必要となる．テスト設計の際にこうした条件を列挙するという作業は，レビューと同様の効果を生み，開発上流での考慮漏れの検出につながることが少なくない．

また，テスト設計で見落とされがちなのが，期待結果の導出である．期待結果とは，バグがない場合に期待されるテストケースの実施結果を意味する．テスト設計の際にきちんと期待結果を導出しておかないと，テスト実施の際にせっかくバグが出現しても，見逃してしまう可能性がある．テスト設計の際に期待結果を導出するという作業は，仕様や構造の具体的な解釈やトレースを伴うため，レビューと同様の効果を生み，考慮漏れや誤りの検出につながることが少なくない．

一般にテストという工程は，テストケースを実行することによってバグを見つけると定義されていることが多い．しかし実際には，テスト実行だけでなく，テスト設計のときにもバグを見つけることが可能となる．しかもテスト設計をするだけであれば実行可能なソフトウェア

が完成している必要がないため，開発の上流でテスト設計を行うことができ，テストを実行しなくてもバグを予防できる．

［テストのライフサイクルと回帰テスト］

テストは，計画，分析，設計，作成，実行，終了条件の評価と報告，およびマネジメントからなるプロセスによって進められる．またテスト対象が単一モジュールレベル，複数モジュールレベル，ないしシステムレベルなのかに応じて，単体テスト，結合テスト，システムテストといったテストレベルでテストを行っていく．テストの規模が大きい場合は，自動化を行うことがある．

テスト対象にバグ修正や仕様変更がなされると，回帰テストを実施し，動作していた他の思わぬ機能が修正や変更の副作用によって動作しなくなるデグレードバグを検出する必要がある．よって効果的な回帰テストのためのテスト設計には，副作用の影響の把握が必須となる．しかしデグレードバグの多発はグローバル変数の多用といった上流での設計の質の低さに起因しているため，副作用の特定が難しい．すなわち回帰テストが必要なテスト対象であるほど，回帰テストが難しいというジレンマがある．

同様のジレンマは，テスト設計そのものにも存在する．効果的なテスト設計を行うには，仕様や設計など上流できちんと検討がなされドキュメントが作成されている必要がある．しかし上流できちんと検討されているとバグが少なくなり，そもそも効果的なテスト設計を行う必要性が低下してしまう．逆に上流でほとんど検討されていないと，ありとあらゆるテストを網羅的に行わなくてはならなかったり，テスト設計に必要な上流の情報の抽出が難しくなるため，効果的なテスト設計が困難になる．

このジレンマのためテスト技術の研究では，上流において仕様や設計をどの程度検討しているのかを仮定するのが難しい．特に実際のソフトウェア開発では，上流でソフトウェア工学技術を十分に活用していない場合も少なくない．したがってテスト技術の研究では，テーマを決めたり研究を進める際に，理論面だけを考慮することなく，実際のソフトウェア開発の状況を十分に反映することがとても重要となる．

さまざまなソフトウェアテストの技術

［テスト技術の分類］

質の高いテスト設計をするためには，多岐にわたるテスト技術を整理して把握しておく必要がある．さもないと，見つけられるはずのバグを見逃したり，テストケース数が膨大になり大幅に工数が超過してしまう．

テスト技術の多くは，確定的テストと呼ばれ[3]，テスト実施以前にテスト設計をしておくことを前提としている．しかしテスト技術の中には，事前にテスト設計をせずにテストを実施する非確定的テストと呼ばれる技術もあるので注意が必要である．

確定的なテスト技術は，テスト対象のある側面に着目してテスト設計をする技術である．そのため，テスト対象の持つ側面を分類することで，テスト技術も分類できる．たとえば，開発技術や非機能特性，ドメイン（適用領域）といった側面で分類することができる．

さまざまな側面でテスト技術を分類しておくと，同じテスト対象に対し異なる側面から同時にテストを行うことが可能になる．それによって，ある側面からのテスト設計が不十分ないし不適切なためバグを見逃しそうになっても，他の側面からのテスト設計で補ってバグを見つけることができる．

ただしテスト技術の中には，テスト対象の持つ側面には着目せず，テスト設計そのものに着目した技術もある．たとえば，単一のテスト条件をグルーピングすることで間引きを行う同値クラス分析，複数のテスト条件の組合せを整理するデシジョンテーブル，複数のテスト条件の間の論理関係を整理し網羅する原因結果グラフ，複数のテスト条件の膨大な組合せに対し数理的に間引きを行う Pairwise 技術や直交配列表を用いる技術などが挙げられる．

［開発技術に対応したテスト技術］

テストは下流に配置されているため，上流で適用されている開発技術に対応したテスト技術を適用すべきことが多い．とはいえ実際のソフトウェア開発では，上流でソフトウェア工学技術を十分に活用していない場合も少なくない．たとえば筆者の経験では，分析や設計の際に状態遷移図を描いておくべきなのに，DFD しか作成していない現場もあった．そのような場合は，上流で適用しておくべき開発技術をテスト工程で適用し，本来あるはずの成果物をリバースエンジニアリングした後に，テスト技術を改めて適用することになる．すると，たとえば状態の抜けや遷移の誤り，イベントの抜けなど，本来上流で考慮しておくべき事項の誤りや抜けによるバグを多く見つけることができる．

上流で適用された開発技術が構造化分析・設計や手続き型言語の場合は，オーソドックスなテスト技術を適用することができる．例として，制御パステストや条件式テスト，データフローパステスト，状態遷移テストなどが挙げられる．

オブジェクト指向で開発されたテスト対象の場合は，オーソドックスなテスト技術に加えて，スレッド状ないしクラスタ状に暫増させていくクラス間結合テストや

UML図を網羅するテスト，ユースケースのテストが必要となる．一方，オーバライドや動的結合のテスト，継承した親クラスのコードに対する子クラス側でのテストは忘れやすいので注意すべきである．

また並列処理やデータベース，GUIなどの要素的な開発技術に対応したテスト技術も研究されている．

［非機能特性に対応したテスト技術］

特にシステムテストレベルの場合，非機能特性を評価したいことが多々ある．したがってテストにおいても，評価すべき非機能特性に対応したテスト技術を適用する必要がある．

たとえば性能やスケーラビリティといった非機能特性をテストする場合は，性能テストや負荷テストなどの技術を適用する．ポータビリティをテストする場合は，構成テストや互換性テストなどの技術を適用する．ただしセキュリティやユーザビリティ，画質や音質など，テストの技術ではなく，それぞれの非機能特性を専門とする領域の技術を適用すべきものもある．

［ドメインに対応したテスト技術］

テスト技術は，ドメイン（適用領域）で分類することもできる．それぞれ特徴や開発技術が異なるからだ．

たとえば組込み系システムの場合，ハードウェアとのインタフェースの齟齬を狙ったテストや，リアルタイム制約に関するテスト，ハードウェアの故障への対応性のテスト，天候などの自然条件を考慮したテスト，さまざまな仕向地（出荷先）のテストなどが必要になる．プラットフォームが多岐にわたるため，テスト環境の構築が難しいという現実的問題もある．また人命にかかわるため安全性を特に評価する必要がある場合も多い．最近では，ハードウェアとの協調検証も議論され始めている．

エンタープライズ系システムの場合，業務フローやビジネスロジックを例外も含めてモデリングし網羅的にテストすることが必要となる．同時に，データモデルを十分に考慮し複雑なデータ間の関係を反映したテストも重要である[4]．またバッチ処理とオンライン処理のバランスを考慮したり，本社や支店とデータセンタといった遠隔拠点を考慮したテストが必要な場合もある．コスト面などからテスト環境の構築が難しい場合に，実稼働環境で連携するサーバを含めたテストが必要となるような現実的問題もある．基幹系や金融系のシステムなど，特に信頼性を重視する必要がある場合も多い．

Web系システムは，エンタープライズ系システムと似ている．負荷テストや性能テストが重要になる点などは，同様である．しかし特に納期とコストを重視した開発が多く，上流での仕様や設計の検討が比較的緩いことがあるため，工数と品質リスクとのトレードオフがシビアになる．また仕様変更が多発するため，回帰テストが重要となる．テストプロセスもウォーターフォールでは難しく，スパイラルやアジャイル型の方が適している．

パッケージソフトウェアは，Webシステムや大量生産型の組込みシステムと同様に，ユーザの特徴を適切に把握し反映したテストが重要となる．そのためユーザシナリオを網羅するテストや，ユーザビリティテストを十分に実施しなくてはならない．また動作環境であるPCで同時に動作する他のソフトウェアや周辺機器との相性評価のために，構成テストや両立性テストが重要となる．

［非確定的なテスト技術］

非確定的なテスト技術の代表例が，テスト実施時に思いつきでテストを挙げるアドホックテストである．モンキーテストとも呼ばれる．乱数を用いる場合はランダムテストと呼ばれる．アドホックテストやランダムテストは，思いもよらない観点でのテストが可能になると言われている．しかし，運良く得られた意外なテスト観点を蓄積し再利用する仕組みが開発やテストの組織になければ，継続的に質の高いテストをすることはできない．

ランダムテストを一部確定的にした技術が，運用プロファイルを用いるテストである．完全にランダムにテストするのではなく，ユーザの特性やユーザシナリオから操作の分布を運用プロファイルとして事前に算出しておき，操作分布に従った乱数を発生させることで実際のユーザの振る舞いをシミュレートするテスト技術である．

アドホックテストに似て非なる技術が，探索型テストである．探索型テスト[5]とは，経験豊富で洞察力に優れたテスト技術者が，その時点までに見つかったバグの傾向，テスト対象の振る舞いの微妙な変化，テスト対象の構造に典型的なバグのパターンなどを観察・洞察し構築した仮説を元にテスト設計をしながら実施する高度なテスト技術である．探索型テストの研究では，経験やバグの傾向の形式知化や，テスト技術者の集中力の向上による観測・洞察の高度化などがテーマとなっている．

［モデルベースドテスト］

ソフトウェア設計技術の進化に伴って，UMLモデルを中心としたモデル駆動開発（MDD, Model Driven Development）や，組込みシステム向けに制御モデルを中心としたモデルベース開発（MBD, Model Based Development）といったMxD技術がかなり発展してきた．MxDによって，モデルからソースコードを自動生成したり，モデルを直接実行できるようになりつつある．

テストでも同様に，テスト設計における技術的進化に伴って，テストのためのモデルやMxDのためのモデ

ルからテストケースを自動で生成し実施するモデルベースドテストの研究が進んでいる．多くは研究段階だが，IBM Haifa 研究所では Hartman[6] らが，Google では Robinson[7] らが一部実用化している．

テストのためのモデルは，テスト技術と同様に多岐にわたる．代表的なモデルとして，制御フローモデルや有限状態機械モデル，統計的な分布モデル，ラテン方格などの数理モデル，ペトリネットなどの代数的モデル，形式記述モデルが挙げられる．単なる機能網羅テストであっても，機能一覧表という簡単なモデルを基にしていると考えてよい．また U2TP（UML2.0 Testing Profile）や TTCN-3（Testing and Test Control Notation Version 3）などテストのモデリングのための記法も提案されている．

モデルベースドテストが実用化されると，テスト設計モデルからテストケースを網羅的に自動生成できるようになり，テスト設計の工数を大幅に削減できる．また自動テスト用のスクリプトへの自動変換と期待結果の自動生成，および期待結果と実際のテスト結果との比較・判定の自動化が可能になれば，テスト実施の工数も大幅に削減できる．ただ現状としては課題も多い．

モデルベースドテストで重要なのは，テストケースの自動生成・実行は手作業オーダーの設計・実施時間を計算機オーダーに（大幅に）低減するだけで，本質的に Myers が指摘したテストケース数の爆発の問題を解決するわけではないという点である．パーキンソンの法則に従うと，テスト時間が低減しても，低減した分だけシステムが大規模化・複雑化するため，テストケース数が爆発するという問題は依然として残る．すなわち本質的なのは，モデルベースドテストによってテストの自動化が実現できたとしても，単に上流のモデルを適用して網羅的にテスト設計をするのではなく，よりピンポイントにバグを見つけられるようなモデルを求め，探索し改善していくべきであるという点である．

ソフトウェアテストの進化

［テストエンジニアリングへの進化］

30 年近く前に Myers がパラダイムシフトを起こしたにもかかわらず，現在でもテスト設計はせずに，Excel シートにテストケースをベタ打ちしコピー＆ペーストを繰り返すのみというテストの現場は少なくない．こうした現場では，いまだにテスト設計を単なる動作確認項目の記述だと思っているのだ．また進んだ現場でも，テスト技術は適用するもののテストの詳細設計にとどまっていることが多い．そのため，大規模なテスト設計やパターン化，再利用に関する取り組みは進んでいない．

そこで今後は，テスト設計から“テストエンジニアリング（テスト開発）”に進化を遂げることが期待されている．テスト設計が，単なる動作確認項目の記述ではなく，テストという大規模な“建造物”を構築する技術に進化していくのである．

まず最初の進化として，テストのアーキテクチャという概念が整備され，大規模化・複雑化したテストを見通しよく設計できるようになるだろう．同時に，テスト設計の表現も豊富になりパターン化が促進されることで，多くのテスト設計パターンが構築され流通するようになると予想される．またテスト設計の再利用技術も進み，テスト設計そのものをプロダクトライン開発の対象として捉えるように進化していくだろう．

すなわち，テスト技術とテストケースによる狭小なテスト設計から，テストアーキテクチャとテスト設計パターンを中心とした包括的なテスト設計に移行していくことが予想される．こうした進化は，特にさまざまなテスト設計を多数担当するような第三者 V&V（IV&V, Independent V&V）の組織では必須となる．

また，ソフトウェア設計が要求分析と密接に結びついているように，テスト設計のための分析技術が研究されていくだろう．テスト分析には大きく分けて，許容できる品質リスクの検討や工数とのトレードオフといったテストの目標の検討と，テスト対象が備えるべき非機能特性や振る舞いといったテスト対象の把握の 2 つがある．前者は開発全体のプロジェクトマネジメント技術や品質保証技術などと，後者は要求分析やアーキテクチャ評価などと協調して進化していくだろう．

このようにテスト設計やテスト分析技術が進化すると，モデルベースドテストが技術としてもツールとしても普及してくる．さらには，モデル駆動開発やモデルベース開発といった MxD と協調し，開発とテストが同期しながらソースコードもテストケースも自動生成し，自動でテストを実行していくようになるだろう．

［バグ工学への進化］

開発やテストの技術が進化し自動化されると，バグは撲滅されるだろうか．歴史はその質問に対し，否という答えを突きつけている．Brooks も「銀の弾丸はない」と述べている．開発やテストの技術が進化し自動化されて余裕ができると，パーキンソンの法則に従い，余裕の分だけ要求が増えてシステムが複雑化・大規模化し，新たな種類のバグが出現するだろう．我々が人間である限り，過ちはなくならず，バグが撲滅されることもない．

そうすると，バグと真っ向から向き合う研究分野が必要となるのは論をまたない．機械設計では失敗学という研究分野が生まれたが，ソフトウェア工学でも同様の

研究分野，いわば“バグ工学”が必要となる．すでにIEEE std. 1004-1993やBeizer[2]などがバグを分類しているが，さらにバグの性質やバグが作り込まれる条件などを深耕した研究が必要になる．

特に，バグが作り込まれやすい仕様や設計のパターンであるアンチ・デザインパターンやバグパターンを，ドメイン，開発技術，上流の仕様や設計のパターン，開発者のヒューマンエラーなど多岐にわたるアプローチで研究することが必要だろう．たとえば，例外的な仕様は検討漏れを誘発しやすいためバグが多いといった，バグを作り込みやすい仕様のパターンを不具合モードとして整理する研究も進んでいる．こうしたアンチ・デザインパターンは，ピンポイント型のテスト設計技術だけでなく，レビューの指摘項目の設計技術の研究も進化させ得る．バグ工学を確立することによって，上流からバグを予防し品質を作り込むことができるようになるのである．

［出荷判定工学への進化］

テストで最も難しいのは，テストの終了判定である．主な技術として網羅性評価とミューテーションテスト，信頼度成長曲線，リスクベースドテストの４つがあるが，いずれも一面的な評価にならざるを得ない．一方，現場でのテスト終了判定，すなわち出荷判定の際には，テストだけでなく仕様や設計のレビュー結果やマネジメントメトリクスも考慮するし，プロセス改善の程度を勘案することもある．すなわちテストの終了判定技術を検討するには，テストだけを踏まえたのではまったく足りず，ソフトウェアの開発ライフサイクルすべてを包括的に考慮した出荷判定という位置づけが必要となる．

しかし現状では，ソフトウェア工学の精緻化に伴い，テスト，レビュー，モデル検査，メトリクス，非機能特性分析，マネジメント，プロセス改善などは連携せず独立して研究されている．そのため，現場の経験で技術を組み合わせて出荷判定を行わざるを得ない．

そこで必要なのは，テストとレビュー，モデル検査などのV&V技術，プロダクトやマネジメントのメトリクス関連の技術，プロセスアセスメント技術などがもっと密に連携し，リスクマネジメント技術やEVMSなどを取り入れながら，包括的な“出荷判定工学”として１つの研究分野を形成することである．また組込み系の場合はハードウェアやシステム全体，エンタープライズ系の場合はインフラや連携システムなど，システム全体を視野に入れて評価しなければならない．

同時に，出荷判定の目標や基準を，残存バグ密度やテスト網羅率といった開発の内部的指標で示すのではなく，顧客価値や事業継続性といった外部的指標で示す必要もある．それによって，ペルソナのような顧客のモデルや企業のビジネスモデルを基にしてソフトウェアの非機能特性を表し，達成すべき価値と品質リスク，コスト，スケジュールなどを一元的にマネジメントしていく工学的な方法論に進化させることが可能になる．

［モダン・ソフトウェア・テスティング］

ソフトウェアテストは，プログラマによるトライアル＆エラーから，バグを見つけるためのテスト設計として技術分野を確立し進化してきた．しかしテストという技術分野の本質は，開発全体の成功こそがテストの目標であるという点である．また，良いテスト設計をするためには，上流の仕様や設計の質が高いことが必要となる．すなわち，これからのテストに求められるのは，テストの質を高めることで上流の質が高まり，上流の質が高まることでテストの質も高まるというフィードバックサイクルとしてテストを捉えることである．それを本稿では，“モダン・ソフトウェア・テスティング”と呼ぶ．

モダン・ソフトウェア・テスティングを実現するためには，テスト技術をテストエンジニアリングへと進化させ，バグ工学を確立することでフィードバックサイクルの活性化を促し，出荷判定工学としてすべてのソフトウェア工学技術が連携することが必要である．それには，テスト以外のソフトウェア工学の研究者が自分の分野の技術をテストに適用すること，そしてテスト分野の研究者が他の分野とどんどん連携することが必要となる．そうすれば我々は，今まさに直面しているソフトウェアの品質事故を未然に防ぐことができ，安心してソフトウェアに身を委ねられるようになるだろう．

参考文献

1）Myers, G. J. : The Art of Software Testing (1979), ソフトウェア・テストの技法，近代科学社 (1980).
2）Beizer, B : Software Testing Techniques (1990), ソフトウェアテスト技法，日経 BP 社 (1994).
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).
4）鈴木三紀夫：テストエンジニアのためのデータモデリング入門，ソフトウェア・テストプレス，技術評論社，Vol.6, pp.98-106 (2007).
5）Kaner C. 等 : Lessons Learned in Software Testing (2002) : ソフトウェアテスト 293 の鉄則，日経 BP 社 (2003).
6）Hartman, A : Model Based Testing: What? Why? How? And Who cares?, ISSTA2007, Portland, ME (Jul, 2006).
7）Robinson, H : The Bionic Tester, CAST2007, Seattle, WA (July 2007).

（平成 20 年 1 月 18 日受付）

西　康晴（正会員）
nishi@se.uec.ac.jp
電気通信大学講師，本会情報処理教育委員会ソフトウェアエンジニアリング委員会幹事．NPO 法人ソフトウェアテスト技術振興協会（ASTER）理事長，NPO 法人組込みソフトウェア管理者・技術者育成研究会（SESSAME）副理事長などを務める．

古川善吾（正会員）
zengo@eng.kagawa-u.ac.jp
香川大学工学部教授，総合情報センター長，本会四国支部監事，NPO 法人ソフトウェアテスト技術振興協会（ASTER）副理事長などを務める．