品質を可視化する“テストフレームワーク”
―テスト工程の自動化による効率化と品質向上―

昨今、短納期のシステム開発が求められるようになり、それに応えるための生産性の向上が必須となっている一方で、品質向上に不可欠なテスト工程を十分に確保できないケースもままある。本稿では、生産性を向上させるとともに、品質を可視化することで品質向上も実現する、テスト工程自動化の“テストフレームワーク”の仕組みについて提言する。

テスト工程自動化の必要性

ここ数年来、ビジネス環境の変化の速さに合わせるべく、システム開発期間短縮の要求がますます強まっている。開発側ではその要求に応えようと、生産性を向上させるためのさまざまな努力を続けている。当然ではあるが、短期間だからといってソフトウェアの品質を落としてよいはずはない。品質を確保するためのテストを確実に実行することは不可欠である。

一方、昨今では変化に対応するためのイテレーション開発（短い単位で設計・実装・テストを繰り返す開発手法）、システムが複雑化してくるときに必要なリグレッションテスト（一部のプログラム変更が他に与える影響を検証するためのテスト）など、テストを繰り返し行うことが重要になってきている。また、ビジネスの変化に合わせた短サイクルでのエンハンス開発（システム稼動後の拡張）も多くなってきている。そのため、必然的に生産性が高く再現実行が容易なテスト方式が求められる。さらに、短納期を実現するためには、それぞれの工程で品質を作り込み、後工程に問題を積み残さないことも重要になる。

テスト工程の自動化は、これまでもテスト工程の生産性を向上させるための解決策とされてきたが、繰り返しテストが少ないときなどは、コスト効果などを理由に見送られるケースも多かった。しかし、繰り返しテストが必要とされている現状ではやはり有効な解決策になる。また、品質を確保するための仕組みをテストに組み込むことができれば、生産性向上にとどまらず品質向上にも大きな効果が得られる。

テスト工程自動化の仕組み

これまでは、テストの自動化というとテストプログラムの実行や画面操作などの自動化、すなわち「テスト実行操作」の自動化が主なものであった。もちろん、これだけでも作業負荷を軽減させることはできる。しかし、テストには実行操作だけでなく、以下のように多くの作業が必要である。

①実行準備

データベースの初期化を行い必要なマスターデータを準備する。スタブ（ダミーのプログラムモジュール）を利用する場合はテストケース（データ、処理内容、処理結果をセットにしたもの）に合わせてスタブを配置する。

野村総合研究所
情報技術本部
開発技術部
上級テクニカルエンジニア
齊藤敬史（さいとうたかし）
専門はJ2EE、モデリング、標準化など

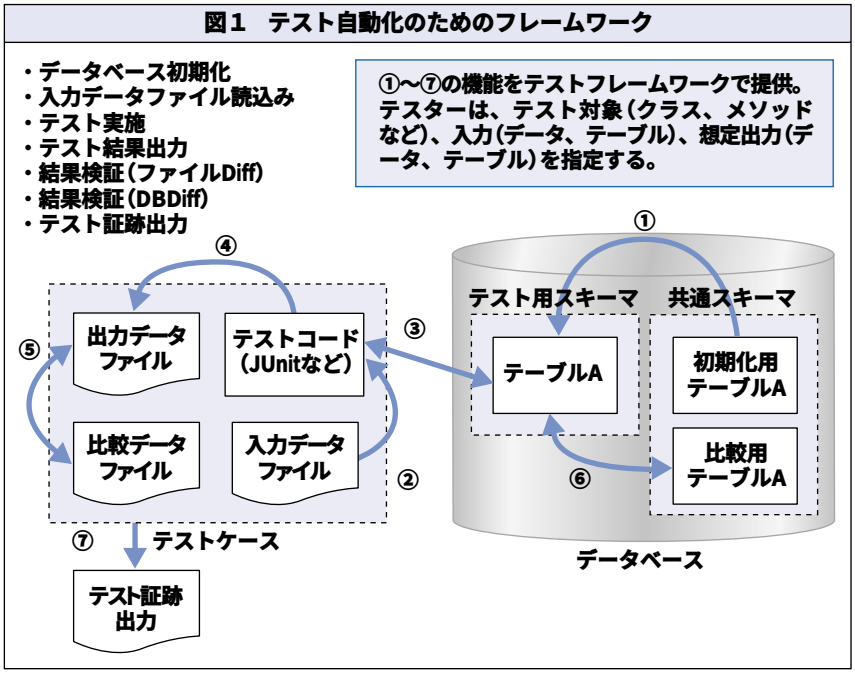
またアプリケーションサーバーの起動などテスト環境を整えるための作業もある。

②実行結果の保存

ドライバーが出力するログ、スクリーンショット（モニター画面をそのまま保存した画像）、実行後のデータベース状態などを記録・保存する。

③結果の検証

目視による結果確認を行うほか、取得した実行結果からテストが成功しているかを検証する。



これらの作業負荷は思いのほか大きい。また、実行準備がテスター任せになっているために、再テストの際に同じ状態を再現できない場合も多い。

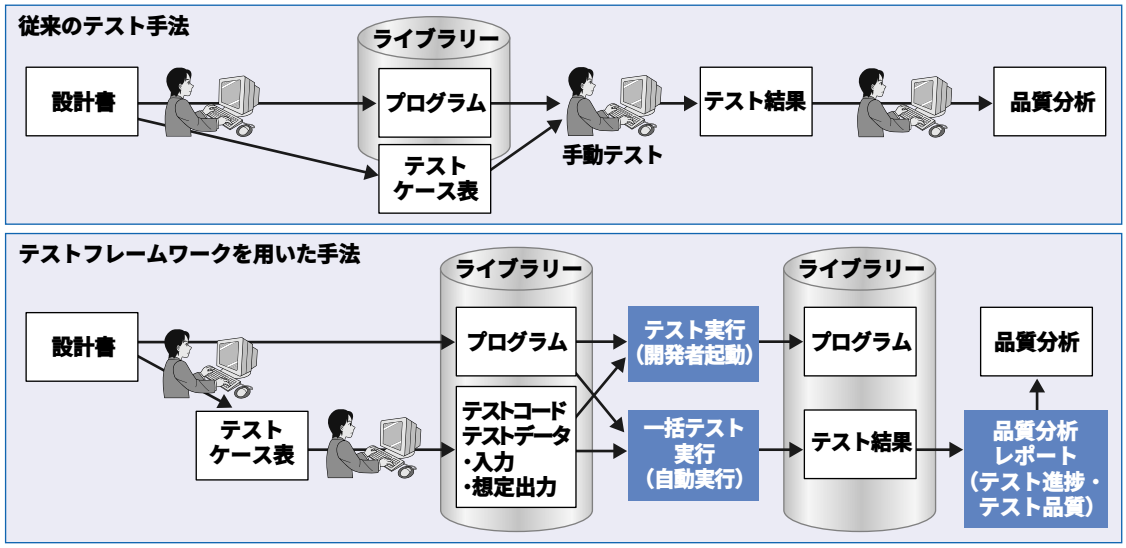
これらの問題を解決するためには、たとえば図1に示すような仕組みで、テスト準備（データベース操作）、テスト実行、結果の取得、結果（出力テーブルなど）の検証を自動化できるようなテストフレームワークを構築することが有効である。テスト結果を確認する方式と内容をこのフレームワークで規定することで、再現テストを容易に実施することができる。

また、テストへの入力、テストの成果物、検証を行うための想定出力（想定テーブル状態）の形式はこのテストフレームワークに従う。そのため、これらを一元管理することも容易になり、これまでテスターに依存しがちであったテストケース、実行結果、検証報告を統一することも可能になってくる。

品質管理の向上につながるテスト自動化

上述のように、テストフレームワークはテストコードやテストデータなど、テスト実行に必要なものを一元管理することができる。また、これまでのテストケースをすべて自動実行することも可能となる。さらに実行結果、検証内容もフレームワークによって統一されるため、それらを利用して品質情報を抽出することも可能になる（次ページ図2参照）。

図2　従来のテスト手法とテストフレームワークを用いたテスト手法の流れ比較



この仕組みによって次のような効果が期待できる。

①漏れのないリグレッションテスト

開発・テストを進めるかたわら、リグレッションテストを自動で行うことができる。そのため、予期しない品質低下の発生や、修正による影響範囲の調査漏れなどを防ぐことができる。

②品質指標の早期確認

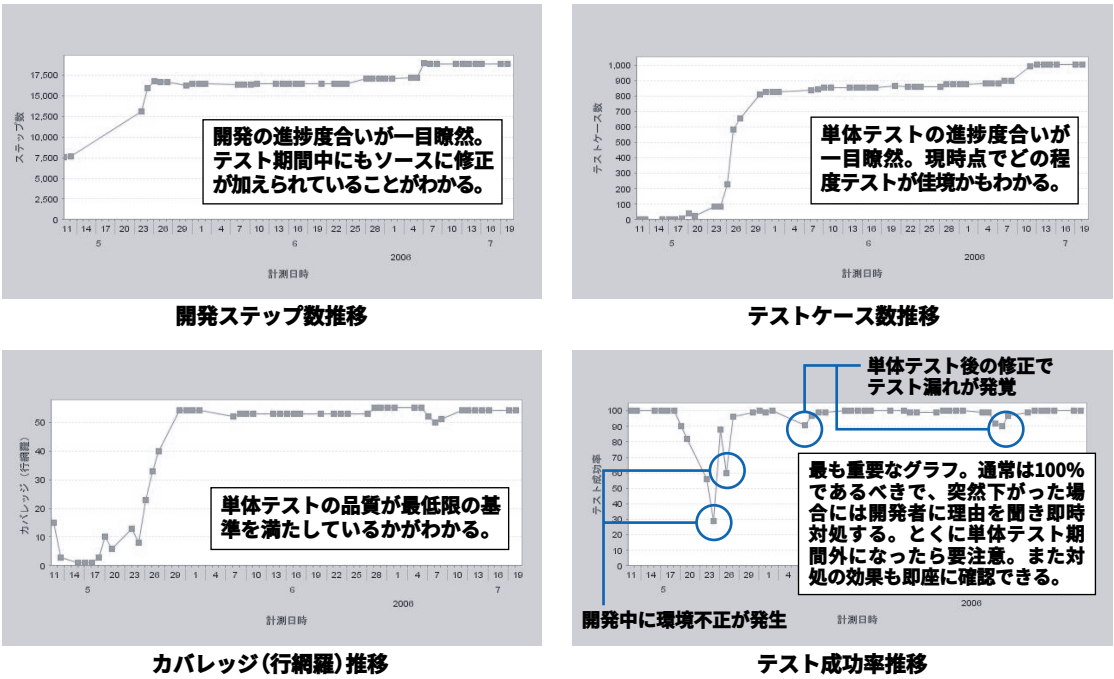
ステップ数、テストケース数といった成果物の進捗管理に加えて、テスト密度、障害発生率、カバレッジ（プログラムに対して実施したテストの範囲）といった品質指標についても日々進捗管理することができる。これまで、工程後の分析にしか利用できていなかったこうした品質指標を、プロジェクトを進める上でのリスク管理として利用することも可能になる（図3参照）。結果として、品質上の問題点をより早い工程で発見でき、より少ないコストで問題解決を図ることができる。

③パートナー会社の品質の可視化

開発などをパートナー会社に委託している場合、この仕組みを利用することで発注側での再現テストも自動で行うことが可能になる。品質指標のレポートも発注側で自動出力することができ、委託先での進捗状態、品質目標の達成度などを可視化することができる。ソースやテストコードなどを保管するリポジトリを発注側と受注側で共有すれば、毎日、場合によってはリアルタイムで進捗状態や品質の確認を行うことができる。

テスト工程を自動化すること、品質指標を早期から確認することは、それぞれ生産性向上、品質向上のための施策になる。しかし従

図3　品質指標出力の例



来は、この両者を同時に実現することは難しい面があった。テストの自動化は、スクリプトの作成など追加作業にともなうコストの増加があるため、繰り返しテストを行うだけでは効果が生まれにくい。品質指標の確認も、仮にグラフ化などを行う仕組みがあったとしても、必要な数値の取得、計算、登録を人手により行えばコストや精度の面で問題も出てくる。

これに対して、ここであげたテストフレームワークの仕組みでは、両者を結合することでコスト面の負担を最小限にし、メリットを最大限に引き出すことができる。たしかに、テストを自動実行させるためのスクリプトの作成はどうしてもコストを引き上げる。しかし、実行だけでなく準備、検証も自動化されることで、全体としてコスト上昇を抑えることができる。それよりも、テスト結果や、それに基づいた客観的な品質レポートを自動で生成し、共有できるメリットをコスト負担なしで得られることが大きい。

品質に直結するテスト工程を自動化することは、短納期で、かつ高品質なシステム開発を成し遂げていくために必須である。それとともに、開発中からつねに品質の達成具合を監視することができるため、品質の向上にも大きく寄与することになる。 ■