15-B-10

Web システムにおける単体テストの品質向上の取組み [1]

1. 概要

住友電工情報システム株式会社では、長年、ソフトウェア開発の品質改善活動と生産性向上活動を継続的に取り組んでいる。

本編では、Web システムの開発において、テストファースト手法の導入、自動テストの導入、および単体テストに用いるテストデータの自動生成システムの構築により、品質が大幅に向上した事例を紹介する。

2. 技術や手法導入の経緯・背景

2.1. 自動テスト導入についての経緯

従来の開発プロセスでは IT[2]以降でプログラム修正が必要になると、開発時に担当したプログラマとは別の担当者が修正することが多く、実現すべき仕様全てを把握することが十分にできないため、変更によって影響がありそうな部分だけをテストするという不確定なテストを実施していた。そのため、デグレード（以降、デグレ）が発生したり、テストのために用意するテストデータやテスト手順が妥当かどうか判断できないなど、テスト品質への懸念があった。

そこで、実現すべき仕様全てに対するテストを実装し、デグレ防止に取り組むため、テステストファーストの手法（設計、コーディング、テストの順に実施する手法）と、自動テストの導入を実施した。また、IT ・ST[3] 、保守のコスト低減を実施し、従来のコスト内で自動テストを開発することを目標とした。

2.2. テストデータ自動生成の構築の背景

当初、単体テストにおけるテストデータの作成工数が大きいという問題があった。テストデータの作成工数が大きくなる理由は、後続機能をテストする場合、先行機能から発生するデータを順に作成する必要があるためテストに必要なレコード（件数）が多くなり、レコードが増えるとテスト値を決定すべき項目も増えるからである。その上、テストデータを作成する際には、担当していない機能の仕様についても理解が必要となり、さらにテストデータ

[1] 事例提供: 住友電工情報システム株式会社 QCD 改善推進部　服部 悦子 氏
[2] Integration Test
[3] System Test

の作成工数が大きくなっていた。

また、自動テストでは、繰り返し何度でもテストを実行するので、テスト実施によって変わってしまうテストデータの状態を毎回テスト前の状態に戻すことが必要になる。例えば、「未発注のデータがプログラムを動作させると発注済みになる」など、データベースのテストデータは一度テストするとロジックによって状態が変わるので、再テストするときには毎回テスト前の状態に戻す必要がある。

このような、テストデータの作成工数の削減と自動テスト実現という課題解決のために、テストデータ自動生成の検討を開始した

検討を進める中で、単体テストで 14%を占めているテストデータ準備コストを半減させるという目標を設定した。

3. 技術・手法の導入時の事前準備や工夫

3.1. 導入体制

新手法の導入にあたっては以下の体制で実施した。

（1）　自動テスト

・　企画 9 名　期間 3 か月　延べ 2 人月

（ワーキンググループ 7 名で単体テスト項目抽出基準を整備、ツール対応 2 名）

・　準備 4 名　期間 2 か月　延べ 1.5 人月（マニュアル作成､自動テストの雛形作成）

（2）　テストデータ自動生成

・　企画 7 名　期間 6 か月　延べ 4 人月

（ワーキンググループ 7 名でシステム化の仕様作成）

・　準備 6 名　期間 3 か月　延べ 9 人月（自動生成システムの開発）

3.2. 自動テスト導入時の工夫

自動テストの導入にあたっては、導入による開発全体コスト増を抑えるために、以下の工夫を実施し、自動テストプログラムの開発工数を捻出した。

・　自動テストツールの選定

・　テストケース増大抑制（単体テスト工数の削減）

・　テストファーストプロセスの工夫

・　単体テスト項目の選定基準見直しの工夫

（1）　自動テストツールの選定

自動テストを検討したとき、Web 操作記録型のツール導入を考えたが、単体テスト（UT）が終了してからでないと同ツールは使用できないため、かえって開発工数が増大することが判明した。一方、コーディング型の JUnit[4] は、テストファーストのテ

[4] Java のプログラムで単体テストの自動化を行うためのフレームワーク　http://junit.org/

ストに JUnit による自動テストが利用できる。このため自動テストツールとしては、JUnit を選定することとした。また、画面からの入力値やセッション情報について JUnit のコードで実装できるようにした（図 15-B-10-1）。

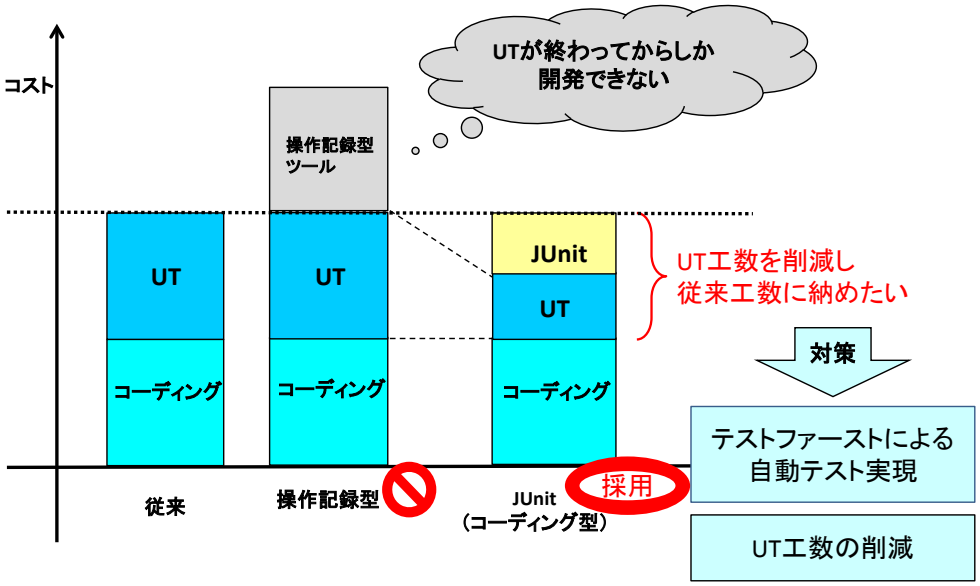

図 15-B-10-1　自動テスト化の検討

(2)　テストケース増大抑制（単体テスト工数の削減）

テスト工数の削減を図るため、フレームワークの機能拡充とアプリケーションプログラムの構造の見直しを行い、テストケースの増大を抑制した（図 15-B-10-2）。

従来のプログラム構造の場合、フレームワークの制約で、アプリケーションプログラムのメソッドですべてのチェックを連続的に実行するため、チェック A、チェック B、チェック C の各エラーチェックにおいて考慮すべき状態の組み合わせでテストケースが増大していた。

これに対して、アプリケーションプログラム内のメソッドを複数個呼び出せるようフレームワークを改善し、個々のメソッド単位でテストできるようにしたため、各エラーチェックの状態を組み合わせたテストケースは不要になり、テストケースを削減することができた。

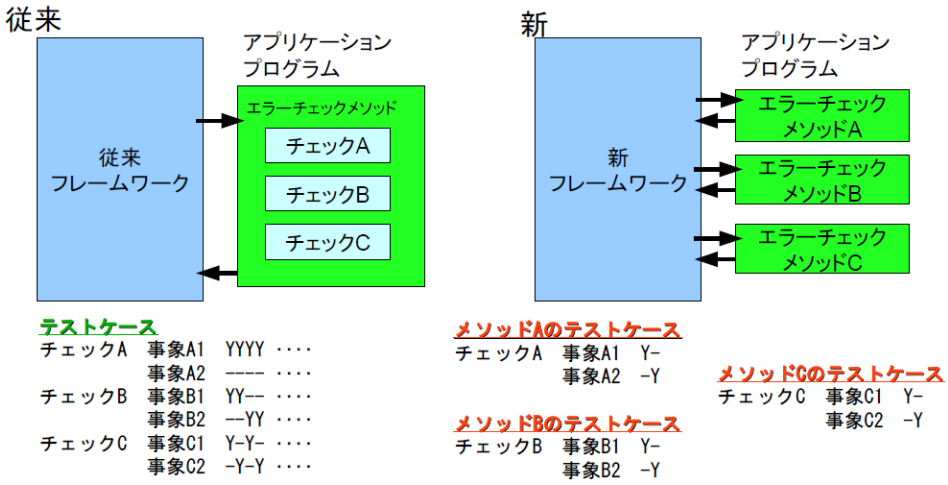

図 15-B-10-2　テストケース抑制のためのプログラム構造見直し

(3)　テストファーストプロセスの工夫

一般的なテストファーストは、最初にテストを作成し、そのテストが動作する必要最低限な実装を行った後、コードを洗練させる、という短い工程を繰り返すスタイルである。従来のプログラム開発は、コーディング、テスト設計、テストとシーケンシャルに各作業を実施していたが、本事例でのテストファーストは、メソッド毎にテスト設計、コーディング、テストの順番に実行するようにプロセスを定義して取り組みを実施した（図 15-B-10-3）。

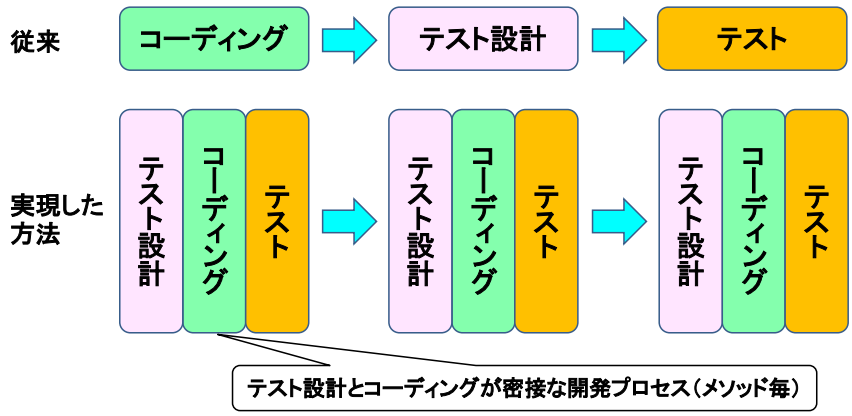

図 15-B-10-3　テストファーストプロセスの工夫

(4)　単体テスト項目の選定基準見直し

新しいフレームワークでは、上流工程での設計成果物（画面イメージ）が最終成果物（動作するプログラム）として再利用できる。したがって、上流工程でレビューしているものは単体テストを不要にできるため、単体テスト項目選定基準の見直しを行

い、テストが必要か否かを明確にすることにより、単体テスト項目の削減をすることができた。

3.3. テストデータ自動生成の工夫

本事例でのテストデータ自動生成システムの概要を図 15-B-10-4 に示す。外部設計者が作成した外部設計（機能設計）やデータベース設計（テーブル定義、項目定義）、プログラマが作成したクエリー設計などが、「楽々Framework SP[5]」上のリポジトリに格納されている。これらの情報とプログラマが作成するテスト設計をテストデータ自動生成システムに入力して、テストデータを自動生成する。作成されたテストデータは、プログラム開発時の単体テストだけでなく、プログラム検収者がテストデータのレビューを実施したり、IT・ST・保守フェーズにおけるプログラム修正時の単体テストにも活用する。

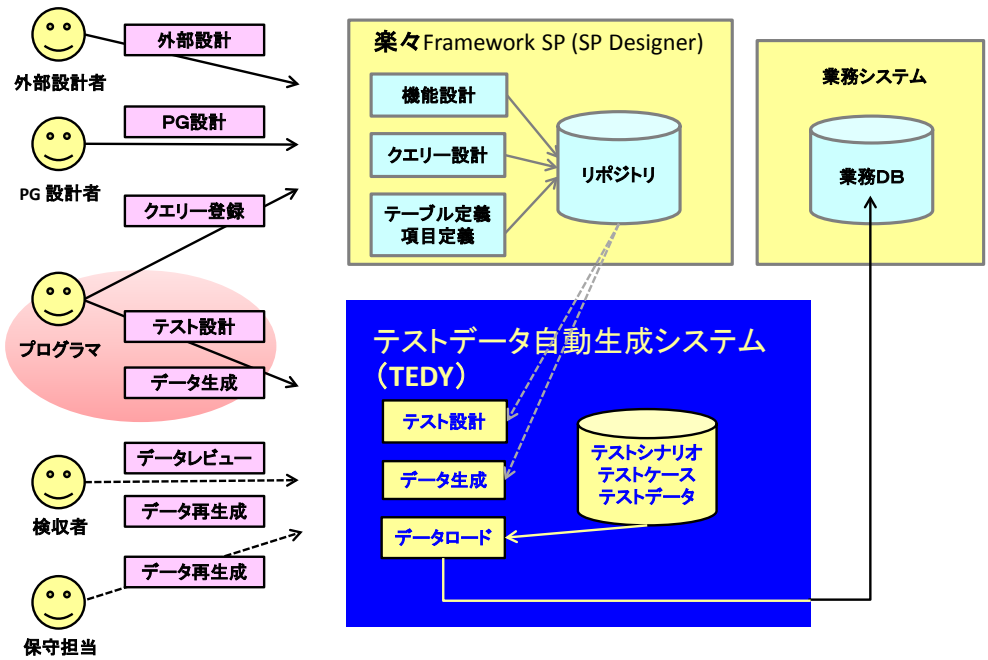

図 15-B-10-4　テストデータ自動生成システムの概要図

工夫の 1 点目は、テストケースとテストデータを一元管理するシステムとしたことである。テストデータはテストケース（テスト仕様）と共に管理すると再利用し易いという現場からの意見があり、これを反映したテストデータ自動生成システムを構築することとした。

工夫の 2 点目は、テストデータ自動生成システムを使用する開発者が、入力する情報を極力減らすことや、操作を簡単にする工夫を行った。具体的には、テストデータ自動生成シス

[5] 住友電工情報システム株式会社の部品組み立て型の Web アプリケーション構築／運用ツール「楽々Framework3」を拡張したもの

テムの作成にあたっては以下を考慮した。

- 必要なテーブルを選択するだけで整合性のとれたレコードが生成できること
- テストに直接関係しない項目は型・桁の制約に則った値で自動生成すること
  (テストに使う項目は値を指定できる)
- テストケース毎にテストデータの管理ができること
- 生成されたデータが繰り返しテスト環境へ反映できること

工夫の 3 点目はリポジトリのメタデータの活用である。データベースのテーブル定義、項目定義をメタデータとしてリポジトリに格納しており、その定義を使って、型・桁の制約に則った値を生成している。また、テーブル間の参照関係（制約）も同様である。

参考にテストデータの自動生成手順を以下に示す。

テストデータの自動生成手順

① テストで必要となるテーブルとテーブル毎に必要件数を指定（入力）する。
② テスト仕様として指定したい値を項目毎に指定（ex.受注数量＝100）する。
③ 「自動生成」するボタンを押すと、まずはテストデータ自動生成システム内にデータ生成する。生成手順は、①で指定されたテーブルの項目毎に、②で値が指定されているものはその値を、それ以外の値は上述のメタデータを活用して値を生成する。数値は桁の範囲でランダムに、文字は桁範囲で文字長をランダムに生成する。①で指定した件数、及び必要テーブル分を生成する。
④ 自動生成されたデータはダウンロード機能を使って生成された値を開発者が確認を実施する。ここで想定値になっていない場合は①、②、③を繰り返す。また、ダウンロードファイルを直接修正してアップロードもできる。
⑤ データができあがったら、最後に単体テストを行う業務システム側のデータベースへ反映する。
⑥ 単体テストを実施する。プログラムの不具合などでテスト項目のやり直しをする場合、⑤を再実施することで、業務システム側のデータがテスト前の状態に戻る。

4. 導入の効果測定と結果

4.1. テストファーストによる自動テスト

今回の事例における自動テストの対象は、図 15-B-10-5 に示すように、エラーチェック、ビジネスロジックの箇所であり、画面、データ抽出、更新処理は対象外である。本事例の Web システムは、楽々FrameworkSP というフレームワーク上で構築されており、リポジトリは Runtime Library で動作する。

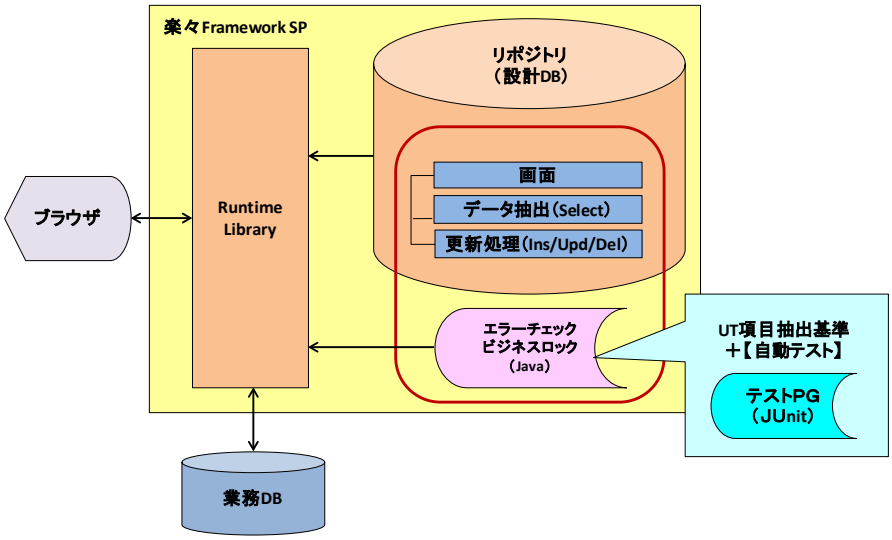

図 15-B-10-5　対象 Web システムの構成と自動テストの実現範囲

4.2. 自動テスト導入の結果

（1）　テストファーストの効果

テストファーストによる自動テスト実施プロジェクトの品質と生産性の結果を表 15-B-10-1 に示す。

品質については、従来のバグ件数が 47%に減少し大きく品質の向上に寄与したことが分かった。47%に減少したバグについて従来と今回の比較を表 15-B-10-2 に示す。ロジックの実装ミスや実装漏れが大幅に減少し、テストファーストの取り組みの効果が大きいことが分かった。

生産性については、目標であるコストを増加させておらず、従来と同等レベルである。

表 15-B-10-1　テストファーストによる自動テスト実施プロジェクトの結果

	組織基準値[注1]	自動テスト実施プロジェクト
バグ数[注2]	100	47
生産性	100	98.9

注 1：組織基準を 100 とした相対値

注 2：プログラム開発工程での作り込みバグ数

表 15-B-10-2　バグの内容

バグ区分内訳トップ 3	従来[注3]	今回[注3]
ロジックの実装ミス	23	2
実装漏れ	19	3
データベース処理（データ抽出、更新処理）	10	9

注 3：組織基準を 100 とした相対値

(2)　単体テスト工数の削減

自動テストにより手動テストの工数が減少した。これは、以下に記載している単体テスト実施時の画面遷移数の推移数からもいえることである。

Web システムにおける単体テスト作業の大半は画面を遷移させることが多い。このため画面遷移数を計数することでその効果を把握することが可能である。結果は以下の通りで単体テスト実施時の画面遷移数が約半減しているので単体テストの工数も大幅な削減ができたといえる。

単体テスト実施時の画面遷移数

従来：1.99 回／JaX[6]

今回：1.03 回／JaX

(3)　自動テストカバレッジ

全本体コードの約 96%が自動テストを実施できた。メソッド単位で見ると、最大は 100%、最小でも 82%の自動テストカバレッジであり、自動テストが実施できていないルートは異常ルートなどのみで､他は問題なく自動テストを実施することができた。

4.3. テストデータ自動生成システムの構築

テストデータ自動生成システムの構築の有効性について、その確認方法を以下に示す。

確認方法

・本番稼動中システムの照会機能について、開発時にテストデータ作成に要した時間（テストデータ自動生成システム未使用）と生成システムを使った場合の時間を比較した。

・また、自動生成システムを使った場合、開発時に手動で作成したデータを全て [7]生成できるか確認した。

テストデータ自動生成システム構築の有効性確認結果を表 15-B-10-3 に示す。テストデータの作成時間は、従来の 4 時間から 1.5 時間へ約 38%に減少している。1 シナリオ平均 15 分でテストデータ生成からデータベース登録、プログラムの動作確認まで実施できたので、シナリオ数が多ければ多いほどその効果が発揮される。また、テスト実行前にデータベースへの登録ができることと、繰り返しテストができるようになった。

表 15-B-10-3　テストデータ自動生成システム構築の有効性確認結果

	生成システム未使用	生成システム使用
データ作成方法	手動で作成・登録	テストデータ自動生成で生成・登録
データ網羅率	100%	100%
データ作成時間	4 時間	1.5 時間

[6] JaX・・・住友電工情報システムで使うプログラム規模（ライン数相当）

[7] 照会機能のテスト数は 6 シナリオ、17 テストケース、テストデータは 6 テーブル、60 レコードである。

4.4. 結果と考察

(1) 品質について

テストファーストによる自動テストの一連の取り組みにより、4.2 の表 15-B-10-2 に示すようにバグの作り込みを大幅に減少させることができたことと、自動テスト化により単体テスト中のデグレの防止ができたことで、品質が大幅に向上した。さらに、IT、ST、保守フェーズにおけるデグレの防止も期待できる。

(2) 生産性について

今回の一連の取り組みでは、工数が増加することもなく、従来のコスト内で自動テストを実現する目標を達成できた。また、テストデータ自動生成システムの構築により、データ作成時間が従来の 38%に減少したため、テストデータ準備コストを半減させる目標を達成することができた。

(3) 検証への影響

単体テストでの品質向上により、その後の結合テストやシステムテストで発生するバグが減少し、そのバグの対応時間も減少できている。また、結合テストやシステムテストで検出されるバグへの対処では、自動テストが整備されているため対処時のデグレ防止に効果がある。

5. 今後の課題と取り組み

(1) 自動テスト

自動テストについては以下の改善を図っていく。

① 習熟スピードの向上

経験者に比べ習得に時間がかかるプログラム初心者でも、1 か月で習得させたい。

② 自動テスト範囲の拡大

自動テストの実現に取り組んできたが、現在は、画面、データ抽出、更新処理については、まだ従来通りに手動テストを実施している。今後は、データ抽出、更新処理についても自動テスト化を目指す（図 15-B-10-6 の今後の自動テスト化範囲）。

また、プログラム中の画面遷移を実際にブラウザで動作させるテストについては自動化できていなため、自動テスト化を取り組む。

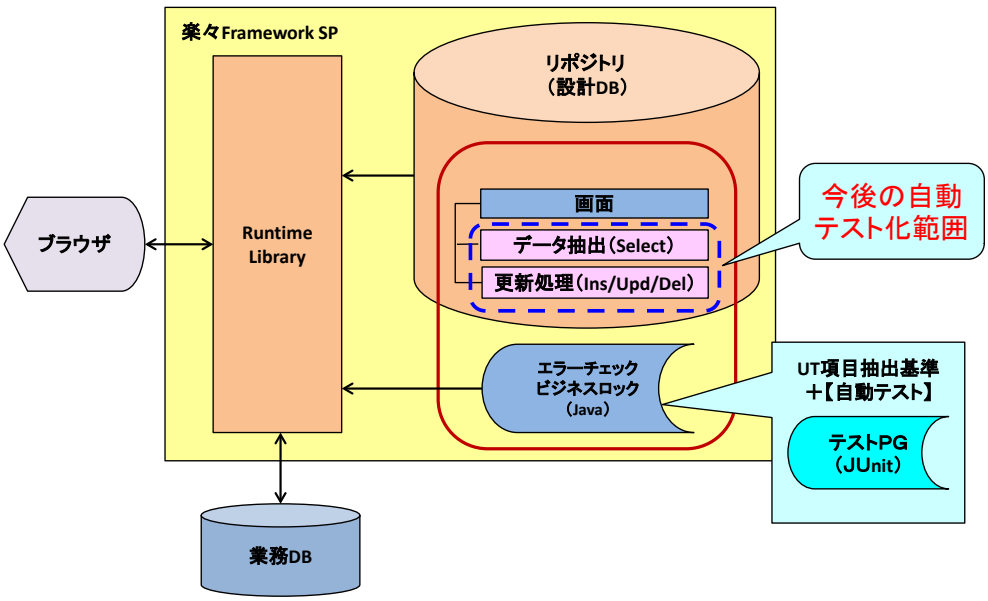

図 15-B-10-6　今後の自動テスト化範囲

(2)　テストデータ自動生成システム

部品表など再帰構造データの自動生成や既存値の流用など、テストデータ自動生成システムの機能強化を図っていく。

(3)　自動テストとテストデータ自動生成システムの連携

自動テストとテストデータ自動生成システムは個別のシステムとして構築したが連携化できていないことが課題である。これらを連携させることにより、テストコードから直接、業務データベースへ反映させることが可能になる。

(4)　水平展開

他のプロジェクトへ展開するためには、新手法の教育コースが必要となる。現在そのコースを整備中で、これが確立した後、他のプロジェクトの要員を教育し、水平展開を実施する予定である。

6. まとめ

テストファースト手法と自動テストの導入、および単体テストに用いるテストデータの自動生成システムの構築により、Web システムの単体テストでの工数を増やさず、品質が向上した事例を紹介した。また、テストファーストの取り組みでデグレを防止し、単体テスト・IT・ST・保守の各コストを低減させることができた。今後は、機能的な改善、要員教育の実施、ノウハウの収集・展開などの取り組みの実施が期待される。

参考文献

[1] 住友電工情報システム株式会社　服部悦子　SPI Japan 2014　テストデータ自動生成による品質・コストの改善～テスト設計システム構築とアイデアを出す工夫～

[2] 住友電工情報システム株式会社　服部悦子　SPI Japan 2013　Software Product Line の実践～テスト資産の構築～

独立行政法人情報処理推進機構 技術本部 ソフトウェア高信頼化センター（IPA/SEC）