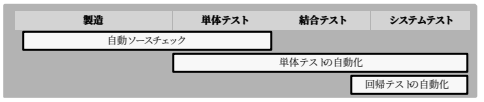
Webシステム開発におけるテスト自動化の実践

JaSST 2004

2004年1月27日
株式会社NTTデータ 井上 康臣

概要

- Webシステム開発時、さまざまな工程でテストツールを適用した それぞれの実践から得られた「自動化のポイント」について概説する
  - **自動ソースチェック**
    - **Checkstyle**（オープンソースのJavaスタイルチェッカー）による製造時のリアルタイム・スタイルチェック
    - **Jtest**（商用のJavaソースコードチェックツール）によるソースコード品質の底上げ
  - **単体テストの自動化**
    - **JUnit**（オープンソースのJava単体テストツール）による単体テスト強化
  - **回帰テストの自動化**
    - **WinRunner**（商用のGUIテストツール）による回帰テストの効率化



ソースチェック自動化の実践

導入事例

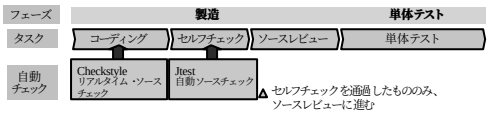
自動ソースチェック

- 概要
  - CheckstyleとJtestを導入することでソースチェックを自動化し、ソースコード品質と作業効率の向上を図った
- 導入の背景
  - 全てのソースコードを目視でレビューする作業が非常に大きな負担だった
  - プログラマのスキルにバラつきがあり、ソースコード品質もバラつきがあった
  - ソースコード品質が低い場合、ソースレビューでは基本事項のチェックに追われ効率が悪かった
  - これらの問題を解消するため、ソースチェックツールを導入した

実施内容

自動ソースチェック

- Checkstyle、Jtest導入によるセルフチェックの自動化
  - 当初、Jtestによりコーディング後の一括チェックを行っていたが、軽微なバグが多発して修正作業に手間がかかっていた
    ⇒Checkstyleを導入し、軽微なバグはコーディング時に発見し修正
  - セルフチェックの確実な実施による、正規のソースレビューの効率化
    ※セルフチェックに合格しないと先に進めないルールとした



実施結果

自動ソースチェック

- 導入効果
  - ソースレビュー、セルフチェックの大幅な作業工数削減
    - 全ソースコード数分～数時間でチェック。
  - ソースコード品質と技術者スキルの平準化ができた
  - 全ソースコードを指摘漏れなくチェック⇒見逃しバグ削減
    - ある担当者：「この手のバグは目視チェックで徹底的につぶしたはずなのに。」
  - ソースチェックの効果
    ～ソースチェックを行った担当者のコメントから～
    - 「委託先のソースコード品質の分析ができた」
    - 「いかに自分たちが常識を知らずにコード汚していたか思い知らされた」
- 注意事項
  - チェックできる内容は限られている
    - イディオムやコーディングスタイルのチェックのみなので、業務仕様にあっているかといった確認は目視にて行う。
  - ルールの絞込みが必須
    - Jtestの344種類のルールの中には自プロジェクトにとって不要なものもあり、重要なルールのみをピックアップする作業が必要。
      ⇒Java有識者によるピックアップを行い100種類程度に絞り込んだ。

今後の展開

自動ソースチェック

- ソースチェックは高い効果が得られたため、今後、積極的に推進していく
  - 社内向け 標準 チェックルール を策定した
    - 社内コーディング 標準に準拠。
    - 各ルールを品質特性 機能性、性能など)に分類し、ランク付けした。
  - 導入が 円滑に 進められるよう、導入 手順書、運用 ルール を作 成した

- 得られたノウハウ
  - コーディングアンチパターン
    - 多くのプロジェクトで検出されたバグに関して取りまとめ、コーディング 標準に 盛り込むようなフィードバックを行っている
  - 下記の ルール は、いくつかの プロジェクトで違反が 検出されたもの 例)
    - finallyブロックで入力または出力をクローズする
    - string.equals ("リテラル")ではなく、"リテラル" .equals(string)を呼び出す
    - 文字列 リテラル を保持するためのStringオブジェクトを作成するのに newは使用しない
    - 直接Exceptionをスローするのを避ける。常にexceptionのサブクラスを使用する
    - Object.equals ()をオーバーライドするときは常に、Object.hashCode()もオーバーライドする
    - ¥nまたは¥r を行分離記号としてハードコーディングしない
    - etc ...

自動チェックツールの'○'と'×'

自動ソースチェック

- Checkstyle
  - ○と考 えられる点
    - フリー であり 公開されている情報も充実してきているため、導入は非常に容易。Sun提供のコーディング 標準への準拠性をチェックするのみであればノンカスタマイズで適用できる。
    - Eclipseプラグインでの利用の場合、ソースコード保存時にチェックが走るため問題がすぐに明らかになり修正が迅速に行える。
  - × 制限事項)
    - 発見できるのは、命名規約やスタイル基準の準拠など、比較的軽微な問題のみ。
- Jtest
  - ○と考 えられる点
    - 344種類という 幅広い ルールに関してのチェックが可能。
    - チェックルールは新規に追加することも含めて高度なカスタマイズが可能 要スキル)。
  - × 制限事項)
    - 重要なルールの絞込みが必須
    - 使いこなすのにスキルが必要 専任の管理者を配置する必要がある)。
      - ツールの仕様上、コンパイルできる環境を用意しなければならず、環境に変更があったりすると、Jtest 側もメンテナンスが必要。メカニズムを把握している管理者が必要。

単体テスト自動化の実践

導入事例

単体テストの自動化

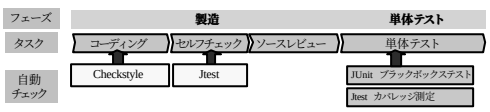
- 概要
  - 品質重視のプロジェクトにJUnitを導入し、ソースコード品質の確保と単体テスト作業効率の向上を図った
  - 自動ソースチェックツールも併用し、より高品質なソースコードの開発を目指した

- 導入の背景
  - すべてのテスト項目について実施証跡 の提出が 求められていた
  - 単体テストレベルの回帰 テストを確実に実行できるようにすることで、高い品質の確保を狙った

実施内容

単体テストの自動化

- JUnit導入による 品質確保と作業効率改善
  - JUnitでテストを実施し、Jtest でカバレッジを測定した
    - デバッガ を使用しなければ通せない 一部 のパスは、JUnitの テストコードを単にドライバとして利用しデバッガ を併 用した。
  - JUnitのテストコードと実施結果の 画面を実施証跡 として利用することで、実施証跡を取得する作業の工数削減を図った



実施結果 (1)

単体テストの自動化

- 導入効果
  - ソースコード品質の確保
    - 例外処理など、結合 テストでないと発 生させづらい部 分を除いた箇所 を網 羅する回帰 テストが用 意できた。
  - 単体テスト作業工数の 削減
    - 実施証跡として入出力時 の画 面キャプチャなどを取得しその ドキュメント類を整備す る労 力に比 べると、JUnit の テストコード作成 は負 担が 少 ない。
  - 導入の容 易性 (JUnitのメリット)
    - JUnit はシンプル であり プログラマにとって導入 についての 抵抗感 や初期コストは少 なくて済む。
    - 当初は、ツール運用が スムーズに行 われるか 心配していたが、実際導入してみると発 生したのは些 細な問題だけだった 。
    - 雑誌や 書籍、Web などでも多く取り上げられているため、各 自でノウハウ情 報が入手しやすいのも大 きい。

実施結果（2）

- 注意事項
  - ツールが使えることと、よいテストができることとは別の話
    - 問題はいかにテスト項目を「正しく」抽出するかという点。
    - NULL値の扱いなどエラー推測的なテストケースが抽出できているか、目視レビューで確認する。
  - テストコードのボリュームは、テスト対象プログラムより大きくなる（平均1.5～2倍程度）
    - プログラム作成の他に、テストコード作成の期間を見込んでおく必要がある。
  - プログラム上の煩雑な手続きを共通化するようなライブラリが必要になる
    - JUnit、Javaの有識者がライブラリを開発し、適用した。

今後の展開

- テストコードを"活かす"取り組み
  - 自作ツールにより、テストコードから情報を取り出し様々な作業を効率化する仕掛けを用意した
    - テストケース一覧表をリバース生成するツール。
      - テストコードにタグつきのコメントを入れておくと、コメント部分を抽出・整形して表示する。
    - テストケース数カウントツール。
      - テストコード内にあるテストメソッド数をカウントするツール。
- 得られたノウハウ
  - テストコード用ライブラリ
    - 多くのテストコードから呼び出される一連の手続きなどは、ユーティリティクラスとして定義しておく
    - 例）
      - 日付比較（Dateクラスをassertで比較）。
      - SQL実行（テーブルのクリア/ロールバックなど）。
      - privateメソッドの呼び出し　など。

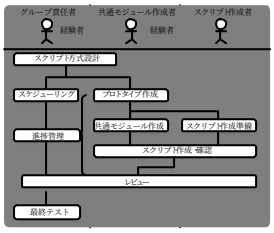
回帰テスト自動化の実践

導入事例

- 概要
  - 頻繁に回帰テストを手動で実施していたプロジェクトにおいて、GUIテストツール・WinRunnerを導入して回帰テストの効率化を図った
- 導入の背景
  - 回帰テストの実施状況
    - リリースした後で頻繁にバージョンアップがあり、その都度手動で回帰テストを実施していた。
    - 毎回10名の担当者によりテストを実施していたが、同一の要員を継続して確保するのは難しく、入れ替えが頻繁にあった。
  - ここには次のような問題があった
    - 回帰テストには非常に大きな工数を必要としていた（1回あたり約50人時）。
    - 手動によるテストでは操作誤りも多く、作業を遅延させる要因となっていた。
    - テストシナリオは煩雑で、テスト実施は担当者にとって大きな負担になっていた。
    - テスト担当者入れ替え時の教育コストが高くついていた。
  - 上記の問題解消のためWinRunnerを導入し、回帰テストの効率化を図った

実施内容（1）

- スクリプト開発体制と作業フロー
  - 開発体制
    - テストスクリプト開発は、専任」で行った。
  - 作業フロー
    - 役割分担と作業フローが必要。



各役割に必要なスキル

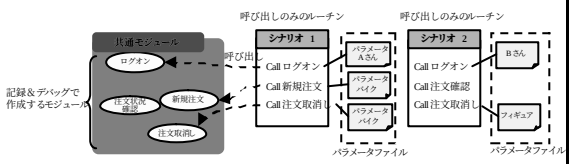
グループ責任者：
GUIテストツールの深い知識
作業管理能力

共通モジュール作成者：
GUIテストツールの深い知識

スクリプト作成者：
GUIテストツールの知識

実施内容（2）

- スクリプト方式について
  - 再利用を考慮した方式の採用
    - 呼び出しをするのみのルーチンから共通モジュールを呼び出す方式。
    - データ駆動型のスクリプト。
  - シナリオを追加する場合は、呼び出しのみのルーチンとパラメータファイル（テストデータ）のみを作成すれば良い



実施結果（1）

- 導入効果
  - テスト担当者の削減
    - 人間に代わってGUIテストツールが画面操作を実施するため、要員を削減できる。
    - 今回の事例では、10人→5人に削減できた。
  - テスト担当者にかかる負荷の削減
    - 回帰テスト時、数時間の連続操作による緊張状態など、担当者に非常に大きな負荷がかかっていた。
    - ツールを用いたテストは、画面を眺めている時折必要な操作を施すのみでよい。
      ⇒テストは「面白く」、「非常に楽」（担当者談）になった。
  - 正確性の向上、テストの実施管理の容易な実現
    - 正確に・確実に実施したい記録が残ることになる。実施漏れなどは記録からすぐに発見できる。
  - スクリプト開発中のディグレード発見による品質向上
    - スクリプト開発中にも数回、画面操作を行う。この際にもディグレードが発見できた。
    - 今回の事例でも、運用時の重大なバグも含め5,6件発見できた。

実施結果（2）

- 注意事項
  - 工数面では、数回以上の繰り返しが想定される場面で有効（付録参照）
  - スクリプト開発の専任体制が必要。導入時は経験者による支援も必要
  - あまり小規模なプロジェクトには向かない
    - 要員コスト、ツールコスト（導入コスト、スクリプト開発コスト）がデメリットになる。
  - 「運用、保守」フェーズでの導入をオススメ
    - 時間的な制約が厳しい新規開発時で頻繁に回帰テストを実践するのは難しい。
  - 次のような導入リスクも考慮しておく必要がある
    - ツールを使いこなすには スキルが必要。スクリプト開発コストに加えて「導入コスト」も見込んでおく必要がある。

「GUIテストツールを導入すれば、非効率な回帰テストが楽々自動化される」と安易に考えるのは禁物
⇒導入に際してはプロジェクトの特性を考慮する

今後の展開

- 得られたノウハウ
  - スクリプト開発作業の標準化
    - 再利用性を考慮したスクリプト方式
    - 作業プロセスの定義
- テストスクリプト作成はソフトウェア開発そのものと心得る
  - プロジェクト管理者と明確な作業規約が必要
    - 必要な作業規約
      - スクリプト規約。
      - 共通モジュールの定義書。
      - GUIテストツールのノウハウ。
  - テストスクリプト開発にも次のような作業を実践した
    - 進捗管理
      - 毎日ミーティングを行い、状況把握・対応を迅速にできるようにした。
      - 問題発生時には、誰」が、「いつまで」に対応するのか明確に一覧管理しておく。
    - スクリプト管理
      - 共通モジュールを複数人で編集することがあり、一日数回バックアップをとっていた。

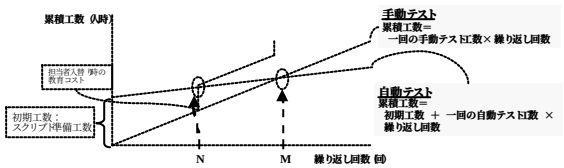
まとめ

まとめ

- テスト自動化により品質面、工数面で成果をあげることが出来た
  - 複数ツールを導入することで高い効果が得られる
    - シンプルなツールが充実し、導入障壁が低くなってきている
    - テストツールの推進担当者が必要
  - より効果的なツール利用のため、適用結果をフィードバックすると良い
    - ソースチェック結果をアンチパターンとして活用
    - テストケースのリバース生成などJUnitのテストコードを活用するしくみ
    - 作業効率を考慮した作業プロセスとスクリプト方式
- ツール導入により万事がうまくいくわけではない
  - よいテスト設計は人間が行う
  - ツールの制限範囲を見極める
    - 人間の作業の一部を肩代わりするものと考える
    - 作業の一部をツールで行うこととした運用ルールを作成する必要がある

付録）手動テストとの工数比較

- 手動テストに必要な工数と自動テストを導入した場合の工数比較
  - 自動テストではスクリプト準備工数が必要になるため、何回か繰り返さないと投資が回収できない
- 今回、および過去の事例から損益分岐となる回数を算出
  - GUIテストツールによるテストをN回でM回繰り返すと「工数での」元が取れる
  - 手動のテスト担当者が頻繁に入れ替わることを考慮するとN回の繰り返しで元が取れる
    - 担当者入れ替えによる教育コストなどを考慮。



- 導入検討時には、N～Mを損益分岐点とし、費用効果を考える