Vol.2015-SE-189 No.16
2015/7/23

ソフトウェア設計ドキュメントを利用したテスト実行スクリプト生成技術の提案と評価

丹野 治門[1,a)]　張 暁晶[1]

概要：本研究は，Web アプリケーションを対象として，テスト工程におけるテスト実施の効率化に取り組む．テスト実施の労力を減らすための既存方法として，テスト自動実行ツールが入力とするスクリプトを作成し，テスト実行を自動で行う方法がある．この方法では，一度スクリプトを作成すれば二度目以降のテストを自動で行えるという利点があるが，初回のスクリプト作成や，仕様や設計が変更された時の，スクリプトのメンテナンスに労力がかかるという問題があった．本研究では，これらの問題を解決するため，ソフトウェア設計工程の成果物である設計ドキュメントから，テスト実行を行うスクリプトを自動生成する手法を提案する．本研究では，既存方法に対して提案手法がどれだけテスト実施の労力削減効果があるかの評価も行い，有効性を確認した．

キーワード：ソフトウェアテスト, テスト実行，テストスクリプト，設計ドキュメント，モデルベーステスト

Test Script Generation Based on Software Design Documents

Abstract: This research focuses on how to make test execution more efficient, when testing web applications. One existing approach for cost reduction is to produce test scripts, the inputs of the test execution tool, and to execute tests automatically. However, when applying this approach, making and maintaining the test scripts are costly tasks. To solve this problem, we propose an approach to automatically generate test scripts from design documents, which are artifacts of the design process, using model-based testing. We confirm the effectiveness of our approach by comparing it to the existing approach.

Keywords: Software Testing, Test Execution, Test Script, Design Document, Model-Based Testing

1. はじめに

ソフトウェアテストはソフトウェアの品質を確保する上で重要である．しかしながら，現状，テストは大部分が手動で行われており，とても労力がかかるという問題がある．テストはテスト設計とテスト実施に分けられる．テスト設計とはソフトウェア仕様に基づいたテストケースの網羅的な洗い出しや，各テストケースにおけるテストデータ作成などの作業であり，テスト実施は各テストケースを実際に実行し，動作結果を確認し合否判定を行う作業である．本研究では，特に手作業だと労力のかかるテスト実施の効率化に着目した．

テスト実施の労力を減らすための既存方法として，テスト自動実行ツール [1,3] が入力とするスクリプトを作成し，テスト実行を自動で行う方法があるが，この方法には以下のような問題点がある．

- 問題 (1) : スクリプト作成の労力がかかる．例えば，図 1 に示すような Web アプリケーションで，オペレータ情報を入力して登録ボタンを押すというテストを実施するスクリプトを作成する場合，それぞれの入力フィールドへテストデータを入力する操作や，ボタンを押すといった操作をスクリプトとして逐一記述していく必要があり，初回のスクリプト作成には手動の労力が多くかかってしまう．
- 問題 (2) : スクリプトメンテナンスの労力がかかる : 仕様や設計が変更された時の，スクリプトのメンテナンスにも労力がかかる．例えば，図 1 の Web アプリケーションの画面設計に変更があり，入力フィールドやボ

[1] NTT ソフトウェアイノベーションセンタ，東京都港区港南 2-13-34 NSS-II ビル 6F
[a)] tanno.haruto@lab.ntt.co.jp

タンなどの画面要素に変更があった場合，設計変更前の画面における画面要素を対象に作成されていたスクリプトは正常に動作しなくなる可能性があるため，設計変更に合わせて適切に修正する必要がある．

本研究の目的は，上述したような問題を解決し，ソフトウェア仕様に基づいた網羅的なテスト実施の労力を削減することである．本研究では，図 1 に示すような業務システムのフロントエンドである Web アプリケーションの，単一画面遷移の動作を確認するためのテストにおけるテスト実施をスコープとし，主にウォータフォール型開発のような設計ドキュメントなどの成果物が工程ごとに必ず作成される開発プロセスを対象とする．

本研究の貢献は以下の 2 点である．

(1) 本研究では，モデルベーステストの考え方に基づき，ソフトウェアの設計ドキュメントから，テスト実行に必要なスクリプトを自動で生成する手法を提案した．提案手法では，設計工程の成果物であるソフトウェア設計ドキュメントのみからテスト実行スクリプトを生成できるため，初回のスクリプト作成の労力がかからない．また，仕様変更があり設計ドキュメントが修正された場合でも，修正した設計ドキュメントからスクリプトを再生成するだけで，設計ドキュメントと整合性の取れた最新のスクリプトが得られる．

(2) 2 種類の簡単な Web アプリケーションを用い，既存方法に対して提案手法がどれだけテスト実施の労力を削減できるかを測定する評価を行い，提案手法の有効性を確認した．

本論文で提案しているテスト実行スクリプトの生成手法は，筆者らの過去の取り組み [15] のコンセプトに基づいており，本論文では関連研究の拡充 (2 章参照)，提案手法の手順の詳細化 (3 章参照)，評価の拡充 (4 章参照) を行っている．

以降，2 章では，スクリプト作成に関する既存技術について述べる．3 章で，本研究が提案する手法について紹介し，4 章では提案手法の評価とその結果について述べる．そして，最後に 5 章で本論文の結論を述べる．

2. 既存技術

本研究では，ソフトウェア仕様に基づいた網羅的なテスト実施の労力を削減することを目指しており，これに関連したテストスクリプト作成の既存技術について述べる．

前章で述べたように，テスト実施の労力を減らすための既存方法としては，テスト自動実行ツール [3] が入力とするスクリプトを作成し，テスト実行を自動で行う方法がよく用いられるが，以下のような問題点がある．

- 問題 (1) スクリプト作成の労力がかかる．
- 問題 (2) スクリプトメンテナンスの労力がかかる．

問題 (1) の解決を目指した既存方法として，キャプチャ&リプレイ機能（例 : SeleniumIDE [2]）を利用することが考えられる． キャプチャ&リプレイとは，画面操作を記録して，記録した内容を繰り返し実行できる機能である．例えば，この機能をもつツールである SeleniumIDE [2] では，画面操作を記録して，SeleniumWebDriver [3] というテスト自動実行ツールへの入力となるスクリプトを作成することができる．このような機能を用いると，2 回目以降のテスト実行を自動化することはできるが，はじめに記録させる画面操作は手動で行わなければならず，操作ミスによる記録のし直しなども発生するため，労力がかかってしまう．他の方法としては，条件分岐や繰り返し処理，サブルーチンなどを導入し，少ない記述量で目的の動作をするスクリプトを作成する構造化スクリプティング [4] という手法がある．この手法でスクリプトを作成すれば，スクリプト作成の労力を低減できるが，完全に無くすことはできない．

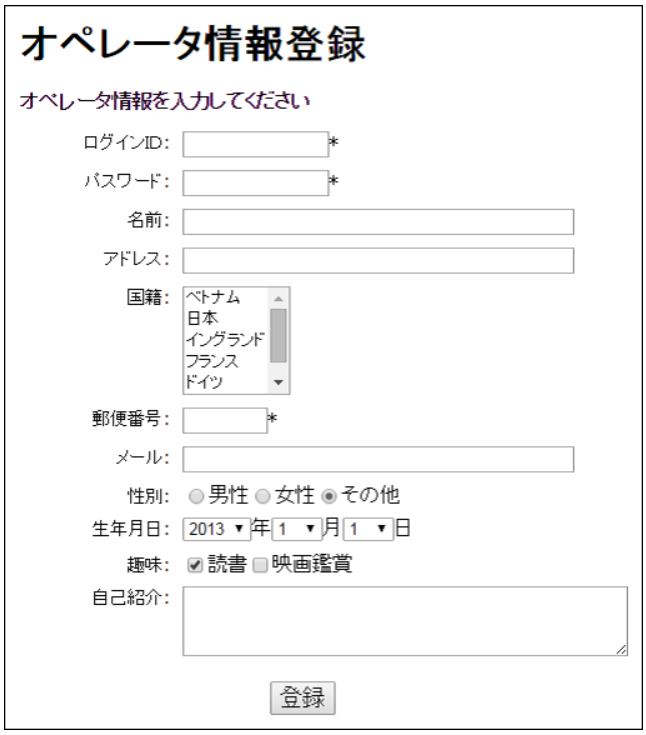

図 1 本研究が対象とする Web アプリケーションの例

問題 (2) の解決を目指した既存方法としては，設計やソフトウェアの変更に伴い，修正が必要となったスクリプトの箇所のサジェストにより，手動のスクリプト修正を効率化する手法 [9] や，作成済みのたくさんのスクリプトの共通部分を自動で見つけ出し共通化することでスクリプトの量を減らし，スクリプト修正を効率化する方法 [12]，モデルベーステストの考え方に基づき，テストケースをモデル化し，テストケースを構成する個々のステップ単位にスクリプトの単位操作を関連づけておくことで，GUI 変更があった場合，その変更に関連する単位操作のみをキャプチャー&リプレイで記録しなおせばスクリプトの修正がで

きる手法 [10] などがある．他の方法としては，スクリプトを作成するときに，ページオブジェクトパターン [11] という，画面を操作するためのページオブジェクト (例えば，Web アプリケーションならば，画面ごとのボタンやテキストフィールドなどの HTML 要素を保持し，各要素を操作するための API を定義する) とページオブジェクトを操作するテストシナリオを分離して記述するデザインパターンを利用する方法がある．この方法では，画面の設計が変更されたとき，修正個所をページオブジェクトのみに局所化できるため，スクリプト修正効率化において有効である．これらの方法を用いることでスクリプトメンテナンスの労力を少なくすることは可能であるが，全て自動化されているわけではないため，やはり手動修正のコストはかかってしまう．

問題 (1),(2) を一挙に解決する方法としては，Web アプリケーションのソースコードや実行情報から動的にテストスクリプトを生成する手法が存在する．例えば，Web アプリケーションのリバースエンジニアリング技術 [6,8] を利用し，ソフトウェアの動作モデルをソースコード，実行情報から構築し，そこからスクリプトを自動生成する手法 [5] が存在する．この方法では，スクリプトの大部分を自動生成することができ，かつ，ソフトウェアの設計変更に伴い，実装が変わった時に，その実装にあわせたスクリプトを生成できるため，現状の実装にあったスクリプトの作成という観点では，スクリプトメンテナンスのコストもかからないといえる．この方法で作成したスクリプトを用いると，アプリケーションの現状の実装に合わせた動作を行わせることができるため，ソフトウェア仕様との突合を行わなくても見つかるバグ（例えば，アプリケーションの異常終了など）の検出には有効であるが，「ソフトウェアの仕様通りに実装されているか」という観点のバグを見つけるには，実装 (ソースコードや実行時情報) に基づいて生成したスクリプトはそのまま信用して用いるわけにはいかないため，適切なレビュー等の追加タスクが必要になってしまう．

3. 提案手法

前章で述べた既存技術の問題点を解決し，ソフトウェア仕様に基づいた網羅的なテスト実施の労力を削減するため，本研究では，モデルベーステストの考え方に基づき，ソフトウェアの設計ドキュメントからテスト実行スクリプトを自動生成する手法を提案する．

本手法は，筆者らが過去に提案した設計ドキュメントからテストケースを自動生成機構 TesMa [17] を拡張して実現した．本手法では，ソフトウェア設計ドキュメント及びその内部モデルである設計モデル，テストモデルを拡張しており，スクリプト生成に必要な物理情報 (HTML における画面要素の ID 等) を保持できるようにし，各テストケースと対応づける点を特徴としている．この特徴により，以下のように既存方法の問題点を解決することができる．

- 問題 (1) の解決：ソフトウェア設計工程の成果物である設計ドキュメントのみからテスト実行スクリプトを生成できるため，初回のスクリプト作成の労力がかからない．
- 問題 (2) の解決：仕様変更があり設計が修正された場合でも，設計ドキュメントからスクリプトを再生成するだけで，設計ドキュメントと整合性の取れた最新のスクリプトが得られる．

これらの特徴により，提案手法を用いることで効率よくテスト実施を行うことが可能となる．

提案手法の全体像を図 2 に示す．本手法では，以下の手順で設計ドキュメントからスクリプト生成を行い，生成したスクリプトをテスト自動実行ツールで実行し，テスト担当者がテストの合否を判定するための，実行結果情報を得る．

- 手順 (1)：設計工程成果物である設計ドキュメント (図 2(a)) を TesMa が読み込み，設計モデル (図 2(b)) へと変換する．
- 手順 (2)：設計モデルの情報に基づき，テスト実行に必要なテスト手順，テストデータを含むテストケース群を網羅的に生成し，それらをテストモデル (図 2(c)) としてまとめる．
- 手順 (3)：テストモデルからテストケース表 (図 2(d) を生成する．
- 手順 (4)：テストモデルからテスト実行スクリプト (図 2(e)) を生成する．
- 手順 (5)：テスト自動実行ツールへ生成したスクリプトを入力として与え，テスト対象ソフトウェア (図 2(f)) に対するテストを実施し，実行結果情報 (図 2(g)) を記録する．
- 手順 (6)：テストの実行結果情報をテスト担当者が確認し，最終的なテストの合否判定を行い，合否判定結果 (図 2(h)) を得る．

この手順のうち (1)〜(5) は全て自動で行われ，(6) は手動で行う必要はある．

以降の節で，手順 (1)〜(6) について，図 1 を題材としながら，詳しい説明を行う．手順 (1)〜(6) において，本研究の特徴的な部分は手順 (1) において入力となる設計ドキュメント，手順 (4) である．

3.1 手順 (1)：設計ドキュメントから設計モデルを生成

手順 (1) では，設計ドキュメントを入力し，TesMa が内部的に用いる設計モデルを生成する．提案手法の入力となる設計ドキュメントは，画面設計書である．画面設計書は Excel ブックであり，画面要素一覧のシートと，画面入力制約のシートで構成される．画面要素一覧シートは，テキストボックスや，ボタンなど，画面の各構成要素に関する

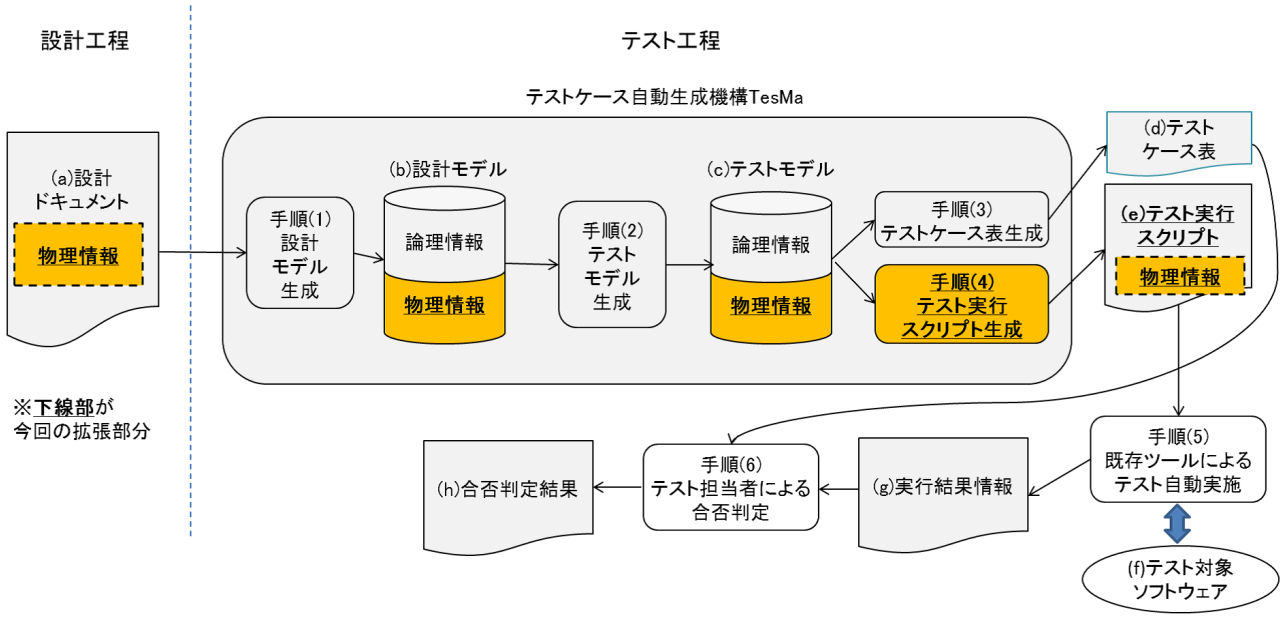

図 2 提案手法の全体像

定義を記述したものであり，画面入力制約シートは，画面要素一覧シートで定義された各画面要素の対する入力制約を定義したものである．これらの設計ドキュメントは，実際に開発現場で用いられている設計ドキュメントを参考にし，ドキュメント要素の記述形式の規定，そしてドキュメント要素への ID 付与等の点で見直しを行っており，機械処理できるようになったもの [16] を用いている．提案手法では，外部設計における論理情報 (画面要素名) と，詳細設計相当の物理情報 (HTML における物理 ID であり，id 属性もしくは name 属性が使える) の対応を記述可能にし，この情報を用いることで，実際にブラウザ上で実行可能なスクリプトを生成することができる．

図 1 の Web アプリケーション画面の画面要素一覧シートを図 3 に示す．図 1 のログイン ID を入力する画面要素の設計情報は，図 3 で，物理 ID は「userid」，画面要素名は「ログイン ID」，項目種別は「テキストボックス」というように記述する．また，この画面要素一覧シートに対応する画面入力制約シートは図 4 のように記述する．例えば，画面要素「ログイン ID」は，「文字列型」であり，「文字数範囲」として最小文字数は 5，最大文字数は 12 という入力範囲の情報が記載されている．

項番	物理ID	画面要素名	項目種別
1	userid	ログインID	テキストボックス
2	passid	パスワード	テキストボックス
3	username	名前	テキストボックス
4	address	アドレス	テキストボックス
5	country	国籍	リストボックス
6	zip	郵便番号	テキストボックス
7	email	メール	テキストボックス
8	gender	性別	グループ
9	msex	性別#男性	ラジオボタン
10	fsex	性別#女性	ラジオボタン
11	year	性別#その他	ラジオボタン
12	month	年	コンボボックス
13	day	月	コンボボックス
14	book	日	コンボボックス
15	movie	読書	チェックボックス
16	link	映画鑑賞	チェックボックス
17	about	自己紹介	テキストエリア
18	register	登録ボタン	送信ボタン

図 3 画面要素一覧シートの例

3.2 手順 (2)：設計モデルからテストモデルを生成

手順 (2) では，設計モデルからテストケースの集合であるテストモデルを生成する．テストケースは，画面設計書の画面入力制約シートにおける各制約ごとに，同値分割，境界値分析 [7] に基づき，既存のテストデータ生成技術 [13, 14] を用いて網羅的に生成される．各テストケースはテストデータそれぞれに紐づき 1 つずつ生成される．

例えば，図 4 の画面入力制約シートのログイン ID に関する文字数範囲の制約ならば，同値分析に基づいて，5 文字〜12 文字という制約を満たすテストデータ (例えば，”abcdef”) と条件を満たさないテストデータ (例えば，”a”) に関するテストケースがそれぞれ 1 つずつ生成される．そして，境界値分析に基づいたテストデータならば，「最少文字数 5」に着目した”abcd”,”abcde”などのテストデータ (及びテストケース) が網羅的に生成される．

項番	画面要素	型	チェックルール名	条件パラメータ
1	ログインID	文字列	文字数範囲	最小文字数=5, 最大文字数=12
2	パスワード	文字列	文字数範囲	最小文字数=8 最大文字数=16
3	名前	文字列	チェック無し	
4	アドレス	文字列	チェック無し	
5	国籍	国籍	チェック無し	
6	郵便番号	文字列	半角数字のみ	
7	メール	文字列	チェック無し	
8	性別	性別	チェック無し	
9	年	年	チェック無し	
10	月	月	チェック無し	
11	日	日	チェック無し	
12	読書	真偽値	チェック無し	
13	映画鑑賞	真偽値	チェック無し	
14	自己紹介	文字列	チェック無し	

図 4 画面入力制約シートの例

No.	系	名称	画面要素名	値
1	正常	入力値	ログインID	“tanaka”
		入力値	パスワード	“abc”
		...	...	...
		入力値	自己紹介	“efg”
2	準正常	入力値	ログインID	“abcd”
		入力値	パスワード	“abc”
		...	...	...
		入力値	国籍	日本
		...	...	...
		入力値	読書	TRUE
		入力値	映画鑑賞	FALSE
		入力値	自己紹介	“efg”
...	...	...	...	...

図 5 テストケース表の例

3.3 手順 (3)：テストモデルからテストケース表を生成

手順 (3) では，手順 (2) で生成したテストモデルから，テストケース表を生成する．テストケース表は，テストケースの一覧を帳票形式にまとめたもので，各テストケースにおいて使用されたテストデータ（画面要素に対応する入力値）が記載されている．テストケース表は，手順 (6) でテスト担当者が，各テストケースのテスト実施結果を確認する際に利用する．

テストケース表の例を図 5 に示す．例えば，テストケースの No2 は，ログイン ID の入力制約「最少文字数 5」に着目し，境界値分析に基づいて生成されたものであり，「最少文字数 5」を違反しているため，準正常系（エラー）となるテストケースである．

3.4 手順 (4)：テストモデルからテスト実行スクリプトを生成

手順 (4) では，テストモデルからテスト自動実行ツールが実行可能なテスト実行スクリプトを生成する．本手法では，テスト自動実行ツールとして，キーワード駆動テストの考え方に基づく GUI テスト自動実施フレームワーク Open2Test [1] を採用し，Open2Test が入力とするスクリプトを生成できるようにした．

図 5 のテストケース No.2 に対応するスクリプトを図 6 に示す．スクリプト本体 (図 6 の (ア)Test Script) では操作対象のオブジェクトに対してどのようなアクションを行うかを示しており，画面要素とその ID の対応表は別ファイル (図 6 の (イ)Object Repository) となっている．例えば，図 6(ア) の 1 行目は，ログイン ID という画面要素に対応するブラウザ上の TextBox へ”abcd”というテストデータを入力することを示す．このようなスクリプトを生成し，テスト自動実行ツールへ入力として与えることで，画面の各画面要素の必要な箇所へ，テストデータである入力値を次々に挿入し，最後に「登録ボタン」を押す．といった一連の操作を自動で行うことが可能となる．提案手法では，このようなスクリプトを全て，設計工程成果物である設計ドキュメント (画面設計書) から自動で生成することが可能であるため，スクリプト作成や，設計変更に伴うスクリプト修正の手間が発生することはない．

3.5 手順 (5)：テスト実行スクリプトを用いてテスト自動実施

手順 (5) では，手順 (4) で生成した各テストケースごとのスクリプトを，テスト自動実行ツールへ入力することで，テスト対象のソフトウェアが，設計通りの動作をするかの確認を行い，テストを実施する．

テスト対象のソフトウェアは，設計ドキュメントで定義した通りに，Web アプリケーションの各画面要素へ画面要素一覧シートで定義した物理 ID が付与されるよう実装される．そのため，スクリプトに記述された物理 ID の情報から，対応する画面要素を特定することが可能であるため，スクリプトを用いてテスト対象のソフトウェアを自動で操作することが可能となる．テスト自動実行ツールで，各テストケースを実施した後に，遷移した画面のスクリーンショットが，テスト実行結果の情報として保存される．

図 1 の Web アプリケーションの画面を表示している HTML ファイルを図 7 に示す．図 7 の点線で囲っている部分が，各画面要素の物理 ID である．例えば，ログイン ID を入力する画面要素に対応するタグには，3 で定義した通り，「userid」が id 属性として記述されている．ログイン ID 以外の画面要素についても同様である．例えば，図 5 のテストケース No.2 に対応するスクリプトが実行されると，これは「ログイン ID」の長さの制約に違反するテストデー

（ア）Test Script

実行	キーワード	操作対象オブジェクト	アクション
r	perform	Page0:ログインID	set:“abcd”
r	perform	Page0:パスワード	set:”abc”
・・・	・・・	・・・	・・・
r	perform	Page0:性別#男性	click
r	perform	Page0:年	select:2015
r	perform	Page0:月	select:1
r	perform	Page0:日	select:1
r	perform	Page0:読書	ckeck:ON
r	perform	Page0:映画化鑑賞	check:OFF
r	perform	Page0:自己紹介	“efg”
r	perform	Page0:登録ボタン	click
・・・	・・・	・・・	・・・

（イ）Object Repository

Object Name	Object Type	Object Path
Page0:ログインID	TextBox	id=userId
Page0:パスワード	TextBox	Id=passid
・・・	・・・	・・・
Page0:性別#男性	RadioButton	Id=msex
Page0:年	ComboBox	id=year
Page0:月	ComboBox	id=month
Page0:日	ComboBox	id=day
Page0:読書	CheckBox	id=book
Page0:映画化鑑賞	CheckBoc	id=movie
Page0:自己紹介	TextBox	id=about
Page0:登録ボタン	Button	id=register
・・・	・・・	・・・

図 6 スクリプトの例

タを用いたテストなので，もし，テスト対象のソフトウェアが設計通りに正しく実装されていれば，文字列の長さに違反しているというエラーを表示する画面へ遷移し，図 8 のようなスクリーンショットが，テストケース No.2 の実行結果情報として保存される．

3.6 手順 (6)：テスト担当者による合否判定

手順 (3) で生成したテストケース表と，手順 (5) の実行結果であるスクリーンショットを用いて，テスト担当者が，各テストケースの合否判定を行う．

例えば，図 5 のテストケース No.2 での合否判定を行う場合は，テストケース No.2 のテストデータ (ログイン ID の入力制約「最少文字数 5」を違反する) と，図 8 を突合し，この場合であれば，設計通りの正しい実行結果になっていることが確認でき，このテストケースは合格であるあるという判断を行うことができる．仮に，このテストケースの実行結果であるスクリーンショットが，「正しく登録を行えた場合に遷移する画面」であった場合は，ソフトウェアが設計通りに実装されておらず，設計と実装の不一致があったことがわかる．

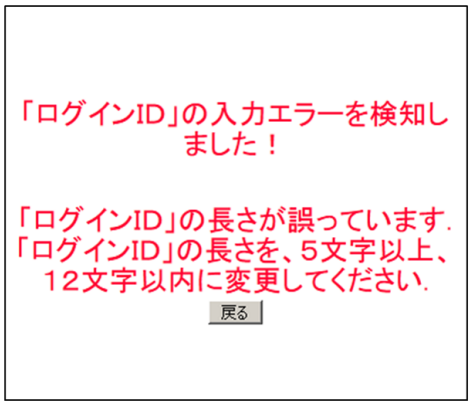

図 8 実行結果（スクリーンショット）の例

4. 評価

4.1 方法

テスト実施の労力削減効果を確認するため，以下の 3 つの方法を用いて，それぞれの方法でテスト実施にどれだけの労力 (手動の時間) がかかるかを測定した．

- (1) 提案手法：提案手法を用いてスクリプトを自動生成する方法．
- (2) 既存方法：スクリプトを全て手動で作成する方法．
- (3) 全て手動：スクリプトを用いず全て手動でテスト実施を行う方法．

テスト設計は，筆者らが過去に開発したテスト設計自動化ツール [17] により自動で行うことを前提とした．テスト自動実行ツールとしては，Open2Test framework for Selenium Web Driver(V2) [1] を用い，標準で提供されているキーワードのセットを使用した．

また，今回の評価では，以下の 2 種類の Web アプリケーションから以下のような画面を 1 つ選び評価を行った．

- 画面 (A)：図 1 に示したような評価用に作成した単純な Web 画面 (画面要素は 13)，テストケース数は 60.
- 画面 (B)：JavaPetStore の 1 画面 (画面要素は 8)，テストケース数が 29.

JavaPetStore は OSS であり，設計書は存在しないため，動的解析により提案手法が入力とする形式の設計ドキュメントを作成した．評価作業は，全て Web アプリケーション開発の経験がある開発者 1 名により行っている．

4.2 結果

評価結果を図 1 に示す．図において，準備とはテストを実際に実行する前のテストスクリプト作成やデバッグ作業など，実行はテストの実行，そして，合否判定はテスト実

```
<body onLoad="document.registration.userid.focus();">
    <h1>オペレータ情報登録</h1>
    <p>オペレータ情報を入力してください</p>
    <form name="registration" onSubmit="return formValidation();" method="get">
        <ul>
            <li><label for="userid">ログインＩＤ:</label></li>
            <li><input type="text" name="userid" id="userid" size="12" /><span>*</span></li>

            <li><label for="passid">パスワード:</label></li>
            <li><input type="password" name="passid" id="passid" size="12" /><span>*</span></li>
```

図 7 Web 画面の HTML ファイル

表 1 評価結果 (人時)

	準備	実行	合否判定	合計
画面 (A)，テストケース数=60，画面要素数=13				
(1) 提案手法	0.50	0.83	1.00	2.33
(2) 既存方法	10.50	0.50	1.00	12.00
(3) 全て手動	0.50	3.42	1.00	4.92
画面 (B)，テストケース数=29, 画面要素数=8				
(1) 提案手法	0.18	0.03	0.15	0.36
(2) 既存方法	1.17	0.03	0.15	1.35
(3) 全て手動	0.75	0.52	0.15	1.42

行結果確認にそれぞれ要した手動の作業時間を人時単位で示している．提案手法によって，既存方法での準備に要する作業時間を画面 (A) では 5% ，画面 (B) では 15% にまで削減し，テスト実施全体 (準備，実行，合否判定) では画面 (A) で 19% ，画面 (B) で 26% にまで作業時間を削減できた．これにより，提案手法では大きく労力を削減できていることが確認できた．

4.3 考察

提案手法については，画面 (A)，(B)，どちらの場合についても，他の方法よりも手動の作業時間を削減できることが確認できたが，スクリプトを手動で作成する場合 (表 1 の (2) 既存方法) と，スクリプト作成を一切行わず，全て手作業でテストを実施する場合 (表 1 の (3) 全て手動) では，画面 (A) と画面 (B) で違いが出た．画面 (A) では，スクリプトを手作業で作成する方が，全て手動で実施するより手間がかかっており，画面 (B) では逆の結果となっている．これは，画面 (A) は画面要素数が多く，1 つ 1 つの要素に対応するスクリプトのステップを作成するために時間がかかってしまったためと考えられる．

5. まとめ

本研究では，モデルベーステストの考え方に基づき，ソフトウェアの設計ドキュメントである画面設計書から，テスト実行に必要なスクリプトを自動で生成する手法を提案した．提案手法では，設計工程の成果物であるソフトウェア設計ドキュメントのみからテスト実行スクリプトを生成できるため，初回のスクリプト作成の労力がかからず，仕様変更があり設計ドキュメントが修正された場合でも，修正した設計ドキュメントからスクリプトを再生成するだけで，設計ドキュメントと整合性の取れた最新のスクリプトが得られるという特徴がある．2 種類の簡単な Web アプリケーションを用い，既存方法に対して提案手法がどれだけテスト実施の労力を削減できるかどうかを測定する評価を行い，既存方法に比べて 19%〜26% にまで，手動の時間を削減できていることを確認した．

今後は，他の既存方法 (キャプチャ＆リプレイ機能など) に対する追加の評価を行い，更なる提案手法の有効性検証を行っていきたい．また，将来的には合否判定を自動化するスクリプトも設計ドキュメントから生成できるようにして自動化の適用範囲を拡大することで，テストの効率化も進めていきたい．

参考文献

[1] Open2Test. http://www.open2test.org/.
[2] SeleniumIDE. http://www.seleniumhq.org/projects/ide/.
[3] SeleniumWebDriver. http://www.seleniumhq.org/projects/webdriver/.
[4] システムテスト自動化 標準ガイド. CodeZine BOOKS. 翔泳社, 2014.
[5] Stocco Andrea, Leotta Maurizio, Ricca Filippo, and Paolo Tonella. Why creating web page objects manually if it can be done automatically? In *2015 IEEE/ACM 10th International Workshop on Automation of Software Test*, AST '15, pp. 70–74, Firenze, Italy, 2015. IEEE Computer Society.
[6] Wontae Choi, George Necula, and Koushik Sen. Guided gui testing of android apps with minimal restart and approximate learning. In *ACM SIGPLAN Notices*, Vol. 48, pp. 623–640. ACM, 2013.
[7] Lee Copeland, 雅彦宗. はじめて学ぶソフトウェアのテスト技法. 日経 BP 社, 日経 BP 出版センター (発売), 2005.
[8] Giuseppe Antonio Di Lucca, Anna Rita Fasolino, and Porfirio Tramontana. Reverse engineering web applications: the ware approach. *Journal of Software maintenance and evolution: Research and practice*, Vol. 16, No. 1-2, pp. 71–101, 2004.
[9] Mark Grechanik, Qing Xie, and Chen Fu. Maintaining and evolving gui-directed test scripts. In *Proceedings of the 31st International Conference on Software Engineering*, ICSE '09, pp. 408–418, Washington, DC, USA, 2009. IEEE Computer Society.
[10] Ali Hokroh, 岸知二. 6L-5 Web アプリケーションの gui を

対象とした MBT 手法の提案. 全国大会講演論文集, mar 2015.

[11] Maurizio Leotta, Diego Clerissi, Filippo Ricca, and Cristiano Spadaro. Improving test suites maintainability with the page object pattern: An industrial case study. In *Proceedings of the 2013 IEEE Sixth International Conference on Software Testing, Verification and Validation Workshops*, ICSTW '13, pp. 108–113, Washington, DC, USA, 2013. IEEE Computer Society.

[12] Yandrapally Rahulkrishna, Sridhara Giriprasad, and Sinha and Saurabh. Automated modularization of gui test cases. In *Proceedings of the 37th International Conference on Software Engineering*, ICSE '15, pp. 44–54, Firenze, Italy, 2015. IEEE Computer Society.

[13] Xiaojing Zhang and Takashi Hoshino. A trial on model based test case extraction and test data generation. *Model-based Testing in Practice*, p. 51, 2010.

[14] 丹野治門, 張暁晶. ドメインテスト技法に基づく網羅的なテストデータ自動生成手法の提案. Technical Report 6, nov 2014.

[15] 丹野治門, 張暁晶. 5A-4 モデルベーステストに基づくテスト実行スクリプト生成手法の提案. 全国大会講演論文集, mar 2015.

[16] 丹野治門, 張暁晶, 星野隆. 結合テストにおけるテスト項目自動生成手法の提案と評価. 電子情報通信学会技術研究報告. SS, ソフトウェアサイエンス, Vol. 110, No. 227, pp. 37–42, oct 2010.

[17] 丹野治門, 張暁晶, 田端啓一, 生沼守英, 村主一仁. ソフトウェアの品質確保と開発コスト削減を目指したテスト自動化技術. NTT 技術ジャーナル, Vol. 25, No. 10, pp. 19–22, oct 2013.