実機テスト自動化によるテスト効率改善について
－大規模なテスト自動化におけるテストスクリプト設計方針と実現－

于　澎†

†株式会社 ACCESS ソフトウェア事業部 〒261-0023 千葉県千葉市美浜区中瀬 1-10-2
E-mail:　Yu.Peng@access-company.com

あらまし　組込みソフトウェア開発において、最低限のコストでビルドの作成毎に改修確認を兼ねてシステムテストとストレステストを実施するため、実機を使用した自動テストを行った。テストスクリプトを適切に作成しなければ逆に工数増加、実施時間増加、テストチームのモチベーション低下に繋がってしまう恐れがあるため、テストスクリプトの保守性、移植性、信頼性の向上に重点を置きながら、効率よくテストスクリプトを作成し、最終的にテスト実施コストが約 70%に短縮することができ、開発全体の効率を改善することができた。また、将来を見据えてテストスクリプト仕様の理想像について考察した。

キーワード　実機、テスト自動化、テストスクリプト

Improving efficiency of test automation test running on a real device
－Design strategy and implementation of automated test scripts for large test－

Peng Yu

† Software Business Unit, ACCESS Co., LTD

1-10-2　Nakase, Mihama-ku, Chiba-shi, Chiba 261-0023 Japan

E-mail:　Yu.Peng@access-company.com

Abstract　In an embedded software development, We performed an automated test in order to make the system tests and stress tests executable for each release at the lowest cost. To reduce testing cost as much as possible, we focused on improving the reliability, the maintainability and the portability of test scripts and tried auto-creating test scripts using Excel Macro. As a result, we succeeded in reducing the testing costs by about 70% eventually. And we also discussed the vision of the specification of test scripts.

Keyword　Real Device，Test Automation，Test Script

1. はじめに

リリースごとに回帰テストを実施することにより、不具合の修正によるコードの変更や新たな実装についての品質を確認することができるが、ソースコードの規模に比例してテストにかかる工数も増大するため、テスト自動化を導入し、テスト工数の削減の必要性があった。テスト自動化の導入効果として以下があると考えられる。

- テスト工数削減（人手の確保）
- テスト期間短縮
- テストチームのモチベーション維持
- 品質の定点観測
- 再利用可能

しかし、実機を使用したテスト自動化は参考文献が少ない上、テストスクリプトの品質への要求が PC 上で行う自動テストのスクリプトより格段に厳しくなる。品質の悪いテストスクリプトは導入効果を薄め、場合によっては逆効果を招いてしまうこともある。

そのため、テストスクリプト設計時にスクリプトの保守性、移植性、信頼性、効率性に重点を置きながら、マクロを利用して効率的にスクリプト生成を行うことにより、導入効果を最大限に引き出すことを実現した。

1.1. テスト対象について

本稿では某メーカー向け端末機器（タッチパネル搭載）に搭載された弊社主力製品Netfront® Browserをテスト対象とする。PC版シミュレータでテストを行った後、実機に搭載された状態で品質評価と品質確保のため、リリース毎に合計 5000 件ほどのシステムテストと 6 時間のストレステストを行った。

1.2. 自動化テストツールの選定について

弊社では自動化テストツールの導入に伴い、複数の

テストツールについて次のような視点でツールの評価を行った。

- 必要とされる全ての実機操作が可能であること
- 強力な画像処理機能が搭載されていること
- 柔軟なフロー制御が可能、かつ習得しやすいスクリプト言語によるコーディングが可能であること
- スクリプトの文法チェック機能を有すること
- WindowsPC を同時に制御可能であること
- 他言語と連携できるインターフェースを持っていること
- 技術面・運用面でのサポートが得られること

これらの条件を元に検討の結果、弊社では日本ノーベル社のテスト自動化ツール『Quality Commander 5』（以下 QC と略す）を導入した。

1.3. テスト自動化ツールの利用時の問題点

本案件より以前にも QC を導入したことがあり、既知の問題点として、以下のものがあった。

- テストスクリプトの作成、品質確保に時間を要する
- 作成済みテストスクリプトを他案件に流用することが困難
- 仕様変更等でテストスクリプトのメンテナンス作業が頻繁に発生する
- リリース毎に顧客に引き渡す時間が決まっており、仕様変更対応に要する時間が十分に与えられなく、一部テストの手動実施を余儀なくされる
- テストスクリプトの不具合により、自動テストが止まる

上記の問題点を解決すべく、テストスクリプト作成に取り組んだ。詳細を 2 章から 3 章で説明する。

1.4. 実績

1.4.1. スクリプト作成

表 1 テストスクリプト作成工数の実績

テストスクリプト作成実績日数	18日
テストスクリプト作成実績工数	2.13人月
テストスクリプト作成実績本数	5270本
テストスクリプト行数 （空行抜き、説明文注釈込み）	188669行
テストスクリプトあたり行数	35.8行/本
一行あたりの作成時間	3.97人秒/行
実質テストスクリプト行数 （空行抜き、注釈抜き）	126497行
期待値画像枚数	21140枚
手動記入テストスクリプト行数	6550行
テストスクリプト自動作成 Excelマクロ行数	330行

1.4.2. テスト実施（工数）

表 2 Testing by Manual(予想)

	1ヶ月目	2ヶ月目	3ヶ月目	4ヶ月目	5ヶ月目	6ヶ月目	7ヶ月目	合計(人月)
テスト工数	3.00	5.50	5.50	5.50	5.50	5.50	5.50	36.00

教育　テスト実施(5パス)

表 3 Testing by Robot(実績)

	1ヶ月目	2ヶ月目	3ヶ月目	4ヶ月目	5ヶ月目	6ヶ月目	7ヶ月目	合計(人月)
スペシャリスト	0.50	1.20	1.00	0.50	0.50	0.20	0.20	3.9
エンジニア	0.50	1.20	1.00	1.00	1.00	1.00	1.00	6.5

スクリプト設計　スクリプト作成　テスト実施(8パス)

※ツールのレンタル料金は約 500,000 円/月

- 合計でコストをおよそ 70%に削減できた
- リリース毎にテスト実施ができ、不具合調査、改修が早くなる
- 人間の休み時間にテスト実施ができ、最終的にテスト 3 回多めに実施できた
- 人間が実施に向いていない長時間ストレス系テスト実施も楽になり、チームモチベーション維持に貢献

1.4.3. テスト実施（実施時間）

表 4 自動テスト実行時間の実績

	1パス目	2パス目	3パス目	4パス目
実行時間(h)	107.57	120.28	107.44	117.00
実質実行時間	78.22	92.55	94.25	96.74
稼働率※1	72.71%	76.94%	87.73%	82.69%
途中で止まった回数	12	1	4	1
平均故障間隔(h)	8.9643	120.28	26.859	117

※1稼働率=実質実行時間÷実行時間×100%

最初の 1 パス目は不具合がたくさん残っており、12 回も止まったが、2 パス目以降止まる回数が減り、安定してテストできるようになった。

1.4.4. 不具合検出件数

表 5 不具合検出件数

	1パス目	2パス目	3パス目	4パス目	5パス目	6パス目	7パス目	8パス目
新規不具合件数	13	0	0	0	0	0	2	0

2. テストスクリプトの品質特性

テストスクリプトもソフトウェアであるため、ISO9126 で定められている品質特性と品質副特性で品質を評価することができる。しかし、ソフトウェアをテストするソフトウェアという特殊な位置づけから、設計時に作りこまなければならない品質特性の優先度は普通のソフトウェアとは異なる。本案件では、以下の品質特性に重点を置いて設計を行った。

2.1. 保守性

テスト対象以上にテストスクリプトに保守性が求められる。

テストスクリプトは一度作成すれば、長く続いて使用できるものでなく、以下のように様々な要因により変更する必要がある。

- テスト対象の仕様変更
- テストコンテンツの変更
- テストケースの変更

テスト対象の仕様変更はテストスクリプト変更の一要因でしかないため、テストスクリプト変更の確率はテスト対象の仕様変更が発生する確率よりも大きいということになる。(図 1　振り幅は変更発生確率を表す)

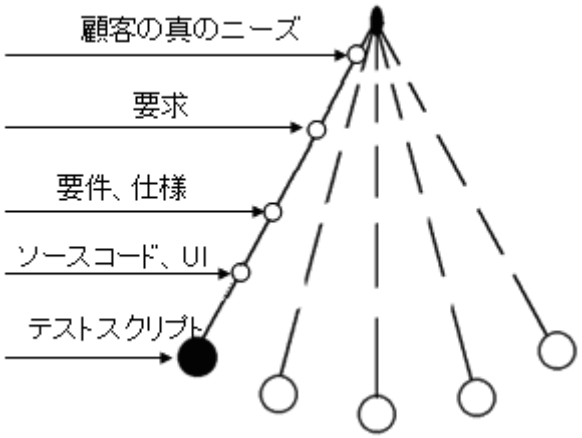

図 1　テストスクリプトの変更確率が大きい

一方、テストスクリプトの修正時間は十分に与えられないことが多い。そのため、どのようにテストスクリプトの保守性を高めるかがテスト自動化を成功に導くための鍵となる。保守性を高めるには、以下の方針でテストスクリプトを作成する必要がある、

- 変更箇所を特定しやすくする
- 変更箇所を少なくする
- 変更漏れ、変更副作用を最低限に抑える

また、テスト対象の仕様変更は過去データを分析すればある程度分かるため、仕様変更を見越したスクリプト作成も重要である。

2.2. 信頼性

テスト対象以上にテストスクリプトに信頼性が求められる。

人間は週末、休日、業務時間外を合計すると、年間78%の時間仕事をしていない。その時間を利用し、テストを可能にすることがテスト自動化の大きなメリットの一つである。しかし、人間が休んでいる間にテストがスクリプトの不具合により止まった場合、テストを再開させることができず、テスト自動化のメリットは薄れてしまう。そのため、テスト対象が不安定でも、テストスクリプトは安定していなければならないわけである。

主なテストスクリプトの不具合：

- 無限ループ
- 画面遷移がずれて、操作が無意味になる
- 致命的なメモリの使用ミスでテストツールがクラッシュする
- 画像判定ミスよる誤分岐
- テスト対象の既知バグによる誤動作

2.3. 移植性

テスト対象並にテストスクリプトに移植性が求められる。

一度作成したテストスクリプトが会社の資産となり、次の案件に流用できることがテスト自動化の大きなメリットの一つである。そのため、

- テストスクリプトの機能を細分化して、機能もしくはモジュール毎に切り出せるように設計する
- なるべくモジュールに汎用性を持たせる
- 弊社の場合、自社製品のソフトウェアを顧客のハードウェアに載せてテストすることが多いため、ハードウェアに依存する部分と依存しない部分を分けて設計する

ことが求められる。

3. テストスクリプト設計

3.1. テストスクリプトの階層化

大規模なテストスクリプトでは、端末操作以外にも座標変換、実行時画面解析、端末の操作に対する反応確認やリトライなどの操作がよく行われるため、テストスクリプトを作成する前に、よく行われる操作をまとめてライブラリ化することでテストスクリプトの保守性を向上させることができると考えられる。また、実機を使用した自動化テストにおいて一般的なライブラリ構造をまとめておいて、本案件以降も同様な構造でテストスクリプトを作成することでライブラリ部分の保守性と移植性を向上させることができると考えられる。本案件では以下の階層化モデルを提案し、その

階層モデルでテストスクリプトの階層化を行った。試験手順層以下の層がライブラリという位置づけである。

図 2　テストスクリプト階層図

- 試験手順層：この層の関数はテストケースをテストケースと同じ粒度で転記したもので、コード行数はテストケースステップ数の1～2倍が望ましい。下位層で定義された関数を呼び出して操作を行う。関数の先頭にテストケース ID、詳細、手順、期待結果などの情報を記入する。試験手順層の関数はテストケースが変更されない限り、変らない
- 操作確認層：操作が行われるたびに呼び出される。操作が正しく行われたかを確認し、正しく行われなかった場合、取るべき操作を取る。
- 座標変換層：座標以外の形で指定された押下位置を座標に変換する
- 判定層：静止画、文字列、音声、動画などを比較し、一致するかを判定する。または片方がもう一方のどこに位置するかを割り出す。
- 情報処理層：画像処理、音声処理などを行う。
- 操作層：座標を受け取って押下する、カメラで写真を撮るなど最も基本的な操作をする
- 物理層：ハードウェアのドライバなど電気的に操作層の操作を実現する

階層化には以下のメリットがあると考えられる。

- 各層の機能が明確になり、問題が出た際に問題特定に要する時間が少なくなる
- 今後の案件も同様な階層化モデルで設計することで、テストスクリプト（ライブラリ部分）の移植性が向上する
- 各層で行われる作業が明確になることでバグを作りこむことが少なくなり、信頼性が向上する。また、修正時の修正漏れと修正副作用も少なくなる

3.2. テスト情報の分離

3.1では、操作の観点でテストスクリプトをメンテナンスしやすいように小分けするが、3.2ではデータの観点でテストスクリプトを小分けする。

実機を使用した自動テストにかかわる要素として、

- テストケース
- テストコンテンツ
- テスト対象ソフトウェア（UI、遷移方法など）
- テスト対象ハードウェア（サイズなど）
- テスト環境（実機の置き方など）
- テストツール

があると考えられる。以上の情報を無秩序に関数に記入すると、可読性、メンテナンス性が低下してしまうことがある。そのため、本案件では以上述べた要素を別々に管理するようにし、効率化を図った。その結果3.1とほぼ同様なメリットが得られた。

3.3. 座標変換

タッチパネル操作の場合、押下位置を座標で指定することが多いが、スクリプトに座標を直接記入すると、可読性が悪くなる上に、メンテナンスが難しいため、本案件では、座標を直接指定せず、文字列で押下位置を示し、実行時に文字列を座標に変換する方法を取った。試みた変換方法には以下があった。

- マクロ定義：実行前に座標に直接名前をつける。実行時にマクロ定義ファイルを引いて座標に変換する。押下位置が頻繁に変らない場合に有効
- 画像変換：実行前に押下位置の画像を切り取ってファイルに保存し、ファイル名に意味のある名前をつけておく。実行時に指定されたファイル名から保存しておいた画像を読み込む。次に読み込んだ画像を現画面から探し、見つかった場所を押下する。押下位置が変っても UI（画像）さえ変らなければ変更する必要はない。押下位置の UI（画像）が頻繁に変らない場合に有効
- 連続番号：実行前に画面要素にある規則で連続番号を振り、各番号に名前をつける。実行時に名前から番号に変換し、現画像を画像処理し画面要素の該当番号が示すところを押下する。画面要素の順番さえ変らなければ、UI と座標が変っても変更する必要はない。変更するときにも番号の書き換えだけで済むため、メンテナンスが簡単である。しかし、高精度で汎用性のある画像処理技術が必要だ。

以上の三つの手法はいずれも一長一短あるが、それらを統合するとよりよい手法になる。

- 連続番号とマクロ定義の併用：連続番号を使用して画面要素を抽出する際、画面要素の認識率は100%にならないことが多い。誤判定が発生した場合でも正しい位置を押下できるように、文字列に連続番号だけでなく、座標も登録しておくことで精度をさらに上げる。座標は参考にしかならないため、座標が少しずれる程度で変更する必要はない。

・ 空気を読む変換方法：マクロ定義、画像変換、連続番号、さらに画面要素の配置などの情報を統合して、どの位置を押下したいかを汲み取って押下する※2。

※2本手法は現在まだ開発中

本案件では、テスト対象の状況を考慮し、連続番号法を採用した。その結果、押下位置の UI、座標、フォントが変っても、メンテナンスする必要はないため、メンテナンス工数を削減することができた。

3.4. テストスクリプトの自動生成

テストスクリプトを手動で作成する場合、作成、テストに時間がかかる上に、大量のメンテナンス作業が必要となってくる。そのため、本案件ではスクリプト量が最も多い試験手順層に対してテストスクリプトの自動生成を試みた。手順は以下に示す。

1. Excel ベースの試験仕様書を用意する。弊社の試験仕様書は一行が一テストケースに対応しており、列に試験概要、上位ドキュメント章番号、試験手順、期待値、テストコンテンツ URL、テスト実施者などの情報が入っている。
2. 使用されていない列の最も左寄りの列から操作を記入していく。
   操作の引数は前の列に記入されていることが多いので、省略して記入することが可能。
   また、スクリプト同士で似た操作があれば、フィルハンドルを押したままドラッグすることで簡単にコピーできる。
3. VBA でマクロを作成し、テストスクリプトを自動生成する。

テストスクリプト自動生成を試みた結果、生産性が自動生成しない場合 50～100 本/人日から約 200 本/人日となり、およそ 2 倍～4 倍まで上がった。

3.5. その他

ここまでに述べてきたのはテストスクリプトの特性に注目して考えられた手法であるが、テストスクリプトもソフトウェアであるため、ソフトウェア開発における一般的な手法も併用すると、更に効果があがる。

・ 上位ドキュメントとのトレーサビリティ確保。テストスクリプト変更となったときに変更箇所が特定でき、変更漏れを防ぐ
・ 分岐を減らし、スクリプト複雑度を低減
・ スクリプト可読性の向上

また、可能であればテストスクリプト設計前に既存バグについて調べておくと、以下のメリットがある。

・ 自動テストが可能な程度の安定性を持っているコンポーネントと持っていないコンポーネントの見分けがつき、テストスクリプト作成の優先度を決めることができる。
・ テストスクリプト設計自体に大きな影響を与えるバグがある場合、設計段階から考慮できる（例：極端に発熱量が多いというバグが未解決の場合、1 時間毎に試験を 10 分停止させてから復帰させるように設計しなければならない）。

4. テストスクリプト仕様の理想像

4.1. オブジェクト指向化

QC を含めて従来のテストスクリプト仕様はほとんどプロセス型である。これはテストスクリプトとはテスト手順に従って書いていけばよいという既成概念から来たものと考えられる。しかし、小規模なテストスクリプトであれば問題ないが、規模が大きくなると、プロセス型テストスクリプトに限界があり、3.1と3.2のように操作とデータの観点で人間がメンテナンスできる規模に小分けしなければならない。

そのため、テストスクリプト仕様をオブジェクト指向化することで、メンテナンス性のよいテストスクリプトが作成しやすくなる上に、オブジェクト指向の継承、ポリモーフィズムの特徴を応用すれば、テストスクリプトの流用もしやすくなると考えられる。

4.2. テストプラットフォームの構築

きちんと設計しなければテスト自動化は効果を挙げられない可能性があるため、毎回設計をする必要がある。しかし、設計できるスペシャリストがいるか、設計するだけの時間があるかなどの問題もあれば、設計したとしても、人間のミスによる設計の欠陥も考えられる。

そのため、業界で統一したオープンソースのテストプラットフォームが必要となってくると考えられる。開発エンジニアには開発用プラットフォームが揃っているのにもかかわらず、テストエンジニアにはテスト用のプラットフォームがないのが現状だ。テスト自動化なら最も基本の

・ データベース
・ 常時監視
・ テスト機材のドライバ
・ テストスクリプトインタプリターとデバッガー
・ その他テストスクリプト開発ツール

などの機能を揃えて提供し、また、誰でも開発ツールなどを開発して組み込めるようになれば、テスト自動化はしやすくなると考えられる。

5. 結論

実機を使用したテスト自動化では、テストスクリプトはテスト手順に沿って書いていけばよい、テスト手

順をツールで記録してそれを毎回再生すればよいなどの考え方があるが、大規模なテストでは、テスト自動化の効果を発揮できないどころか、逆効果になってしまうことさえある。

テスト自動化の効果を最大限に発揮するには、テストスクリプトの保守性､移植性､信頼性に重点を置き、きちんとした設計を行う必要がある。

本案件では、テストスクリプト設計、テストスクリプト自動作成を実施し、最終的にテスト工程で試験回数を倍近く増やし、手動により実施不可能なテストを可能にしながら、コストを 70%に削減し、製品品質の向上に貢献できた。

今後はさらにテストスクリプト設計を追及しながら、テスト工程の時間短縮、画像処理技術をテスト自動化への導入などに力を入れていきたい。

文　献

[1] 大西建児，勝亦匡秀，加藤大受，佐々木方規，鈴木三紀夫，町田欣史，湯本剛，吉澤智美，ソフトウェアテスト教科書 JSTQB Foundation（第 2 版），有限会社風工社（編），株式会社翔泳社，東京，2009

[2] http://www.jnovel.co.jp/qc/